

Received 11 May 2023, accepted 23 May 2023, date of publication 29 May 2023, date of current version 7 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3280638

RESEARCH ARTICLE

A New Digital Signature Primitive and Its Application in Blockchain

XIANG ZOU¹ AND PENG ZENG^{1,2} , (Member, IEEE)

¹Third Research Institute of Ministry of Public Security, Shanghai 200031, China

²Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China

Corresponding author: Peng Zeng (pzeng@sei.ecnu.edu.cn)

This work was supported in part by the 2014 Shanghai Leading Talents Program, in part by the Key Laboratory of Information Network Security of Ministry of Public Security (The Third Research Institute of Ministry of Public Security) under Grant C22600, and in part by the Shanghai Natural Science Foundation under Grant 23ZR1417800.


ABSTRACT In this paper, we propose a new digital signature primitive, called expander signature, and discuss its application in blockchain. The most promising advantage of expander signature is that a signer can generate all signatures at once using a powerful computer, and stores expander keys personally. Each time the signer wants some of his signatures to be verified, he releases the related expander key. No matter when or where, the signer can do this via a resource-limited device, for example, a personal portable terminal. We formally define the syntax and security of expander signature. Under our precisely defined security model, we give generic constructions of expander signature from both public key infrastructure-based and identity-based signature schemes. The security of our constructed expander signature schemes rigorously depends on the underlying public key signature schemes. The expander keys do not leak any information about the signer's secret key and the size of the expander keys is constant no matter how many times the expander has been occurred. Finally, we give an application example of expander signature in blockchain.

INDEX TERMS Digital signature, expander signature, forward security, blockchain, smart contract.

I. INTRODUCTION

Digital signature is a powerful cryptographic mechanism to ensure the integrity of data. In a conventional digital signature scheme, a signer signs a message using his signing key SK so that the produced signature σ can be verified by his public key PK . Since SK of the signer is secret, nobody can forge a valid signature on behalf of the signer as long as SK is not compromised. The public key PK is usually assumed to be publicly available for all, so once the signature σ is generated, anybody can verify its validation.

All the digital signature schemes in the literature have a common characteristic that once a signature σ is published by the signer, it can be verified immediately using the signer's public key PK . It results in that these signature schemes fail to meet the following requirement, i.e., one produces a set S of signatures which can be divided into n subsets S_i , $1 \leq i \leq n$, with $S_1 \subset S_2 \subset \dots \subset S_n \subset S$ and

The associate editor coordinating the review of this manuscript and approving it for publication was Mueen Uddin .

only the signatures in each subset S_i can be verified when certain conditions are satisfied over time. We can think of this kind of digital signature as a fine-grained digital signature. Taking into account that the nature of the signature's verifying is gradually expanded, we call this kind of new signature primitive as expander signature.

In expander signature, the signer publishes an expander key ek each time when he wishes the signatures related to ek to be verified. With ek and the public key PK of the signer, the signatures related to ek can be verified. With the evolution of the process, the number of the verifiable signatures is increasing, and the expander stops when all the signatures have been verified.

Expander signature is specially demanded in some scenarios. For example, suppose the signer is a one-year loaner of bank B for buying a car from company C . The signer signs a buying car contract Con_{SC} with C and a 12-month repayment contract Con_{SB} with B . Con_{SB} stipulates the amount of money the signer should repay each month and how to deal with when the signer violates the items in the contract. When

the contract Con_{SC} is valid, the bank B transforms the whole car money to the company C . For the 12 repayment plans in the contract Con_{SB} , the signer generates 12 signatures, but they should not be verified unless the signer transforms his money to the bank B . In other words, the signer transforms his month repayment to B and releases a token ek (we call it an expander key later) each month. With the signer's public key pk and the token ek , anyone can verify the signer's signatures related to the current or previous ek . When all of the 12 signatures are verified according to the contract Con_{SB} , the signer will own the car; otherwise, the bank B can legitimately take back the car and auction it.

Nowadays many people around the world buy things (e.g. cars or houses) in instalments. It may benefit us if we record personal's repayment on blockchain. Since blockchain is of openness and unforgeability, individual honesty and trustworthiness can be publicly verified on blockchain. Whenever a signature is verified as valid, the transformation is recorded in the blockchain. It is obvious, in the above example of buying car, if the signer fails to transform the money to the bank B , or cannot provide a token to verify his signature, he will lose his money. As a result, there is a requirement to design a signature scheme that generates signatures for which only a part of them can be verified when some conditions are satisfied. However, as far as we know, there are no digital signature schemes satisfying the requirement and thus we try to solve this kind of scalability problem of blockchain in this paper.

Our construction of expander signature is generic since it can transform any traditional signature scheme to an expander signature scheme without losing its security. Besides, it is worthy of mentioning that the additional expander algorithm (please see the algorithm `Expand` in Section IV-A) is efficient since it only needs operations on a cryptographic collision-resistant hash function (e.g., MD5, SHA256, etc). It is generally considered that hash operation is a kind of lightweight computation and it has many important applications, for examples, message authentication, digital signature, encryption, password protection [1], [2], [3], [4], [5].

Another feature of our construction is that the size of expander keys is constant regardless how many times the expander occurs. Our expander algorithm works as follows. Suppose that there are n ordered tags $t_1, t_2, \dots, t_n \in \{0, 1\}^*$ in the system. The signer starts the expander algorithm by selecting a random seed $r \in \{0, 1\}^*$ and then computes the n expander keys ek_i , $1 \leq i \leq n$, in reverse order: $\text{ek}_n = H(r, (t_1, 1), (t_2, 2), \dots, (t_n, n))$, $\text{ek}_{n-1} = H(\text{ek}_n)$, $\text{ek}_{n-2} = H(\text{ek}_{n-1})$, \dots , $\text{ek}_1 = H(\text{ek}_2)$, where H is a cryptographic collision-resistant hash function. It is obvious that we have $\text{ek}_i = H^{[n-i+1]}(r, (t_1, 1), (t_2, 2), \dots, (t_n, n))$ for each $1 \leq i \leq n$, where the notation $H^{[j]}(x)$ denotes j consecutive applications of the hash function H on x (e.g., $H^{[1]}(x) = H(x)$ and $H^{[2]}(x) = H(H(x)) = H(H^{[1]}(x))$). With the one-way property of H , it is easy to compute $\text{ek}_{i-1} = H(\text{ek}_i)$ for a given ek_i , but cannot compute the pre-image ek_{i+1} of ek_i (i.e. the backward expander key). In this way, the size of the

expander keys remains constant (i.e. the output length of H). Figure 1 shows the expander key evolution paradigm.

Finally we mention that our proposed expander signature can be used in blockchain and make the blockchain more scalable. Though blockchain has been regarded to be suitable for many application scenarios [6], [7], [8], it needs to be improved further and the scalability is one of these problems to be addressed. There are some works discussed the scalability of payment channel in blockchain [9], but no works on solving the scalability of conditioned verification in blockchain up to now. In this paper, we try to implement expander signature in blockchain to achieve flexible verification of transactions. If the verification of the signatures under the expander key and all previous expander keys is successful according to the items in the loan contract, the money is transformed to the payee and then the transaction is recorded into the blockchain.

A. OUR CONTRIBUTIONS

In this paper we construct a new digital signature primitive, called expander signature. The most promising advantage of expander signature is that a signer can generate all signatures at once using a powerful computer, and store expander keys personally. Each time the signer wants some of his signatures to be verified, he can release the related expander key. No matter when or where, the signer can do this via a resource-limited device, for example, a personal portable terminal. Other merits of expander signature include that the expander keys do not leak any information about the signer's secret key and the size of the expander keys is constant no matter how many times the expander has been occurred.

Our contributions in this paper are summarized as below:

- We propose a new digital signature primitive, called expander signature, and discuss its application on blockchain. As an example, a smart contract instance is given in this paper.
- We formally define the syntax and security of expander signature.
- We construct expander signature schemes from traditional signature schemes, including public key infrastructure-based (PKI-based) and identity-based (ID-based) expander signature schemes. The security of the expander signature schemes rigorously depends on the underlying digital signature schemes. Our construction is simple and efficient.
- We instantiate two expander signature schemes based on BLC short signature scheme [10] and Boyen and Water's ID-based signature scheme [11].

