https://doi.org/10.1007/s11704-025-41269-5

RESEARCH ARTICLE

Scalable batch verification of ECDSA for blockchain using IVC

Li LIU[1,2], Puwen WEI[1,2,3]✉, Shuchang LIU[2], Zirui WANG[2], Da HU[2], Zengjie KOU[4]

1. Key Laboratory of Cryptologic Technology and Information Security (Ministry of Education), Shandong University, Qingdao 266231, China
2. School of Cyber Science and Technology, Shandong University, Qingdao 266231, China
3. Quancheng Laboratory, Jinan 250100, China
4. Topsec Network Technology Inc., Beijing 100000, China

Received November 25, 2024; accepted February 13, 2025

E-mail: pwei@sdu.edu.cn

Abstract

With the rising volume of transactions on blockchains, signature verification becomes a critical bottleneck of efficiency, hindering scalability and performance. This paper presents a general approach to batch verification of arbitrary signatures on blockchain. By leveraging the memory-friendliness of incremental verifiable computation (IVC) and optimizing for blockchain environments, the proposed scheme can enhance scalability, reduce memory consumption, and ensure compatibility with common devices while supporting an arbitrary number of signature verifications. This approach allows for the concurrent generation of IVC proofs while receiving signatures from other nodes, making it particularly well-suited for low-latency blockchain applications. As a concrete instantiation of our approach, we introduce BEATS (Batch ECDSA Transaction verification Scheme), where the underlying SNARK is instantiated by Spartan with Bulletproof commitment. Our implementation, evaluated on a virtual machine with 8 cores and 16 GB RAM, shows significant performance gains compared to $\text{Spartan}_{\text{BP}}$, which is the direct construction using Spartan with Bulletproof commitment to verify a batch of ECDSA. The comparison shows that BEATS speeds up the prover by 3–7 times and the verifier by 48–240 times when handling up to 2^{11} ECDSA signatures, the maximum batch size supported by $\text{Spartan}_{\text{BP}}$. For larger batches exceeding 2^{10}, our scheme outperforms the baseline approach, which verifies ECDSA signatures one by one without any proof system. Our verifier achieved a speedup of 21–174 times compared to the baseline as the batch size grows to 2^{20}. Furthermore, BEATS exhibits a remarkably low memory footprint, with peak memory usage remaining below 1 GB.

Keywords

batch verification; ECDSA; IVC; blockchain

1 Introduction

A digital signature is a cryptographic mechanism that allows a signer to authenticate a message using his private key. Anyone can then verify the validity of the signature using the corresponding public key. This authentication functionality has led to the widespread adoption of digital signatures in various applications. The rise of blockchain technology has further increased the popularity of digital signatures. In a blockchain system, when a transfer occurs, it must be accompanied by a digital signature. The transaction can be recorded on the blockchain only if it is verified by a sufficient number of nodes (participants). As the number of transactions grows, the workload for verifying these signatures increases dramatically. This presents a significant challenge for blockchain scalability, as the time it takes to verify signatures can become a bottleneck, limiting the blockchain's overall throughput and performance.

Batch verification of digital signatures offers a promising approach to address this problem. By verifying multiple signatures simultaneously, batch verification significantly reduces the overall verification time compared to individual verification. In particular, customized signatures tailored for batch verification can leverage homomorphic properties to significantly improve the verification process. For example, the aggregate BLS signature [1] can be constructed by adding up multiple individual BLS signatures simply, resulting in a nearly 50% reduction in verification time.

While customized signatures like BLS offer significant advantages for batch verification, adapting standard schemes like ECDSA for efficient batch verification remains an active area of research, due to ECDSA's widespread adoption and strong security track record. However, existing batch verification techniques for standard signature schemes often face limitations. Harn [2] proposes schemes for batch verification of RSA signatures, but these schemes only support multiple signatures signed by the same private key. Karati

et al. [3–5] present improvements for batch verification of ECDSA signature, but the batch size is bounded by a constant number $t \leqslant 9$, making it impractical for a large number of signatures. Although there are efficient methods for batch verification of variants of ECDSA [6,7], these non-standard ECDSA signatures either have security issues or are inefficient.

Recently, packing signatures inside SNARK, which is used in zkRollup [8–11], has been an emerging approach to prove the validity of a batch of transactions. The advantage of this approach lies in its ability to support arbitrary signature schemes without restrictions on the algebraic structure. Verifying a signature inside SNARK allows a prover to prove the validity of the signature by running the SNARK protocol and sending a succinct proof to the verifier. The verifier can check the validity of the signature by verifying the SNARK proof instead of running the signature verification algorithm.

However, many SNARKs that are highly efficient in verification often take substantial time and memory resources to generate proofs due to complex computation. For instance, consider the implementation of SNARKs on a virtual machine with 8 cores and 16 GB of RAM, where the actual available memory is about 12.78 GB. If we apply Spartan [12] with Bulletproof commitment [13], which has a fast prover for a batch of ECDSA signatures, the memory consumption grows linearly with the number of signatures. The resulting scheme can support at most about 2^{11} signatures with peak memory usage exceeding 10 GB of RAM. On the other hand, if applying Marlin [14] with univariate polynomial KZG commitment [15], which has a short proof and a fast verifier, it can support only one ECDSA signature, with peak memory usage exceeding 9.76 GB of RAM. Consequently, generating proofs for a large number of signatures within a reasonable timeframe becomes a significant bottleneck for blockchain systems. This resource-intensive process hinders the participation of low-performance devices in batch verification, potentially limiting the scalability and accessibility of blockchain systems.

1.1 Our contributions

To address the aforementioned challenges, we provide a general construction for the batch verification of signatures in blockchain. Our approach takes advantage of the memory-friendliness of incremental verifiable computation (IVC) and the blockchain setting to enhance scalability and reduce memory consumption, making it compatible with common devices and supporting an arbitrary number of signature verifications. Combined with the state-of-the-art folding scheme [16], our construction can achieve a balance between prover/verifier time and proof size, optimizing proof generation efficiency while introducing minimal overhead for verification. Another notable advantage of our approach is that the prover can generate IVC proofs concurrently with receiving signatures from other nodes in a blockchain system. Unlike most existing batch verification schemes that start generating proofs only after collecting enough valid signatures, our approach leverages the incremental nature of IVC to efficiently utilize the waiting time for signatures. This allows for a reduction in the overall block validation time, making it particularly advantageous in low-latency blockchain environments where minimizing delays is crucial.

Our construction is general and can apply to any type of signature. For concrete construction, we present BEATS (Batch ECDSA Transaction verification Scheme) for batch verification of the ECDSA signature. In BEATS, we instantiate the SNARK after the IVC by Spartan [12] with Bulletproof [13] commitment, which is transparent without any trusted setup. We implement it on a virtual machine with 8 cores and 16 GB of RAM, and make a comparison with the original construction SNARK $\text{Spartan}_{\text{BP}}$, which invokes the SNARK Spartan [12] with Bulletproof [13] commitment to verify a batch of ECDSA directly. The results of our comparison show that BEATS speeds up the prover by 3–17 times and the verifier by 48–240 times when the number of ECDSA signatures is up to 2^{11}. This batch size is the maximum that can be supported by $\text{Spartan}_{\text{BP}}$ with 16 GB of RAM. For batch sizes exceeding 2^{10}, our scheme becomes more efficient compared to the baseline approach, which verifies ECDSA signatures one by one without any proof system. Our verifier achieved a speedup of 21–174 times compared to the baseline when the batch size reaches 2^{20}. Notably, even when the baseline approach was optimized for parallelism, our single-threaded scheme outperformed it when the batch size surpassed 2^{13}. The performance gap continued to widen with larger batch sizes, and at 2^{20} signatures, our verifier demonstrated a speedup of 3–30 times. Furthermore, the peak memory usage of BEATS grows slowly and stays almost constant at less than 1 GB, which ensures compatibility with most low and mid-range devices.

We also conduct a comparison with a non-transparent SNARK Marlin with KZG polynomial commitment [15], which has a short proof and a fast verifier. Due to the bilinear pairing, the R1CS in Marlin should be defined on a pairing-friendly elliptic curve instead of the curve of ECDSA. Therefore, the number of constraints will increase significantly and result in a slower prover and limited scalability. Due to memory limitations, Marlin could only support a single ECDSA signature in 16 GB of RAM. When there is only one ECDSA signature, BEATS speeds up the prover by 10 times, while the verifier time and the proof size are larger than Marlin by about 12 times. However, we improved the scalability significantly that supports an arbitrary number of signatures and the proof size is nearly constant when the batch size is less than 2^{14}. Since Marlin needs to generate proofs for multiple signatures separately, the verifier time and proof size of BEATS can be amortized efficiently.

Other applications. We present a construction that efficiently verifies a large batch of signatures within a proof system, trading increased prover time for significantly reduced verification complexity. This approach is particularly beneficial for zkRollup applications. Furthermore, our construction has promising applications in image authenticity verification, a critical concern in the era of AI-generated content. Standards like C2PA [17], supported by Adobe, Arm, Intel, Microsoft, and Truepic, advocate for embedding digital signatures within each photograph captured by cameras. When acquiring images from photography platforms, verifying these signatures is crucial to ensure their origin from

authentic cameras and to mitigate the risk of AI-generated forgeries. Our scheme offers an efficient solution for verifying the authenticity of a large number of downloaded images.

2 Technique overview

Given a batch of signatures, verifying them inside a SNARK requires transforming the signature verification algorithm into an appropriate arithmetic circuit. In this work, we use the widely studied R1CS constraints system as the arithmetized form. Intuitively, pairing-based SNARK schemes like Marlin are well-suited for blockchain applications due to their succinctness. However, applying it to verify concrete signature schemes such as ECDSA presents challenges. Most of the ECDSA verifier's operations occur on the base field of the elliptic curve `secp256k1` while the R1CS in Marlin is defined on the scalar field of pairing-friendly elliptic curves, such as `BLS12_381` whose scalar field size is smaller than that of `secp256k1`.

Consequently, it needs to simulate operations on a larger field using a smaller field. It will result in many non-native field operations and lead to a significant increase in the number of constraints. Even worse, the prover's time and space complexity in Marlin is asymptotically $O(N\log N)$, where N is the number of constraints. Since the prover must execute complex computations including Fast Fourier Transformation (FFT) for long high-degree polynomials, the prover cost will limit the number of signatures handled in a batch.

One of our ideas in this work is to use SNARK whose finite field matches that of ECDSA. That is, we represent the ECDSA verification algorithm in R1CS constraints over some elliptic curve whose scalar field is the same as the base field of `secp256k1`. One of the candidate elliptic curves is `secq256k1` [18]. Since `secq256k1` is not pairing-friendly, we use $\text{Spartan}_{\text{BP}}$ under the discrete logarithm hardness assumption without bilinear pairing. This approach is similar to the work [19][1)]. The distinct advantages of $\text{Spartan}_{\text{BP}}$ are the fast prover and the succinct proof size. However, using it alone will not achieve the desired high throughput. If applying $\text{Spartan}_{\text{BP}}$ to each signature, t proofs would be generated for t signatures. Verifying t proofs may not always be more efficient than verifying t ECDSA signatures. Alternatively, if invoking the $\text{Spartan}_{\text{BP}}$ one time to prove a batch of ECDSA, it is hard to support large batch sizes of signatures due to the memory limits.

To compress proof length, reduce verification time, and increase throughput, we propose a folding-based technique for batch verification of signatures. This technique leverages the folding scheme from the IVC protocol Nova [16] to compress t ECDSA signatures into a constant size, independent of t. In our scheme, a batch of signatures is divided into blocks of equal size, and the folding scheme is applied between these blocks. However, the folding scheme in Nova is embedded in an IVC protocol, where the full computation is a chain connected by the inputs and outputs of each step rather than independent sub-computations. If simply applying the folding scheme and proving the folding process step by step, the verifier will only receive the last proof. While the validity of this proof indicates that there exist t valid instances that have been folded, they are not necessarily the t signatures claimed by the prover. In other words, the proven statements may not be consistent with the signatures recorded on the blockchain.

To address this issue, we introduce an additional commitment, which is a chain of collision-resistant hash functions, to bind the statements and construct the IVC chain. Specifically, we have the verifier compute the hash chain locally to ensure the consistency of the signature to be proven. Note that this commitment must also be proven in R1CS, and using hash functions like SHA-256 over an extended binary field will dramatically increase the number of constraints due to the bitwise operations involved. To improve the prover efficiency, we choose the SNARK-friendly hash function Poseidon to minimize the prover overhead. Details of the construction can be found in Section 5.2.

We run Nova with the SNARK $\text{Spartan}_{\text{BP}}$ on our modified IVC chain. In contrast to verifying each ECDSA signature inside $\text{Spartan}_{\text{BP}}$ directly and separately, we package multiple signatures via the folding scheme before executing the SNARK. The prover time for each signature is amortized. Finally, only one IVC proof is produced, independent of the number of signatures. The proof size is $O(N+\log N)$ where N is the number of R1CS constraints of one ECDSA verifier.

Remark (General Arithmetization) In this work, we use the R1CS arithmetized form consistent with the basic implementation of Nova. Note that some variants of Nova can support the Plonkish circuits [20] and even the more general CCS (customizable constraint system) [21], and our scheme also supports these arithmetization methods. To avoid abstract definitions and make it easier to understand, we use the original R1CS-based Nova that is sufficient to describe our construction intuitively. When considering Plonkish circuits, we can get a construction that replaces the Spartan with Plonk, which is a non-transparent SNARK with an updatable CRS.

3 Related work

In the last decade, researchers have designed a lot of distinctive proof systems. The trade-off between the prover and verifier time and proof length has always been a hard task. Known for its constant-size succinctness and efficient verification, SNARK schemes such as [14,22,23] are widely used in blockchain [9–11,24] but are often limited by the size of the circuit because the proof time is too slow. Other proof systems [12,25–28] that do not contain complex computations focus on efficient provers that can generate proofs in optimal linear time about the circuit scale, but either the proof size or the verifier time is at least an order of magnitude higher than above SNARK schemes. A recent work [29] proposes an efficient multilinear polynomial commitment scheme, Basefold, which has a faster prover. They construct SNARK from Basefold and prove a single ECDSA signature takes only 122 ms and verifier time is 24

[1)] The implementation in [19] uses Spartan to prove only one instance of the twisted ECDSA signature, which is not the standard ECDSA and it does not provide a well-established proof of security.

ms on a server with 256 GB of RAM. However, the proof size is up to 5.5 MB. [30] also proposes a SNARK with a fast prover and a proof size of about 1 MB for ECDSA. They are both much larger than our scheme. When considering a large batch of signatures, even if using the batching techniques in [31], the prover time and verifier time are linear about the number of signatures. Our verifier time is independent of this. Some IVC can be used in Ethereum's zero-knowledge virtual machines (zkEVMs). But unlike our work, they view the entire block verification process as a virtual machine execution and apply IVC to the computation and memory operations, rather than designing for the specific algorithms. Some projects like [32] use Plonky2 [33] to prove the execution of zkEVM, including verifying ECDSA signatures. However, Plonky2 is not friendly to batch verification, it will be inefficient when proving multiple ECDSA signatures.

To reduce the time and memory requirements of SNARK while maintaining its succinctness and verification efficiency, a new approach involves delegating the complex proving algorithm to powerful devices. A resource-limited prover can simply rent a resource-rich server or distributed computing in a cluster of machines like [34]. However, this will leak the information of the prover to these machines, and it cannot be used for zero-knowledge proof. To protect privacy, [35,36] distributes the secret witness among $n > 1$ parties and uses the MPC protocol to complete the proving algorithm. A drawback of delegating the proof generation to multiple servers is that at least a subset of the servers must be trusted. Garg et al. [37] propose a private delegation with a single server, which is based on the fully homomorphic commitment scheme (FHCom), a heavyweight cryptographic component. They give an analysis of asymptotic complexity without concrete implementation. The delegation scheme gains efficiency entirely due to the higher performance of the servers, each of which executes nearly the same amount of work as the original prover and does not reduce it. Liu et al. [8] construct a distributed Plonk by splitting one large circuit into many small circuits, reducing the computation of each server. The execution of MPC and distributed computations require communication between participants or participants and the delegator or the so-called master node, where network throughput is a critical factor that contributes to the prover time. In this work, we construct a protocol in which low-performance devices can engage locally without delegation.

