Forward-secure Multi-user Aggregate Signatures based on zk-SNARKs

Jeonghyuk Lee, Jihye Kim, and Hyunok Oh

[1] Hanyang University, Seoul, Korea
[2] Kookmin University, Seoul, Korea
ahoo791@hanyang.ac.kr jihyek@kookmin.ac.kr hoh@hanyang.ac.kr

Abstract. As a solution to mitigate the key exposure problems in the digital signature, the forward security has been proposed. The forward security guarantees the integrity of the messages generated in the past despite leaks of a current time period secret key by evolving a secret key on each time period. However, there is no forward secure signature scheme whose all metrics have constant complexities. Furthermore, existing works do not support multi-user aggregation of signatures. In this paper, we propose a forward secure aggregate signature scheme utilizing recursive zk-SNARKs (zero knowledge Succinct Non-interactive ARguments of Knowledge), whose all metrics including size and time have $O(1)$. The proposed forward secure signature scheme can aggregate signatures generated by not only a single user but also multiple users. The security of the proposed scheme is formally proven under zero-knowledge assumption and random oracle model.

Keywords: Digital signature · Forward security · aggregate signature · zero-knowledge proof · zk-SNARK · recursive proof composition

1 Introduction

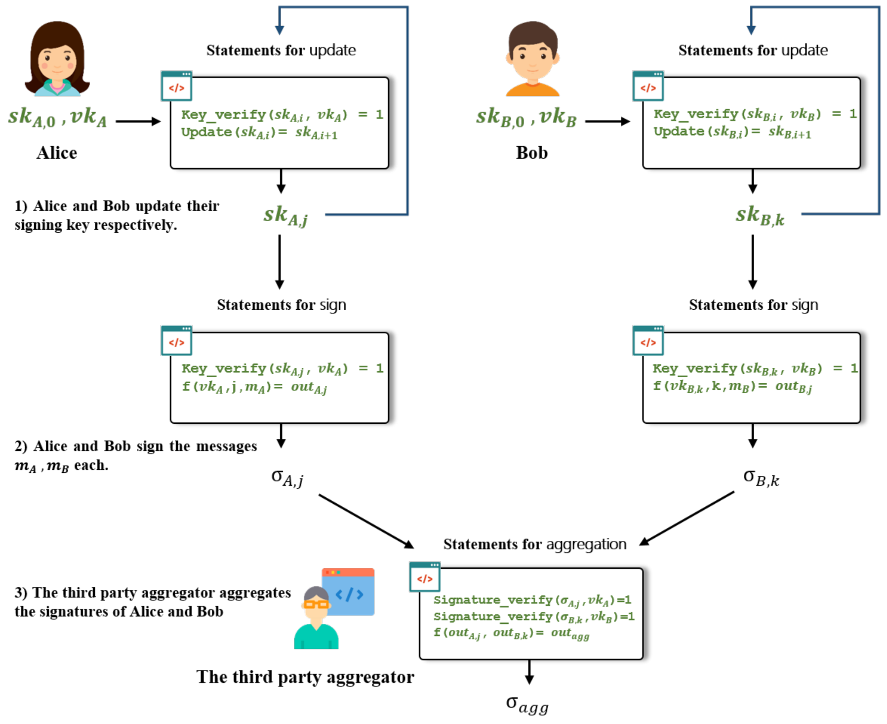
The forward security which assigns a different signing key to each time period alleviates a problem induced by the key exposure. After the security notion is firstly proposed by Anderson [3], several forward secure signature schemes have been devised [5, 2, 26, 32, 11, 37, 33] for decades. However, these works have a limitation in that the maximum time period T should be fixed in setup for constant public key size. It causes the necessity of remaking signing key and the public key when the maximum time period T ends. To avoid the problem, the maximum time period T is set to a large value, however, it results in inefficiency of the signature scheme of which complexities are total time period dependent [5, 2, 26]. For instance, Abdalla's construction [2] has $O(T)$ time complexity in setup, signing, and the verification. Although recent optimization of Abdalla's construction [30] reduces signing cost to $O(1)$ with some setup time trade-off, the verification cost remains $O(T)$. Several works attain a constant verification time complexities [26, 11, 25, 27]. However, they have a total time period dependent time complexity at least in one of other metrics.

In different view of the signature, an aggregation is a useful tool for alleviating a storage problem. Specifically, an aggregate signature is used in blockchain applications to reduce the space required for the signature storage, and it can mitigate the burden of blockchain network caused by the immense size of the transactions [44, 40]. Although there exist many aggregate signature schemes to merge the signatures [8, 12, 42, 22, 14], only a few researches support an aggregation of forward secure signatures [36, 31, 35, 43, 18]. However, the aggregate signature schemes are able to either aggregate signatures generated from a single user [36, 31, 35, 43] or multi-user signatures at which period is equivalent [36, 31, 35, 43]. This requirement makes difficult to be deployed in a decentralized environment where the time periods are often not synchronized. In this paper, we propose a forward secure multi-user aggregate signature scheme where all the complexities are totally independent of the time period T. The aggregation supports different messages and different users flexibly applicable for the decentralized environment. In our construction, we use simulation extractable zk-SNARK(zero-knowledge Succinct Non-interactive ARgument of Knowledge) [24, 9, 34, 28] as a building block of the signature scheme. zk-SNARK enables proving arbitrary statements, and the arbitrariness of the statements facilitates the removal of restrictions in existing signature schemes. That is, it is able to consider zk-SNARK proof as a signature if the proof proves the statements that suit the requirement of the signature. Naively, we can have an idea to construct a forward secure signature scheme by proving following statements.

– A public verification key and a secret signing key are well constructed.
– The secret signing key is connected with a message.
– The secret signing key is updated correctly.

Though all of the statements can be proven by simply including these statements in a zk-SNARK circuit, it is not enough for the forward secure signature since signing keys of all time periods are required as witnesses in the proof generation. Furthermore, it causes the inefficiency in that the circuit size increases proportional to the number of key updates. We resolve the issue by adopting PCD(Proof Carrying Data) [7] where a proof proves the verification result of the other proof to prove the update process without the previous signing key.

In Fig.1, we describe the flow of our forward secure aggregate signature construction. Sign, update, and aggregate algorithms are designed by subsuming zk-SNARK proof construction that proves required statements respectively. All the statements consist of verification of previous proof and additional computation. As described in 1) in Fig.1, when the initial signing key $sk_{A,0}$ and $sk_{B,0}$ are assumed to be composed of user secret value and the proof that proves a relation of user secret value and the verification key, the secret signing key can be updated recursively by proving statements for update. The statement stipulates that the verification result of the previous signing key is passed and the signing key of the next time period is generated correctly. By updating the signing key recursively, we can keep the circuit size constant regardless of how many times the signing key is updated. When the updated signing key $sk_{A,j}$ and the message m_A are given, the signature generation is conducted similar to the update.



1) Alice and Bob update their signing key respectively.

```
Key_verify($sk_{A,i}$, $vk_A$) = 1
Update($sk_{A,i}$)= $sk_{A,i+1}$
```

```
Key_verify($sk_{B,i}$, $vk_B$) = 1
Update($sk_{B,i}$)= $sk_{B,i+1}$
```

2) Alice and Bob sign the messages m_A , m_B each.

```
Key_verify($sk_{A,j}$, $vk_A$) = 1
f($vk_A$,j,$m_A$)= $out_{A,j}$
```

```
Key_verify($sk_{B,k}$, $vk_B$) = 1
f($vk_{B,k}$,k,$m_B$)= $out_{B,j}$
```

3) The third party aggregator aggregates the signatures of Alice and Bob

```
Signature_verify($\sigma_{A,j}$,$vk_A$)=1
Signature_verify($\sigma_{B,k}$,$vk_B$)=1
f($out_{A,j}$, $out_{B,k}$)= $out_{agg}$
```

Fig. 1: Basic structure of our proposal

For instance, when Alice proves that the secret signing key for j period $sk_{A,j}$ is verified and the message m_A and the time period j is connected with the secret signing key, the proof can be a forward secure signature itself. In aggregation described in 3), the third party aggregator verifies $\sigma_{A,j}$ and $\sigma_{B,k}$ that are signature of Alice and Bob in j,k time period respectively. The third party aggregator proves the verification results and the aggregation of two signatures. That is, if two verification of signatures are converged into one zk-SNARK proof, it can be an aggregate signature itself. Since the aggregation needs only the public information required in the verification, it can be conducted publicly.