II. RELATED WORKS

A. FORWARD-SECURE DIGITAL SIGNATURE

The notion of digital signature was first proposed by Diffie and Hellman in their classic paper [12]. Digital signature can be used to protect the authenticity and integrity of the message and has many important applications today. Forward-secure

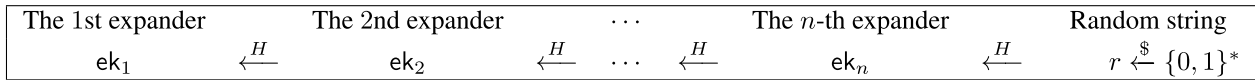


FIGURE 1. Paradigm for expander key chain: $ek_n = H(r)$ and $ek_i = H(ek_{i+1})$, $i = n - 1, n - 2, \dots, 2, 1$.

digital signature is a special digital signature which was proposed by Bellare and Miner [13], followed by the works [14], [15], [16], [17], [18], [19], [20], [21]. In a forward-secure digital signature scheme, the signer's public key is fixed, but the signing key is updated at regular time intervals. At each end of the interval, the signer computes a new signing key using the current signing key and time for the next interval, and deletes the current signing key. It is obvious that in such a scheme, the compromise of the current signing key does not enable an adversary to forge signatures pertaining to the past (the time prior to key exposure) so that forward security achieves. However, the existing forward-secure digital signature schemes in the literature only consider the update of signing keys with time and thus the signer suffers from the burden of constantly updating signing keys.

B. PUNCTURABLE ENCRYPTION

Puncturable encryption (PE) was first proposed by Green and Miers [22]. It provides a new mechanism for achieving forward-secure encryption in store and forward messaging systems. PE can be viewed as a form of tag-based encryption [23] with an additional efficient puncture algorithm. Secret keys in PE can be repeatedly and sequentially punctured at many different points, replicating the experience of normal message deletion [24], [25], [26]. More times the secret keys are punctured, the less ciphertexts can be decrypted. Followed of the work [22], Sun et al. [27] proposed the first symmetric puncturable encryption (SPE) using pseudorandom function, and further realised a backward-secure searchable encryption based on SPE. Though it is a relative new cryptographic primitive, PE has been found many applications, such as designing chosen-ciphertext secure fully homomorphic encryption [28], fully forward-secure key exchange for 0-RTT [29], [30], and forward and backward secure private searchable encryption [27], [31]. However, there are no works on exploring digital signature with the similar property as PE, which also has its special applications.

In this work, we start to study the new primitive by first giving the formal definition of expander signature and its security model. Then we construct generic expander signature schemes from traditional PKI-based and ID-based signature schemes. Finally, considering the importance of smart contracts to blockchain [32], [33], [34], [35], we design a smart contract based on expander signature as an application example of expander signature in blockchain.

III. PRELIMINARY AND DEFINITIONS

We denote by $|\mathcal{S}|$ the cardinality of a finite set \mathcal{S} and $x \xleftarrow{\$} \mathcal{S}$ the operation of choosing an element x from \mathcal{S} uniformly at

random. Further, we denote by $\mathcal{N} = \{1, 2, \dots, n\}$ the set of the first n positive integers and $\mathcal{T} = \{(t_i, i) \mid i \in \mathcal{N}\}$ the set of n tag-index pairs.

A. EXPANDER SIGNATURE

Definition 1: A PKI-based expander signature scheme is a tuple PKES = (KeyGen, Expand, Sign, Verify) of four algorithms defined as follows:

- **KeyGen**(par, U) \rightarrow (pk, sk): On input a public parameter par and a user U , it outputs a public-private key pair (pk, sk) for U , where sk and pk are for signing and verifying, respectively. We assume that par includes the system's security parameter κ and the description of the tag-index pair set \mathcal{T} .
- **Expand**(par, (t_i, i)) \rightarrow ek_i : On input the public parameter par and a tag-index pair $(t_i, i) \in \mathcal{T}$, the algorithm outputs an expander key ek_i on tag t_i .
- **Sign**(par, sk, (t_i, i) , M_i) \rightarrow σ_i : On input the public parameter par, a signing key sk, a tag-index pair $(t_i, i) \in \mathcal{T}$, and a message M_i to be signed, the algorithm outputs a signature σ_i on M_i conditioned on the i -th tag t_i . σ_i can be verified by pk and ek_i .
- **Verify**(par, pk, ek_i , M_i , σ_i) \rightarrow $\{1, 0\}$: On input the public parameter par, a verifying key pk, an expander key ek_i , a message M_i and its signature σ_i , the algorithm outputs 1 if the verification on σ_i is successful; Otherwise, it outputs 0.

Correctness. We say that a PKI-based expander signature scheme PKES = (KeyGen, Expand, Sign, Verify) is correct if the following equation holds unconditionally: for any public-private key pair (pk, sk) = KeyGen(par, U) of user U , message M_i , tag-index pair $(t_i, i) \in \mathcal{T}$,

$$\text{Verify}(\text{par}, \text{pk}, \text{Expand}(\text{par}, (t_i, i)), M_i, \sigma_i) = 1,$$

where $\sigma_i = \text{Sign}(\text{par}, \text{sk}, (t_i, i), M_i)$.

Definition 2: An ID-based expander signature scheme is a five-tuple IBES = (Setup, KeyGen, Expand, Sign, Verify) which can be defined as following:

- **Setup**(1^κ) \rightarrow (par, msk): On input a security parameter κ , it outputs the public parameter par and a master secret key msk. As in Definition 1, we assume that par includes the system's security parameter κ and the description of the tag-index pair set \mathcal{T} .
- **KeyGen**(par, msk, ID) \rightarrow sk: On input the public parameter par, the master secret key msk, and an identity ID of user U , it outputs the private key sk for U .
- **Expand**(par, (t_i, i)) \rightarrow ek_i : It is the same as the algorithm Expand in PKES.
- **Sign**(par, sk, (t_i, i) , M_i) \rightarrow σ_i : It is the same as the algorithm Sign in PKES.

- $\text{Verify}(\text{par}, \text{ID}, \text{ek}_i, M_i, \sigma_i) \rightarrow \{1, 0\}$: On input the public parameter par , an identity ID , an expander key ek_i , a message M_i and its signature σ_i conditioned on ek_i , it outputs 1 if the verification on σ_i is successful; Otherwise, it outputs 0.

Correctness. We say that an ID-based expander signature scheme $\text{IBES} = (\text{Setup}, \text{KeyGen}, \text{Expand}, \text{Sign}, \text{Verify})$ is correct if the following equation holds unconditionally: for any identity ID , message M_i , and tag-index pair $(t_i, i) \in \mathcal{T}$,

$$\text{Verify}(\text{par}, \text{ID}, \text{Expand}(\text{par}, (t_i, i)), M_i, \sigma_i) = 1,$$

where $(\text{par}, \text{msk}) = \text{Setup}(1^k)$, $\text{sk} = \text{KeyGen}(\text{par}, \text{msk}, \text{ID})$, and $\sigma_i = \text{Sign}(\text{par}, \text{sk}, (t_i, i), M_i)$.

B. SECURITY MODEL

In PKES (or IBES), the expander keys $\text{ek}_i, i \in \mathcal{N}$, are evolving which means that, given the i -th expander key ek_i , it is easy to compute the j -th expander key as $\text{ek}_j = H^{[i-j]}(\text{ek}_i)$ for $j < i$, but no polynomial probability time (PPT) adversary can compute any k -th expander key ek_k for $k > i$ with non-negligible probability. In this sense, the security of PKES includes two parts: (1) the existential unforgeability under adaptive chosen-message attack (EUF-CMA); and (2) the one-way property of the expander keys. In order to formally describe these actions, we pin down an appropriate security model for PKES based on the idea from [36] (the security model for IBES is similar).