Recently, several studies have utilized SNARK to construct aggregate signatures, multi-signatures, and threshold signatures [38–40]. In these constructions, the prover only needs to provide SNARK proof for the existence of valid signatures for fixed public keys and public messages, without revealing the signatures to the verifier. Similarly, in some zkRollup designs, transactions are packaged at Layer 2, with metadata sent to Layer 1 along with a SNARK proof verifying the validity of these transactions. This metadata only includes the users' public keys and the transaction details, omitting the signatures, which helps compress the blockchain size by saving storage space. In this context, signatures are treated as part of the SNARK witness, needing only to prove their existence, so the verifier lacks any information about the signatures. This differs from the batch verification approach considered in this paper, where signatures serve as common inputs for both the prover and verifier. While our scheme can also transfer the signatures to the witness of the IVC and eliminate the need for signature storage, retaining the original signatures of transactions is essential for practical blockchain applications when considering accountability and dispute resolution. For instance, when a user wants to retrieve a historical transaction, they only need to access the relevant transaction and its signature, rather than the entire block (or batch). Furthermore, transaction signatures are crucial for resolving disputed transactions in financial scenarios. Without the original signatures, dispute resolution, audits, and reviews would be significantly more complicated.

4 Preliminaries

4.1 Notation

We use λ as the security parameter and $\mathsf{negl}(\cdot)$ as the negligible function. Let $\mathbb{F}$ be the finite field with $|\mathbb{F}| = 2^{\Theta(\lambda)}$. The bold letter $\boldsymbol{a}$ denote the vector and $\boldsymbol{a}_i$ denote the ith element of $\boldsymbol{a}$. For vector $\boldsymbol{a} \in \mathbb{F}^m$ and $\boldsymbol{b} \in \mathbb{F}^m$, $\boldsymbol{a} \circ \boldsymbol{b} = (a_1 b_1, \ldots, a_m b_m)$ is their Hadamard product. $[n]$ represents the set $\{1, \ldots, n\}$, $[a, b]$ represents integers in the range from a to b.

4.2 Building block

Definition 1 (Non-interactive Argument and SNARK). An argument for a language L is a protocol between the prover $\mathcal{P}$ and verifier $\mathcal{V}$ and consists of a tuple of PPT algorithms $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ with the following interface:

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: On input security parameter λ, outputs public parameters pp.
- $\mathcal{K}(\mathsf{pp}, \mathcal{R}) \to (\mathsf{pk}, \mathsf{vk})$: On input public parameters pp and the relation $\mathcal{R}$, outputs proving key pk and verifier key vk.
- $\mathcal{P}(\mathsf{pk}, \mathbb{u}, \mathbb{w}) \to \pi$: On inputs pp, instance $\mathbb{u}$ and witness $\mathbb{w}$, outputs the proof π.
- $\mathcal{V}(\mathsf{vk}, \mathbb{u}, \pi) \to \{0, 1\}$: On inputs pp, instance $\mathbb{u}$ and proof π, outputs 1 or 0 to indicate accept or reject, respectively.

$(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is a non-interactive argument if the communication between the prover $\mathcal{P}$ and verifier $\mathcal{V}$ only includes a proof π.

Let $\mathcal{R}_L$ be a relation consists of all instance-witness pairs $(\mathbb{u}, \mathbb{w})$ with $\mathbb{u} \in L$. A non-interactive argument $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ needs to satisfy the following properties:

- **Perfect completeness**: For any PPT adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{l|l} \exists \mathbb{w}, \text{s.t. } (\mathbb{u}, \mathbb{w}) \in \mathcal{R}_L, & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathcal{R}_L), & \mathbb{u} \leftarrow \mathcal{A}(\mathsf{pp}, L), \\ \pi \leftarrow \mathcal{P}(\mathsf{pk}, \mathbb{u}, \mathbb{w}), & \mathbb{u} \in L \\ \mathcal{V}(\mathsf{vk}, \mathbb{u}, \pi) = 1 & \end{array}\right] = 1.$$

- **Soundness**: For any polynomial time adversaries $\mathcal{P}^*$, for $\epsilon = \mathsf{negl}(\lambda)$,

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathsf{vk}, \mathbb{u}, \pi) = 1, & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ \mathbb{u} \notin L & (\mathbb{u}, \pi) \leftarrow \mathcal{P}^*(\mathsf{pp}), \\ & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathcal{R}_L) \end{array}\right] \leqslant \epsilon.$$

We call the non-interactive argument with knowledge-soundness

defined as below a non-interactive argument of knowledge (NARK).

- **Knowledge-soundness**: For any PPT potentially malicious prover $\mathcal{P}^*$, there exists a polynomial-time extractor $\mathcal{E}$ such that such that for all randomness ρ, for $\epsilon = \mathsf{negl}(\lambda)$,

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathsf{vk}, \mathbb{u}, \pi) = 1, & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathbb{u}, \mathbb{w}) \notin \mathcal{R}_L & (\mathbb{u}, \pi) \leftarrow \mathcal{P}^*(\mathsf{pp}; \rho), \\ & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathcal{R}_L), \\ & \mathbb{w} \leftarrow \mathcal{E}(\mathsf{pp}; \rho) \end{array}\right] \leqslant \epsilon.$$

A NARK $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is succinct if the length of the proof π is sublinear to the length of the witness $\mathbb{w}$, and we call it succinct non-interactive argument of knowledge (SNARK).

Definition 2 (Incrementally Verifiable Computation (IVC)) An incrementally verifiable computation scheme consists of a triple of PPT algorithms $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ with the following interface:

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$: On input security parameter λ, outputs public parameters pp.
- $\mathcal{K}(\mathsf{pp}, \mathsf{F}) \to (\mathsf{pk}, \mathsf{vk})$: On input public parameters pp and a function $\mathsf{F} : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}^a$, outputs prover and verifier key $(\mathsf{pk}, \mathsf{vk})$.
- $\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \mathsf{aux}_{i-1}, \Pi_{i-1}) \to \Pi_i$: On input pk, an index $i \in \mathbb{N}^*$, the input $z_0 \in \{0,1\}^a$ with optional advice $\mathsf{aux}_{i-1} \in \{0,1\}^b$ and a claimed output $z_i \in \{0,1\}^a$, an IVC proof Π_{i-1}, outputs the updated IVC proof Π_i.
- $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \to \{0,1\}$: On input vk, an index $i \in \mathbb{N}^*$, the input $z_0 \in \{0,1\}^a$ and a claimed output $z_i \in \{0,1\}^a$, an IVC proof π_i, outputs 1 or 0 to indicate accept or reject, respectively.

An IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies perfect completeness if for any PPT adversary $\mathcal{A}$:

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathsf{vk}, (i+1, z_0, z_{i+1}), \Pi_{i+1}) = 1 & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ & (\mathsf{F}, i, z_0, z_i, \mathsf{aux}_i, \Pi_i) \leftarrow \mathcal{A}(\mathsf{pp}), \\ & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{F}), \\ & z_{i+1} = \mathsf{F}(z_i, \mathsf{aux}_i), \\ & \mathcal{V}(i, z_0, z_i, \Pi_i) = 1, \\ & \Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i+1, z_0, z_{i+1}), \mathsf{aux}_i, \Pi_i) \end{array}\right] = 1.$$

An IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies knowledge-soundness if for any constant $n \in \mathbb{N}^*$, and PPT potentially malicious prover $\mathcal{P}^*$, there exists a polynomial-time extractor $\mathcal{E}$ such that for all randomness ρ, for $\epsilon = \mathsf{negl}(\lambda)$,

$$\Pr\left[\begin{array}{l|l} z_n \neq z, & \mathsf{pp} \leftarrow \mathcal{G}\left(1^\lambda\right), \\ \mathcal{V}(\mathsf{pp}, z_0, z, \Pi) = 1 & (\mathsf{F}, z_0, z, \Pi) \leftarrow \mathcal{P}^*(\mathsf{pp}, \rho), \\ & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{F}), \\ & (\mathsf{aux}_i)_{i=0}^{n-1} \leftarrow \mathcal{E}(\mathsf{pp}; \rho), \\ & \forall i \in [n]: \\ & z_i \leftarrow \mathsf{F}(z_{i-1}, \mathsf{aux}_{i-1}) \end{array}\right] \leqslant \epsilon.$$

Definition 3 (Commitment Scheme). A commitment scheme is a pair of PPT algorithms $(\mathsf{Setup}, \mathsf{Commit})$ with the following interface:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input security parameter λ, outputs public parameter pp.
- $\mathsf{Commit}(\mathsf{pp}, m) \to c$: On input public parameter pp and the message $m \in \mathbb{F}$, outputs a commitment c.

The Commitment scheme in this work needs to satisfy the following properties:

- **Binding**: for any PPT adversary $\mathcal{A}$ and $\epsilon = \mathsf{negl}(\lambda)$,

$$\Pr\left[\begin{array}{l|l} m_0 \neq m_1 \wedge & \mathsf{pp} \leftarrow \mathsf{Setup}\left(1^\lambda\right), \\ \mathsf{Commit}(\mathsf{pp}, m_0) & (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ = \mathsf{Commit}(\mathsf{pp}, m_1) & \end{array}\right] \leqslant \epsilon.$$

- **Additively homomorphic**: Given two commitments c_0, c_1 respect to message m_0, m_1, it must hold that $c_0 + c_1 = \mathsf{Commit}(\mathsf{ck}, m_0 + m_1)$.

Another property of the commitment scheme is called *hiding*, which requires that the commitment itself does not reveal any information about the committed value. The commitment scheme usually commits to a message with randomness to satisfy the *hiding* property. For simplicity, the randomness in the commitment is omitted in this work. Note that we aim to generate proofs for public signatures without using secret inputs and the hiding property is not necessary for the commitment in our scheme. In particular, the concrete instantiation of our commitment does not need to take randomness as input.

Definition 4 (Hash Functions) A hash function on $\mathbb{F}$ is a pair of efficient algorithms $(\mathsf{Setup}, \mathsf{H})$

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input security parameter λ, outputs public parameters pp.
- $\mathsf{H}(\mathsf{pp}, m) \to h$: On inputs public parameters pp and message $m \in \{0,1\}^*$, outputs a hash value $h \in \mathbb{F}$.

- **Collision-resistant:** A hash function is collision-resistant if for every efficient adversary $\mathcal{A}$ and $\epsilon = \mathsf{negl}(\lambda)$,

$$\Pr\left[\begin{array}{l|l} \mathsf{H}(\mathsf{pp}, m_0) = \mathsf{H}(\mathsf{pp}, m_1) & \mathsf{pp} \leftarrow \mathsf{Setup}\left(1^\lambda\right) \\ \wedge\, m_0 \neq m_1 & (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] \leqslant \epsilon.$$

- **Zero-testing:** Let Commit be the commit algorithm of a commitment scheme that has the binding property, and $\mathsf{P} : \mathbb{F} \to \mathbb{F}^d[X]$ be a deterministic function that maps a field element to a polynomial with degree $d = O(\lambda)$. A hash function has the general zero testing property if for every PPT adversary $\mathcal{A}$, for $\epsilon = \mathsf{negl}(\lambda)$,

$$\Pr\left[\begin{array}{l|l} h \leftarrow \mathsf{H}(\mathsf{pp}, \mathsf{Commit}(\mathsf{pp}, p)), & \mathsf{pp} \leftarrow \mathsf{Setup}\left(1^\lambda\right), \\ \mathsf{P}(p)(h) = 0 & p \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] \leqslant \epsilon.$$

Definition 5 (Committed Relaxed R1CS). Let $\mathbb{F}$ denotes a finite field, consider the parameters $m, \ell \in \mathbb{N}$ where $m > \ell$. A committed relaxed R1CS instance is a tuple $\mathbb{U} = (\overline{E}, s, \overline{W}, x)$, where $s \in \mathbb{F}, x \in \mathbb{F}^\ell$, and $\overline{E}$ and $\overline{W}$ are commitments to $E \in \mathbb{F}^m$ and $W \in \mathbb{F}^{m-\ell-1}$. A committed relaxed R1CS instance $\mathbb{U} = (\overline{E}, s, \overline{W}, x)$

is satisfiable with respect to the structure $\mathsf{s}=(A,B,C)$ if there exist a relaxed witness $\mathbb{W}=(E,W)$ such that $(\mathbb{U},\mathbb{W})\in\mathcal{R}_{\mathsf{s}}$, where the committed relaxed R1CS relation $\mathcal{R}_{\mathsf{s}}$ is defined by

$$\mathcal{R}_{\mathsf{s}}:=\left\{(\mathbb{U},\mathbb{W}):\begin{array}{l}\overline{E}=\mathsf{Commit}(\mathsf{pp}_{\mathsf{E}},E),\\ \overline{W}=\mathsf{Commit}(\mathsf{pp}_{\mathsf{W}},W),\\ Z=(W,x,s),\\ A\cdot Z\circ B\cdot Z=s\cdot(C\cdot Z)+E\end{array}\right\}.$$

There are at most $k=\Omega(m)$ non-zero elements in each of the matrix $A,B,C\in\mathbb{F}^{m\times m}$. When setting $s=1,\overline{E}=\vec{\mathbf{0}}$, the committed relaxed R1CS instance is a standard R1CS instance, where the constraint equation is defined as $A\cdot Z\circ B\cdot Z=C\cdot Z$.

As explained in Definition 3, unlike Nova, the randomness of the commitment in this work is omitted.

Definition 6 (Folding Scheme). A folding scheme for committed relaxed R1CS consists of a tuple of algorithms $(\mathcal{G},\mathcal{K},\mathcal{P},\mathcal{V})$:

- $\mathcal{G}(1^{\lambda})\to\mathsf{pp}$: On input security parameter λ, outputs public parameters pp.
- $\mathcal{K}(\mathsf{pp},\mathsf{s})\to\mathsf{pk}$: On inputs public parameters pp, a common structure s, outputs proving key pk.
- $\mathcal{P}(\mathsf{pk},(\mathbb{U},\mathbb{W}),(\mathbb{u},\mathbb{w}))\to(\overline{T},(\mathbb{U}',\mathbb{W}'))$: On inputs proving key pk, two committed relaxed R1CS instance-witness pair $(\mathbb{U},\mathbb{W}),(\mathbb{u},\mathbb{w})$, outputs a folded committed relaxed R1CS instance-witness pair $(\mathbb{U}',\mathbb{W}')$ with a folding proof $\overline{T}$.
- $\mathcal{V}(\mathsf{pp},\mathbb{U},\mathbb{u},\overline{T})\to\mathbb{U}'$: On inputs public parameters pp, two committed relaxed R1CS instance $\mathbb{U},\mathbb{u}$, and a folding proof $\overline{T}$, outputs a folded committed relaxed R1CS instance $\mathbb{U}'$.

A folding scheme satisfies perfect completeness if for all PPT adversaries $\mathcal{A}$,

$$\Pr\left[\begin{array}{l}\mathbb{U}'=\mathbb{U}'',\\ (\mathbb{U}',\mathbb{W}')\in\mathcal{R}_{\mathsf{s}}\end{array}\;\middle|\;\begin{array}{l}\mathsf{pp}\leftarrow\mathcal{G}\left(1^{\lambda}\right),\\ ((\mathbb{U},\mathbb{W}),(\mathbb{u},\mathbb{w}))\leftarrow\mathcal{A}(\mathsf{pp}),\\ (\mathbb{U},\mathbb{W})\in\mathcal{R}_{\mathsf{s}},(\mathbb{u},\mathbb{w})\in\mathcal{R}_{\mathsf{s}},\\ \mathsf{pk}\leftarrow\mathcal{K}(\mathsf{pp},\mathsf{s}),\\ (\overline{T},\mathbb{U}',\mathbb{W}'),\\ \leftarrow\mathcal{P}(\mathsf{pk},(\mathbb{U},\mathbb{W}),(\mathbb{u},\mathbb{w})),\\ \mathbb{U}''\leftarrow\mathcal{V}(\mathsf{pp},\mathbb{U},\mathbb{u},\overline{T})\end{array}\right]=1.$$

A folding scheme satisfies knowledge soundness if for any PPT potentially malicious $\mathcal{P}^*$ and $\epsilon=\mathsf{negl}(\lambda)$, there is a PPT extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{array}{l}(\mathbb{U},\mathbb{W})\in\mathcal{R}_{\mathsf{s}},\\ (\mathbb{u},\mathbb{w})\in\mathcal{R}_{\mathsf{s}}\end{array}\;\middle|\;\begin{array}{l}\mathsf{pp}\leftarrow\mathcal{G}\left(1^{\lambda}\right),\\ (\mathsf{s},\mathbb{U},\mathbb{u})\leftarrow\mathcal{P}^*(\mathsf{pp},\rho),\\ (\mathbb{w},\mathbb{W})\leftarrow\mathcal{E}(\mathsf{pp},\rho)\end{array}\right]\geqslant$$
$$\Pr\left[(\mathbb{U}',\mathbb{W}')\in\mathcal{R}_{\mathsf{s}}\;\middle|\;\begin{array}{l}\mathsf{pp}\leftarrow\mathcal{G}\left(1^{\lambda}\right),\\ (\mathsf{s},\mathbb{U},\mathbb{u})\leftarrow\mathcal{P}^*(\mathsf{pp},\rho),\\ \mathsf{pk}\leftarrow\mathcal{G}(\mathsf{pp},\mathsf{s}),\\ (\overline{T},\mathbb{U}',\mathbb{W}')\leftarrow\mathcal{P}^*(\mathsf{pk},\rho),\\ \mathbb{U}'\leftarrow\mathcal{V}(\mathsf{vk},\mathbb{U},\mathbb{u},\overline{T})\end{array}\right]-\epsilon.$$

Let H denote the Hash function in Denifition 4, which satisfies the collision-resistant property and general zero testing property. A folding scheme $(\mathcal{G},\mathcal{K},\mathcal{P},\mathcal{V})$ is non-interactive if the communication between $\mathcal{P}$ and $\mathcal{V}$ only includes a proof $\overline{T}$, and the randomness r used to generate the outputs is computed as $\mathsf{H}(\mathsf{vk},\mathbb{U},\mathbb{u},\overline{T})$ by $\mathcal{P}$ and $\mathcal{V}$ locally.