1.1 Our contributions

Generic construction of a forward secure signature using zk-SNARK We propose a generic construction of the forward secure signature via zk-SNARKs. In our construction, any kind of simulation extractable zk-SNARK from the preprocessing SNARKs such as GM17,KLO20 [24, 29] to universal structured reference string based SNARK such as MBKM19,GWC19,BBB+18 [38, 20, 13] can be utilized as a building block. Various zk-SNARK libraries such as libsnark [1], snarkjs [4] can be used in the signature construction freely.

$O(1)$ efficiency We construct a forward secure signature of which complexities are independent on the time period. In fact, our scheme does not require any maximum time period in the scheme setup. Our forward secure signature makes all of the metrics in setup, update, sign, and verify algorithms have $O(1)$ time complexities. The verification key size, secret key size and the signature size have $O(1)$ space complexities. Table 1 compares performance and space requirement of

Table 1: Performance and size comparison in forward secure signature schemes: T and l denote the maximum period and the message length.

	Ours	BM [5]	AR [2]	IR [26]	Boyen et al. [11]	KO17 [30]
Key generation time	$O(1)$	$O(lT)$	$O(lT)$	$O(lT)$	$O(\log T)$	$O(lT)$
Update time	$O(1)$	$O(l)$	$O(l)$	$O(lT)$	$O(1)$	$O(l)$
Signing time	$O(1)$	$O(T+l)$	$O(lT)$	$O(l)$	$O(\log T + l)$	$O(1)$
Verification time	$O(1)$	$O(T+l)$	$O(lT)$	$O(l)$	$O(1)$	$O(lT)$
Secret key size	$O(1)$	$O(l)$	$O(1)$	$O(1)$	$O(\log^2 T)$	$O(1)$
Verification key size	$O(1)$	$O(l)$	$O(1)$	$O(1)$	$O(\log T + l)$	$O(1)$
Signature size	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$

forward secure signature schemes [5, 2, 26, 11, 30]. While all metrics are constant in our signature scheme, the other schemes have at least one metric that is dependent on the maximum period T. Except Boyen et al. [11] and our scheme, at least one metric is $O(T)$ while the secret key size is not constant in [11].

Table 2: Comparison of forward secure aggregate signature schemes where n indicates the number of aggregated signatures.

	Ours	MT07 [36]	YNR12 [43]	KO19 [31]
Aggregation period	Any	Any	Sequential	Sequential
Aggregation user	Multiple	Single	Single	Single
Public aggregation	Yes	Yes	No	Yes
Key generation time	$O(1)$	$O(T)$	$O(T)$	$O(lT)$
Update time	$O(1)$	$O(1)$	$O(1)$	$O(l)$
Signing time	$O(1)$	$O(1)$	$O(1)$	$O(lT)$
Aggregation time	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Verification time	$O(n)$	$O(n)$	$O(n)$	$O(ln + lT)$
Verification key size	$O(1)$	$O(T)$	$O(T)$	$O(l)$
Signature size	$O(1)$	$O(1)$	$O(1)$	$O(1)$

Multi-user signature aggregation for any time period We propose a generic aggregate construction that aggregates multi-user signatures removing all restrictions on the time period and the message. While the time period and the message should be fixed [18] or the time periods should be sequential in existing schemes [43, 31], our scheme can aggregate signatures of different messages generated by multi-user in unsequential time periods. In summary, our public aggregation technique not only allows the aggregation of multi-user signatures without any constraint of the time period, and the message but maintains $O(1)$ complexity in the signature size and the key size. Table 2 shows the aggregation possibility, the performance, and the size requirement in various forward secure aggregate signature schemes. While our scheme and MT07 can aggregate signatures in arbitrary periods, YNR12 and KO19 can aggregate signatures in consecutive periods. In addition, our scheme can aggregate signatures generated by multiple users while the other forward secure schemes can only aggregate signatures generated by a single user. YNR12 scheme can aggregate signatures by only a signer while the other schemes allow anyone to aggregate signatures.

In summary, we present a new forward secure signature methodology whose complexities are fully independent of the time period and its aggregation has high flexibility. Our new forward secure signature scheme can be utilized usefully in a decentralized environment where entities have their own time periods independently. However, we should bear zk-SNARK computation that is relatively heavier than those of other schemes to obtain a time period independent efficiency and the flexibility in the aggregation. Though the computation is independent of the time period and message length, it is reliant on zk-SNARK circuit size. Thus, we measure the actual performance of our signature scheme via the implementation.

We describe related works in section 2. In section 3, we explain preliminaries of our design and define the security notion. We demonstrate specific construction of our signature scheme in section 4. Section 5 presents an extension of our forward secure signature scheme to the forward secure multi-user aggregate signature. Section 6 presents the security proofs of all constructions. Section 7 analyzes the experiment results of our scheme. We draw a conclusion in section 8.

2 Related work

The forward security notion is firstly introduced by Anderson [3] aiming to alleviate the damage from the key exposure problem. The forward security divides overall time periods into separate time period and utilizes a different secret key in each period. A subsequent secret key is derived from the one in the previous time period, and extracting the previous one from the current secret key is computationally hard. Because of the hardness of the extraction, a past signature cannot be fabricated even if the secret key in the current time period is exposed. Bellare and Miner [5] formalize the security notion and present the first practical forward secure signature scheme. After the formalization, many pieces of research try to improve the efficiency of forward secure signature since it is ideal that all the operations have $O(1)$ time complexity, $O(1)$ signature size and $O(1)$ key size respectively. Abdalla [2] reduces the secret key size from $O(l)$ to $O(1)$ where l is message length while maintaining all the time complexity of operations. While Kozlov et al. [32] design a forward secure signature scheme updating a key in $O(1)$ time complexity, it has a limitation in that its signing time complexity and verification time complexity are linear with overall time period T. Itkis and Reyzin [26] present a forward secure signature scheme whose signing and verification have the time complexity irrelevant to overall time period T while taking a trade-off in the update time. Malkin et.al [37] construct a generic forward secure signature scheme that has almost unbounded time period. Boyen et al. [11] propose the forward secure signature methodology whose signing key can be updated as an encrypted form. Kim and Oh [30] make AR [2] have the $O(1)$ signing time complexity with adding some computations in the setup time. Despite the results of many pieces of research, any research could not reach a methodology where all operations can be conducted in $O(1)$ time while maintaining $O(1)$ space.

Meanwhile, Several researches introduce aggregation technique to existing forward secure signature schemes to reduce a space required to store multiple signatures [35, 31]. Ma and Tsudik [36] firstly construct a forward secure aggregate signature, however, the methodology has $O(T)$ verification key space complexity. Although Ma [35] evolves BM [5] and AR [2] into a forward secure aggregate signature scheme that has a constant size of verification key, the scheme is ascertained insecure by Kim and Oh [31]. Yavuz et al. [43] propose the aggregation technique that can be conducted by only a single addition. However, the technique has a drawback in that the verification key size is linear with T. Kim and Oh [31] devise the aggregation technique where the verification key size is irrelevant to overall time period T. However, these techniques [43, 31] only allow the aggregation of single-user signatures whose time periods are consecutive.

We use simulation extractable zk-SNARK [24, 9, 34, 28] as a building block of signature scheme where many zk-SNARK schemes work in pairing-based environment [39, 23, 24, 28]. Gennaro et al.[21] firstly propose Non-interactive argument system where a general function is supported. Groth [23] reduces the number of verification equations from three to one and the number of elements of proof from eight to three. Groth and Maller [24] propose a simulation extractable zk-SNARK firstly maintaining the proof size as three. However, Groth and Maller construction should use SAP(Square Arithmetic Program) representation that incurs double size of common reference string of QAP(Quadratic Arithmetic Program) representation. Several pieces of research try to develop SE-SNARK that can be represented as QAP representation. Bowe and Gabizon [9] construct QAP based SE-SNARK that has five proof elements. Lipmaa [34] reduces the proof size required in QAP representation to four, and Kim et al. [28] have three proofs in QAP representation and a single verification equation.