Let \mathcal{F} be a forger against $\text{PKES} = (\text{KeyGen}, \text{Expand}, \text{Sign}, \text{Verify})$ who knows the public tag-index pair set $\mathcal{T} = \{(t_i, i) \mid i \in \mathcal{N}\}$. The EUF-CMA security of PKES can be formally defined by the following game $\text{Game}_{\text{PKES}, \mathcal{F}}^{\text{EUF-CMA}}$ interacted between a challenger \mathcal{C} and \mathcal{F} .

Game $_{\text{PKES}, \mathcal{F}}^{\text{EUF-CMA}}$: The challenger \mathcal{C} first runs the algorithm $\text{KeyGen}(\text{par}, U)$ to get a public-private key pair (pk, sk) for the signer U . Meanwhile, \mathcal{C} initializes an empty list $\mathcal{L}_{\text{sign}}$ to record \mathcal{F} 's signing queries on behalf of U . Then \mathcal{C} outputs pk to \mathcal{F} and allows \mathcal{F} to adaptively make the following queries:

- Expander key query $\mathcal{O}_{\text{ek}}(t_i, i)$: On input a tag-index pair (t_i, i) , \mathcal{C} returns \perp if \mathcal{F} has made an expander key query $\mathcal{O}_{\text{ek}}(t_j, j)$ on a tag-index pair (t_j, j) with $j \geq i$; Otherwise, it returns an expander key $\text{ek}_i = \text{Expand}(\text{par}, (t_i, i))$ to \mathcal{F} .
- Signing query $\mathcal{O}_{\text{sign}}((t_i, i), M_i)$: On input a tag-index pair (t_i, i) and a message M_i from the message space \mathcal{M} , \mathcal{C} first checks if \mathcal{F} has made a signing query on the pair $((t_i, i), M_i)$. If yes, \mathcal{C} returns \perp . Otherwise, \mathcal{C} computes $\sigma_i = \text{Sign}(\text{par}, \text{sk}, (t_i, i), M_i)$ and returns it to \mathcal{F} . Meanwhile, \mathcal{C} adds the new item $((t_i, i), M_i, \sigma_i)$ into the list $\mathcal{L}_{\text{sign}}$.

Finally, \mathcal{F} outputs a forgery $F^* = ((t_{i^*}, i^*), M_{i^*}, \sigma_{i^*})$. We call \mathcal{F} wins the game $\text{Game}_{\text{PKES}, \mathcal{F}}^{\text{EUF-CMA}}$ if F^* satisfies the following conditions:

- 1) $(t_{i^*}, i^*) \in \mathcal{T}$ and $F^* \notin \mathcal{L}_{\text{sign}}$;
- 2) $\text{Verify}(\text{par}, \text{pk}, \text{ek}_{i^*}, M_{i^*}, \sigma_{i^*}) = 1$, where $\text{ek}_{i^*} = \text{Expand}(\text{par}, (t_{i^*}, i^*))$.

Definition 3: We say that PKES is EUF-CMA secure if for any PPT forger \mathcal{F} , the advantage, denoted by $\text{Adv}_{\text{PKES}}^{\text{EUF-CMA}}(\mathcal{F})$, that \mathcal{F} wins $\text{Game}_{\text{PKES}, \mathcal{F}}^{\text{EUF-CMA}}$ is negligible.

Remark 1: Note that the forger \mathcal{F} can even make an expander query on (t_{i^*}, i^*) as long as its forgery $F^* = ((t_{i^*}, i^*), M_{i^*}, \sigma_{i^*}) \notin \mathcal{L}_{\text{sign}}$.

For the one-way property of PKES, we define a game, denoted by $\text{Game}_{\text{PKES}, \mathcal{F}}^{\text{OW}}$, which is also interacted between a forger \mathcal{F} and a challenger \mathcal{C} . \mathcal{F} is allowed to adaptively make the same queries \mathcal{O}_{ek} and $\mathcal{O}_{\text{sign}}$ as in the game $\text{Game}_{\text{PKES}, \mathcal{F}}^{\text{EUF-CMA}}$. Then \mathcal{F} outputs an expander key ek_{i^*} . We call \mathcal{F} wins the game $\text{Game}_{\text{PKES}, \mathcal{F}}^{\text{OW}}$ if ek_{i^*} meets the following conditions:

- 1) $(t_{i^*}, i^*) \in \mathcal{T}$;
- 2) \mathcal{F} makes neither the query $\mathcal{O}_{\text{ek}}(t_i, i)$ nor the query $\mathcal{O}_{\text{sign}}((t_i, i), M_i)$ for any $i \geq i^*$;
- 3) For any message M_{i^*} , it holds the equation $\text{Verify}(\text{par}, \text{pk}, \text{ek}_{i^*}, M_{i^*}, \sigma_{i^*}) = 1$, where $\sigma_{i^*} = \text{Sign}(\text{par}, \text{sk}, (t_{i^*}, i^*), M_{i^*})$.

Definition 4: We say that PKES is of one-way property if for any PPT attacker \mathcal{F} , the advantage, denoted by $\text{Adv}_{\text{PKES}}^{\text{OW}}(\mathcal{F})$, that \mathcal{F} wins $\text{Game}_{\text{PKES}, \mathcal{F}}^{\text{OW}}$ is negligible.

IV. UNIVERSAL CONSTRUCTION OF EXPANDER SIGNATURE

In this section, we present a universal construction of expander signature. Using our method, any traditional digital signature scheme can be easily transformed to an expander signature scheme, meanwhile keeping the security of the expander signature scheme. Our construction is divided to two cases: PKI-based and ID-based.

A. UNIVERSAL CONSTRUCTION OF PKI-BASED EXPANDER SIGNATURE

Let $\text{PKS} = (\text{Gen}, \text{Sig}, \text{Ver})$ be a secure PKI-based digital signature scheme and H a cryptographic collision-resistant hash function. Based on PKS, our PKI-based expander signature scheme $\text{PKES} = (\text{KeyGen}, \text{Expand}, \text{Sign}, \text{Verify})$ can be constructed as follows:

- $\text{KeyGen}(\text{par}, U) \rightarrow (\text{pk}, \text{sk})$: It runs the algorithm $\text{PKS.Gen}(\text{par}, U)$ to generate a public-private key pair (pk, sk) for the user U .
- $\text{Expand}(\text{par}, (t_i, i)) \rightarrow \text{ek}_i$: For a tag-index pair $(t_i, i) \in \mathcal{T}$, the algorithm first checks if $i = 1$. If yes, it selects $r \xleftarrow{\$} \{0, 1\}^*$ and computes

$$\text{ek}_1 = H^{[n]}(r, (t_1, 1), (t_2, 2), \dots, (t_n, n)).$$

Then it outputs the first expander key ek_1 and establishes a list

$$\mathcal{L}_{\text{ek}} = \{r, (\text{ek}_1, 1)\}$$

locally. Otherwise (i.e. in the case of $i > 1$), it finds r from \mathcal{L}_{ek} and computes

$$\text{ek}_i = H^{[n-i+1]}(r, (t_1, 1), (t_2, 2), \dots, (t_n, n)).$$

Finally, it outputs the i -th expander key \mathbf{ek}_i and adds the new item (\mathbf{ek}_i, i) into $\mathcal{L}_{\mathbf{ek}}$.

- $\text{Sign}(\text{par}, \text{sk}, (t_i, i), M_i) \rightarrow \sigma_i$: It first runs the algorithm $\text{Expand}(\text{par}, (t_i, i))$ to generate the i -th expander key \mathbf{ek}_i . Then it returns the signature $\sigma_i = \text{PKS.Sig}(\text{par}, \text{sk}, H(M_i) \oplus \mathbf{ek}_i)$.
- $\text{Verify}(\text{par}, \text{pk}, \mathbf{ek}_i, M_i, \sigma_i) \rightarrow \{0, 1\}$: It first finds a public expander key \mathbf{ek}_j with the greatest index j and checks if the equation $\mathbf{ek}_j = H^{[i-j]}(\mathbf{ek}_i)$ holds. If not, it returns 0. Otherwise, it returns $\text{PKS.Ver}(\text{pk}, H(M_i) \oplus \mathbf{ek}_j, \sigma_i)$.