Definition 7 (Digital Signature in Hash-and-Sign Paradigm) A digital signature scheme consists of a tuple of PPT algorithms $(\mathsf{Setup}_{\mathsf{sig}},\mathsf{Gen}_{\mathsf{sig}},\mathsf{Sign},\mathsf{Verify})$, which are defined as follows.

- $\mathsf{Gen}(\mathsf{pp})$: On input public parameters, outputs a keypair (sk,pk).
- $\mathsf{Sign}(sk,m)\to\sigma$: On input a signing key sk and a message m, outputs the σ as signature of m.
- $\mathsf{Verify}(pk,\sigma,m)\to\{0,1\}$: On input a signature σ with its message m and public key pk, outputs 1 or 0 to indicate whether the signature passes the verification or not.

Given a hash function hash with collision-resistant property, if the algorithm Sign and Verify are executed as the following process, the digital signature scheme is in the hash-and-sign paradigm.

- $\mathsf{Sign}(sk,m)\to\sigma$: On input a signing key sk and a message m, computes that
  - $e\leftarrow\mathsf{hash}(m)$,
  - $\sigma\leftarrow\mathsf{Sign}'(sk,e)$.
- $\mathsf{Verify}(pk,\sigma,m)\to\{0,1\}$: On input a signature σ with its message m and public key pk,
  - $e\leftarrow\mathsf{hash}(m)$,
  - $\{0,1\}\leftarrow\mathsf{Verify}'(pk,e)$.

The algebraic group model (AGM). In the algebraic group model, AGM [41], an algebraic adversary should output a representation vector whenever it outputs a group element in terms of all the group elements that have been seen by the adversary so far. We use capital letter X to denote a group element, let $(G_1,\ldots,G_k)$ be all group elements known to the adversary, $\boldsymbol{x}$ is the corresponding representation vector of X such that $X=\sum_{i=1}^{k}\boldsymbol{x}_iG_i$.

Refined AGM for Nova. To cover the situations in Nova, a refined AGM was introduced in [42] that proposes three requirements:

(1) All group elements in the input of the algebraic adversary are common random strings.

(2) Adversarial algorithms are non-interactive and cannot obtain any additional information by querying.

(3) Pre-declared multiple encodings are allowed and the adversary should output all relevant representation vectors. In particular, when the adversary outputs a represent vector $\boldsymbol{a}$ of group element A, if a group element B is embedded in $\boldsymbol{a}$ by another encoding way, the algebraic adversary should output the representation vector $\boldsymbol{b}$ of B as well.

4.3 Overview of Nova

Consider an IVC statement as shown in Fig. 1. For a function $\mathsf{F}:\mathbb{F}^{a\times b}\to\mathbb{F}^{a}$, there exist auxiliary inputs $(\mathsf{aux}_0,\ldots,\mathsf{aux}_{n-1})$ such that

$$\mathsf{F}(\ldots\mathsf{F}(\mathsf{F}(z_0,\mathsf{aux}_0),\mathsf{aux}_1)\ldots,\mathsf{aux}_{n-1})=z_n.$$

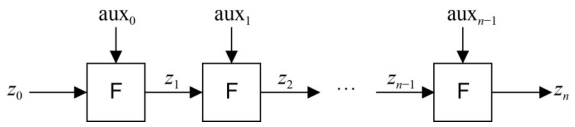

Fig. 1 IVC chain

To prove this statement, the function F will be arithmetized as some committed relaxed R1CS constraints by the IVC prover and verifier before running the Nova IVC scheme [16] iteratively. For $i \in [n]$, in the ith iteration, the prover claims that the i-th step is evaluated correctly, i.e., $\mathsf{F}(z_i, \mathsf{aux}_i) = z_{i+1}$, and all the previous $i-1$ steps are also evaluated correctly, i.e.,

$$\mathsf{F}(\ldots(\mathsf{F}(\mathsf{F}(z_0, \mathsf{aux}_0), \mathsf{aux}_1), \ldots), \mathsf{aux}_{i-1}) = z_i.$$

As shown in Fig. 2, in the ith iteration, a folding scheme is invoked to fold the current instance $\mathbb{u}_i$ into a folded instance $\mathbb{U}_i$ that will be updated to $\mathbb{U}_{i+1}$. In addition, a new instance $\mathbb{u}_{i+1}$ is introduced to represent the execution of the step function and the folding process. Precisely, the instance $\mathbb{u}_{i+1}$ is a standard R1CS instance arithmetizing the trace of an augment function F', which includes computing $\mathsf{F}(z_i, \mathsf{aux}_i) = z_{i+1}$, generating $\mathbb{U}_{i+1}$ and checking some consistency between two neighboring iterations by a hash function H. The satisfiability of $\mathbb{u}_{i+1}$ implies the computation from z_i to z_{i+1} and the folding process for getting $\mathbb{U}_{i+1}$ are correctly executed. The instance $\mathbb{U}_{i+1}$ is a committed relaxed R1CS instance, and its satisfiability implies the satisfiability of the instance of $\mathbb{U}_i$ and $\mathbb{u}_i$, which further implies the correctness of the computation from z_0 to z_i by recursion. The hash function H can be seen as a binding commitment to z_i, the consistency between the current input and last output is checked by it in the instance $\mathbb{u}_i$. Therefore, the correctness of the IVC statement can be delayed until the end to be checked.

Nova on cycle of elliptic curves. Let E_1, E_2 represent two elliptic curves that satisfy the "cyclic" property. More precisely, the base field $\mathbb{F}_p$ of E_1 is exactly the scalar field of E_2, and the base field $\mathbb{F}_q$ of E_2 is exactly the scalar field of E_1.

In the actual structure, due to the need for additive homomorphism in the folding scheme, the commitment scheme in the R1CS instance is realized by the Pedersen vector commitment, which is defined over elliptic curve groups. Note that operations on the elliptic curve's base field dominate the folding scheme's verification algorithm, while the R1CS constraints are specified on its scalar field. If one describes the verification algorithm of the folding scheme on the scalar field directly, it will lead to many operations on the non-native field that incur a lot of R1CS constraints. An innovative approach in the implementation of Nova [43] is using a cycle of elliptic curves, the correctness of the folding scheme on one curve can be checked on another curve. Precisely, the folding scheme is executed on both elliptic curves E_1 and E_2 while the step function F is represented only on one elliptic curve, without loss of generality, E_1. The folding scheme verifier on E_1 is represented as R1CS constraints over the scalar field $\mathbb{F}_p$ of the curve E_2, while the folding scheme verifier on E_2 is represented as R1CS constraints over the scalar field $\mathbb{F}_q$ of E_1.

In the original Nova, operations on two curves are switched in a crossover manner, finally, the IVC protocol outputs a proof consisting of 4 instance-witness pairs. Nguyen et al. [44] found security problems with such an implementation and proposed the refined Nova, where the operations on the two elliptic curves are performed in alternating order, ultimately generating 3 instance-witness pairs as the IVC proof.

Compress with SNARK. If one wants to check the correctness of the computation of the first i iterations of the IVC scheme, one can easily run the IVC verifier on the IVC proof of the ith iteration, which results in a slower verification. In Nova, a SNARK protocol is introduced additionally at the end of the IVC, making it sufficient to check the validity of the succinct proofs of the SNARK. This compression process with SNARK is implemented by Spartan, which in fact can be replaced by any appropriate SNARK with a commitment scheme that satisfies the requirements of the folding scheme. Due to the concrete efficiency requirements in practice.

In addition to the efficiency benefits, compressing with SNARK at the end of the IVC chain also serves some security purposes. As mentioned in [44], there may exist a malleability attack in Nova IVC when the step function is non-deterministic. Adding a SNARK protocol with zero-knowledge property can prevent this attack by hiding auxiliary inputs from the adversary, as long as the zkSNARK is simulation extractable. Looking ahead, since the step function in our scheme is deterministic, we do not require the zero-knowledge property and the attack in [44] cannot apply to our scheme.

5 Batch verification of ECDSA in Nova

In this section, we present a general construction for batch verification of signatures based on the Nova IVC scheme. We

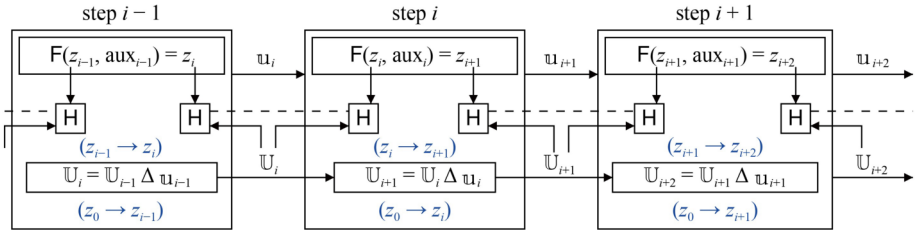

Fig. 2 Simplified Nova IVC for $i > 1$. Here we only sketch the main functionality of each step of the augment function in Nova, omitting some detailed processes such as hash checking $\mathbb{u}_i.x = \mathsf{H}(\mathsf{pp}, i, z_0, z_i, \mathbb{U}_i)$ and the trivial case $i = 1$. The symbol Δ denotes the verifier algorithm for the folding scheme, which means that $\mathbb{U} \Delta\, \mathbb{u} = \mathcal{V}_{\mathsf{fs}}(\mathsf{pp}_{\mathsf{fs}}, \mathbb{U}, \mathbb{u}, \overline{\mathsf{T}})$. The blue font indicates the state implied if the statements in the box above it are true. Two parts connected by a dotted line are checked for equivalence in the protocol, but they are not explicitly input-output relationships

optimize the instantiation and construct a transparent scheme BEATS for batch verification of ECDSA. Before introducing our construction, we show a simple attempt and its security problem in Section 5.1. We argue that a regular IVC scheme cannot support batch verification of signatures.

5.1 A simple attempt

As mentioned in Section 2, applying the folding scheme directly and proving the folding process cannot guarantee the consistency of the signatures claimed by the prover and those received by the verifier. To achieve consistency, a hash function can be added as in Fig. 2 for checking the equality of instances between two neighboring steps, as shown in Fig. 3. Symbol definitions of Fig. 3 are the same as in Fig. 2. "=1" in step i means z_i will be set to 1 when running the hash function H to check the equivalence. The blue font indicates the state implied if the statements in the box above it are true. We treat the output of F as a variable, denoted by z. [2] In addition to running the verification algorithm of the signature, the function F checks the correctness of the previous step. For each $i \in [n]$, $z_i = 1$ if and only if $\mathsf{Verify}(pk_i, \sigma_i, m_i) = 1$ and $z_{i-1} = 1$.

After building the connection, we find that the construction is equivalent to the IVC chain in Fig. 4. In the IVC construction, the prover should prove the computation of $\mathsf{F}(\ldots\mathsf{F}(\mathsf{F}(z_0, y_0), y_1)\ldots y_{n-1}) = z_n$. Since the function F is a common parameter and $z_0 = z_n = 1$ is the public instance, the IVC statement will be accepted if the IVC proof passes the verification. Thus all n executions of the function F are correct, which implies that all n signatures are valid.

However, in the IVC context, each y_i is the auxiliary input of F, which is not explicitly sent to the verifier. An important fact is that auxiliary inputs are non-deterministic and not unique. In applications such as blockchain transactions where the verifier must receive all signatures, each y_i must be public and deterministic. The construction in Fig. 3 only claims that the prover knows some valid signature triples $(pk_0^*, \sigma_0^*, m_0^*), \ldots, (pk_n^*, \sigma_n^*, m_n^*)$ if there are n steps in total. Still, these n signatures may not be consistent with the n signatures $(pk_0, \sigma_0, m_0), \ldots, (pk_n, \sigma_n, m_n)$ sent to the verifier in advance. So a malicious prover can send a batch of invalid signatures and instead generate proofs using some trivial valid signatures, such as $(pk^*, \sigma^*, \mathbf{0})$ where the pk^* is the public key of the prover who naturally knows its private key pk^*, thereby tricking the verifier without attacking against the unforgeability of the signature scheme. In blockchain, this attack will lead to fake transactions, so we must ensure the consistency of the verified signatures between the prover and verifier.

5.2 Batch verification of signature in Nova

We first propose a general IVC-based construction for batch verification of arbitrary digital signatures. Suppose the prover has verified all signatures at the beginning of the protocol and sent the valid ones among them to the verifier. In Section 5.1 for simplicity, we use the function F to represent only one signature verification. In fact, multiple signatures can be contained in one step of the IVC, where the concrete efficiency depends on the number of signatures in each step.

Suppose that the verifier receives $t \in \mathbb{N}^*$ signatures in total. The t signatures will be divided into $n = \lceil \frac{t}{b} \rceil$ blocks, where $b \in \{1, \ldots, t\}$ is the block size that describes the number of signatures in a block. Each step of the IVC handles one block so that n is the number of steps of IVC. For each $i \in [n]$, we set the input to each F is a vector $\boldsymbol{y}_i = \left[\left(pk_{i,j}, \sigma_{i,j}, m_{i,j}\right)\right]_{j \in [b]}$. Let $\boldsymbol{z}_i$ denote the output vector of F, which indicates the verification result of the signatures in $\boldsymbol{y}_i$. The description of $\mathsf{F} : \mathbb{F}^a \to \mathbb{F}^b$ where $|\boldsymbol{y}_i| = a, |\boldsymbol{z}_i| = |\mathbf{1}^b| = b$ is public.

The main idea of our construction is depicted in Fig. 5. Instead of taking F as a step function in IVC simply, we introduce an additional collision-resistant hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{F}$. In the naïve construction in Section 5.1, since each input $\boldsymbol{y}_i$ is the auxiliary input and $\boldsymbol{z}_i$ as an intermediate value will not be sent to the verifier, the function F is treated as a non-deterministic function. It fails to work in the context of verifying deterministic signatures. We use hash function H as a commitment to these non-deterministic inputs and outputs, and the binding property relies on the collision resistance of the hash function. We initialize h_0 to a public value, and then sequentially take $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$ as input to H like the Merkle-Damgard structure until h_n is output as the resulting commitment. In applications such as blockchain, it is not enough for the prover only to send the commitment to the signatures and then apply a probabilistic proof protocol. Since the verifier can access all these signatures, the verifier must also compute the commitment. Finally, all hash values are connected into an IVC chain as Fig. 1, where the statement is that $h_i = \mathsf{H}(h_{i-1}, \boldsymbol{y}_{i-1}, \mathsf{F}(\boldsymbol{y}_{i-1}))$ for all $i \in [n]$.

We modify the scheme proposed in [16] to obtain the formal description of the IVC scheme. Compared to the original definition,

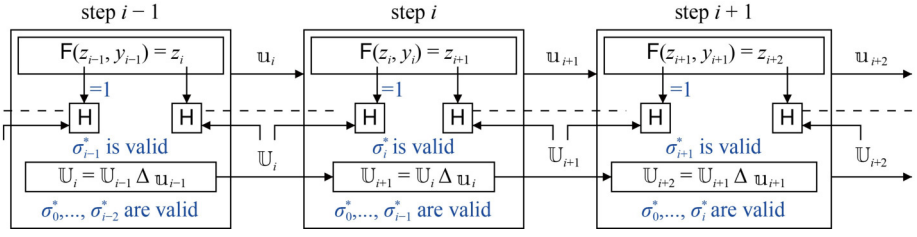

Fig. 3 Modification of Fig. 2 with consistency check

[2] Although the output of F can be viewed as a constant and embedded into a fixed R1CS structure, it is more general to provide explicit output for circuits. F here describes only the evaluation, not the arithmetized circuit or constraints system.