We use PCD(Proof Carrying Data) as a specific building block of our construction. PCD is a special case of zk-SNARK, and the proof proves the result of verification of other proofs recursively. Chiesa and Tromer firstly define the notion firstly [16], and Bitansky et al. [7] devise the recursive proof composition where any zk-SNARK can be used as a builiding block. Ben-Sasson et al. [6] enhance the practicality of the notion via using 2-cycles of pairing friendly elliptic curves. In addition, Bowe et al. [10] propose the recursive proof composition that does not require a trusted setup using Bulletproof [13] and Sonic [38] as building blocks. Chiesa et al. [15] not only grant the transparency as Bowe's construction [10] but enable the recursive proof composition to work in post-quantum environments.

3 Background

3.1 Notation

We write $y \leftarrow x$ for substitution x on y. We write $y \leftarrow S$ for sampling y from S if S is a set. We write $y \leftarrow A(x)$ for a probabilistic algorithm on input x returning output y. When a probabilistic algorithm $A(x)$ has a private input r, we denote $A(x; r)$. We state $f(\lambda)$ is *negligible* if $f(\lambda) \approx 0$. We denote a concatenation as $||$. Given a scheme Π, its all operations are denoted by $\Pi.\mathsf{name}$. Let $\mathcal{R}$ be a relation generator that given a security parameter λ in unary returns a polynomial time decidable relation $R \leftarrow \mathcal{R}(1^\lambda)$. We denote $\mathcal{R}_\lambda$ as the set of relations that $\mathcal{R}(1^\lambda)$ outputs. We call ϕ the instance and w the witness for $(\phi, w) \in R$. We denote all of $\mathcal{A}$'s inputs and outputs for an algorithm $\mathcal{A}$ by $trans_{\mathcal{A}}$.

3.2 Simulation extractable zk-SNARK

Definition 1. *A zero-knowledge succinct non-interactive arguments of knowledge(zk-SNARK) for $\mathcal{R}$ is a set of quadruple algorithms* $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{SimProve})$ *as follows.*

– Setup *is a PPT setup algorithm that takes as input a relation* $R \in \mathcal{R}_\lambda$ *and returns a common reference string* crs *and a simulation trapdoor* τ.
– Prove *is a PPT algorithm that takes as input a common reference string* crs, *an instance* ϕ *and a witness* w *for* $(\phi, w) \in R$, *and returns a proof* π.
– Verify *is a deterministic polynomial time algorithm which takes as input a common reference string* crs, *an instance* ϕ *and a proof* π, *and returns 0(reject) or 1(accept).*
– $\mathsf{SimProve}$ *is a PPT algorithm which takes as input a common reference string* crs, *a simulation trapdoor* τ, *and an instance* ϕ. *The algorithm returns a simulated proof* π.

zk-SNARK Π satisfies completeness, knowledge soundness, zero-knowledge, and succinctness as described below.

Perfect completeness : Perfect completeness stipulates that a prover with a witness who is given a true statement can convince a verifier.

For all $\lambda \in \mathbb{N}$, for all $R \in \mathcal{R}_\lambda$ and for all $(\phi, w) \in R$:

$$\Pr[(crs, \tau) \leftarrow \mathsf{Setup}(R); \pi \leftarrow \mathsf{Prove}(crs, \phi, w) : \mathsf{Verify}(crs, \phi, \pi) = 1] = 1 \quad (1)$$

Computational soundness : Computational knowledge soundness states that the prover must know a witness and the witness should be extracted efficiently from a knowledge extractor. Proof of knowledge requests every adversarial prover $\mathcal{A}$ to generate an accepting proof, there must be an extractor $\chi_\mathcal{A}$ which outputs a valid witness taking a same input of $\mathcal{A}$. Formally, we define $\mathbf{Adv}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda) = \Pr[\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)]$ where $\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}$ is defined as follows.

Algorithm 1 Knowledge soundness game $\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}$

```
$\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)$
  $R \leftarrow \mathcal{R}(1^\lambda)$
  $(crs, \tau) \leftarrow \mathsf{Setup}(R)$
  $(\phi, \pi) \leftarrow \mathcal{A}(crs)$
  $w \leftarrow \chi_\mathcal{A}(trans_\mathcal{A})$
  $\mathcal{A}$ wins if $\mathsf{Verify}(crs, \phi, \pi) = 1$ and $(\phi, w) \notin R$ and fails otherwise.
```

An argument system Arg is considered computationally sound if for any PPT adversary adversary $\mathcal{A}$, there exists a PPT extractor $\chi_\mathcal{A}$ where $\mathbf{Adv}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda) \approx 0$

Perfect zero-knowledge : Perfect zero-knowledge stipulates that a proof does not disclose any information about the witness besides the truth of the instance.

The statement is certified by a simulator which cannot access a witness but has some trapdoor information that allows simulating proofs. Formally, we define $\mathbf{Adv}^{zk}_{Arg,\mathcal{A}}(\lambda) = 2\Pr[\mathcal{G}^{zk}_{Arg,\mathcal{A}}(\lambda)] - 1$ such that the game $\mathcal{G}^{zk}_{Arg,\mathcal{A}}$ is defined as follows.

Algorithm 2 Zero-knowledge game $\mathcal{G}^{zk}_{Arg,\mathcal{A}}$

```
$\mathcal{G}^{zk}_{Arg,\mathcal{A}}(\lambda)$
  $R \leftarrow \mathcal{R}(1^\lambda)$
  $(crs, \tau) \leftarrow \mathsf{Setup}(R)$
  $b \leftarrow \{0,1\}$
  if $b = 0$ then
    $P^b_{crs,\tau}(\phi_i, w_i)$ returns $\pi_i$ where $\pi_i \leftarrow \mathsf{Prove}(crs, \phi, w)$ and $(\phi_i, w_i) \in R$
  else
    $P^b_{crs,\tau}(\phi_i, w_i)$ returns $\pi_i$ where $\pi_i \leftarrow \mathsf{SimProve}(crs, \phi, \tau)$ and $(\phi_i, w_i) \in R$
  end if
  $b' \leftarrow \mathcal{A}^{P^b_{crs,\tau}(\phi_i, w_i)}$
  $\mathcal{A}$ wins if $b = b'$ and fails otherwise.
```

The argument system is considered perfect zero-knowledge if $\mathbf{Adv}^{zk}_{Arg,\mathcal{A}}(\lambda) = 0$ for all PPT adversaries $\mathcal{A}$.

Definition 2. *A simulation extractable zk-SNARK for* $\mathcal{R}$ $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{SimProve})$ *satisfies simulation-extractability described below.*

Simulation-Extractability : Simulation-Extractability stipulates that any adversary $\mathcal{A}$ who can access a simulated proof for a false instance cannot forge the proof to another proof for a false instance. Formally, we define $\mathbf{Adv}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda) = \Pr[\mathcal{G}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)]$ where the game $\mathcal{G}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}$ is defined as follows.

Algorithm 3 Simulation extractable knowledge soundness game $\mathcal{G}^{proof-ext}_{Arg,\mathcal{A},\chi_\mathcal{A}}$

```
$\mathcal{G}^{sound}_{Arg,\mathcal{A},\chi_\mathcal{A}}(\lambda)$
  $R \leftarrow \mathcal{R}(1^\lambda)$ , $Q \leftarrow 0$
  $(crs, \tau) \leftarrow \mathsf{Setup}(R)$
  repeat
    $\pi_i \leftarrow \mathsf{SimProve}(crs, \tau, \phi_i)$
    $Q \leftarrow Q \cup \{\phi_i, \pi_i\}$
  until $(\phi, \pi) \leftarrow \mathcal{A}^{\mathsf{SimProve}_{crs,\tau}}(crs)$
  $w \leftarrow \chi_\mathcal{A}(trans_\mathcal{A})$
  $\mathcal{A}$ wins if $\mathsf{Verify}(crs, \phi, \pi) = 1$, $(\phi, w) \notin R$ and $(\phi, \pi) \notin Q$ and fails otherwise.
```

zk-SNARK is considered simulation extractable if there exists an extractor $\chi_{\mathcal{A}}$, for any PPT adversary $\mathcal{A}$, where $\mathbf{Adv}^{proof-ext}_{Arg,\chi_{\mathcal{A}},\mathcal{A}}(\lambda) \approx 0$.