Remark 2: We mention that it is easy to combine the above two algorithms Expand and Sign to efficiently generate the n expander keys and the corresponding n signatures at once for some specific applications. Specially, it can reduce the number of hash operations from $n(n-1)/2$ to n for the generation of the n expander keys $\mathbf{ek}_i, i \in \mathcal{N}$. We denote by Expand-Sign the combined algorithm which can be described as follows:

- $\text{Expand-Sign}(\text{par}, \text{sk}, \mathfrak{M}) \rightarrow (EK, \Sigma)$: Given the public parameter par , the signing key sk , and the n message set $\mathfrak{M} = \{M_i \mid i \in \mathcal{N}\}$ to be signed, it chooses a random string $r \xleftarrow{\$} \{0, 1\}^*$ and computes the n -th expander key as $\mathbf{ek}_n = H(r, (t_1, 1), (t_2, 2), \dots, (t_n, n))$ and the corresponding signature as $\sigma_n = \text{PKS.Sig}(\text{par}, \text{sk}, H(M_n) \oplus \mathbf{ek}_n)$. Then for $i = n-1, n-2, \dots, 2, 1$, it computes $\mathbf{ek}_i = H(\mathbf{ek}_{i+1})$ and $\sigma_i = \text{PKS.Sig}(\text{par}, \text{sk}, H(M_i) \oplus \mathbf{ek}_i)$ in sequence. Finally, it outputs the expander key list $EK = (\mathbf{ek}_1, \mathbf{ek}_2, \dots, \mathbf{ek}_n)$ and the signature list $\Sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$.

B. UNIVERSAL CONSTRUCTION OF ID-BASED EXPANDER SIGNATURE

Let $\text{IBS} = (\text{Setup}, \text{Extract}, \text{Sig}, \text{Ver})$ be a secure ID-based signature scheme and H a cryptographic collision-resistant hash function. Based on IBS , the ID-based expander signature scheme $\text{IBES} = (\text{Setup}, \text{KeyGen}, \text{Expand}, \text{Sign}, \text{Verify})$ can be constructed as follows:

- $\text{Setup}(1^\kappa) \rightarrow (\text{par}, \text{msk})$: On input a security parameter κ , it runs $\text{IBS.Setup}(1^\kappa)$ to generate the public parameter par and the master secret key msk .
- $\text{KeyGen}(\text{par}, \text{msk}, \text{ID}) \rightarrow \text{sk}_{\text{ID}}$: It runs the algorithm $\text{IBS.Extract}(\text{par}, \text{msk}, \text{ID})$ to generate the private key sk_{ID} of the user U_{ID} with identity ID .
- $\text{Expand}(\text{par}, (t_i, i)) \rightarrow \mathbf{ek}_i$: It is the same as the one in PKES .
- $\text{Sign}(\text{par}, \text{sk}_{\text{ID}}, (t_i, i), M_i) \rightarrow \sigma_i$: It first runs the algorithm $\text{Expand}(\text{par}, (t_i, i))$ to get the expander key \mathbf{ek}_i and then returns $\sigma_i = \text{IBS.Sig}(\text{par}, \text{sk}_{\text{ID}}, H(M_i) \oplus \mathbf{ek}_i)$.
- $\text{Verify}(\text{par}, \text{ID}, \mathbf{ek}_i, M_i, \sigma_i) \rightarrow \{1, 0\}$: It first finds a public expander key \mathbf{ek}_j with the greatest index j and checks if the equation $\mathbf{ek}_j = H^{[i-j]}(\mathbf{ek}_i)$ holds. If not, it returns 0. Otherwise, it returns the result of $\text{IBS.Ver}(\text{par}, \text{ID}, H(M_i) \oplus \mathbf{ek}_j, \sigma_i)$.

V. SECURITY ANALYSIS

Theorem 1: Suppose that $\text{PKS} = (\text{Gen}, \text{Sig}, \text{Ver})$ is a (ε, t) EUF-CMA secure PKI-based signature scheme, then our universal expander signature scheme PKES on PKS (refer to Section IV-A) is $(\varepsilon, t + \mu \cdot t_e + \nu \cdot t_s)$ EUF-CMA secure, where μ (resp. ν) is the times of the expander key (resp. signing) queries and t_e (resp. t_s) is the running time of one expander key (resp. signing) query.

Proof: Let \mathcal{F} be a forger of PKES , \mathcal{B} and \mathcal{C} the simulators of PKES and PKS , respectively. During the whole game $\text{Game}_{\text{PKES}, \mathcal{F}}^{\text{EUF-CMA}}$ (refer to Section III-B), \mathcal{B} maintains two lists $\mathcal{L}_{\mathbf{ek}}$ and $\mathcal{L}_{\text{sign}}$, which are initially empty.

At first, \mathcal{C} runs PKS.Gen to generate the signer's public-private key pair (pk, sk) . Then \mathcal{C} sends pk to \mathcal{B} and \mathcal{B} forwards it to \mathcal{F} . \mathcal{F} adaptively makes the following queries to \mathcal{B} :

- Expander key query $\mathcal{O}_{\mathbf{ek}}(t_i, i)$ ¹: On receiving this kind of queries, \mathcal{B} first checks if $i = 1$. If yes, \mathcal{B} selects $r \xleftarrow{\$} \{0, 1\}^*$, computes

$$\mathbf{ek}_1 = H^{[n]}(r, (t_1, 1), (t_2, 2), \dots, (t_n, n)),$$

and sets $\mathcal{L}_{\mathbf{ek}} = \{r, (\mathbf{ek}_1, 1)\}$; Otherwise, i.e. in the case of $i \geq 2$, by looking up the list $\mathcal{L}_{\mathbf{ek}}$, \mathcal{B} checks if \mathcal{F} has made an expander key query on some tag-index pair (t_j, j) with $j \geq i$. If it did, \mathcal{B} returns \perp to \mathcal{F} ; Otherwise, \mathcal{B} gets the value r from the list $\mathcal{L}_{\mathbf{ek}}$ and returns the expander key

$$\mathbf{ek}_i = H^{[n-i+1]}(r, (t_1, 1), (t_2, 2), \dots, (t_n, n))$$

on the tag-index pair (t_i, i) to \mathcal{F} . Finally, \mathcal{B} records (\mathbf{ek}_i, i) in the list $\mathcal{L}_{\mathbf{ek}}$.

- Signing query $\mathcal{O}_{\text{sign}}((t_i, i), M_i)$: On receiving this kind of queries by the forger \mathcal{F} , \mathcal{B} first computes the expander key $\mathbf{ek}_i \leftarrow \text{Expand}(\text{par}, (t_i, i))$ and makes an original signing query for the message $H(M_i) \oplus \mathbf{ek}_i$ to \mathcal{C} . Then \mathcal{C} computes $\sigma_i = \text{PKS.Sig}(\text{par}, \text{sk}, H(M_i) \oplus \mathbf{ek}_i)$ for \mathcal{B} . Finally, \mathcal{B} returns σ_i to \mathcal{F} and updates $\mathcal{L}_{\text{sign}} = \mathcal{L}_{\text{sign}} \cup \{(t_i, i), M_i, \sigma_i\}$.

Finally, the forger \mathcal{F} outputs a forgery $F^* = ((t_i^*, i^*), M_{i^*}, \sigma_{i^*})$ with the following conditions:

- 1) $(t_i^*, i^*) \in \mathcal{T}$;
- 2) $((t_i^*, i^*), M_{i^*}, \sigma_{i^*}) \notin \mathcal{L}_{\text{sign}}$;

On receiving a forgery $F^* = ((t_i^*, i^*), M_{i^*}, \sigma_{i^*})$, \mathcal{B} computes $\mathbf{ek}_{i^*} = H^{[n-i^*+1]}(r, (t_1, 1), (t_2, 2), \dots, (t_n, n))$ and sends $(H(M_{i^*}) \oplus \mathbf{ek}_{i^*}, \sigma_{i^*})$ to \mathcal{C} as a forgery to PKS . The advantage that \mathcal{C} resolves the hard problem related to PKS is the same as that \mathcal{B} resolves that hard problem, too, with running time $t + \mu \cdot t_e + \nu \cdot t_s$, where μ (resp. ν) is the times

¹Note that \mathcal{F} is allowed to make expander key queries on any tag-index pairs not strictly according to the order of the tags. That is, \mathcal{F} can make expander key queries on (t_j, j) , $j \in \mathcal{N}$, even if he did not make expander key queries on tag-index pairs $(t_1, 1) \dots (t_{j-1}, j-1)$. But if \mathcal{F} has made an expander key query on (t_i, i) , he is not allowed to make queries on tag-index pairs $(t_1, 1) \dots (t_{i-1}, i-1)$, since he can compute these expander keys from \mathbf{ek}_i .

of the expander key (resp. signing) queries and t_e (resp. t_s) is the running time of one expander key (resp. signing) query.