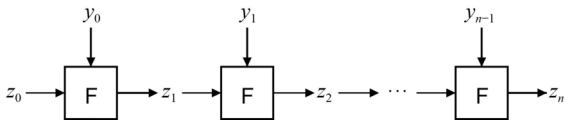

Fig. 4 IVC chain respect to Fig. 3

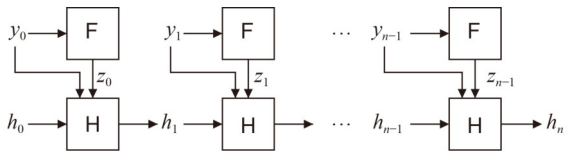

Fig. 5 Our construction

we view $\mathsf{H} \circ \mathsf{F}$ as a composite function. In the i-th step of IVC, the prover computes and proves an augment function F' shown in Fig. 6. The modified IVC scheme is shown in Fig. 7. NIFS shown in Fig. 8 denotes the non-interactive folding scheme in Definition 6. Let trace denote the execution of the augment function F', which can be represented as a committed relaxed R1CS with structure $\mathsf{s}_{F'}$. The instance-witness pair $(\mathbb{u}, \mathbb{w})$ describes the trace of F' in each step of IVC. First, F' executes the step function $\mathsf{H} \circ \mathsf{F}$ on $h_{i-1}, \boldsymbol{y}_{i-1}$ to obtain the incremental computation result $h_i = \mathsf{H}(h_{i-1}, \boldsymbol{y}_{i-1}, \mathsf{F}(\boldsymbol{y}_{i-1}))$. Second, F' runs the verifier of NIFS to fold two committed relaxed R1CS instances $\mathbb{U}_i$ and $\mathbb{u}_i$ into a new instance $\mathbb{U}_{i+1}$. Then the IVC prover generates $\mathbb{u}_{i+1}$ to describe the trace of the ith invocation of the augment function F'. The instance $\mathbb{u}_{i+1}$ claims the correctness of $\mathsf{H} \circ \mathsf{F}$ and folding process in the ith step.

We provide our full protocol $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for batch signature verification in Fig. 9. Similarly, we assume that the prover has sent the signature information $\boldsymbol{y} = [\boldsymbol{y}_{i,j}]_{i\in[n], j\in[b]}$ to the verifier and claims that all these signatures are valid, i.e., $\forall i \in [n], j \in [b]$, $\mathsf{F}(\boldsymbol{y}_{i,j}) = \boldsymbol{z}_{i,j} = 1$, which implies that $\boldsymbol{z} = [\boldsymbol{z}_{i,j}]_{i\in[n], j\in[b]} = \mathbf{1}^t$. Firstly, the prover computes the hash chain with length n to obtain the commitment h_n. Then the prover runs the IVC prover for the IVC chain in 5 to generate an IVC proof Π. Due to the large proof size, instead of running the IVC verifier, the SNARK prover SNARK.P will be invoked on the folded IVC proof. After receiving a proof from the prover, the verifier checks the commitment by comparing the h_n from the prover and h that is computed locally on $\boldsymbol{y}$ and $\boldsymbol{z}$, where $\boldsymbol{z}$ is set to $\mathbf{1}^t$ to enforce the prover can only prove valid signatures. Then the verifier runs the folding scheme verifier

$F'(\mathsf{pp}_{\mathsf{fs}}, \mathbb{U}_i, \mathbb{u}_i, (i, h_0, h_i), \boldsymbol{y}_i, \overline{T}_i) \to x.$

If i is 1, output $\mathsf{H}(\mathsf{pp}, 1, h_0, \mathsf{H}(h_0, \boldsymbol{y}_0, \mathsf{F}(\boldsymbol{y}_0)), \mathbb{u}_\perp)$.

Otherwise,

1. Check that
   $\mathbb{u}_i.x = \mathsf{H}(\mathsf{pp}, i, h_0, \mathsf{H}(h_{i-1}, \boldsymbol{y}_{i-1}, \mathsf{F}(\boldsymbol{y}_{i-1})), \mathbb{U}_{i-1})$,
   where $\mathbb{u}_i.x$ is the public IO of $\mathbb{u}_i$.
2. Check that $(\mathbb{u}_i.\overline{E}, \mathbb{u}_i.s) = (\mathbb{u}_\perp.\overline{E}, 1)$.
3. Compute $\mathbb{U}_{i+1} = \mathsf{NIFS.V}(\mathsf{pp}, \mathbb{U}_i, \mathbb{u}_i, \overline{T}_i)$.
4. Output $\mathsf{H}(\mathsf{pp}, i, h_i, \mathsf{H}(h_i, \boldsymbol{y}_i, \mathsf{F}(\boldsymbol{y}_i)), \mathbb{u}_i)$.

Fig. 6 Modified augment function

$\mathcal{G}(1^\lambda) \to \mathsf{pp}$: Output $\mathsf{pp} \leftarrow \mathsf{NIFS.G}(1^\lambda)$.

$\mathcal{K}(\mathsf{pp}, \mathsf{F}, \mathsf{H}) \to (\mathsf{pk}, \mathsf{vk})$:

1. $\mathsf{pk} \leftarrow \mathsf{NIFS.K}(\mathsf{pp}, \mathsf{s}_{F'})$.
2. Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((\mathsf{F}, \mathsf{H}, \mathsf{pk}_{\mathsf{fs}}), (\mathsf{F}, \mathsf{H}, \mathsf{pp}))$.

$\mathcal{P}(\mathsf{pk}, i, h_0, h_i, \boldsymbol{y}_i, \boldsymbol{z}_i, \Pi_i) \to \Pi_{i+1}$:

Parse Π_i as $((\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i))$ and then

1. If $i = 0$, compute
   $\left(\mathbb{U}_{i+1}, \mathbb{W}_{i+1}, \overline{T}_{i+1}\right) \leftarrow (\mathbb{u}_\perp, \mathbb{w}_\perp, \mathbb{u}_\perp.\overline{E})$.
   Otherwise, compute
   $\left(\mathbb{U}_{i+1}, \mathbb{W}_{i+1}, \overline{T}_{i+1}\right) \leftarrow \mathsf{NIFS.P}(\mathsf{pk}, (\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i))$.
2. Compute
   $(\mathbb{u}_{i+1}, \mathbb{w}_{i+1}) \leftarrow \mathsf{trace}\left(F', \left(\mathsf{vk}, \mathbb{U}_i, \mathbb{u}_i, (i, h_0, h_i), \boldsymbol{y}_i, \overline{T}_i\right)\right)$.
3. Output $\Pi_{i+1} \leftarrow ((\mathbb{U}_{i+1}, \mathbb{W}_{i+1}), (\mathbb{u}_{i+1}, \mathbb{w}_{i+1}))$.

$\mathcal{V}(\mathsf{vk}, i, h_0, h_i, \Pi_i) \to \{0, 1\}$:

If $i = 0$, check that $h_0 = h_i$.

Otherwise,

1. Check that $h_i = h$.
2. Parse Π_i as $((\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i))$.
3. Check that $\mathbb{u}_i.x = \mathsf{H}(\mathsf{vk}, i, h_0, h_i, \mathbb{U}_i)$.
4. Check that $\left(\mathbb{u}_i.\overline{E}, \mathbb{u}_i.s\right) = (\mathbb{u}_\perp.\overline{E}, 1)$.
5. Check that $(\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i) \in \mathcal{R}_{s_{F'}}$.

Fig. 7 Modified IVC scheme

NIFS.V and the SNARK verifier SNARK.V on the folded IVC instance. If all conditions are met, the verifier accepts the statement, which implies all received signatures are valid.

5.3 Batch verification of ECDSA

Intuitively, we can apply the protocol in Fig. 9 to the batch verification of ECDSA by setting the function F as the verification of ECDSA. Although this protocol is general for arbitrary signature schemes, the selection of parameters and realization of some functionalities may affect the efficiency in practical applications.

5.3.1 Optimizations on frontend implementation

The standard ECDSA signature [45] is shown in Fig. 10. The process of compiling the verification algorithm to an arithmetic representation such as R1CS is often called the frontend of the protocol. The frontend efficiency plays an important role in the overall proving process, and it is measured by the size of the arithmetic representation, which in this work refers to the number of R1CS formed constraints.

Hash functions. Since the hash function H is a part of the step function in the IVC statement, it needs to be arithmetized as R1CS defined over a large field that the commitment scheme and SNARK protocol defined on. To reduce the number of constraints, we use the SNARK-friendly Hash function, Poseidon [46], to instantiate it. Similarly, the folding scheme verification algorithm as shown in Fig. 2 will also be proven in R1CS, so we can also use Poseidon to

$\mathcal{G}(1^\lambda) \to \mathsf{pp}$: Output commitment keys $\mathsf{pp_E} \leftarrow \mathbb{G}^m, \mathsf{pp_W} \leftarrow \mathbb{G}^{m-\ell-1}$ where $\mathbb{G}$ is a group with additive homomorphism.

$\mathcal{K}(\mathsf{pp}, \mathsf{s} = (A, B, C)) \to \mathsf{pk}$: Output $\mathsf{pk} \leftarrow (\mathsf{pk}, \mathsf{s}))$.

$\mathcal{P}(\mathsf{pk}, (\mathbb{U}, \mathbb{W}), (\mathbb{u}, \mathbb{w})) \to \overline{T}, (\mathbb{U}', \mathbb{W}')$:

1. Parse $(\mathbb{U}, \mathbb{W})$ as $\left(\overline{E}^{(1)}, s^{(1)}, \overline{W}^{(1)}, x^{(1)}\right), (E^{(1)}, W^{(1)})$ and then set $z^{(1)} = (W^{(1)}, x^{(1)}, s^{(1)})$.
   Parse $(\mathbb{u}, \mathbb{w})$ as $\left(\overline{E}^{(2)}, s^{(2)}, \overline{W}^{(2)}, x^{(2)}\right), (E^{(2)}, W^{(2)})$ and then set $z^{(2)} = (W^{(2)}, x^{(2)}, s^{(2)})$.
2. Compute $T = Az^{(1)} \circ Bz^{(2)} + Az^{(2)} \circ Bz^{(1)} - u^{(1)} \cdot Cz^{(2)} - u^{(2)} \cdot Cz^{(1)}$.
3. Compute $\overline{T} = \mathsf{com}(\mathsf{pp_E}, T) = \langle T, \mathsf{pp}_E \rangle$.
4. $r \leftarrow \mathsf{H}(\mathbb{U}, \mathbb{u}, \overline{T})$.
5. Compute

$$E' \leftarrow E^{(1)} + r \cdot E^{(2)},\ W' \leftarrow W^{(1)} + r \cdot W^{(2)} \quad (1)$$

$$\overline{E'} \leftarrow \overline{E'}^{(1)} + r \cdot \overline{E'}^{(2)},\ \overline{W'} \leftarrow \overline{W'}^{(1)} + r \cdot \overline{W'}^{(2)} \quad (2)$$

$$s' \leftarrow s^{(1)} + r \cdot s^{(2)},\ x' \leftarrow x^{(1)} + r \cdot x^{(2)} \quad (3)$$

6. Output $\overline{T}, (\mathbb{U}', \mathbb{W}')$, where $(\mathbb{U}', \mathbb{W}') = \left((\overline{E'}, s', \overline{W'}, x'), (E', W')\right)$.

$\mathcal{V}(\mathsf{pp}, (\mathbb{U}, \mathbb{u}), \overline{T}) \to \mathbb{U}'$:

1. Parse $\mathbb{U}$ as $\left(\overline{E}^{(1)}, s^{(1)}, \overline{W}^{(1)}, x^{(1)}\right)$, and parse $\mathbb{u}$ as $\left(\overline{E}^{(2)}, s^{(2)}, \overline{W}^{(2)}, x^{(2)}\right)$.
2. $r \leftarrow \mathsf{H}(\mathbb{U}, \mathbb{u}, \overline{T})$.
3. $\overline{E'} \leftarrow \overline{E'}^{(1)} + r \cdot \overline{E'}^{(2)}, \overline{W'} \leftarrow \overline{W'}^{(1)} + r \cdot \overline{W'}^{(2)}$.
   $s' \leftarrow s^{(1)} + r \cdot s^{(2)}, x' \leftarrow x^{(1)} + r \cdot x^{(2)}$.
4. Output $\mathbb{U}'$ where $\mathbb{U}' = \left(\overline{E'}, s', \overline{W'}, x'\right)$.

Fig. 8 Group-based non-interactive folding scheme in [42]

generate the randomness in the NIFS as long as it satisfies the general zero testing property in Definition 4. Since the Poseidon is used as a random oracle in the original non-interactive folding scheme in Nova, it can be deduced that Poseidon has the general zero-testing property in the AGM According to Lemma 1.

Lemma 1 ([42]). The random oracle has the zero-testing property.

Notice that the hash function hash in ECDSA is instantiated by Keccak-256 in many blockchain applications. Applying the general protocol in Fig. 9 on the function ECDSA.Verify requires to compile the Keccak-256 defined over the expansion of binary field $\mathbb{F}_2$ as constraints over a large field $\mathbb{F}_p$, which will significantly increase the size of the R1CS.

In the SNARK scheme, the verifier must compute the local commitment to the public inputs, which include the signed messages in this work. Furthermore, as Keccak-256 can be evaluated very efficiently on the CPU, it is not necessary for the verifier to delegate this computation to the prover. Therefore, for a batch of signatures in the hash-and-sign paradigm, we replace the message $m_{i,j}$ in the input

Suppose that the size of the package is $t = nb > 0$, where $n, b \in \mathbb{N}^*$ are fixed public parameters that depend on the size of the signature verification function F. Let $\boldsymbol{y} = (pk_{i,j}, \sigma_{i,j}, m_{i,j})_{i\in[n], j\in[b]}$ be the package of signature triples and $\boldsymbol{y}_i = (pk_{i,j}, \sigma_{i,j}, m_{i,j})_{j\in[b]}$. The statement is that for $i \in [n]$, $\mathsf{F}(\boldsymbol{y}_i) = \boldsymbol{z}_i = \mathbf{1}^b$, where $|\boldsymbol{y}_i| = |\boldsymbol{z}_i| = b$.

$\mathcal{G}(1^\lambda) \to \mathsf{pp}$:

1. Compute $\mathsf{pp_{fs}} \leftarrow \mathsf{NIFS.G}(1^\lambda)$.
2. Compute $\mathsf{pp_{snark}} \leftarrow \mathsf{SNARK.G}(1^\lambda)$.
3. Output $(\mathsf{pp_{fs}}, \mathsf{pp_{snark}})$.

$\mathcal{K}(\mathsf{pp}, \mathsf{F}, \mathsf{H}) \to (\mathsf{pk}, \mathsf{vk})$:

1. $\mathsf{pk_{fs}} \leftarrow \mathsf{NIFS.K}(\mathsf{pp.pp_{fs}}, \mathsf{s}_{F'})$.
2. $(\mathsf{pk_{snark}}, \mathsf{vk_{snark}}) \leftarrow \mathsf{SNARK.K}(\mathsf{pp.pp_{snark}}, \mathsf{s}_{F'})$.
3. Output

$$(\mathsf{pk}, \mathsf{vk}) \leftarrow ((\mathsf{F}, \mathsf{H}, \mathsf{pk_{fs}}, \mathsf{pk_{snark}}), (\mathsf{F}, \mathsf{H}, \mathsf{pp.pp_{fs}}, \mathsf{vk_{snark}})).$$

$\mathcal{P}(\mathsf{pk}, n, h_0, \boldsymbol{y}, \boldsymbol{z}) \to \pi$:

1. For $i \in [n-1]$, compute
   $h_i \leftarrow \mathsf{H}(h_{i-1}, \boldsymbol{y}_i, \boldsymbol{z}_i)$.
   $\Pi_i \leftarrow \mathsf{IVC.P}((\mathsf{F}, \mathsf{H}, \mathsf{pk.pk_{fs}}), (i, h_0, h_{i-1}), \boldsymbol{y}_{i-1}, \boldsymbol{z}_{i-1}, \Pi_{i-1})$.
2. Set $\Pi = \Pi_n := ((\mathbb{U}, \mathbb{W}), (\mathbb{u}, \mathbb{w}))$.
3. Compute
   $\left(\mathbb{U}', \mathbb{W}', \overline{T'}\right) \leftarrow \mathsf{NIFS.P}\left(\mathsf{pk.pk_{fs}}, (\mathbb{U}, \mathbb{W}), (\mathbb{u}, \mathbb{w})\right)$.
4. Compute $\pi' \leftarrow \mathsf{SNARK.P}\left(\mathsf{pk.pk_{snark}}, \mathbb{U}', \mathbb{W}'\right)$.
5. Output $\pi := \left(h_n, \mathbb{U}, \mathbb{u}, \overline{T'}, \pi'\right)$.

$\mathcal{V}(\mathsf{vk}, n, h_0, \boldsymbol{y}, \pi) \to \{0, 1\}$:

1. Compute
   $h \leftarrow \mathsf{H}\left(\ldots \mathsf{H}\left(\mathsf{H}\left(h_0, \boldsymbol{y}_0, \mathbf{1}^b\right), \boldsymbol{y}_1, \mathbf{1}^b\right) \ldots, \boldsymbol{y}_n, \mathbf{1}^b\right)$.
2. Parse π as $\left(h_n, \mathbb{U}, \mathbb{u}, \overline{T'}, \pi'\right)$.
3. Check that $h_n = h$.
4. Check that $\mathbb{u}.x = \mathsf{H}\left(\mathsf{vk.vk_{fs}}, n, h_0, h_n, \mathbb{U}\right)$.
5. Check that $\left(\mathbb{u}.\overline{E}, \mathbb{u}.s\right) = \left(\mathbb{u}_\perp.\overline{E}, 1\right)$.
6. Compute $\mathbb{U}' \leftarrow \mathsf{NIFS.V}\left(\mathsf{vk.vk_{fs}}, (\mathbb{U}, \mathbb{u}), \overline{T'}\right)$.
7. Check that $\mathsf{SNARK.V}\left(\mathsf{vk.vk_{snark}}, \mathbb{U}', \pi'\right) = 1$.