Proof-carrying data : Proof-carrying data(PCD) [16, 17] is a cryptographic primitive that guarantees the validity of all previous proofs via the recursive proof composition. It is a special case of zk-SNARK where the relation R is composed of some functions that are required for the verification of other proofs. To guarantee the verification result of the other proof in the proof circuit, the instance ϕ includes a proof set $\vec{\pi}$ and inputs needed for the verification of the proof additionally. It has a proof size independent of the number the recursion, and it has $O(1)$ verification time regardless of the number of recursion. PCD inherits all the properties of zk-SNARK such as the completeness, the knowledge soundness and zero-knowledge. The concrete proofs of these properties are described in Ben-Sasson et al.'s work [6].

3.3 Forward secure signatures

Definition 3. *A forward secure signature is a set of four algorithms* $\mathsf{FSS} = (\mathsf{Keygen}, \mathsf{Update}, \mathsf{Sign}, \mathsf{Verify})$ *where*

- Keygen *takes as input a security parameter* λ *and returns a key pair* sk_0, vk *the initial signing key and the verification key and the time period* j*.*
- Update *takes as input a secret key* sk_j*, the time period* j *and returns the secret key* sk_{j+1} *and the next time period* $j+1$*.*
- Sign *takes as input a message* m*, the secret key* sk_j*, the time period* j*, the verification key* vk *and returns* σ_j *that is a signature for time period* j*.*
- Verify *takes as input a message* m*, the time period* j*, the verification key* vk*, the signature* σ *and returns 1 if the* σ *is valid signature or 0, otherwise.*

We define the security of FSS similarly to the existing works do [5, 2, 26, 11].The only difference is that we do not assume the maximal period anymore. Informally, an adversary who wants to succeed a valid signature forgery executes chosen message attack cma until a secret signing key of the current time period is leaked. The adversary succeeds a valid forgery if the adversary generates a signature of the previous time period on a new message.

Formally, we define $\mathbf{Adv}^{fwsec}_{\mathsf{FSS},\mathcal{F}}(\lambda) = \Pr[\mathcal{G}^{fwsec}_{\mathsf{FSS},\mathcal{F}}(\lambda)]$ where the game $\mathcal{G}^{fwsec}_{\mathsf{FSS},\mathcal{F}}$ is defined in Algorithm 4.

The adversary $\mathcal{F}$ works in three phases : the chosen message attack cma phase, the break-in phase, $\mathsf{breakin}$, the forgery phase forge. FSS is considered forward secure if $\mathbf{Adv}^{fwsec}_{\mathsf{FSS},\mathcal{F}}(\lambda) \approx 0$ for any PPT adversary $\mathcal{F}$ where the execution time is at most t and the number of signing queries is at most q_{sig}.

Definition 4. *A hash function* H *is extractable for a ppt adversary* $\mathcal{A}$ *when there exists an extractor* ε *such that, for large enough security parameter* λ *and*

Algorithm 4 Forward security game $\mathcal{G}^{fwsec}_{\mathsf{FSS},\mathcal{F}}(\lambda)$

```
$\mathcal{G}^{fwsec}_{\mathsf{FSS},\mathcal{F}}(\lambda)$
  $(sk_0, 0, vk) \leftarrow \mathsf{Keygen}(\lambda)$
  $j \leftarrow 0$
  **repeat**
      $j \leftarrow j+1$; $sk_j \leftarrow \mathsf{Update}(sk_{j-1}, j-1, vk)$; $d \leftarrow \mathcal{F}^{\mathsf{Sign}(\cdot,j)}(\mathsf{cma}, vk)$
  **until** $d = \mathsf{breakin}$
  $(m^*, b, \sigma) \leftarrow \mathcal{F}(\mathsf{forge}, sk_j)$
  **if** $\mathsf{Verify}(m^*, b, vk, \sigma) = 1$ and $m^*$ was not queried of $\mathsf{Sign}(\cdot, b)$ and $0 \leq b < j$ **then**
      return 1 **else** return 0
  **end if**
```

auxiliary input $\mathsf{aux} \in \{0,1\}^{poly(\lambda)}$*, the adversary* $\mathcal{A}$ *wins the game below with negligible probability.*

We define $\mathbf{Adv}^{Hash-ext}_{H,\varepsilon,\mathcal{A}}(\lambda) = \Pr[\mathcal{G}^{Hash-ext}_{H,\varepsilon,\mathcal{A}}(\lambda)]$ where the game $\mathcal{G}^{Hash-ext}_{H,\varepsilon,\mathcal{A}}$ is defined in Algorithm 5.

Algorithm 5 Hash extraction game $\mathcal{G}^{Hash-ext}_{H,\varepsilon,\mathcal{A}}$

```
$\mathcal{G}^{Hash-ext(\lambda)}_{H,\varepsilon,\mathcal{A}}$
  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
  $(\sigma; x_e) \leftarrow (\mathcal{A}||\mathcal{E})(\mathsf{pp}, \mathsf{aux})$
  $\mathcal{A}$ wins if $\exists x$ such that $H(\mathsf{pp}, x) = \sigma \wedge \sigma \neq H(\mathsf{pp}, x_e)$ and there is a PPT algorithm
  $\mathsf{Check}(pp, \sigma)$ that returns 1 if $\exists x$ such that $H(\mathsf{pp}, x) = \sigma$ and 0 otherwise.
```

The hash extraction game needs the function Check which allows the verifier to check the well-formedness of hashes received from the adversary [19].

3.4 Forward secure multi aggregate signatures

Definition 5. *A forward secure multi aggregate signature* FSMAS *is a set of six algorithms that adds additional algorithms to algorithms of* FSS*. The additional algorithms are defined as follows.*

- Agg *takes as input a multi-user signature set* $((m_1, j_1, vk_1, \sigma_1), ..., (m_n, j_n, vk_n, \sigma_n))$ *and returns an aggregate signature* σ_{agg}*.*
- AggVerify *takes as input a set* $((m_1, j_1, vk_1), ..., (m_n, j_n, vk_n))$*, the aggregate signature* σ_{agg} *and returns 1 if* σ_{agg} *is a valid signature or 0, otherwise.*

We define the security of forward secure multi aggregate signature similar to Algorithm 4. An adversary $\mathcal{F}$ can freely choose all of the user verification keys $\vec{vk} = (vk_1, ..., vk_n)$ except the verification key of one honest user

Algorithm 6 Forward security game of FMSAS $\mathcal{G}^{fwsec}_{\mathsf{FSMAS},\mathcal{F}}(\lambda)$

```
$\mathcal{G}^{fwsec}_{\mathsf{FMSAS},\mathcal{F}}(\lambda)$
  $(sk^*_0, 0, vk^*) \leftarrow \mathsf{Keygen}(\lambda)$
  $j \leftarrow 0$
  **repeat**
    $j \leftarrow j+1$; $sk^*_j \leftarrow \mathsf{Update}(sk^*_{j-1}, j-1, vk^*)$; $d \leftarrow \mathcal{F}^{\mathsf{Sign}(\cdot,j)}(\mathsf{cma}, vk^*)$
  **until** $d = \mathsf{breakin}$
  $((m_1, j_1, vk_1, ), ..., (m^*, b, vk^*), ..., (m_n, j_n, vk_n)), \sigma^*_{agg}) \leftarrow \mathcal{F}(\mathsf{forge}, sk^*_j)$
  **if** $\mathsf{AggVerify}((m_1, j_1, vk_1), ..., (m_n, j_n, vk_n), \sigma^*_{agg}) = 1$ and $vk^* \in \{vk_1, ..., vk_n\}$ and
  $m^*$ was not queried of $\mathsf{Sign}(\cdot, b)$ and $0 \leq b < j$ **then**
    return 1 **else** return 0
  **end if**
```

vk^*. When the adversary $\mathcal{F}$ is given the verification key of the honest user vk^*, the adversary tries to forge an aggregate signature that involves the signature of the honest user. An adversary $\mathcal{F}$ can access a sign oracle and an update oracle freely, and can request break-in query also. Since the aggregation is allowed to everyone, the adversary does not need to request an oracle of aggregation. If the adversary outputs an aggregate signature σ^*_{agg} where $\mathsf{AggVerify}((m_1, j_1, vk_1), ..., (m_n, j_n, vk_n), \sigma^*_{agg}) = 1$ and the aggregate signature includes the signature of m^* that was not queried of $\mathsf{Sign}(\cdot, b)$ and $0 \leq b < j$ and $vk^* \in \vec{vk}$, then the adversary wins the forgery game. A formal security notion is described in Algorithm 6. Formally, we define $\mathbf{Adv}^{fwsec}_{\mathsf{FSMAS},\mathcal{F}}(\lambda) = \Pr[\mathcal{G}^{fwsec}_{\mathsf{FSMAS},\mathcal{F}}(\lambda)]$ where the game $\mathcal{G}^{fwsec}_{\mathsf{FSMAS},\mathcal{F}}$ is defined in Algorithm 6.