Theorem 2: Suppose that $\text{IBS} = (\text{Setup}, \text{Extract}, \text{Sig}, \text{Ver})$ is a (ε, t) EUF-CMA secure ID-based signature scheme, then our universal expander signature scheme IBES (refer to Section IV-B) is $(\varepsilon, t + \mu \cdot t_e + \nu \cdot t_s)$ EUF-CMA secure, where the notations μ, ν, t_e, t_s have the same meanings as in the above Theorem 1.

The proof of Theorem 2 is similar to the one of Theorem 1 and thus we omit it here.

Theorem 3: Both our universal expander signature constructions PKES and IBES are one-way secure, in the sense that given an expander key, no PPT adversary can compute a backward expander key, but anybody can compute the former expanded keys.

Proof: We take $\text{PKES} = (\text{KeyGen}, \text{Expand}, \text{Sign}, \text{Verify})$ as the example to prove this theorem. Let \mathcal{F} be a forger of PKES on a PKI-based signature scheme $\text{PKS} = (\text{Gen}, \text{Sig}, \text{Ver})$ and let \mathcal{B} and \mathcal{C} be the simulators of PKES and PKS , respectively. During the whole game $\text{Game}_{\text{PKES}, \mathcal{F}}^{\text{OW}}$ (refer to Section III-B), \mathcal{B} maintains two lists \mathcal{L}_{ek} and $\mathcal{L}_{\text{sign}}$, which are initially empty.

At first, \mathcal{C} runs PKS.Gen to generate the signer's public-private key pair (pk, sk) . Then \mathcal{C} sends pk to \mathcal{B} and \mathcal{B} forwards it to \mathcal{F} . During the game $\text{Game}_{\text{PKES}, \mathcal{F}}^{\text{OW}}$, \mathcal{F} can adaptively make the expander key query $\mathcal{O}_{\text{ek}}(t_i, i)$ and the signing query $\mathcal{O}_{\text{sign}}((t_i, i), M_i)$ as in the game $\text{Game}_{\text{PKES}, \mathcal{F}}^{\text{EUF-CMA}}$ (refer to the proof of Theorem 1). Finally, \mathcal{F} outputs an expander key ek_{i^*} which meets the following conditions:

- 1) $(t_{i^*}, i^*) \in \mathcal{T}$;
- 2) \mathcal{F} makes neither expander key query $\mathcal{O}_{\text{ek}}(t_i, i)$ nor signing query $\mathcal{O}_{\text{sign}}((t_i, i), M_i)$ for $i \geq i^*$;
- 3) For any message M_i and $1 \leq i \leq i^*$, it holds $\text{Verify}(\text{par}, \text{pk}, \text{ek}_i, M_i, \text{Sign}(\text{par}, \text{sk}, (t_i, i), M_i)) = 1$.

On receiving \mathcal{F} 's output ek_{i^*} , \mathcal{B} computes $\text{ek}'_{i^*} = H^{[n-i^*+1]}(r, (t_1, 1), (t_2, 2), \dots, (t_n, n))$. If $\text{ek}'_{i^*} = \text{ek}_{i^*}$, \mathcal{B} sends $(H(M_{i^*}) \oplus \text{ek}_{i^*}, \sigma_{i^*})$ to \mathcal{C} as a forgery to PKS . The advantage of \mathcal{C} 's resolving the hard problem related to PKS is the advantage of \mathcal{B} 's resolving that hard problem related to PKES , too. Otherwise, if $\text{ek}'_{i^*} \neq \text{ek}_{i^*}$, with the above condition 3), it must hold $\text{ek}'_{i^*-1} = H(\text{ek}_{i^*}) = H(\text{ek}'_{i^*})$, which means that \mathcal{B} finds a collusion of the hash function H . This contradicts the fact that H is a collision-resistant hash function. This shows the one-way property of PKES . The one-way security of IBES can be proved accordingly.

VI. INSTANTIATION OF EXPANDER SIGNATURES

A. INSTANTIATION OF PKES FROM BLC SHORT SIGNATURE

In this section, we briefly review BLC short signature scheme [10] and then give the concrete construction of our PKES scheme from it.

Let $\text{par} = (q, g, \mathbb{G}, \mathbb{G}_T, e, H)$ be the public parameter, where \mathbb{G} and \mathbb{G}_T are two multiplicative groups of same prime

order q , g is a generator of \mathbb{G} , $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a bilinear pairing, and $H : \{0, 1\}^* \rightarrow \mathbb{G}$ is a collision-resistant cryptographic hash function. Then BLC short signature scheme $\text{PKS} = (\text{Gen}, \text{Sig}, \text{Ver})$ can be described as follows [10].

- $\text{Gen}(\text{par}, U) \rightarrow (\text{pk}, \text{sk})$: On input the public parameter par and a user U , it chooses $x \xleftarrow{\$} \mathbb{Z}_q^*$ and sets the public-private key pair as $(\text{pk} = g^x, \text{sk} = x)$ for U , where \mathbb{Z}_q^* is the multiplicative group of the finite field \mathbb{Z}_q .
- $\text{Sig}(\text{par}, \text{sk}, M) \rightarrow \sigma$: On input the public parameter par , the signer's private key sk , a message $M \in \{0, 1\}^*$, it computes $\sigma = H(M)^{\text{sk}} = H(M)^x$.
- $\text{Ver}(\text{par}, \text{pk}, M, \sigma) \rightarrow \{0, 1\}$: On input the public parameter par , a public key pk of U , a message M and its signature σ , it checks if the following equation holds:

$$e(g, \sigma) \stackrel{?}{=} e(\text{pk}, H(M)). \quad (1)$$

If yes, the algorithm outputs 1; and 0, otherwise.

Based upon the above $\text{PKS} = (\text{Gen}, \text{Sig}, \text{Ver})$ [10], our expander signature scheme $\text{PKES} = (\text{KeyGen}, \text{Expand}, \text{Sign}, \text{Verify})$ can be constructed as below:

- $\text{KeyGen}(\text{par}, U) \rightarrow (\text{pk}, \text{sk})$: On input the public parameter par and a user U , it runs $\text{PKS.Gen}(\text{par}, U)$ to generate the public-private key pair $(\text{pk} = g^x, \text{sk} = x)$ for U .
- $\text{Expand}(\text{par}, (t_i, i)) \rightarrow \text{ek}_i$: If $i = 1$, the algorithm selects $r \xleftarrow{\$} \{0, 1\}^*$ and computes the expander key on the first tag-index pair $(t_1, 1)$ as

$$\text{ek}_1 = H^{[n]}(r, (t_1, 1), (t_2, 2), \dots, (t_n, n)).$$

Then it outputs ek_1 and establishes a list $\mathcal{L}_{\text{ek}} = \{r, (\text{ek}_1, 1)\}$. If $i > 1$, it finds the random string r from \mathcal{L}_{ek} and computes the expander key on the tag-index pair (t_i, i) as

$$\text{ek}_i = H^{[n-i+1]}(r, (t_1, 1), (t_2, 2), \dots, (t_n, n)).$$

Finally, it outputs ek_i and stores (ek_i, i) in \mathcal{L}_{ek} .