Fig. 9 General batch verification of signatures

$\boldsymbol{y}_i$ of $\mathsf{F} \circ \mathsf{H}$ with its digest $e_{i,j} = \mathsf{hash}(m_{i,j})$, and the function F is changed to Verify′ which is the part of Verify in Fig. 10 except for the step 2.

Elliptic curve. The standard ECDSA is defined on the elliptic curve `secp256k1`, denoted E_1. Let $\mathbb{F}_p$ denote its base field and $\mathbb{F}_q$ denote its scalar field. When considering verifying the ECDSA inside a proof system, we usually represent the evaluation of the verification process as R1CS constraints over a finite field $\mathbb{F}$. Due to the group-based folding scheme, we instantiate the homomorphic commitment on the elliptic curve group where the discrete logarithm problem is hard. Therefore the finite field $\mathbb{F}$ should be a scalar field of some elliptic curve E_2. If the field $\mathbb{F}$ differs from the domain of the

Gen(pp) $\rightarrow (pk, sk)$.

1. $sk \xleftarrow{\$} [1, q-1]$, $pk \leftarrow sk \cdot G$.

   // G is a genrator of the group with order q.
2. Output (pk, sk).

Sign$(sk, m) \rightarrow \sigma$

1. $k \xleftarrow{\$} [1, q-1]$, $(x, y) \leftarrow k \cdot G$.
2. $r \leftarrow x \bmod q$, if $r = 0$, then go to step 1.
3. $e \leftarrow \mathsf{hash}(m)$.
4. $s \leftarrow k^{-1}(e + sk \cdot r) \bmod q$, if $s = 0$ then go to step 1.
5. Output (r, s).

Verify$(pk, \sigma, m) \rightarrow \{0, 1\}$.

1. If $r, s \notin [1, q-1]$, then output 0.
2. $e \leftarrow \mathsf{hash}(m)$.
3. $w \leftarrow s^{-1} \bmod q$.
4. $u \leftarrow ew \bmod q$, $v \leftarrow rw \bmod q$.
5. $(x', y') \leftarrow u \cdot G + v \cdot pk$.

   If $(x', y') = (0, 0)$, then output 0.

   Otherwise, $r' \leftarrow x \bmod q$.
6. If $r = r'$, output 1, otherwise, output 0.

Fig. 10 ECDSA signature

evaluation, all non-native operations must be simulated over $\mathbb{F}$. In particular, simulating operations over larger fields on a smaller field additionally increases extensive constraints. Using the state-of-the-art frontend techniques in [47,48], we compute the number of constraints required to simulate each operation of E_1 on the scalar field of E_2, which is set to three cases of elliptic curves. The results are shown in Table 1.

Therefore, to minimize the prover time, we avoid non-native operations as much as possible in our scheme BEATS. We compare the number of constraints of simulation for non-native operations over the native curves in Table 1. The number of constraints of non-native group operations is larger than that of non-native field operations, so group operations are more expensive to simulate than common field operations. In the verification of Fig. 10, there are not only field operations over $\mathbb{F}_q$ such as multiplication and inverse modulo q, but also group operations such as point addition and scalar multiplication. Since the group operations are implemented over the base field $\mathbb{F}_p$ of E_1, we let $\mathbb{F}_p$ to be the scalar of E_2 so that we only need to simulate a few of operations of $\mathbb{F}_q$ on $\mathbb{F}_p$. On the other hand, the verification of the folding scheme is dominated by group operations and it will also be represented as R1CS constraints. Like Nova, we use a cycle of elliptic curves, E_1 and E_2, to make it efficient. We set the E_2 as the primary curve and E_1 as the secondary curve to execute the verification of the folding scheme alternately, the ECDSA verification is only represented on the primary curve. Fortunately, the elliptic curve `secq256k1` [18] perfectly forms a cycle with `secp256k1`, so we instantiate the E_2 by `secq256k1` to complete the IVC scheme. Furthermore, the scalar field $\mathbb{F}_p$ is larger than $\mathbb{F}_q$, so there are no additional constraints for simulating the multiplication and addition modulo q. Therefore, `secq256k1` is optimal to instantiate E_2.

5.3.2 Optimization on backend implementation

Corresponding to the frontend, the process of executing a protocol on a given arithmeticized representation is called the backend. At the end of the IVC scheme, we add a general SNARK to compress the proof size as shown in Fig. 9. Using Spartan with Bulletproof commitment scheme as the SNARK is natural because it is not only designed for R1CS satisfiability but can be compatible with the additive homomorphic commitment based on the discrete logarithm problem. Moreover, it is transparent and has a prover with linear complexity about the size of a committed relaxed R1CS instance. Although the proof size of the $\mathsf{Spartan}_{\mathsf{BP}}$ is not a constant, the size of the final proof π in our scheme is independent of the number of signatures. The reason is that we apply the $\mathsf{Spartan}_{\mathsf{BP}}$ only once on a committed relaxed R1CS instance-witness pair $(\mathbb{U}', \mathbb{W}')$ with a constant scale, instead of verifying a batch of ECDSA signatures inside $\mathsf{Spartan}_{\mathsf{BP}}$ directly. The proof size depends only on the fixed block size and the ECDSA verification itself, not on the total number of signatures.

Table 2 compares our scheme with the approaches used $\mathsf{Spartan}_{\mathsf{BP}}$ and Marlin directly for batch verification of ECDSA. The batch size is $t = nb$ where b is the number of signatures in each step of IVC, and n is the number of steps. Let $O(k)$ MSM represent $O(k)$-sized multi-scalar multiplication, $O(k)$ FFT represent $O(k)$-sized FFT costs $O(k \log k)$ field operations. N denotes the number of constraints of one ECDSA verification over the scalar field of `secp256k1`, $N' (> N)$ denotes the number of constraints of one ECDSA verification simulated on `BLS12_381`, and $N'' (\gtrsim N)$ denotes N plus the number of constraints of a hash function H. Since the prover time is dominated by complex computations such as MSM

Table 1 Comparison of the number of constraints of simulation for non-native operations over the native curves. "Mult" and "Add" denote the multiplication and addition operations of the elements on the scalar field of the non-native curve, respectively. "Point_Add" denotes the addition of elements of the group on the non-native curve. "Scalar_Mult" denotes the multiplication between a scalar field element and a group element of the non-native curve

Curve E_2	Scalar field	Curve E_1	Mult	Add	Point_Add	Scalar_Mult
`secp256k1`	$\mathbb{F}_q$ (256-bit)	`secp256k1`	1	0	338	100353
`secq256k1`	$\mathbb{F}_p$ (256-bit)	`secp256k1`	1	0	3	1793
`BLS12_381`	$\mathbb{F}$ (255-bit)	`secp256k1`	71	25	338	100353

Table 2 Asymptotic costs of our scheme and its baselines to generate and verify a proof for $t = nb$ ECDSA signatures. $|x|$ denotes the size of the public IO of the R1CS

Scheme	CRS	Prover	Proof size	Verifier
$\mathsf{Spartan_{BP}}$	Transparent	$O(tN)$ MSM	$O_\lambda(\log t)$	$O_\lambda(\|x\|+t)$
Marlin	Non-transparent	$O(tN')$MSM + $O(tN')$FFT	$O_\lambda(1)$	$O_\lambda(\|x\|+\log t)$
BEATS	Transparent	$O(tN'')$ MSM	$O_\lambda(1)$	$O_\lambda(\|x\|)$

and FFT, we count the times they must be executed in the protocol. We evaluate the proof size and verifier time by elements and operations of the cryptographic group or field. In the batch verification of ECDSA, the number of constraints of each instance is fixed, so N, N', and N'' are constants and the unique variable in this evaluation is the batch size t. The proof size of $\mathsf{Spartan_{BP}}$ increases with t, the proof size of Marlin is always a constant, and the proof of our scheme is also a constant since the block size b is pre-determined.

6 Security

In this section, we analyze the security of our general construction of the non-interactive argument in Section 5.2. The completeness relies on the completeness of the IVC scheme and the SANRK schemes. The soundness is determined by the knowledge-soundness of the IVC and the soundness of the SANRK schemes.

The knowledge soundness of the original Nova IVC relies on the knowledge soundness of the non-interactive folding scheme (NIFS) in the standard model, where the non-interactive property is realized by instantiating the random oracle with a cryptographic hash function. However in the batch verification of signatures, if the parameter n is as large as $O(\lambda)$, this proof strategy will not work. As mentioned in [42], the original Nova follows a general recursive proof strategy, where the IVC extractor $\mathcal{E}$ generates $\mathcal{E}_i$ from $\mathcal{E}_{i+1}$ for $i \in [n]$ and it holds that

$$\mathsf{time}(\mathcal{E}_i) > \mathsf{time}(\widetilde{\mathcal{E}}_i) + \mathsf{time}(\mathcal{A}_i) > 2 \cdot \mathsf{time}(\mathcal{E}_{i+1}).$$

This proof strategy is applicable when considering a batch of no more than t signatures, where $t = nb, n = O(\log \lambda)$ for a fixed b. When the number of signatures is so large that $n = O(\lambda)$, the running time of the extractor $\mathcal{E}$ will be $2^{O(\lambda)}$, it cannot extract all IVC witness in the polynomial time. If the $\lambda \leqslant 2^{128}$, the original knowledge soundness proof can only support $n < 128$ steps and bounds the $128b$ signatures in total, limiting the scalability in the blockchain in practice. Although the throughput may be improved by choosing a bigger b, an oversized circuit in one step will cost a lot of memory and offset the benefit of IVC to compress the circuit size. Therefore, we use the knowledge soundness proof in AGM proposed by [42] on the Group-based Non-Interactive Folding Scheme shown in Fig. 8. Another advantage of this proof strategy is eliminating the heuristic manner in the arithmetization of oracles.

Lemma 2 (Completeness of IVC). The construction in Fig. 7 is an IVC scheme that satisfies completeness.

Proof Sketch For $i \geqslant 0$, given a valid IVC proof $\Pi_i = ((\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i))$. Suppose the IVC prover runs the protocol honestly as Fig. 7 and outputs a new IVC proof $\Pi_{i+1} = ((\mathbb{U}_{i+1}, \mathbb{W}_{i+1}), (\mathbb{u}_{i+1}, \mathbb{w}_{i+1}))$. Because Π_i is valid, then $(\mathbb{U}_i, \mathbb{W}_i) \in \mathcal{R}_\mathsf{s}$ and $(\mathbb{u}_i, \mathbb{w}_i) \in \mathcal{R}_\mathsf{s}$ for a certain public relation $\mathcal{R}_\mathsf{s}$. Because the prover run the folding scheme to obtain $(\mathbb{U}_{i+1}, \mathbb{W}_{i+1})$, it is the folded result of $(\mathbb{U}_i, \mathbb{W}_i)$ and $(\mathbb{u}_i, \mathbb{w}_i)$ so that $(\mathbb{U}_{i+1}, \mathbb{W}_{i+1}) \in \mathcal{R}_\mathsf{s}$ by the completeness of the folding scheme. Finally, the instance-witness pair $(\mathbb{u}_i, \mathbb{w}_i)$ describes the correct execution of the ith augment function, it must hold that $(\mathbb{u}_i, \mathbb{w}_i) \in \mathcal{R}_\mathsf{s}$ as long as the prover run the entire protocol honestly. The formal proof is provided in Appendix A. □

Lemma 3 (Knowledge Soundness of IVC). If the hash function H has the general zero-testing property and the discrete logarithm assumption holds in the group $\mathbb{G}$, then the IVC scheme in Fig. 7 combined with the group-based folding scheme based on $\mathbb{G}$ and H (Fig. 8) satisfies knowledge soundness in the refined AGM.

Proof We follow the proof strategy of [44]. For an IVC scheme with $n = poly(\lambda)$ steps, the functions F, H, $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{F}, \mathsf{H})$, consider a polynomial time algebraic adversary $\mathcal{P}^*$ on inputs $\mathsf{pp} = (\mathsf{pp}_E, \mathsf{pp}_W)$ and outputs h_0, h, Π with set S of representation vectors following multi-encoding. We construct a extractor $\mathcal{E}$ on input (pp, h_0, h), outputs $(h_i)_{i=1}^{n-1}$ and $(\boldsymbol{y}_i, z_i)_{i=0}^{n-1}$ such that $h_n = h$ and $\mathsf{H}(h_{i-1}, \boldsymbol{y}_{i-1}, z_{i-1}) = h_i$ where $z_{i-1} = \mathsf{F}(\boldsymbol{y}_{i-1})$. We construct $\mathcal{E}$ by sub-extractor $\mathcal{E}_i$ for all $i \in [n-1]$ in reverse order. In each step, given $(h_i, \ldots, h_n)$, $((\boldsymbol{y}_i, z_i), \ldots, (\boldsymbol{y}_{n-1}, z_{n-1}))$, and an accepting IVC proof Π_i, $\mathcal{E}_i$ outputs h_{i-1}, $(\boldsymbol{y}_{i-1}, z_{i-1})$ and an IVC proof Π_{i-1}. After all $n-1$ sub-extractors $\mathcal{E}_i$ are constructed, the IVC extractor $\mathcal{E}$ runs them to extract the final outputs $((\boldsymbol{y}_0, z_0), \ldots, (\boldsymbol{y}_{n-1}, z_{n-1}))$. The constructions of $\mathcal{E}_i$ and $\mathcal{E}$ are described in Fig. 11.

The correctness of the extracted witness from $\mathcal{E}$ relies on the correctness of the outputs of each $\mathcal{E}_i$. By Lemma 4, each Π_{i-1} are valid and $h_{i-1}, \boldsymbol{y}_{i-1}, z_{i-1}$ extracted in the ith step satisfies that $h_i = \mathsf{H}(h_{i-1}, \boldsymbol{y}_{i-1}, z_{i-1})$, where $z_{i-1} = \mathsf{F}(\boldsymbol{y}_{i-1})$.

Lemma 4 Consider the input and output

$$(h_i, \ldots, h_n), ((\boldsymbol{y}_i, z_i), \ldots, (\boldsymbol{y}_{n-1}, z_{n-1})),$$

such that $h_{j+1} = \mathsf{H}(h_j, \boldsymbol{y}_j, z_j)$ and $z_j = \mathsf{F}(\boldsymbol{y}_j)$ for all $j = i, \ldots, n-1$. Let S be a set of representation vectors output by $\mathcal{P}^*$, and assume that $\Pi_i = ((\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i))$ is a valid IVC proof combined with the group-based folding scheme in Fig. 8. If the hash

$\mathcal{E}_i(\mathsf{pp}, (h_i, \ldots, h_n), ((\boldsymbol{y}_i, \boldsymbol{z}_i), \ldots, (\boldsymbol{y}_{n-1}, \boldsymbol{z}_{n-1})), S)$

$\rightarrow h_{i-1}, (\boldsymbol{y}_{i-1}, \boldsymbol{z}_{i-1})$:

1. Parse $\Pi_i = ((\mathbb{U}_i, \mathbb{W}_i), (\mathfrak{u}_i, \mathfrak{w}_i))$.
2. Retrieve $h_{i-1}, (\boldsymbol{y}_{i-1}, \boldsymbol{z}_{i-1}), (\mathbb{U}_{i-1}, \mathfrak{u}_{i-1})$.
3. Retrieve $\overline{E}_{i-1}^{(1)}, \overline{W}_{i-1}^{(1)}, \overline{E}_{i-1}^{(2)}, \overline{W}_{i-1}^{(2)}$ from $(\mathbb{U}_{i-1}, \mathfrak{u}_{i-1})$.
4. Retrieve $E_{i-1}^{(1)}, W_{i-1}^{(1)}, E_{i-1}^{(2)}, W_{i-1}^{(2)}$ from representation set S.
5. Set $\mathbb{W}_{i-1} = (E_{i-1}^{(1)}, W_{i-1}^{(1)})$, $\mathfrak{w}_{i-1} = (E_{i-1}^{(2)}, W_{i-1}^{(2)})$.
6. Output

$h_{i-1}, (\boldsymbol{y}_{i-1}, \boldsymbol{z}_{i-1}), \Pi_{i-1} = ((\mathbb{U}_{i-1}, \mathbb{W}_{i-1}), (\mathfrak{u}_{i-1}, \mathfrak{w}_{i-1}))$.