4 Construction

4.1 Main idea

In this section, we describe a formal construction of the proposed scheme. We present intuition of our construction first then specify details of the construction.

Verification of forward secure signature [5, 2, 26] checks whether the signature satisfies the following properties.

- A verification key is generated from a secret signing key which is implied in the signature.
- The signing key is updated correctly corresponding to a time period.
- The signing key is connected to a correct message with the time period.

While it is not easy to construct a signature scheme which satisfies the above properties simultaneously, it is more straightforward to devise a circuit to satisfy the above properties by stating them. Even if more properties are required, it is not difficult to include them in a circuit. Hence, we devise a relation circuit to efficiently represent each property in zk-SNARKs.

In key generation, a signing and verification key pair is generated with a proof which proves a connectivity between the signing key and the verification key. When the secret signing key is updated, the existing key is checked and it is updated through one-way hash function. In a sign relation, the secret key should be checked against the verification key, and a signature is a hash output of the message, the time period, and the verification key.

When aggregation of signatures is required, a circuit checks two signatures to be aggregated and generates a single hash output from two input data.

4.2 Forward secure signature construction

Algorithm 7 Relation

```
update_relation($s_{j'}, j', vk; \pi_j, s_j, j$)
    **if** $t = 0$ **then**
        $vk = H(s_j||r)$
    **else**
        $j' = j + 1$
        $\Pi$.verify$(\pi_j, s_j, j, vk, crs) = 1$
        $s_{j'} = H(s_j)$
    **end if**
sign_relation($\phi_{sig}; \pi_j, s_j, j, vk$)
    $\Pi$.verify$(\pi_j, s_j, vk, crs) = 1$
    $\phi_{sig} = H(m||j||vk)$
```

We provide a formal construction of the proposed forward secure signature scheme. Note that we use simulation-extractable zk-SNARK Π as a building block in our signature scheme. We assume that a common reference string crs is hard-coded as an integer value in algorithm. Algorithm 7 describes zk-SNARK relation of update and sign process. In key generation (a time period j is set to 0), a proof that proves a correlation of the signing key and the verification key is generated. update$_{\mathsf{relation}}$ describes a key generation relation and update relation. If the time period j' is 0, the proof proves that the verification key vk is a hash output of signing key s_j. When the signer updates the signing key(the time period $j' > 0$), the signer proves that a verification output of π_j which proves correctness of s_j is 1 and an updated signing key $s_{j'}$ is hash output of s_j. As described in sign$_{\mathsf{relation}}$, the signer should prove verification of the proof π_j. Then the signer proves that the committed value ϕ_{sig} is a hash output of a message m, the time period j and the verification key vk.

Algorithm 8 shows an overall construction of our proposed forward secure signature. Let H be a collision-resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^l$ where l is a bit length of hash function output. In Setup, this algorithm takes as input a relation for update R_{update}, and relation for sign R_{sig}. The algorithm

Algorithm 8 Forward secure signature (FSS) scheme

```
Setup($R_{update}, R_{sig}$)
  $crs_{update} \leftarrow \Pi.\mathsf{setup}(R_{update})$
  $crs_{sig} \leftarrow \Pi.\mathsf{setup}(R_{sig})$
  **return** $crs_{update}, crs_{sig}$
Keygen($\lambda$)
  $s_0 \xleftarrow{\$} \mathbb{Z}_l^*$
  $r \xleftarrow{\$} \mathbb{Z}_l^*$
  $vk \leftarrow H(s_0||r)$
  $j \leftarrow 0$
  $\pi_{init} \leftarrow \Pi.\mathsf{prove}(s_j, j, vk, crs_{update})$
  $sk_j \leftarrow (s_j, \pi_{init})$
  **return** $sk_j, j, vk$
Update($sk_j, j, vk$)
  **if** $\Pi.\mathsf{verify}(\pi_j, s_j, j, vk, crs_{update}) = 1$ **then**
    $j' \leftarrow j + 1$
    $s_{j'} \leftarrow H(s_j)$
  **else**
    **abort**
  **end if**
  $\pi_{j'} \leftarrow \Pi.\mathsf{prove}(s_{j'}, j', vk, crs_{update}; \pi_j, s_j, j)$
  **delete** $sk_j$
  $sk_{j'} \leftarrow (s_{j'}, \pi_{j'})$
  **return** $sk_{j'}, j'$
Sign($m, sk_j, j, vk$)
  **if** $\mathsf{verify}(\pi_j, s_j, j, vk, crs_{update}) = 1$ **then**
    $\phi_{sig} \leftarrow H(m||j||vk)$
    $\pi_{sig} \leftarrow \Pi.\mathsf{prove}(\phi_{sig}, crs_{sig}; \pi_j, s_j, j, vk)$
  **else**
    **abort**
  **end if**
  type $\leftarrow 0$
  $\sigma \leftarrow (\pi_{sig}, \phi_{sig}, \mathsf{type})$
  **return** $\sigma$
Verify($m, j, vk, \sigma$)
  Check $\phi_{sig} = H(m||j||vk)$
  $b \leftarrow \Pi.\mathsf{verify}(\pi_{sig}, \phi_{sig}, crs_{sig})$
  **return** $b$
```

generates all common reference strings for relations and hardcodes crs_{update}, crs_{sig} into the algorithms respectively.

Keygen takes a security parameter λ as input. The algorithm sets an initial secret signing key s_0 from $\mathbb{Z}_p$ randomly and yields a verification key vk which is a hash output of concatenation of s_0 and the random value r. An initial time period j is set to 0. The algorithm computes a proof π_{init} which proves the correctness of verification key vk based on update_relation. Keygen sets (s_j,π_j) to sk_j and outputs sk_j,j,vk.

Update takes as input a previous signing key sk_j, time period j, and verification key vk. The algorithm verifies the proof π_j that implies the correctness of signing keys in all previous periods first. If it is verified, it updates the signing key by computing a new signing key s_{j+1} that is a hash output of s_j. Update generates a proof π_{j+1} for update_relation taking $s_{j+1}, j+1, vk$ as input and π_j, s_j, j as witness then deletes the previous signing key sk_j. The algorithm outputs the updated signing key sk_{j+1} and the new time period $j+1$. Sign takes as input a message m, a signing key sk_j, a time period j, and a verification key vk. If the proof π_j which proves validity of s_j on update_relation is verified, ϕ_{sig} that is a hash output of $m||j||vk$ is generated. The algorithm makes a proof π_{sig} taking ϕ_{sig} as input based on sign_relation. And then type that reveals the type of signatures is 0. The algorithm finally outputs $\sigma = (\phi_{sig}, \pi_{sig}, \mathsf{type})$. Since the normal signature verification and the aggregate signature verification have different relations, the type of signature should be available to a verifier (and an aggregator) for efficient aggregation. Though a secret signing key is connected with a message directly in the normal signature scheme, our scheme connects a message m with the verification key vk to support aggregation of multi-user signatures.

Verify first check the correctness of ϕ_{sig} by computing $H(m||j||vk)$. Verify calls Π.verify and returns b which is a verification result of π_{sig} taking input ϕ_{sig}.