- $\text{Sign}(\text{par}, \text{sk}, (t_i, i), M_i) \rightarrow \sigma_i$: On input the public parameter par , the signer's private key sk , a tag-index pair $(t_i, i) \in \mathcal{T}$, and a message $M_i \in \{0, 1\}^*$, the algorithm computes $\text{ek}_i = \text{Expand}(\text{par}, (t_i, i))$ and generates the signature on M_i as $\sigma_i = H(H(M_i) \oplus \text{ek}_i)^{\text{sk}}$.
- $\text{Verify}(\text{par}, \text{pk}, \text{ek}_i, M_i, \sigma_i) \rightarrow \{0, 1\}$: On input the public parameter par , a verification public key pk , an expander key ek_i , a message M_i and its signature σ_i conditioned on ek_i , it first finds a public expander key ek_j with the greatest index j and checks whether $\text{ek}_j = H^{[i-j]}(\text{ek}_i)$. If not, it returns 0. Otherwise, it checks if the following equation holds:

$$e(g, \sigma_i) \stackrel{?}{=} e(\text{pk}, H(H(M_i) \oplus \text{ek}_i)). \quad (2)$$

If yes, the algorithm outputs 1; Otherwise, it outputs 0.

B. INSTANTIATION OF IBES FROM ID-BASED SIGNATURE

In this section, we instantiate our expander signature scheme IBES from Boyen and Waters’s ID-based signature scheme IBS = (Setup, Extract, Sig, Ver) which can be depicted simply as below [11]:

- **Setup**(1^κ) \rightarrow (par, msk): On input a security parameter κ , it first generates a six-tuple $(p, q, g, \mathbb{G}, \mathbb{G}_T, e)$, where p, q are two random κ -bit primes, \mathbb{G}, \mathbb{G}_T are two cyclic groups of order $p \cdot q$, $\mathbb{G}_p = \langle g \rangle$ is subgroup of \mathbb{G} with $|\mathbb{G}_p| = p$, and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a bilinear pairing. Then it selects $3 + k + m$ integers $\alpha, y', z', y_i, z_j \xleftarrow{\$} \mathbb{Z}_p^*$, $1 \leq i \leq k, 1 \leq j \leq m$, and computes $A = e(g, g)^\alpha, u' = g^{y'}, v' = g^{z'}, u_i = g^{y_i} (1 \leq i \leq k), v_j = g^{z_j} (1 \leq j \leq m)$. Finally, it outputs the master secret key as $\text{msk} = g^\alpha$ and the system public parameter as

$$\text{par} = (g, u', u_1, u_2, \dots, u_k, v', v_1, v_2, \dots, v_m, A).$$

Here, we assume that par implicitly includes the parameter k (the length of user identity), m (the length of message), and a description of $(p, q, g, \mathbb{G}, \mathbb{G}_T, e)$.

- **Extract**(par, msk, ID) \rightarrow sk_{ID} : On input the public parameter par, the master secret key msk, and an identity $\text{ID} = (\lambda_1, \lambda_2, \dots, \lambda_k) \in \{0, 1\}^k$ of user U_{ID} , it selects $r \xleftarrow{\$} \mathbb{Z}_p^*$ and outputs the private key sk_{ID} for U_{ID} as

$$\text{sk}_{\text{ID}} = (\text{msk} \cdot (u' \cdot U)^r, g^{-r}) \in \mathbb{G}^2,$$

where $U = \prod_{i=1}^k u_i^{\lambda_i}$.

- **Sig**(par, sk_{ID} , M) \rightarrow σ : On input the public parameter par, a signer’s private key $\text{sk}_{\text{ID}} = (\text{sk}_1, \text{sk}_2) \in \mathbb{G}^2$, and a message $M = (\mu_1, \mu_2, \dots, \mu_m) \in \{0, 1\}^m$ to be signed, it chooses $s \xleftarrow{\$} \mathbb{Z}_p^*$ and outputs the signature on M as

$$\sigma = (\text{sk}_1 \cdot (v' \cdot V)^s, \text{sk}_2, g^{-s}) \in \mathbb{G}^3,$$

where $V = \prod_{j=1}^m v_j^{\mu_j}$.

- **Ver**(par, ID, M, σ) \rightarrow $\{0, 1\}$: On input the public parameter par, an identity $\text{ID} = (\lambda_1, \lambda_2, \dots, \lambda_k) \in \{0, 1\}^k$, a message $M = (\mu_1, \mu_2, \dots, \mu_m) \in \{0, 1\}^m$ and its signature $\sigma = (\sigma_1, \sigma_2, \sigma_3) \in \mathbb{G}^3$, it outputs 1 if the equation

$$e(\sigma_1, g) \cdot e(\sigma_2, u' \cdot U) \cdot e(\sigma_3, v' \cdot V) \stackrel{?}{=} A \quad (3)$$

holds, where $U = \prod_{i=1}^k u_i^{\lambda_i}$ and $V = \prod_{j=1}^m v_j^{\mu_j}$. Otherwise, it outputs 0.

Based on the scheme IBS = (Setup, Extract, Sig, Ver) [11], our ID-based expander signature scheme IBES = (Setup, KeyGen, Expand, Sign, Verify) can be constructed as follows.

- **Setup**(1^κ) \rightarrow (par, msk): It first runs IBS.Setup(1^κ) to generate the parameter par_1 and the master secret key msk. Then it chooses a collision-resistant cryptographic hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^m$. Finally, the

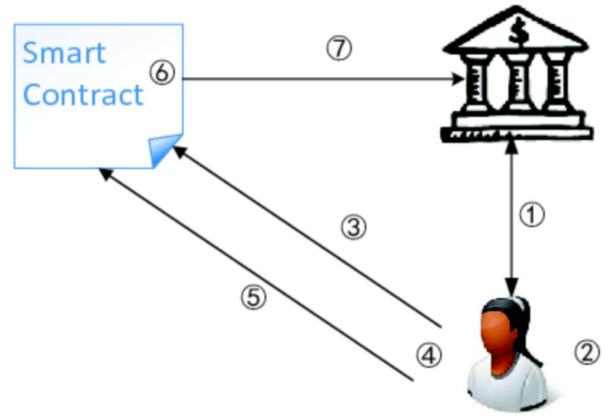


FIGURE 2. The interaction process of proposed smart contract.

algorithm outputs the system public parameter $\text{par} = (\text{par}_1, H)$ and the master secret key msk.

- **KeyGen**(par, msk, ID) \rightarrow sk_{ID} : For an identity ID of user U_{ID} , the algorithm runs IBS.Extract(par₁, msk, ID) to generate the private key sk_{ID} and sends it to U_{ID} via a secure channel.
- **Expand**(par, (t_i, i)) \rightarrow ek_i : It is the same as the algorithm PKES.Expand, except that the range of the hash function H is $\{0, 1\}^m$, not \mathbb{G} (please refer to Section VI-A). That is, the algorithm outputs an expander key ek_i and maintains a list $\mathcal{L}_{\text{ek}} = \{r, (\text{ek}_1, 1), (\text{ek}_2, 2), \dots, (\text{ek}_i, i)\}$.
- **Sign**(par, sk_{ID} , (t_i, i) , M_i) \rightarrow σ_i : It computes $\text{ek}_i = \text{Expand}(\text{par}, (t_i, i))$ and outputs the signature $\sigma_i = \text{IBS.Sig}(\text{par}, \text{sk}_{\text{ID}}, H(M_i) \oplus \text{ek}_i)$ on M_i .
- **Verify**(par, ID, $\text{ek}_i, M_i, \sigma_i$) \rightarrow $\{0, 1\}$: It first finds a public expander key ek_j with the closest index j to i and checks whether $\text{ek}_j = H^{[i-j]}(\text{ek}_j)$. If not, it returns 0. Otherwise, it returns IBS.Ver(par, ID, $H(M_i) \oplus \text{ek}_i, \sigma_i)$.

Remark 3: The instantiations of PKES and IBES are computationally efficient since they only have a few more lightweight hash operations than the original schemes [10] and [11], respectively.

VII. APPLICATION ON BLOCKCHAIN: SMART CONTRACT INSTANCE

Blockchain is an underlying technology for digital currencies and it has attracted great attention from government departments, financial institutions and Internet giants. In this section, we focus on designing a smart contract SC with repayment transaction as an application example of expander signature in blockchain. The interaction process of SC includes the following seven steps (refer to Figure 2) in which we assume that the signer is a loanee from bank B.