$\mathcal{E}(\mathsf{pp}, \rho) \rightarrow ((\boldsymbol{y}_0, \boldsymbol{z}_0), \ldots, (\boldsymbol{y}_{n-1}, \boldsymbol{z}_{n-1}))$:

1. Run the malicious prover $\mathcal{P}^*$: $(h_0, h_n, \Pi_n, S) \leftarrow P^*$.
2. Parse $\Pi_n = ((\mathbb{U}_n, \mathbb{W}_n), (\mathfrak{u}_n, \mathfrak{w}_n))$.
3. Retrieve $\overline{E}_{n-1}, \overline{W}_{n-1}$ from $(\mathfrak{u}_{i-1})$.
4. For $i = n-1, \ldots, 1$:

$h_{i-1}, (\boldsymbol{y}_{i-1}, \boldsymbol{z}_{i-1}),$

$\Pi_{i-1} \leftarrow \mathcal{E}_i(\mathsf{pp}, (h_i, \ldots, h_n), ((\boldsymbol{y}_i, \boldsymbol{z}_i), \ldots, (\boldsymbol{y}_{n-1}, \boldsymbol{z}_{n-1})), S)$.

5. Output $((\boldsymbol{y}_0, \boldsymbol{z}_0), \ldots, (\boldsymbol{y}_{n-1}, \boldsymbol{z}_{n-1}))$.

Fig. 11 Construction IVC extractor

function H has the general zero-testing property and the discrete logarithm assumption holds in the group $\mathbb{G}$, then the PPT sub-extractor $\mathcal{E}_i$ outputs a valid IVC proof $\Pi_{i-1} = ((\mathbb{U}_{i-1}, \mathbb{W}_{i-1}), (\mathfrak{u}_{i-1}, \mathfrak{w}_{i-1}))$ and $(h_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{z}_{i-1})$ satisfy $\boldsymbol{z}_{i-1} = \mathsf{F}(\boldsymbol{y}_{i-1})$ and $h_i = \mathsf{H}(h_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{z}_{i-1})$.

The proof of Lemma 4 is the same as the Lemma 3 of [44] and is provided in Appendix B. □

Theorem 1 (Completeness). The general construction for batch verification of valid signatures in Fig. 9 is a non-interactive argument that satisfies completeness.

Proof Given a batch of signatures, suppose the prover verifies them and takes $t = nb$ valid signatures $\boldsymbol{y}$ with their verification result $\boldsymbol{z}$ as inputs, and runs the protocol as in Fig. 9, outputs a proof π. Because the t signatures are valid, for $i \in [n]$, $\mathsf{F}(\boldsymbol{y}_i) = \boldsymbol{z}_i = \mathbf{1}^b$, where b is the block size. Because the prover computes the hash chain and runs the IVC prover in Fig. 7 honestly, $h_i = \mathsf{H}\left(h_{i-1}, \boldsymbol{y}_i, \mathbf{1}^b\right)$ and Π_i is a valid IVC proof for all $i \in [n]$. The last IVC proof $\Pi_n = ((\mathbb{U}, \mathbb{W}), (\mathfrak{u}, \mathfrak{w}))$ is valid implies that $(\mathbb{U}, \mathbb{W}), (\mathfrak{u}, \mathfrak{w}) \in \mathcal{R}_{\mathsf{s}}$, and checks 4 and 5 of the verifier will pass by the completeness of the IVC scheme. Then the prover obtains $\left(\mathbb{U}', \mathbb{W}', \overline{T'}\right)$ by the folding scheme, so the $(\mathbb{U}', \mathbb{W}') \in \mathcal{R}_{\mathsf{s}}$ under the completeness of the folding scheme. Furthermore, the completeness of the SNARK scheme guarantees that π' is valid. All conditions of the verifier can be met. □

Theorem 2 (Soundness). If the hash function H has the collision-resistant property, the IVC scheme in Fig. 7 satisfies knowledge soundness for $n = O(\lambda)$ steps, and SNARK is a SNARK of a valid IVC proof satisfies knowledge soundness, the general construction in Fig. 9 is a non-interactive argument for batch verification of $t = O(\lambda)$ valid signatures satisfies soundness.

Proof Let $\mathcal{P}^*$ be an adversary against the soundness of the general construction in Fig. 9 for batch verification of $t = O(\lambda)$ signatures. $\mathcal{P}^*$ takes public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{fs}}, \mathsf{pp}_{\mathsf{snark}})$ and outputs $h_0, \boldsymbol{y} = (\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{n-1})$ with a proof $\pi = (h_n, \mathbb{U}, \mathfrak{u}, \overline{T'}, \pi')$ for the statement that all $t = nb$ signatures described in $\boldsymbol{y}$ are valid, i,e, $\forall i \in [n], \mathsf{F}(\boldsymbol{y}_{i-1}) = \mathbf{1}^b$. Suppose there exist at least one invalid signature, without loss of generality, the jth $(0 \leqslant j < t)$ signature, the proof π from $\mathcal{P}^*$ pass the verification with probability $\epsilon(\lambda)$, we demonstrate that $\epsilon(\lambda)$ is negligible.

If $\mathcal{P}^*$ follows the protocol honestly, precisely, $\mathcal{P}^*$ runs the prover algorithm in the Fig. 9 to prove that $\mathsf{F}(\boldsymbol{y}_{i-1}) = \boldsymbol{z}_{i-1}$ for all $i \in [n]$, where $\mathsf{F}(\boldsymbol{y}_{j'}) = \boldsymbol{z}_{j'} \neq \mathbf{1}^b$, $j' = \lfloor \frac{j}{n} \rfloor$ and $\mathsf{F}(\boldsymbol{y}_{i-1}) = \boldsymbol{z}_{i-1} = \mathbf{1}^b$ for all $i \neq j'$. Because the proof is accepted, it must hold that $h_n = \mathsf{H}\left(\ldots \mathsf{H}\left(\mathsf{H}\left(h_0, \boldsymbol{y}_0, \mathbf{1}^b\right), \boldsymbol{y}_1, \mathbf{1}^b\right) \ldots, \boldsymbol{y}_n, \mathbf{1}^b\right)$ by the condition 1-3 of the verifier. As the hash function is collision-resistant, it will contract to the Lemma 5. Thus all $\boldsymbol{z}_{i-1}$ for $i \in [n]$ are $\mathbf{1}^b$ with overwhelming probability. Similarly, if $\mathcal{P}^*$ honestly generate an accepted proof for $\mathsf{F}(\boldsymbol{y}'_{i-1}) = \mathbf{1}^b$ for all $i \in [n]$ but $\boldsymbol{y}' \neq \boldsymbol{y}$ that sent to the verifier before the protocol, it also cannot pass the verification without a negligible probability.

Lemma 5 [49] If H is collision-resistant, then the Merkle-Damgard construction of H is collision-resistant.

If $\mathcal{P}^*$ pass the verification by cheating, all conditions of the verifier will be met. By step 7 of the verifier, π' passes the verification of the SNARK scheme for the instance $\mathbb{U}'$ computed by running the verifier of the folding scheme. We first construct a SNARK extractor $\widetilde{\mathcal{E}}$, which takes $\mathbb{U}'$ and the accepted SNARK proof π', outputs a witness $\mathbb{W}'$ such that $(\mathbb{U}', \mathbb{W}') \in \mathcal{R}_{\mathsf{s}}$ with an overwhelming probability. This implies that instances $\mathbb{U}$ and $\mathfrak{u}$ are satisfiable, we can run the folding extractor $\widehat{\mathcal{E}}$ to extract $\mathbb{W}$ and $\mathfrak{w}$ such that $(\mathbb{U}, \mathbb{W}), (\mathfrak{u}, \mathfrak{w}) \in \mathcal{R}_{\mathsf{s}}$. Then we run the IVC extractor $\mathcal{E}$ that takes (h_0, h) and the accepted IVC proof $\Pi = ((\mathbb{U}, \mathbb{W}), (\mathfrak{u}, \mathfrak{w}))$, outputs $\boldsymbol{y} = (\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{n-1})$ satisfies that $\mathsf{H}(h_{i-1}, \boldsymbol{y}_{i-1}, \mathsf{F}(\boldsymbol{y}_{i-1})) = h_i$ for all $i \in [n]$. Following the construction of the IVC extractor in Fig. 11, we sequentially construct the sub-extractors $\mathcal{E}_i$ for all $i \in [n]$. If there exists any $j \in [n]$ such that $\mathsf{F}(\boldsymbol{y}_{j-1}) = \boldsymbol{z}_{j-1} \neq \mathbf{1}^b$, then it contradicts the collision-resistant property of the hash function H as above. Suppose the SNARK scheme has the knowledge error $\epsilon_{\mathsf{snark}}(\lambda)$, the IVC scheme has the knowledge error $\epsilon_{\mathsf{ivc}}(\lambda)$ for $n = O(\lambda)$ rounds iterations, the non-interactive folding scheme has the knowledge error $\epsilon_{\mathsf{nifs}}(\lambda)$ and the probability of the event that finding a collision for the Merkle-Damgard formed hash chain with length n is $\epsilon_{\mathsf{hash}}(\lambda) \leqslant \mathsf{negl}(\lambda)$. Then, our scheme in Fig. 9 has the soundness error

$$\epsilon(\lambda) = \epsilon_{\mathsf{snark}}(\lambda) + \epsilon_{\mathsf{nifs}}(\lambda) + \epsilon_{\mathsf{ivc}}(\lambda) + \epsilon_{\mathsf{hash}}(\lambda) \leqslant \mathsf{negl}(\lambda).$$

Finally, because the extractor $\widetilde{\mathcal{E}}$, $\widehat{\mathcal{E}}$, and $\mathcal{E}$ can extract the witness in polynomial time, and the hash function can achieve at least 100-bit security, the probability for any PPT $\mathcal{P}^*$ generating an accepted proof for a false statement is negligible. □

7 Implementation

Following the general construction of Section 5.2 and employing the

optimization strategies in Section 5.3, we implement our scheme for batch verification of ECDSA in Rust. The experiments were executed on a Virtual Machine with an Intel i7-11800H CPU with 2.30 GHz, 8 cores, and 16 GB of RAM.

The arithmetic circuit R1CS is compiled using the library bellpepper [50]. We implement the IVC with SNARK scheme following the code of Nova [43], where we choose `secp256k1` and `secq256k1` as the elliptic curve cycle. We set `secq256k1` as the primary curve and `secp256k1` as the secondary curve. The verification of ECDSA is only defined on the scalar field $\mathbb{F}_p$ of `secq256k1`. The number of R1CS constraints is about 4,000 per ECDSA verification. In experimental testing, we can adjust the block size b. In particular, the function F represents b ECDSA verification processes on the digest of the message. We have the verifier compute the digest $e = \mathsf{hash}(m)$ locally and add this computation time to the verifier time. We fix the message m as a 160-byte value as account information in transactions. We instantiate the hash function H in our scheme by Poseidon, which contains an initialization phase independent of the message and a computation phase that processes messages through the sponge structure. Since the initialization of Poseidon is fixed and reusable, it can be seen as the offline computation together with the setup of the proof system for the prover and verifier. We only focus on the online phase related to signatures.

7.1 BEATS

We instantiate the SNARK in our scheme via $\mathsf{Spartan}_{\mathsf{BP}}$ to obtain a transparent scheme BEATS. As shown in Fig. 12, we test the prover time and verifier time with batch sizes t ranging from 1 to 2^{20} ECDSA signatures. We set the block size $b = 1, 2, 8, 32$ in each IVC step, respectively.

We first compare our protocol with the scheme that verified ECDSA inside $\mathsf{Spartan}_{\mathsf{BP}}$ directly. Since BEATS in this paper uses the cycle of curves `secp256k1/secq256k1`, it is natural to compare it with Spartan using `secp256k1` in the experiments. For the prover time, when the batch size is larger than 2^3, BEATS begins to show its advantages. If the batch size is less than 2^9, the prover time will be lower to set $b = 8$, while setting the block size b larger, such as $b = 32$, would make the prover faster if one intends to pack many more signatures. In BEATS, the parameter b determines the number of constraints proven by the final SNARK after IVC. Increasing b speeds up the prover for large batch sizes, while decreasing b speeds up the prover for small batch sizes. Additionally, smaller b results in faster verification and smaller proofs. When $t = 2^{11}$, $\mathsf{Spartan}_{\mathsf{BP}}$ reaches its upper bound on the number of signatures, the prover time of $\mathsf{Spartan}_{\mathsf{BP}}$ is about 776 s, while the prover time for $b = 1, 2, 8$ and 32 in BEATS is 248, 160, 62, and 45 s respectively. We speed up the prover by 3~17 times.

For the verifier time, it grows linearly with the number of signatures in $\mathsf{Spartan}_{\mathsf{BP}}$, while in BEATS it is almost constant when $t \leqslant 2^{14}$. When $t = 2^{11}$, $\mathsf{Spartan}_{\mathsf{BP}}$ reaches its upper bound with a verifier time of 21 s, while the mean of verifier time for

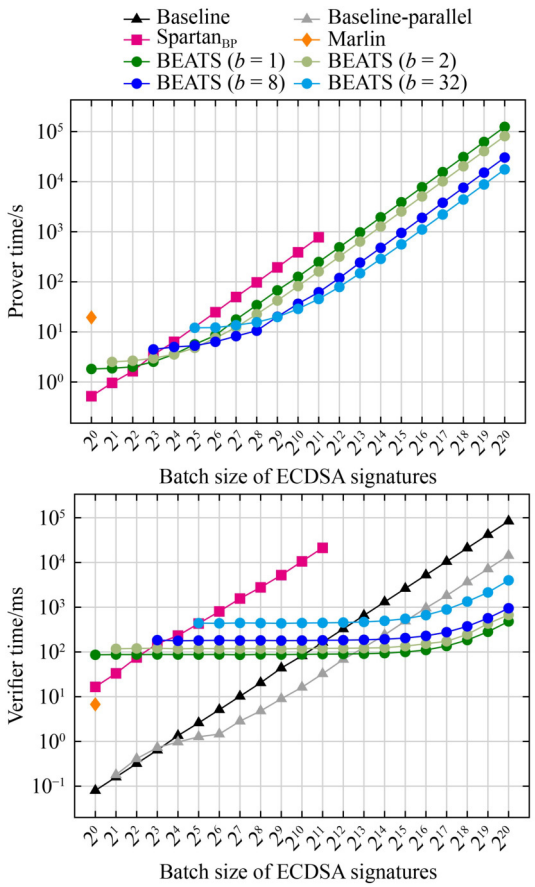

Fig. 12 Comparison of the prover and verifier time between BEATS, $\mathsf{Spartan}_{\mathsf{BP}}$ and Marlin at different batch sizes of ECDSA signatures

different values of $b = 1, 2, 8$ and 32 in BEATS is 87, 118, 180, and 440 ms respectively. We speed up the verifier by 48–240 times. The verifier time of BEATS starts to show an increasing trend only when $t \geqslant 2^{15}$; the reason is the verifier time is dominated by the SNARK verification process before that and the hash computation of the message is negligible. As the size increases, the hash computation is proportional to the number of messages, so the hash computation starts to dominate the verifier time when $t \geqslant 2^{15}$. The baseline with black triangle legend in Fig. 12 indicates the verification time of ECDSA signatures without any proof system. Obliviously, BEATS will be more efficient when the batch size is larger than 2^{10} compared to the baseline. When $t = 2^{20}$, we speed up the verifier by 21–174 times. Furthermore, even if the baseline is optimized for parallelism as the gray triangle legend, BEATS still using a single thread becomes advantageous when the batch size exceeds 2^{13}. The gap when $t \geqslant 2^{13}$ grows with the batch size. When $t = 2^{20}$, we speed up the verifier by 3~30 times.

Then we compare our scheme with Marlin. In Section 1.1, we mentioned that Marlin is based on bilinear pairing, but `secp256k1/secq256k1` is not pairing-friendly as noted in Section 2. Thus it is difficult to implement the bilinear pairing mapping of Marlin on `secp256k1` efficiently. To ensure fair comparisons, we use the optimal finite field for Marlin, given its reliance on non-

native operations. Following the result of [48] that there are about 2^{20} constraints [3] for each ECDSA verification, we substitute the result into the benchmark test to obtain the experiment performance. Since the prover time of Marlin is quasi-linear and a large-scale MSM and FFT dominate the time and memory of the prover, the execution quickly reaches the upper bound. Marlin can prove only one ECDSA signature in the virtual machine of this experiment. When $b = 1$, the prover takes 1.82 s in BEATS and takes 19.46 s in Marlin for one ECDSA signature. The single prover in BEATS is faster than Marlin by about 10 times. Marlin has fast verifier with 6.74 ms and an extremely short proof with 0.73 KB. In BEATS, the verifier takes 86.36 ms and the proof size is 9.22 KB. Consequently, the verifier time and the proof size of BEATS are larger than Marlin by about 12 times. However, these advantages of Marlin come at the cost of longer prover time and higher memory consumption, and it is non-transparent with a trusted setup. We improve the scalability significantly that supports an arbitrary number of signatures and the proof size is nearly constant when the batch size is less than 2^{14}. Since Marlin needs to generate proofs for a batch of signatures separately, the verifier time and proof size of BEATS can be amortized efficiently.