5 Extended construction

5.1 Forward secure aggregate signature construction

Algorithm 9 represents a relation for the signature aggregation. We assume that the aggregation proceeds one by one repeatedly regardless of the signature's time period, message, verification key. The intuitive idea of the aggregation is that a proof can be an aggregate signature if the proof proves multiple verifications of the signatures. For instance, when Alice wants to aggregate Bob's two signatures which have different time periods and messages, she can aggregate two signatures by verifying signatures respectively and proving the verification process. As described in Algorithm 9, an aggregator verifies two signatures then generates a hash value that implies all the components of signatures(the message, the time period, the verification key).

Algorithm 10 shows a whole construction of aggregation. AggSetup takes as input a relation R_{agg} that is described in Algorithm 9, and generates a common

Algorithm 9 Agg relation

```
agg_relation($\phi_{agg}; \sigma_i, \sigma_j$)
  Parse $\sigma_i$ as ($\phi_i$,$\pi_i$,$\mathsf{type}_i$)
  Parse $\sigma_j$ as ($\phi_j$,$\pi_j$,$\mathsf{type}_j$)
  if $\mathsf{type}_i = 0$ then
      $crs \leftarrow crs_{sig}$
  else
      $crs \leftarrow crs_{agg}$
  end if
  $\Pi.\mathsf{verify}(\pi_i, \phi_i, crs) = 1$
  $\Pi.\mathsf{verify}(\pi_j, \phi_j, crs_{sig}) = 1$
  $\phi_{agg} = H(\phi_i||\phi_j)$
```

reference string for aggregation crs_{agg} and it is hardcoded in the algorithm. The aggregate algorithm Agg takes as input n signature sets $(m_1, j_1, vk_1, \sigma_1), ..., (m_n, j_n, vk_n, \sigma_n)$ where m_i,j_i,vk_i are the message, the time period, the verification key respectively. The algorithm first checks the validity of ϕ_{sig_i} value of all σ_i by computing $H(m_i||j_i||vk_i)$ then verifies first two signatures by running Π.verify with crs_{sig}. After they are verified, the algorithm generates a new signature value ϕ_{agg} which is a hash output of $(\phi_{sig_1}||\phi_{sig_2})$. The algorithm proves the aggregation result according to algorithm 9. If the number of the signature set is over three, the aggregator verifies the previous aggregate signature with crs_{agg} and input signature σ_i with crs_{sig}. Similarly to the above aggregation, the aggregator generates a hash output value ϕ_{agg} and proves the verification results repeatedly. The algorithm returns aggregate signature $(\phi_{agg}, \pi_{agg}, \mathsf{type})$ lastly.

A verification algorithm AggVerify takes n messages, time periods, verification keys and the aggregate signature σ_{agg}. A verifier first checks the validity of ϕ_{agg} in σ_{agg} using the hash-chain computation, and verifies the aggregate signature using Π.verify.

6 Security proof

Theorem 1. *Let* FSS *be our key evolving signature scheme. Then for parameters modulus size λ, the execution time t, the common reference generation time t_{crs},the number of sign queries q_{sig},*

$$\mathbf{Adv}^{fwsec}_{\mathsf{FSS},\mathcal{F}}(\lambda) \leq \mathbf{Adv}^{Hash-ext}_{H,\varepsilon,\mathcal{A}}(\lambda) \tag{2}$$

where $t' = t + t_{crs}$

Theorem 2. *Let* FSMAS *be our forward secure multi aggregate signature scheme. Then for parameters modulus size λ, the execution time t, the common reference generation time t_{crs},the number of sign queries q_{sig},*

$$\mathbf{Adv}^{fwsec}_{\mathsf{FSMAS},\mathcal{F}}(\lambda) \leq \mathbf{Adv}^{Hash-ext}_{H,\varepsilon,\mathcal{A}}(\lambda) \tag{3}$$

where $t' = t + t_{crs}$

Algorithm 10 Aggregation construction

```
AggSetup($R_{agg}$)
    $crs_{agg} \leftarrow \Pi.\mathsf{setup}(R_{agg})$
    return $crs_{agg}$
Agg($(m_1, j_1, vk_1, \sigma_1), ..., (m_n, j_n, vk_n, \sigma_n)$)
    parse $\sigma_i$ as $(\phi_{sig_i}, \pi_{sig_i}, \mathsf{type}_i)$
    Check all $\phi_{sig_i} = H(m_i||j_i||vk_i)$
    $b_1 \leftarrow \Pi.\mathsf{verify}(\pi_{sig_1}, \phi_{sig_1}, crs_{sig})$
    $b_2 \leftarrow \Pi.\mathsf{verify}(\pi_{sig_2}, \phi_{sig_2}, crs_{sig})$
    if $b_1 \&\& b_2 = 1$ then
        $\phi_{agg} \leftarrow H(\phi_{sig_1}||\phi_{sig_2})$
        $\pi_{agg} \leftarrow \Pi.\mathsf{prove}(\phi_{agg}, crs_{agg}; \sigma_1, \sigma_2)$
        $\mathsf{type} \leftarrow 1$
    end if
    if $n < 3$ then $\sigma_{agg} \leftarrow (\phi_{agg}, \pi_{agg}, \mathsf{type})$
        return $\sigma_{agg}$
    else
        for $i \leftarrow 3$ to $n$ do
            $\sigma_{old} \leftarrow (\phi_{agg}, \pi_{agg})$
            $b_1 \leftarrow \Pi.\mathsf{verify}(\pi_{agg}, \phi_{agg}, crs_{agg})$
            $b_2 \leftarrow \Pi.\mathsf{verify}(\pi_i, \phi_i, crs_{sig})$
            if $b_1 \& b_2 \neq 1$ then abort
            else
                $\phi_{agg} \leftarrow H(\phi_{agg}||\phi_i)$
                $\pi_{agg} \leftarrow \Pi.\mathsf{prove}(\phi_{agg}, crs_{agg}; \sigma_{old}, \sigma_i)$
                $\mathsf{type} \leftarrow 1$
                $\sigma_{agg} \leftarrow (\phi_{agg}, \pi_{agg}, \mathsf{type})$
            end if
        end for
    end if
    return $\sigma_{agg}$
AggVerify($(m_1, j_1, vk_1), ..., (m_n, j_n, vk_n), \sigma_{agg}$)
    parse $\sigma_{agg}$ as $(\phi_{agg}, \pi_{agg})$
    $\phi_{sig_1} \leftarrow H(m_1||j_1||vk_1)$
    $\phi_{sig_{old}} \leftarrow \phi_{sig_1}$
    for $i \leftarrow 2$ to $n$ do
        $\phi_{sig_i} \leftarrow H(m_i||j_i||vk_i)$
        $\phi_{sig_{old}} \leftarrow H(\phi_{sig_{old}}||\phi_{sig_i})$
    end for
    if $\phi_{sig_{old}} = \phi_{agg}$ then
        $b \leftarrow \Pi.\mathsf{verify}(\phi_{agg}, crs_{agg})$
    end if
    return $b$
```

6.1 Proof of Theorem 1

We construct an adversary $\mathcal{A}$ that conducts a hash extraction game described in Algorithm 5. The adversary $\mathcal{A}$ utilizes the adversary $\mathcal{F}$ which executes $\mathcal{G}^{fwsec}_{\mathsf{FSS},\mathcal{F}}(\lambda)$ experiment described in Algorithm 4 as a subroutine. We suppose the adversary $\mathcal{F}$ succeeds with $\mathbf{Adv}^{fwsec}_{\mathsf{FSS},\mathcal{F}}(\lambda)$ in execution time t.

Initial key generation. The adversary $\mathcal{A}$ gets a hash value σ from the challenger, and then set σ to vk. The adversary runs Setup and acquires common reference strings crs_{update}, and crs_{sig}. Note that since the adversary has a trapdoor of the common reference string, the adversary can generate simulated proof using $\mathsf{\Pi.SimProve}$. Unusually, the adversary does not need to choose an initial secret signing key s_0, since the correctness of vk is guaranteed by a simulated proof which deceives a relation between vk and the blank input. The adversary runs subroutine $\mathcal{F}$ taking as input vk, crs_{update}, and crs_{sig}.