- 1) The signer signs a contract Con_{SB} with the bank B. Con_{SB} specifies the items which should be followed by the signer and B, including how much money B should lend to the signer, how many months this money should be returned to B by the signer, the amount of money

```

pragma solidity ^0.8.18;
contract Paper{
    mapping(string => string[]) signatureMap;
    mapping(string => string[]) ekMap;
    mapping(string => address[]) bankAddr;
    mapping(string => uint[]) totalPay;
    mapping(string => uint[]) payPerMon;
    mapping(string => uint[]) payNum;

    //set the initial value
    function setContractValue(string _pubKey, uint _payPerMon, uint _payNum,
        address _bankAddr) public{
        payPerMon[_pubKey] = _payPerMon;
        payNum[_pubKey] = _payNum;
        totalPay[_pubKey] = 0;
        bankAddr[_pubKey] = _bankAddr;
    }

    //take the algorithm PKES.Expand-Sign in Section IV-A as the example to
    //upload all signatures and expander keys in a single call
    function uploadSignature(string _pubKey, string _priKey, string[] _tagMap)
    public{
        ekMap[_pubKey] = PKES.Expand-Sign(par, _priKey, _tagMap)[0];
        // par is the system parameter that every user knows
        signatureMap[_pubKey] = PKES.Expand-Sign(par, _priKey, _tagMap)[1];
    }

    //verify signature based on public key and expander key
    function verifySignature(string _pubKey, string _ek, uint _value, string _signature)
    private returns (bool){
        if(PKES.Verify(par, _pubKey, _ek, _value, _signature) == 1) return true;
        return false;
    }

    //verify if the enough money has been paid in current month
    function verify(string _pubKey, uint _idx, uint _value)
    payable public returns(bool){
        string[] storage signatures = signatureMap[_pubKey];
        string[] storage eks = ekMap[_pubKey];
        if(signatures.length == 0) return false;
        uint havePaied = totalPay[_pubKey];
        havePaied += _value;
        require(havePaied == _idx * payPerMon[_pubKey]);
        for(uint k = 0; k < _idx; k++){
            if(verifySignature(par, _pubKey, eks[k], _value, signatures[k]) == false)
                return false;
        }
        bankAddr[_pubKey].transfer(_value);
        totalPay[_pubKey] = havePaied;
        return true;
    }
}

```

FIGURE 3. Key pseudo-codes of proposed smart contract.

the signer should return to B each month, the amount of money the signer should pay to B if he fails to transform money before current transformation, and B 's privilege if the signer violates the contract.

- 2) The signer generates all signatures, each for one-month payment to B . These signatures could only be verified when the signer transforms his monthly payment to B .
- 3) The signer uploads all the signatures to the smart contract SC . SC initializes $idx = 0$ and $totalPay = 0$.
- 4) The signer transforms the current repayment to B .
- 5) The signer releases an expander key to SC . The expander key together with the signer's public key are used to verify all the transforms before current month.
- 6) On receiving the expander key, SC first checks if the signer has returned enough money to B (e.g. checking whether $totalPay = idx * payPerMonth$). If not, the signer should transform $idx * payPerMonth - totalPay$ to B in current month and the verification is on the

new message $idx * payPerMonth - totalPay$. Otherwise, SC computes all the previous expander keys, and then verifies all the previous months' signatures.

- 7) If the verification is successful, SC updates $idx = idx + 1$ and $totalPay = idx * payPerMonth$. Finally the money is transformed to B 's address.

Figure 3 gives some key codes of SC based on the Solidity programming language.

VIII. CONCLUSION

In this paper we proposed a new digital signature primitive, called expander signature, which allows a signer to generate all signatures and expander keys at once using a powerful computer. The signer needs only to release the related expander keys when some of his signatures to be verified. This can be done with a resource-limited device, for example, a personal portable terminal. Further the expander keys do not leak any information about the signer's secret key and the size of the expander keys is constant no matter how many times the expander has been occurred. We gave two general constructions of expander signatures with PKI-based and ID-based digital signatures and instantiated two expander signature schemes based on two traditional signature schemes. We formally defined the security model of expander signature and gave rigorous security proof. Finally we designed a smart contract as an application example of expander signature in blockchain. Future study includes the code implementation of the designed smart contract and a new design of expander signature which does not depend on any existing digital signature schemes.

REFERENCES

- [1] D. Yu, R.-H. Hsu, J. Lee, and S. Lee, "EC-SVC: Secure CAN bus in-vehicle communications with fine-grained access control based on edge computing," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 1388–1403, 2022, doi: 10.1109/TIFS.2022.3152405.
- [2] H. N. Noura, O. Salman, R. Couturier, and A. Chehab, "A single-pass and one-round message authentication encryption for limited IoT devices," *IEEE Internet Things J.*, vol. 9, no. 18, pp. 17885–17900, Sep. 2022, doi: 10.1109/JIOT.2022.3161192.
- [3] V. Srivastava, S. K. Debnath, B. Bera, A. K. Das, Y. Park, and P. Lorenz, "Blockchain-envisioned provably secure multivariate identity-based multi-signature scheme for Internet of Vehicles environment," *IEEE Trans. Veh. Technol.*, vol. 71, no. 9, pp. 9853–9867, Sep. 2022, doi: 10.1109/TVT.2022.3176755.
- [4] M. Rasori, M. L. Manna, P. Perazzo, and G. Dini, "A survey on attribute-based encryption schemes suitable for the Internet of Things," *IEEE Internet Things J.*, vol. 9, no. 11, pp. 8269–8290, Jun. 2022, doi: 10.1109/JIOT.2022.3154039.
- [5] S. Wiefeling, P. R. Jørgensen, S. Thunem, and L. L. Iacono, "Pump up password security! Evaluating and enhancing risk-based authentication on a real-world large-scale online service," *ACM Trans. Privacy Secur.*, vol. 26, no. 1, pp. 1–36, Feb. 2023, doi: 10.1145/3546069.
- [6] K. Qin, L. Zhou, and A. Gervais, "Quantifying blockchain extractable value: How dark is the forest?" in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 198–214, doi: 10.1109/SP46214.2022.9833734.
- [7] S. Qahtan, K. Y. Sharif, A. A. Zaidan, H. A. Alsatat, O. S. Albahri, B. B. Zaidan, H. Zulzalil, M. H. Osman, A. H. Alamoodi, and R. T. Mohammed, "Novel multi security and privacy benchmarking framework for blockchain-based IoT healthcare Industry 4.0 systems," *IEEE Trans. Ind. Informat.*, vol. 18, no. 9, pp. 6415–6423, Sep. 2022, doi: 10.1109/TII.2022.3143619.