Tables 3 and 4 describe the comparison of proof size between BEATS and $\text{Spartan}_{\text{BP}}$ and Marlin. The batch size t in Table 3 represents the number of the ECDSA signatures in total and the block size b in Table 4 represents the number of the ECDSA signatures handled in each step of IVC in BEATS, the proof size of BEATS is a constant independent of t for fixed b. Our proof size is larger than the $\text{Spartan}_{\text{BP}}$, because the implementation is over the cycle of elliptic curves, there are two instance-witness pairs at the last of the IVC scheme, i.e., $(\mathbb{U}', \mathbb{W}')$ in Fig. 9 is replaced by $(\mathbb{U}^{(1)}, \mathbb{W}^{(1)})$ and $(\mathbb{U}^{(2)}, \mathbb{W}^{(2)})$. Finally, there are two SNARK proofs. When a short proof is desired, one can use only one curve to halve the proof size.

For scalability, the upper bound on the number of ECDSA signatures that can be proved at once is 2^{11} in $\text{Spartan}_{\text{BP}}$ and only one in Marlin. BEATS can support arbitrary batch size theoretically, and we show only a portion of the practical intervals in Fig. 12 within a practically acceptable time. Thanks to the chain-based IVC construction, our scheme is memory-friendly. The ith step in the chain claims that the previous i steps were executed correctly, which is a compression of the historical state. Each generated IVC proof can be viewed as an update to the current state and the past state can be released. The prover generates the IVC proof and takes a constant memory and a linear time about the number of steps of the IVC chain. On the other hand, we can replace chain-based construction with tree-based construction, which provides a faster prover but costs more memory. This construction is similar to the PCD-based SNARK in [51], but the batch verification for multiple signatures is a natural uniform format, so we do not require a compiler nor do we need to consider the wiring connections between each sub-computation. In this parallel implementation, the time for executing $O(tN'')$-sized MSM of the prover in Table 1 will be reduced to the time for $O(\log n \cdot bN'')$-sized MSM, where $t = nb$ and $b << n$. In addition, the width of the tree can be adjusted according to the optimal number of threads for the device.

8 Conclusion

We present a general approach to batch verification of arbitrary signatures on the blockchain, addressing scalability, memory efficiency, and compatibility with common devices while supporting an arbitrary number of verifications. As a concrete instantiation, we introduce BEATS for batch ECDSA verification, demonstrating significant performance improvements over existing methods. The comparison shows that BEATS speeds up the prover and the verifier of $\text{Spartan}_{\text{BP}}$ significantly for the large batch of ECDSA signatures. Furthermore, BEATS has high scalability that supports an arbitrary number of signatures, which is far superior to the existing SNARK such as Marlin. Finally, BEATS exhibits a remarkably low memory footprint, making it compatible with resource-limited devices.

A research direction in the future is how to realize proof acceleration on small finite fields. However, several challenges must be addressed. Firstly, while small finite fields may offer faster field operations, they often lead to increased soundness errors. This necessitates additional security mechanisms, potentially increasing scheme complexity and introducing overhead for the prover. Secondly, simulating non-native operations on smaller fields can result in larger circuit sizes. Since prover time in many proof systems scales linearly with the number of constraints, this circuit size growth can significantly impact prover performance. Finally, in the context

Table 3 Proof size of $\text{Spartan}_{\text{BP}}$ and Marlin in different batch sizes

Batch size t	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}
$\text{Spartan}_{\text{BP}}$/KB	4.08	4.36	4.65	4.93	5.22	5.50	5.79	6.09	6.37	6.66	6.95	7.24
Malrin/KB	0.73	–	–	–	–	–	–	–	–	–	–	–

Table 4 Proof size of BEATS in different block sizes

Block size b	1	2	8	32
BEATS/KB	9.22	9.51	9.79	10.37

[3] Although the field size in [48] is 254-bit while the scalar field of BLS12_381 is 255-bit, representing a 256-bit field element on either of them requires 3 of registers. All techniques for simulating the operations of a larger field on a smaller field are general, there are 1,508,136 constraints in total.

of Nova IVC, the use of a cycle of elliptic curves is crucial for efficient proofs. Finding suitable curve pairs that both enhance proof efficiency and adhere to the cycle form presents a significant challenge.

Acknowledgements

We would like to thank Kunxian Xia for his valuable discussions. This work was supported by the National Key Research and Development Program of China (Grant No. 2022YFB2701700).

Competing interests

The authors declare that they have no competing interests or financial conflicts to disclose.

Appendixes

Appendix A Proof of Lemma 2

Proof Suppose $\mathsf{F}(\boldsymbol{y}_i)=\boldsymbol{z}_i$ and $\mathsf{H}(h_i,\boldsymbol{y}_i,\boldsymbol{z}_i)=h_{i+1}$ for some $i\geqslant 0$. Let $\mathsf{pp}\leftarrow\mathcal{G}(1^\lambda)$ and $(\mathsf{pk},\mathsf{vk})\leftarrow\mathcal{K}(\mathsf{pp},\mathsf{F},\mathsf{H})$. Given an IVC proof Π_i satisfies that

$$\mathcal{V}(\mathsf{vk},i,h_0,h_i,\Pi_i)=1.$$

We should demonstrate that given

$$\Pi_{i+1}\leftarrow\mathcal{P}(\mathsf{pk},i,h_0,h_i,\boldsymbol{y}_i,\boldsymbol{z}_i,\Pi_i),$$

it must hold that

$$\mathcal{V}(\mathsf{vk},i+1,h_0,h_{i+1},\Pi_{i+1})=1.$$

When $i=0$**.** Given an IVC proof Π_0 satisfies that

$$\mathcal{V}(\mathsf{vk},0,h_0,h_0,\Pi_0)=1.$$

By the algorithm $\mathcal{P}$ and augment function F', we have

$$\Pi_1=((\mathbb{U}_1,\mathbb{W}_1),(\mathbb{u}_1,\mathbb{w}_1)),$$

where $\mathbb{U}_1=\mathbb{u}_\perp$ and $\mathbb{W}_1=\mathbb{W}_\perp$. By definition, the trivial instance-witness pair $(\mathbb{u}_\perp,\mathbb{w}_\perp)\in\mathcal{R}_{\mathsf{s}_{F'}}$. Since $(\mathbb{u}_1,\mathbb{w}_1)$ describe the trace of F' in terms of $(\mathbb{U}_1,\mathbb{W}_1)$, by correctness of F, H and the satisfiability of $(\mathbb{U}_1,\mathbb{W}_1)$, it also holds that $(\mathbb{u}_1,\mathbb{w}_1)\in\mathcal{R}_{\mathsf{s}_{F'}}$. By the construction of F', we have

$$\mathbb{u}_1.x=\mathsf{H}(\mathsf{vk},1,h_0,\mathsf{H}(h_0,\boldsymbol{y}_0,\boldsymbol{z}_0),\mathbb{U}_1),$$

where $\boldsymbol{z}_0=\mathsf{F}(\boldsymbol{y}_0)$. By the construction of $\mathcal{P}$, we have $\mathbb{w}_1.E=\mathbf{0}$ that implies that $\mathbb{u}_1.\overline{E}=0=\mathbb{u}_\perp.\overline{E}$, and $\mathbb{u}.x=1$. Therefore, it must hold that

$$\mathcal{V}(\mathsf{vk},1,h_0,h_1,\Pi_1)=1.$$

When $i\geqslant 1$**.** Suppose given an IVC proof $\Pi_i=((\mathbb{U}_i,\mathbb{W}_i),(\mathbb{u}_i,\mathbb{w}_i))$ satisfies that

$$\mathcal{V}(\mathsf{vk},i,h_0,h_i,\Pi_i)=1.$$

Given

$$\begin{aligned}\Pi_{i+1}=&((\mathbb{U}_{i+1},\mathbb{W}_{i+1}),(\mathbb{u}_{i+1},\mathbb{w}_{i+1}))\\&\leftarrow\mathcal{P}(\mathsf{pk},i,h_0,h_i,\boldsymbol{y}_i,\boldsymbol{z}_i,\Pi_i).\end{aligned}$$

By construction, $\mathcal{P}$ runs the prover of the non-interactive folding scheme such that

$$(\mathbb{U}_{i+1},\mathbb{W}_{i+1},\overline{T}_i)\leftarrow\mathsf{NIFS.P}(\mathsf{pk},(\mathbb{U}_i,\mathbb{W}_i),(\mathbb{u}_i,\mathbb{w}_i)).$$

By completeness of non-interactive folding scheme and $(\mathbb{U}_i,\mathbb{W}_i),(\mathbb{u}_i,\mathbb{w}_i)\in\mathcal{R}_{\mathsf{s}_{F'}}$, we have that $(\mathbb{U}_{i+1},\mathbb{W}_{i+1})\in\mathcal{R}_{\mathsf{s}_{F'}}$. Since $\mathbb{u}_i.x=\mathsf{H}(\mathsf{vk},i,h_0,\mathsf{H}(h_i,\boldsymbol{y}_i,\boldsymbol{z}_i),\mathbb{U}_i)$, $\mathbb{u}_i.\overline{E}=0$, and $\mathbb{u}_i.s=1$, $\mathcal{P}$ will output $(\mathbb{u}_{i+1},\mathbb{w}_{i+1})$ that describe the trace of F' on inputs $(\mathbb{U}_i,\mathbb{u}_i,i,h_0,h_i,\boldsymbol{y}_i,\boldsymbol{z}_i,\overline{T}_i)$. By the construction of F' and the completeness of the non-interactive folding scheme. we have

$$\begin{aligned}\mathbb{u}_{i+1}.x&=\mathsf{H}(\mathsf{vk},i+1,h_0,\mathsf{H}(h_i,\boldsymbol{y}_i,\boldsymbol{z}_i),\mathsf{NIFS.V}(\mathsf{vk},\mathbb{U}_i,\mathbb{u}_i,\overline{T}_i))\\&=\mathsf{H}(\mathsf{vk},i+1,h_0,\mathsf{H}(h_i,\boldsymbol{y}_i,\mathsf{F}(\boldsymbol{y}_i)),\mathbb{U}_i).\end{aligned}$$

By the construction of $\mathcal{P}$, we have $\mathbb{w}_{i+1}.E=\mathbf{0}$ that implies that $\mathbb{u}_{i+1}.\overline{E}=0=\mathbb{u}_\perp.\overline{E}$, and $\mathbb{u}.s=1$. Therefore,

$$\mathcal{V}(\mathsf{vk},i+1,h_0,h_{i+1},\Pi_{i+1})=1.$$

□

Appendix B Proof of Lemma 4

Proof By the construction of the group-based folding scheme in Fig. 10, instance-witness pairs in the valid Π_i have the following format,

$$\mathbb{U}_i=(\overline{E}_i^{(1)},s_i^{(1)},\overline{W}_i^{(1)},x_i^{(1)}),\mathbb{W}_i=(E_i^{(1)},W_i^{(1)}),\tag{A1}$$

$$\mathbb{u}_i=(\overline{E}_i^{(2)},s_i^{(2)},\overline{W}_i^{(2)},x_i^{(2)}),\mathbb{w}_i=(E_i^{(2)},W_i^{(2)}).\tag{A2}$$

Since $(\mathbb{U}_i,\mathbb{W}_i),(\mathbb{u}_i,\mathbb{w}_i)\in\mathcal{R}_{\mathsf{s}_{F'}}$, they satisfy the Pedersen commitment relation, i,e.,

$$\overline{E}_i^{(1)}=\mathsf{com}(\mathsf{pp}_E,E_i^{(1)})=\langle E_i^{(1)},\mathsf{pp}_E\rangle,\tag{A3}$$

$$\overline{W}_i^{(1)}=\mathsf{com}(\mathsf{pp}_W,W_i^{(1)})=\langle W_i^{(1)},\mathsf{pp}_W\rangle,\tag{A4}$$

$$\overline{E}_i^{(2)}=\mathsf{com}(\mathsf{pp}_E,E_i^{(2)})=\langle E_i^{(2)},\mathsf{pp}_E\rangle,\tag{A5}$$

$$\overline{W}_i^{(2)}=\mathsf{com}(\mathsf{pp}_W,W_i^{(2)})=\langle W_i^{(2)},\mathsf{pp}_W\rangle.\tag{A6}$$

Given Π_i, the extractor $\mathcal{E}_i$ should extract

$$\Pi_{i-1}=((\mathbb{U}_{i-1},\mathbb{W}_{i-1}),(\mathbb{u}_{i-1},\mathbb{w}_{i-1})),$$

and extract $(h_{i-1},\boldsymbol{y}_{i-1})$, such that

$$h_i = \mathsf{H}(h_{i-1}, \boldsymbol{y}_{i-1}, \mathsf{F}(\boldsymbol{y}_{i-1})).$$

Since Π_i is a valid IVC proof, $\mathbb{w}_i$ is the witness for the trace of F' in the ith step and is formed as

$$(\mathsf{pp}, \mathbb{U}_{i-1}, \mathbb{u}_{i-1}, (i, h_0, h_{i-1}), \boldsymbol{y}_{i-1}, \boldsymbol{z}_{i-1}, T_{i-1}).$$

Thus $\mathcal{E}_i$ can extract $(\mathbb{U}_{i-1}, \mathbb{u}_{i-1})$ from $\mathbb{w}_i$ efficiently. Since the witness $\mathbb{w}_{i-1}$ contains the input and output of the execution $\mathsf{H}(h_{i-1}, \boldsymbol{y}_{i-1}, \mathsf{F}(\boldsymbol{y}_{i-1}))$, $\mathcal{E}_i$ can efficiently extract $(h_{i-1}, \boldsymbol{y}_{i-1})$ from Π_{i-1}. By the satisfiability of $\mathbb{u}_i$ and the construction of F', it must hold that $h_i = \mathsf{H}(h_{i-1}, \boldsymbol{y}_{i-1}, \mathsf{F}(\boldsymbol{y}_{i-1}))$.

Then $\mathcal{E}_i$ can retrieve $(\overline{E}^{(1)}_{i-1}, \overline{W}^{(1)}_{i-1})$ from $\mathbb{U}_{i-1}$ and retrieve $(\overline{E}^{(2)}_{i-1}, \overline{W}^{(2)}_{i-1})$ from $\mathbb{u}_{i-1}$. Since $(\overline{E}^{(1)}_{i-1}, \overline{W}^{(1)}_{i-1})$ and $(\overline{E}^{(2)}_{i-1}, \overline{W}^{(2)}_{i-1})$ are group elements, in the refined AGM, the algebraic adversary must output the represent vectors in the set S. Therefore, $\mathcal{E}_i$ can efficiently find the corresponding representation vectors $(E^{(1)}_{i-1}, W^{(1)}_{i-1})$ and $(E^{(2)}_{i-1}, W^{(2)}_{i-1})$ under the base $\mathsf{pp}_E \,\|\, \mathsf{pp}_W$. Then $\mathcal{E}_i$ sets $\mathbb{W}_{i-1} = (E^{(1)}_{i-1}, W^{(1)}_{i-1})$ and $\mathbb{w}_{i-1} = (E^{(2)}_{i-1}, W^{(2)}_{i-1})$.