Interactive query phase

- **Update query** : When $\mathcal{F}$ requests to update the signing key, $\mathcal{A}$ runs $\mathsf{\Pi.SimProve}$ and acquires a simulated proof on any arbitrary signing key at the queried time period. Since the simulated proof can prove any relation of false input, $\mathcal{A}$ can update the signing key fraudulently.
- **Sign query** : $\mathcal{A}$ generates a signature dishonestly without the secret signing key when $\mathcal{A}$ receives a sign query (a message m, time period j). $\mathcal{A}$ sets $H(m||j||vk)$ as ϕ_{sig} and generates a simulated proof π_{sig}. Finally $\mathcal{A}$ outputs a signature where σ=(π_{sig},ϕ_{sig},type) to $\mathcal{F}$.
- **Break-in query** : $\mathcal{A}$ randomly chooses a secret signing key s_b that is unrelated to the previous signing key and the vk. Likewise the above, relation between s_b and vk is proven by simulated proof π_b. $\mathcal{A}$ outputs current time period signing key s_b and π_b to $\mathcal{F}$.

Final forgery When $\mathcal{F}$ acquires a signing key s_b and π_b where b is the time period at $\mathsf{break-in}$, $\mathcal{F}$ outputs forged signature $(\pi_{sig^*}, \phi_{sig^*}, \mathsf{type})$ on a new message m^* where m^* was not queried of $\mathsf{Sign}(\cdot, j)$ and $0 \leq j < b$. After receiving the forged signature set, $\mathcal{A}$ runs extractor $\mathcal{E}$ and extracts a secret signing key s_j which is a witness of π_{sig^*}. The adversary $\mathcal{A}$ can compute a pre-image of s_b through hashing s_j repeatedly.

Success probability We analyze a probability of above execution. Note that we do not need to guess the time period of break-in since all the signing keys are irrelevant to the signing key at the time period of break-in via the simulated proof. Likewise, there is no probability to fail responding the sign query because all the queries can be responded through the simulated proof. Therefore, a success probability of $\mathcal{A}$ converges on the success probability of extractor $\mathcal{E}$ completely.

6.2 Proof of Theorem 2

A proof of Theorem 2 is almost identical to the proof of Theorem 1. The adversary $\mathcal{F}$ forges an aggregate signature on behalf of forging the normal forward secure signature. Thus, all the proof procedure proceeds as subsection 6.1 except for the final forgery. When $\mathcal{A}$ is given a forged aggregate signature, $\mathcal{A}$ should run the extractor $\mathcal{E}$ recursively until the extractor outputs the sk_j of signature on message m^* where $0 \leq j < b$. Since the aggregate signature proves only the aggregation result, an inner proof which is a witness of the forged aggregate signature should be extracted n times in worst case (n is the number of aggregations). Thus the success probability of $\mathcal{A}$ converges on ϵ^n where ϵ is the success probability of extractor $\mathcal{E}$. Note that if the aggregation is performed balanced then n becomes $O(\log(N))$ where N denotes the total number of signatures in an aggregate signature and the success probability is $N\epsilon$.

7 Experiment

In this section, we measure the performance of our forward secure signature. Basically, our forward secure signature scheme supports the aggregation of multi-user signatures regardless of their time periods, and all the efficiencies are not dependent on the time period. In particular, only zk-SNARK circuit size affects the time efficiencies of our construction, and the performance of the construction can differ depending on the zk-SNARK proof scheme. We implement the forward secure signature scheme using the plonk [20] implemented by aztec as the zero-knowledge proof scheme [41]. We compare the performance of our signature scheme with the performances of BM [5], AR [2], IR [26] and KO17 [30], KO19 [31] via an experiment. The experiment is performed on Intel i7 4.2GHz laptop with 64GB RAM under Ubuntu 18.04.

Table 3: Performance experiment results in forward secure signature schemes

		Ours	BM [5]	AR [2]	IR [26]	KO17 [30]
Key generation time		$19.407s$	$642.4375s$	$398.9233s$	$1970.0101s$	$4.4893s$
Update time		$12.874s$	$0.6ms$	$0.4ms$	$43ms$	$0.4ms$
Signing time		$12.555s$	$2.5412s$	$398.3939s$	$1ms$	$0.4ms$
Verification time		$1ms$	$2.5118s$	$389.2896s$	$0.2ms$	$389.3412s$
Signing key size	Secret key	$1.6KB$	$65KB$	$0.2KB$	$0.5KB$	$0.2KB$
	Public parameter	$654MB$				
Verification key size	Verification key	$32B$	$65KB$	$0.5KB$	$0.5KB$	$1KB$
	Public parameter	$1KB$				
Signature size		$1.6KB$	$0.5KB$	$0.2KB$	$0.3KB$	$0.5KB$

Figure 2 represents the key setup time, the signing time, the update time, and the verification time of all comparative signatures by varying the total time

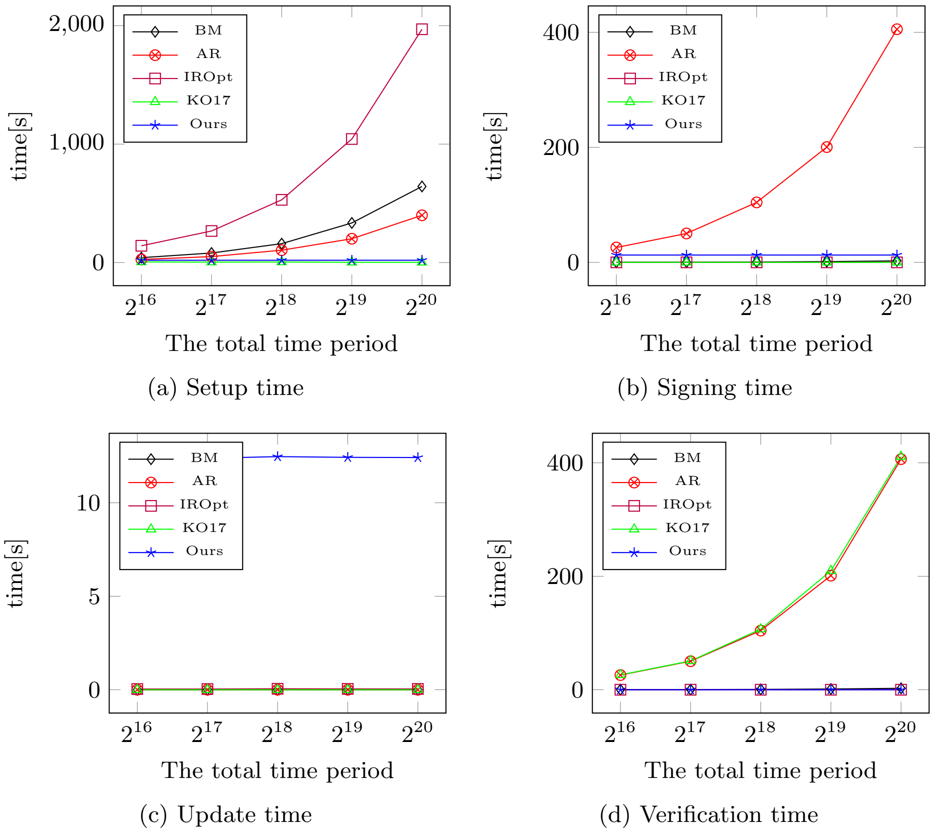


Fig. 2: Comparison of time performances with other forward secure signatures

periods T. The security parameter is fixed to 2048 bits in cases of RSA-based signature schemes [5, 2, 26, 30, 31] and 256 bits in our case since the construction utilizes a pairing function. The message length is fixed to 160 bits in all cases.

As shown in Figure 2(a), the setup time is proportional to the total time periods T in BM, AR, IR(optimized version). Specifically, the setup time of IROpt reaches about 2000 seconds when the total time period is 2^{20}. In case of KO17, the time complexity is $O(lT)$ originally, however, the setup time can be optimized by using the RSA group order in the implementation. Our scheme has 19 seconds in all time periods cases. Figure 2(b) illustrates the signing time of the signature schemes. IROpt, KO17, and our signature scheme have a constant signing time on the total time period T. However, since the zk-proof circuit size affects the signing time of our signature scheme, it takes 12 seconds approximately. Meanwhile, BM and AR have a signing time that is proportional to total time periods T in the same manner, the signing time of AR takes more than those of BM since the computation in AR is composed of exponentiation. In case of the update time, as shown in Figure 2(c), all the signature schemes have a constant update time. Like in the case of the signing time, our update time is affected by the zk-SNARK circuit size and it takes 12 seconds identical to the signing time. Figure 2(d) illustrates the verification time of all signature schemes. Only IROpt and our signature scheme have a constant verification time, on the other hand, other signature schemes have a total time period dependent verification time. Table 3 shows detailed data of the performance when $T = 2^{20}$, the message bits $l = 160$.