- [8] M. K. Hasan, M. D. Akhtaruzzaman, S. R. Kabir, T. R. Gadekallu, S. Islam, P. Magalingam, R. Hassan, M. Alazab, and M. A. Alazab, "Evolution of industry and blockchain era: Monitoring price hike and corruption using BIoT for smart government and Industry 4.0," *IEEE Trans. Ind. Informat.*, vol. 18, no. 12, pp. 9153–9161, Dec. 2022, doi: [10.1109/TII.2022.3164066](https://doi.org/10.1109/TII.2022.3164066).
- [9] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 949–966, doi: [10.1145/3243734.3243856](https://doi.org/10.1145/3243734.3243856).
- [10] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *Proc. 7th Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Dec. 2001, pp. 514–532, doi: [10.1007/3-540-45682-1_30](https://doi.org/10.1007/3-540-45682-1_30).
- [11] X. Boyen and B. Waters, "Compact group signatures without random oracles," in *Proc. 25th Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, St. Petersburg, Russia, May 2006, pp. 427–444, doi: [10.1007/11761679_26](https://doi.org/10.1007/11761679_26).
- [12] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 6, pp. 644–654, Nov. 1976, doi: [10.1109/TIT.1976.1055638](https://doi.org/10.1109/TIT.1976.1055638).
- [13] M. Bellare and S. K. Miner, "A forward-secure digital signature scheme," in *Proc. 19th Annu. Int. Cryptol. Conf.*, Santa Barbara, CA, USA, Aug. 1999, pp. 431–448, doi: [10.1007/3-540-48405-1_28](https://doi.org/10.1007/3-540-48405-1_28).
- [14] H. Jingxin, "A new forward-secure digital signature scheme," in *Proc. Int. Workshop Anti-Counterfeiting, Secur. Identificat. (ASID)*, Kyoto, Japan, Apr. 2007, pp. 116–129, doi: [10.1109/iwasid.2007.373738](https://doi.org/10.1109/iwasid.2007.373738).
- [15] W. Tzeng and Z. Tzeng, "Robust forward-secure signature schemes with proactive security," in *Proc. 4th Int. Workshop Pract. Theory Public Key Cryptography*, Cheju Island, (South) Korea, Feb. 2001, pp. 264–276, doi: [10.1007/3-540-44586-2_19](https://doi.org/10.1007/3-540-44586-2_19).
- [16] M. Abdalla, S. K. Miner, and C. Namprempe, "Forward-secure threshold signature schemes," in *Proc. Cryptographer's Track RSA Conf.*, San Francisco, CA, USA, Apr. 2001, pp. 441–456, doi: [10.1007/3-540-45353-9_32](https://doi.org/10.1007/3-540-45353-9_32).
- [17] G. Itkis and L. Reyzin, "Forward-secure signatures with optimal signing and verifying," in *Proc. 21st Annu. Int. Cryptol. Conf.*, Santa Barbara, CA, USA, 2001, pp. 332–354, doi: [10.1007/3-540-44647-8_20](https://doi.org/10.1007/3-540-44647-8_20).
- [18] T. Malkin, D. Micciancio, and S. K. Miner, "Efficient generic forward-secure signatures with an unbounded number of time periods," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, Amsterdam, The Netherlands, Apr. 2002, pp. 400–417, doi: [10.1007/3-540-46035-7_27](https://doi.org/10.1007/3-540-46035-7_27).
- [19] X. Boyen, H. Shacham, E. Shen, and B. Waters, "Forward-secure signatures with untrusted update," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, Alexandria, VA, USA, Nov. 2006, pp. 191–200, doi: [10.1145/1180405.1180430](https://doi.org/10.1145/1180405.1180430).
- [20] B. Libert, J.-J. Quisquater, and M. Yung, "Forward-secure signatures in untrusted update environments: Efficient and generic constructions," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, Alexandria, VA, USA, Oct. 2007, pp. 266–275, doi: [10.1145/1315245.1315279](https://doi.org/10.1145/1315245.1315279).
- [21] M. Abdalla, F. B. Hamouda, and D. Pointcheval, "Tighter reductions for forward-secure signature schemes," in *Proc. 16th Int. Conf. Pract. Theory Public-Key Cryptography*, Nara, Japan, Feb. 2013, pp. 292–311, doi: [10.1007/978-3-642-36362-7_19](https://doi.org/10.1007/978-3-642-36362-7_19).
- [22] M. D. Green and I. Miers, "Forward secure asynchronous messaging from puncturable encryption," in *Proc. IEEE Symp. Secur. Privacy*, San Jose, CA, USA, May 2015, pp. 305–320, doi: [10.1109/SP.2015.26](https://doi.org/10.1109/SP.2015.26).
- [23] P. D. MacKenzie, M. K. Reiter, and K. Yang, "Alternatives to non-malleability: Definitions, constructions, and applications (extended abstract)," in *Proc. 1st Theory Cryptography Conf.*, Cambridge, MA, USA, Feb. 2004, pp. 171–190, doi: [10.1007/978-3-540-24638-1_10](https://doi.org/10.1007/978-3-540-24638-1_10).
- [24] J. Wei, X. Chen, J. Wang, W. Susilo, and I. You, "Towards secure asynchronous messaging with forward secrecy and mutual authentication," *Inf. Sci.*, vol. 626, pp. 114–132, May 2023, doi: [10.1016/j.ins.2023.01.052](https://doi.org/10.1016/j.ins.2023.01.052).
- [25] S. Nuta, J. C. N. Schuldt, and T. Nishide, "PoS blockchain-based forward-secure public key encryption with immutable keys and post-compromise security guarantees," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. 106, no. 3, pp. 212–227, 2023, doi: [10.1587/transfun.2022cip0016](https://doi.org/10.1587/transfun.2022cip0016).
- [26] Y. Wang, P. Wei, M. Miao, and X. Zhang, "Verifiable dynamic search over encrypted data in cloud-assisted intelligent systems," *Int. J. Intell. Syst.*, vol. 37, no. 12, pp. 11830–11852, Dec. 2022, doi: [10.1002/int.23065](https://doi.org/10.1002/int.23065).
- [27] S.-F. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Toronto, ON, Canada, Oct. 2018, pp. 763–780, doi: [10.1145/3243734.3243782](https://doi.org/10.1145/3243734.3243782).
- [28] R. Canetti, S. Raghuraman, S. Richelson, and V. Vaikuntanathan, "Chosen-ciphertext secure fully homomorphic encryption," in *Proc. 20th Int. Conf. Pract. Theory Public-Key Cryptogr. (PKC)*, Amsterdam, The Netherlands, Mar. 2017, pp. 213–240, doi: [10.1007/978-3-662-54388-7_8](https://doi.org/10.1007/978-3-662-54388-7_8).
- [29] F. Gunther, B. Hale, T. Jager, and S. Lauer, "0-RTT key exchange with full forward secrecy," in *Proc. 36th Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, Paris, France, Apr. 2017, pp. 519–548, doi: [10.1007/978-3-319-56617-7_18](https://doi.org/10.1007/978-3-319-56617-7_18).
- [30] D. Derler, T. Jager, D. Slamanig, and C. Striecks, "Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange," in *Proc. 37th Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, Tel Aviv, Israel, Apr. 2018, pp. 425–455, doi: [10.1007/978-3-319-78372-7_14](https://doi.org/10.1007/978-3-319-78372-7_14).
- [31] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Dallas, TX, USA, Oct. 2017, pp. 1465–1482, doi: [10.1145/3133956.3133980](https://doi.org/10.1145/3133956.3133980).
- [32] C. Wu, J. Xiong, H. Xiong, Y. Zhao, and W. Yi, "A review on recent progress of smart contract in blockchain," *IEEE Access*, vol. 10, pp. 50839–50863, 2022, doi: [10.1109/ACCESS.2022.3174052](https://doi.org/10.1109/ACCESS.2022.3174052).
- [33] T. Górski, "Reconfigurable smart contracts for renewable energy exchange with re-use of verification rules," *Appl. Sci.*, vol. 12, no. 11, p. 5339, May 2022, doi: [10.3390/app12115339](https://doi.org/10.3390/app12115339).
- [34] W. Xiong and L. Xiong, "Data trading certification based on consortium blockchain and smart contracts," *IEEE Access*, vol. 9, pp. 3482–3496, 2021, doi: [10.1109/ACCESS.2020.3047398](https://doi.org/10.1109/ACCESS.2020.3047398).
- [35] P. Santamaría, L. Tobarra, R. Pastor-Vargas, and A. Robles-Gómez, "Smart contracts for managing the chain-of-custody of digital evidence: A practical case of study," *Smart Cities*, vol. 6, no. 2, pp. 709–727, Feb. 2023, doi: [10.3390/smartcities6020034](https://doi.org/10.3390/smartcities6020034).
- [36] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, vol. 17, no. 2, pp. 281–308, Apr. 1988, doi: [10.1137/0217017](https://doi.org/10.1137/0217017).



XIANG ZOU received the Ph.D. degree in computer science and technology from the University of Science and Technology of China, Anhui, China, in 2004. He is currently a Research Professor with the Third Research Institute of the Ministry of Public Security. His current research interests include network information security, applied cryptography, and personal identity information protection.



PENG ZENG (Member, IEEE) received the Ph.D. degree in computer science and technology from Shanghai Jiao Tong University, Shanghai, China, in 2009. He is currently an Associate Professor with East China Normal University, Shanghai. His current research interests include applied cryptography, network information security, and coding theory.

...