Next, we demonstrate the correctness of the Π_{i-1} extracted by $\mathcal{E}_i$. A valid IVC proof should pass all checks of the verifier in Fig. 8. By definition of the committed relaxed R1CS, a satisfied instance-witness should first pass the opening check of the commitment. By the premise, $(\mathbb{u}_i, \mathbb{w}_i) \in \mathcal{R}_{\mathsf{s}_{F'}}$. The description of the instance $\mathbb{u}_i$ contains the folding process $\mathbb{U}_i \leftarrow \mathsf{NIFS.V}(\mathbb{U}_{i-1}, \mathbb{u}_{i-1}, \overline{T}_{i-1})$. Thus it must hold that

$$\begin{aligned} \overline{E}^{(1)}_i &= \overline{E}^{(1)}_{i-1} + r\overline{T}_{i-1} + r^2\overline{E}^{(2)}_{i-1}, \quad s^{(1)}_i = s^{(1)}_{i-1} + r s^{(2)}_{i-1}, \\ \overline{W}^{(1)}_i &= \overline{W}^{(1)}_{i-1} + r\overline{W}^{(2)}_{i-1}, \quad x^{(1)}_i = x^{(1)}_{i-1} + r x^{(2)}_{i-1}, \end{aligned} \tag{A7}$$

where $r = \mathsf{H}(\mathbb{U}_i, \mathbb{u}_i, \overline{T}_{i-1})$. Since $\overline{T}_{i-1}$ is a group element contained in the witness $\mathbb{w}_i$, $\mathcal{E}_i$ can find its representation vector T_{i-1} from R. Due to the AGM, these extracted witnesses satisfy the following equations,

$$\begin{aligned} &\overline{E}^{(1)}_{i-1} = \langle E^{(1)}_{i-1}, \mathsf{pp}_E \,\|\, \mathsf{pp}_W\rangle, \ \overline{W}^{(1)}_{i-1} = \langle W^{(1)}_{i-1}, \mathsf{pp}_E \,\|\, \mathsf{pp}_W\rangle, \\ &\overline{E}^{(2)}_{i-1} = \langle E^{(2)}_{i-1}, \mathsf{pp}_E \,\|\, \mathsf{pp}_W\rangle, \ \overline{W}^{(2)}_{i-1} = \langle W^{(2)}_{i-1}, \mathsf{pp}_E \,\|\, \mathsf{pp}_W\rangle, \\ &\overline{T}_{i-1} = \langle T_{i-1}, \mathsf{pp}_E \,\|\, \mathsf{pp}_W\rangle. \end{aligned} \tag{A8}$$

By combining Eqs. (A7) and (A8) with Eqs. (A3) and (A4),

$$\begin{aligned} \overline{E}^{(1)}_i &= \mathsf{com}(\mathsf{pp}_E, E^{(1)}_i) = \langle E^{(1)}_i, \mathsf{pp}_E\rangle \\ &= \overline{E}^{(1)}_{i-1} + r\overline{T}_{i-1} + r^2\overline{E}^{(2)}_{i-1} \\ &= \langle E^{(1)}_{i-1} + rT_{i-1} + r^2 E^{(2)}_{i-1}, \mathsf{pp}_E \,\|\, \mathsf{pp}_W\rangle \\ &= \langle E^{(1)}_{i-1} + rT_{i-1} + r^2 E^{(2)}_{i-1}, \mathsf{pp}_E\rangle. \end{aligned} \tag{A9}$$

The last equitation holds under the discrete logarithm assumption of the group. Similarly, we have

$$\begin{aligned} \overline{W}^{(1)}_i &= \mathsf{com}(\mathsf{pp}_W, W^{(1)}_i) = \langle W^{(1)}_i, \mathsf{pp}_W\rangle \\ &= \overline{W}^{(1)}_{i-1} + r\overline{W}^{(2)}_{i-1} \\ &= \langle W^{(1)}_{i-1} + rW^{(2)}_{i-1}, \mathsf{pp}_E \,\|\, \mathsf{pp}_W\rangle \\ &= \langle W^{(1)}_{i-1} + rW^{(2)}_{i-1}, \mathsf{pp}_W\rangle. \end{aligned} \tag{A10}$$

By the Eqs. (A8)–(A10) and the additive homomorphism of the inner product, we obtain the following relation,

$$\begin{aligned} &\overline{E}^{(1)}_{i-1} = \mathsf{com}(\mathsf{pp}_E, E^{(1)}_{i-1}), \ \overline{W}^{(1)}_{i-1} = \mathsf{com}(\mathsf{pp}_W, W^{(1)}_{i-1}), \\ &\overline{E}^{(2)}_{i-1} = \mathsf{com}(\mathsf{pp}_E, E^{(2)}_{i-1}), \ \overline{W}^{(2)}_{i-1} = \mathsf{com}(\mathsf{pp}_W, W^{(2)}_{i-1}). \end{aligned} \tag{A11}$$

Therefore, the instance-witness pairs $(\mathbb{U}_{i-1}, \mathbb{W}_{i-1})$, $(\mathbb{u}_{i-1}, \mathbb{w}_{i-1})$ pass the opening check of the commitment. Additionally, let $T = T_{i-1}$, $s = s^{(1)}_i$, $Z = (W^{(1)}_i, x^{(1)}_i, s^{(1)}_i)$, $E = E^{(1)}_i$, $s_b = s^{(b)}_{i-1}$, $Z_b = (W^{(b)}_{i-1}, x^{(b)}_{i-1}, s^{(b)}_{i-1})$, $E_b = E^{(b)}_{i-1}$ for $b \in \{0, 1\}$, and δ represents a redundant term contains Z_1, Z_2, A and B. By the premise that $(\mathbb{U}_i, \mathbb{W}_i) \in \mathcal{R}_{\mathsf{s}_{F'}}$ and Eq. (A7), we have

$$\begin{aligned} &AZ \circ BZ - s \cdot CZ - E \\ &\quad = A(Z_1 + rZ_2) \circ B(Z_1 + rZ_2) - (s_1 + rs_2) \cdot C(Z_1 + rZ_2) \\ &\qquad - (E_1 + rT + r^2 E_2) \\ &\quad = (AZ_1 \circ BZ_1 - s_1 \cdot CZ_1 - E_1) + r\delta \\ &\qquad + r^2 (AZ_2 \circ BZ_2 - s_2 \cdot CZ_2 - E_2) \\ &\quad = 0. \end{aligned} \tag{A12}$$

The last term of Eq. (15) is a degree-2 polynomial in r whose coefficients are determined by $p := (Z_1, E_1, Z_2, E_2, T)$. Let $\mathsf{P} : \mathbb{F}^{5m} \to \mathbb{F}^2[X]$ be a deterministic function, we define this polynomial as $\mathsf{P}(p)$. Since $r = \mathsf{H}(\mathbb{U}_{i-1}, \mathbb{u}_{i-1}, \overline{T}_{i-1})$, where $\mathbb{U}_{i-1}, \mathbb{u}_{i-1}, \overline{T}_{i-1}$ can be seen as commitments to p. By the general zero test property of the hash function H, the polynomial $\mathsf{P}(p)$ is a zero-polynomial with overwhelming probability. Therefore, we can deduce that

$$AZ_1 \circ BZ_1 - s_1 \cdot CZ_1 - E_1 = 0 = AZ_2 \circ BZ_2 - s_2 \cdot CZ_2 - E_2,$$

and $(\mathbb{U}_{i-1}, \mathbb{W}_{i-1}), (\mathbb{u}_{i-1}, \mathbb{w}_{i-1}) \in \mathcal{R}_{\mathsf{s}_{F'}}$. The last check of the verifier in Fig. 9 is passed.

The valid Π_i implies that $(\mathbb{u}_i, \mathbb{w}_i) \in \mathcal{R}_{\mathsf{s}_{F'}}$, where $(\mathbb{u}_i, \mathbb{w}_i)$ describes the trace of F' on inputs

$$(\mathbb{U}_{i-1}, \mathbb{u}_{i-1}, i-1, h_0, h_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{z}_{i-1}, \overline{T}_{i-1}).$$

By the construction of F' we have

$$\mathbb{u}_{i-1}.x = \mathsf{H}(\mathsf{vk}, i-1, h_0, \mathsf{H}(h_{i-1}, \boldsymbol{y}_{i-1}, \mathsf{F}(\boldsymbol{y}_{i-1})), \mathbb{U}_{i-1}),$$

and $(\mathbb{u}_{i-1}.\overline{E}, \mathbb{u}_{i-1}.s) = (\mathbb{u}_\perp.\overline{E}, 1)$. Therefore,

$$\mathcal{V}(\mathsf{vk}, i-1, h_0, h_{i-1}, \Pi_{i-1}) = 1.$$

□

References

[1] Boneh D, Gentry C, Lynn B, Shacham H. Aggregate and verifiably encrypted signatures from bilinear maps. In: Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology. 2003, 416−432

[2] Harn L. Batch verifying multiple RSA digital signatures. Electronics Letters, 1998, 34(12): 1219−1220

[3] Karati S, Das A, Roychowdhury D, Bellur B, Bhattacharya D, Iyer A. Batch verification of ECDSA signatures. In: Proceedings of the 5th International Conference on Cryptology in Africa on Progress in Cryptology. 2012, 1−18

[4] Karati S, Das A, Roychoudhury D. Randomized batch verification of standard ECDSA signatures. In: Proceedings of the 4th International Conference on Security, Privacy, and Applied Cryptography Engineering. 2014, 237−255

[5] Karati S, Das A. Faster batch verification of standard ECDSA signatures using summation polynomials. In: Proceedings of the 12th International Conference on Applied Cryptography and Network Security. 2014, 438−456

[6] Antipa A, Brown D, Gallant R, Lambert R, Struik R, Vanstone S. Accelerated verification of ECDSA signatures. In: Proceedings of the 12th International Workshop on Selected Areas in Cryptography. 2005, 307−318

[7] Cheon J H, Yi J H. Fast batch verification of multiple signatures. In: Proceedings of the 10th International Conference on Practice and Theory in Public-Key Cryptography. 2007, 442−457

[8] Liu T, Xie T, Zhang J, Song D, Zhang Y. Pianist: scalable zkRollups via fully distributed zero-knowledge proofs. In: Proceedings of 2024 IEEE Symposium on Security and Privacy (SP). 2024, 1777−1793

[9] Polygon. See Polygon.technology/polygon-zkevm website, 2024

[10] Scroll. See Scroll.io/ website, 2025

[11] ZKsync. See Zksync.io/ website , 2024

[12] Setty S. Spartan: efficient and general-purpose zkSNARKs without trusted setup. In: Proceedings of the 40th Annual International Cryptology Conference on Advances in Cryptology. 2020, 704−737

[13] Bünz B, Bootle J, Boneh D, Poelstra A, Wuille P, Maxwell G. Bulletproofs: short proofs for confidential transactions and more. In: Proceedings of 2018 IEEE Symposium on Security and Privacy (SP). 2018, 315−334

[14] Chiesa A, Hu Y, Maller M, Mishra P, Vesely N, Ward N. Marlin: preprocessing zkSNARKs with universal and updatable SRS. In: Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology. 2020, 738−768

[15] Kate A, Zaverucha G M, Goldberg I. Constant-size commitments to polynomials and their applications. In: Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security on Advances in Cryptology. 2010, 177−194

[16] Kothapalli A, Setty S, Tzialla I. Nova: recursive zero-knowledge arguments from folding schemes. In: Proceedings of the 42nd Annual International Cryptology Conference on Advances in Cryptology. 2022, 359−388

[17] C2PA technical specification. See C2pa.org/specifications/specifications/1.1/specs/C2PA_Specification website, 2024

[18] Curve with group order 2^255 − 19. See Moderncrypto.org/mail-archive/curves/2018/000992 website, 2018

[19] Spartan-ECDSA. See Github.com/personaelabs/spartan-ecdsa website, 2024

[20] Bünz B, Chen B. Protostar: generic efficient accumulation/folding for special-sound protocols. In: Proceedings of the 29th International Conference on the Theory and Application of Cryptology and Information Security. 2023, 77−110

[21] Kothapalli A, Setty S. HyperNova: recursive arguments for customizable constraint systems. In: Proceedings of the 44th Annual International Cryptology Conference on Advances in Cryptology. 2024, 345−379

[22] Groth J. On the size of pairing-based non-interactive arguments. In: Proceedings of the 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology. 2016, 305−326

[23] Gabizon A, Williamson Z J, Ciobotaru O. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. See Eprint.iacr.org/2019/953 website, 2019

[24] Zcash. See Zcash.readthedocs.io/en/latest/ website, 2019

[25] Zhang J, Xie T, Zhang Y, Song D. Transparent polynomial delegation and its applications to zero knowledge proof. In: Proceedings of 2020 IEEE Symposium on Security and Privacy (SP). 2020, 859−876

[26] Golovnev A, Lee J, Setty S, Thaler J, Wahby R S. Brakedown: Linear-time and field-agnostic snarks for R1CS. In: Proceedings of Annual International Cryptology Conference. 2023, 193–226

[27] Xie T, Zhang Y, Song D. Orion: zero knowledge proof with linear prover time. In: Proceedings of the 42nd Annual International Cryptology Conference on Advances in Cryptology. 2022, 299−328

[28] Kales D, Zaverucha G. Efficient lifting for shorter zero-knowledge proofs and post-quantum signatures. See Eprint.iacr.org/2022/588 website, 2022

[29] Zeilberger H, Chen B, Fisch B. BaseFold: efficient field-agnostic polynomial commitment schemes from foldable codes. In: Proceedings of the 44th Annual International Cryptology Conference on Advances in Cryptology. 2024, 138−169

[30] Block A R, Fang Z, Katz J, Thaler J, Waldner H, Zhang Y. Field-agnostic SNARKs from expand-accumulate codes. In: Proceedings of the 44th Annual International Cryptology Conference on Advances in Cryptology. 2024, 276−307

[31] Chen B, Bünz B, Boneh D, Zhang Z. HyperPlonk: plonk with linear-time prover and high-degree custom gates. In: Proceedings of the 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology. 2023, 499−530

[32] 0xPolygonZero/zk_evm. See Github.com/0xPolygonZero/zk_evm website, 2024

[33] Plonky2. See Github.com/0xPolygonZero/plonky2/blob/main/plonky2/plonky2 website, 2022

[34] Wu H, Zheng W, Chiesa A, Popa R A, Stoica I. DIZK: a distributed zero knowledge proof system. In: Proceedings of the 27th USENIX Conference on Security Symposium. 2018, 675−692

[35] Chiesa A, Lehmkuhl R, Mishra P, Zhang Y. EOS: efficient private delegation of zkSNARK provers. In: Proceedings of the 32nd USENIX Conference on Security Symposium. 2023, 361

[36] Sha J, Liu S. Delegable zk-SNARKs with proxies. Frontiers of Computer Science, 2024, 18(5): 185812

[37] Garg S, Goel A, Wang M. How to prove statements obliviously? In: Proceedings of the 44th Annual International Cryptology Conference on Advances in Cryptology. 2024, 449−487

[38] Das S, Camacho P, Xiang Z, Nieto J, Bünz B, Ren L. Threshold signatures from inner product argument: succinct, weighted, and multi-threshold. In: Proceedings of 2023 ACM SIGSAC Conference on Computer and Communications Security. 2023, 356−370

[39] Garg S, Jain A, Mukherjee P, Sinha R, Wang M, Zhang Y. hinTS: threshold signatures with silent setup. In: Proceedings of 2024 IEEE Symposium on Security and Privacy (SP). 2024, 3034−3052

[40] Qiu T, Tang Q. Predicate aggregate signatures and applications. In: Proceedings of the 29th International Conference on the Theory and Application of Cryptology and Information Security on Advances in Cryptology. 2023, 279−312

[41] Fuchsbauer G, Kiltz E, Loss J. The algebraic group model and its applications. In: Proceedings of the 38th Annual International Cryptology Conference on Advances in Cryptology. 2018, 33−62

[42] Lee H, Seo J H. On the security of nova recursive proof system. See Eprint.iacr.org/2024/232 website, 2024

[43] Nova. See Github.com/microsoft/Nova website, 2025

[44] Nguyen W, Boneh D, Setty S. Revisiting the nova proof system on a cycle of curves. See Eprint.Iacr.Org/2023/969 website, 2023

[45] National Institute of Standards and Technology. Digital signature standard (DSS). See Csrc.nist.gov/pubs/fips/186–5/final website, 2023

[46] Grassi L, Khovratovich D, Rechberger C, Roy A, Schofnegger M. Poseidon: a new hash function for zero-knowledge proof systems. In: Proceedings of the 30th USENIX Conference on Security Symposium. 2021, 519−535

[47] Kosba A, Papamanthou C, Shi E. xJsnark: a framework for efficient verifiable computation. In: Proceedings of 2018 IEEE Symposium on Security and Privacy (SP). 2018, 944−961

[48] 0xPARC. zk-ECDSA part 2: under the hood. See 0Xparc.org/blog/zk-ecdsa-2 website, 2024

[49] Katz J, Lindell Y. Introduction to Modern Cryptography. 2nd ed. Chapman & Hall/CRC, 2014

[50] bellpepper. See Github.com/lurk-lab/bellpepper website, 2024

[51] Nguyen W, Datta T, Chen B, Tyagi N, Boneh D. Mangrove: a scalable framework for folding-based SNARKs. In: Proceedings of the 44th Annual International Cryptology Conference on Advances in Cryptology. 2024, 308−344

Li LIU is a PhD candidate at the School of Cyber Science and Technology, Shandong University, China. Her current research interests include zero-knowledge proof, verifiable computation, and privacy protection.

Puwen WEI is currently a professor at the School of Cyber Science and Technology, Shandong University, China. His research interests include cryptography and blockchain security.

Shuchang LIU is an undergraduate at the School of Cyber Science and Technology, Shandong University, China. His current research interests include zero-knowledge proof and its applications in machine learning.

Zirui WANG is an undergraduate at the School of Cyber Science and Technology, Shandong University, China. His current research interests include zero-knowledge proof, machine learning, and full homomorphism encryption.

Da HU is an undergraduate at the School of Cyber Science and Technology, Shandong University, China. His current research interests include cryptanalysis and the design of symmetric key ciphers.

Zengjie KOU received his Bachelor’s degree in Mathematics from Shandong University, China and a Master’s degree in Management from Tsinghua University, China. Now, he is the Senior Vice President of Topsec Network Technology Inc., China. His current research interests include Cybersecurity and Industrial Internet.