Table 4: Performance experiment results on aggregation

	Ours	BM-FAS [31]	AR-FAS [31]
Setup time	$38.54s$	$1701.22s$	$1159.04s$
Signing time	$12.55s$	$2242.76s$	$3477.03s$
Aggregation time	$25.23s$	$0.004ms$	$0.004ms$
Aggregate signature verification time	$1ms$	$1680.63s$	$1739.39s$

Table 4 represents the experiment results on the forward secure aggregate schemes. We implement forward secure aggregate schemes via applying KO19 [31] technique to BM and AR respectively. The total time period, the message bits, and the security parameter are set to those of Table 3 equally and the number of signatures are two. As shown in Table 4, the aggregation time in our signature scheme has a relatively heavy computation than other forward secure signatures. It is caused by the zk-proof circuit size that are composed of approximately 1,600,000 gates. However, the verification time of the aggregate signature has an equal time with the single signature verification. In addition, the heavy zk-proof circuit size can be reduced by using the Halo technique [10] that extracts the proof verification circuit to outside the proof circuit via the proof aggregation.

8 Conclusion

In this paper, we propose a new forward secure multi-user aggregate signature using zk-SNARK. Our new forward secure signature scheme supports all constant complexities and a flexible aggregation where all restrictions that exist in previous works are eliminated. In the proposed scheme, forward secure signature properties are stated and a corresponding circuit is constructed. Finally, an aggregation circuit is built to merge two signatures, in which the circuit implies the verification results of signatures. Since the circuit size remains constant, the key/signature size and the performance remain constant irrelevant to the time period. The proposed scheme can remove any restriction existing in the previous work and show the best complexity. The security of the proposed scheme is formally proven. In future, our methodology that uses zk-SNARK as a building block can be generalized to more properties such as group signature, blind signature, etc. We additionally minimize the cost from zk-SNARK computation.

References

1. libsnark (2014). https://github.com/scipr-lab/libsnark (2020)
2. Abdalla, M., Reyzin, L.: A new forward-secure digital signature scheme. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 116–129. Springer (2000)
3. Anderson, R.: In fourth annual conference on computer and communications security. ACM (1997)
4. Baylina, J.: iden3/snarkjs. https://github.com/iden3/snarkjs (2020)
5. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: Annual International Cryptology Conference. pp. 431–448. Springer (1999)
6. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. Algorithmica **79**(4), 1102–1160 (2017)
7. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for snarks and proof-carrying data. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing. pp. 111–120 (2013)
8. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 416–432. Springer (2003)
9. Bowe, S., Gabizon, A.: Making groth's zk-snark simulation extractable in the random oracle model. IACR Cryptol. ePrint Arch. **2018**, 187 (2018)
10. Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive proof composition without a trusted setup. IACR Cryptol. ePrint Arch. **2019**, 1021 (2019)
11. Boyen, X., Shacham, H., Shen, E., Waters, B.: Forward-secure signatures with untrusted update. In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 191–200 (2006)
12. Brown, D.R., Vanstone, S.A.: Aggregate signature schemes (May 22 2012), uS Patent 8,185,744
13. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315–334. IEEE (2018)
14. Chatterjee, S., Hankerson, D., Knapp, E., Menezes, A.: Comparing two pairing-based aggregate signature schemes. Designs, Codes and Cryptography **55**(2-3), 141–167 (2010)
15. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 769–793. Springer (2020)
16. Chiesa, A., Tromer, E.: Proof-carrying data and hearsay arguments from signature cards. In: ICS. vol. 10, pp. 310–331 (2010)
17. Chiesa, A., Tromer, E.: Proof-carrying data: Secure computation on untrusted platforms (high-level description). The Next Wave: The National Security Agency's review of emerging technologies **19**(2), 40–46 (2012)
18. Drijvers, M., Neven, G.: Forward-secure multi-signatures. IACR Cryptol. ePrint Arch. **2019**, 261 (2019)
19. Fiore, D., Fournet, C., Ghosh, E., Kohlweiss, M., Ohrimenko, O., Parno, B.: Hash first, argue later: Adaptive verifiable computations on outsourced data. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1304–1316 (2016)
20. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. **2019**, 953 (2019)

21. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 626–645. Springer (2013)
22. Gentry, C., Ramzan, Z.: Identity-based aggregate signatures. In: International workshop on public key cryptography. pp. 257–273. Springer (2006)
23. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 305–326. Springer (2016)
24. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 581–612. Springer (2017). https://doi.org/10.1007/978-3-319-63715-0_20, https://doi.org/10.1007/978-3-319-63715-0_20
25. Hohenberger, S., Waters, B.: New methods and abstractions for rsa-based forward secure signatures. In: International Conference on Applied Cryptography and Network Security. pp. 292–312. Springer (2020)
26. Itkis, G., Reyzin, L.: Forward-secure signatures with optimal signing and verifying. In: Annual International Cryptology Conference. pp. 332–354. Springer (2001)
27. Jurkiewicz, M.: Binary tree based forward secure signature scheme in the random oracle model
28. Kim, J., Lee, J., Oh, H.: Qap-based simulation-extractable snark with a single verification. Tech. rep., Cryptology ePrint Archive, Report 2019/586, 2019. https://eprint. iacr. org . . . (2019)
29. Kim, J., Lee, J., Oh, H.: Simulation-extractable zk-snark with a single verification. IEEE Access **8**, 156569–156581 (2020)
30. Kim, J., Oh, H.: Forward-secure digital signature schemes with optimal computation and storage of signers. In: IFIP International Conference on ICT Systems Security and Privacy Protection. pp. 523–537. Springer (2017)
31. Kim, J., Oh, H.: Fas: Forward secure sequential aggregate signatures for secure logging. Information Sciences **471**, 115–131 (2019)
32. Kozlov, A., Reyzin, L.: Forward-secure signatures with fast key update. In: International Conference on Security in Communication Networks. pp. 241–256. Springer (2002)
33. Krawczyk, H.: Simple forward-secure signatures from any signature scheme. In: Proceedings of the 7th ACM conference on Computer and communications security. pp. 108–115 (2000)
34. Lipmaa, H.: Simulation-extractable snarks revisited. Tech. rep., Cryptology ePrint Archive, Report 2019/612, 2019. http://eprint. iacr. org . . . (2019)
35. Ma, D.: Practical forward secure sequential aggregate signatures. In: Proceedings of the 2008 ACM symposium on Information, computer and communications security. pp. 341–352 (2008)
36. Ma, D., Tsudik, G.: Forward-secure sequential aggregate authentication. In: 2007 IEEE Symposium on Security and Privacy (SP'07). pp. 86–91. IEEE (2007)
37. Malkin, T., Micciancio, D., Miner, S.: Efficient generic forward-secure signatures with an unbounded number of time periods. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 400–417. Springer (2002)

38. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2111–2128 (2019)
39. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE (2013)
40. Qiao, K., Tang, H., You, W., Zhao, Y.: Blockchain privacy protection scheme based on aggregate signature. In: 2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA). pp. 492–497. IEEE (2019)
41. zac williamson: Aztecprotocol/barretenberg. https://github.com/AztecProtocol/barretenberg.git (2020)
42. Xiong, H., Guan, Z., Chen, Z., Li, F.: An efficient certificateless aggregate signature with constant pairing computations. Information Sciences **219**, 225–235 (2013)
43. Yavuz, A.A., Ning, P., Reiter, M.K.: Baf and fi-baf: Efficient and publicly verifiable cryptographic schemes for secure logging in resource-constrained systems. ACM Transactions on Information and System Security (TISSEC) **15**(2), 1–28 (2012)
44. Yuan, C., Xu, M.x., Si, X.m.: Research on a new signature scheme on blockchain. Security and Communication Networks **2017** (2017)