Batch Range Proof: How to Make Threshold ECDSA More Efficient

Guofeng Tang[1], Shuai Han[2⋆], Li Lin[1], Changzheng Wei[1], and Ying Yan[1]

[1] Digital Technologies, Ant Group
[2] Shanghai Jiao Tong University

Abstract. With the demand of cryptocurrencies, threshold ECDSA recently regained popularity. So far, several methods have been proposed to construct threshold ECDSA, including the usage of OT and homomorphic encryptions (HE). Due to the mismatch between the plaintext space and the signature space, HE-based threshold ECDSA always requires zero-knowledge range proofs, such as Paillier and Joye-Libert (JL) encryptions. However, the overhead of range proofs constitutes a major portion of the total cost.

In this paper, we propose efficient batch range proofs to improve the efficiency of threshold ECDSA. At the heart of our efficiency improvement is a new technical tool called *Multi-Dimension Forking Lemma*, as a generalization of the well-known general forking lemma [Bellare and Neven, CCS 2006]. Based on our new tool, we construct efficient batch range proofs for Paillier and JL encryptions, and use them to give batch multiplication-to-addition (MtA) protocols, which are crucial to most threshold ECDSA. Our constructions improve the prior Paillier-based MtA by a factor of 2 and the prior JL-based MtA by a factor of 3, in both computation and bandwidth in an amortized way. Our batch MtA can be used to improve the efficiency of most Paillier and JL based threshold ECDSA. As three typical examples, our benchmarking results show:

- We improve the Paillier-based CGGMP20 [Canetti et al., CCS 2020] in bandwidth by a factor of 2.1 to 2.4, in computation by a factor of 1.5 to 1.7.
- By implementing threshold ECDSA with the batch JL MtA of XAL+23 [Xue et al., CCS 2023] and our batch JL MtA respectively, our batch construction improves theirs in bandwidth by a factor of 2.0 to 2.29, in computation by a factor of 1.88 to 2.09.
- When replacing OT-based MtA in DKLs24 [Doerner et al., S&P 2024] with our Paillier-based batch MtA, we improve the bandwidth efficiency by 7.8× at the cost of 5.7× slower computation.

1 Introduction

A (t, n) threshold signature builds on a secret-sharing scheme, which splits the signing key across n parties, such that: (1) any subset of t honest parties can produce a valid signature, without reconstructing the key; (2) any subset of fewer than t parties can neither produce a signature, nor find anything about the key [40,58]. Threshold signature has received much attention in cryptocurrencies, as it can be used to provide a high level of key protection [20]. In the industry, threshold ECDSA is the most widely deployed threshold signature, which has become a major tool for protecting hundreds of billions of dollars in cryptocurrency wallets, and is currently powering the wallets of Coinbase [3], Binance [1], Zengo [5], BitGo [2], Fireblocks [4] and many other fintech companies, with servicing thousands of financial institutions and hundreds of millions of end-user consumers. Besides cryptocurrency wallets, threshold cryptography has also been found to have other applications in multiple scenarios, including distributed key management [45], decentralized identity systems [52] and Byzantine fault tolerance (BFT) consensus algorithms [63,37].

Recall that an ECDSA signature on a message msg involves computing $\sigma = k^{-1} \cdot (\mathcal{H}(\mathsf{msg}) + r \cdot x)$, where x is the secret key, k is a secret nonce and r is a public nonce. The intuitive approach for designing threshold ECDSA is to secretly share x and k, with each one holding an additive share (x_i, k_i), but a challenge arises when attempting to compute k^{-1} and $k^{-1} \cdot x$ in a distributed manner. A large number of practical protocols [31,32,33,39,16,50,51,38,23,25,26,60] aim to address this obstacle using a two-party multiplication-to-addition (MtA) protocol. Specifically, with shares a, b as inputs, the MtA protocol securely computes shares α, β such that $\alpha + \beta = a \cdot b$ as output. By having each pair of parties invoke

⋆ Guofeng Tang (tang.guofeng789@gmail.com) and Shuai Han (dalen17@sjtu.edu.cn) contributed equally to this research.

Fig. 1: **Our Paillier-based batch MtA protocol.** Let (pk_B, sk_B) is Bob's key pair of Paillier encryption scheme. $\Sigma_{\mathsf{enc}}[\ell]$ and $\Sigma_{\mathsf{aff}}[\ell]$ are our batch range proofs, which respectively prove that the plaintexts in ciphertexts $\{C_i\}_{i\in[\ell}$ and the coefficients in $\{D_i\}_{i\in[\ell]}$ are within given ranges.

Alice($\{a_i\}_{i\in[\ell]}$)		Bob($pk_B, sk_B, \{b_i\}_{i\in[\ell]}$)
	$\xleftarrow{\{C_i\}_{i\in[\ell]}, \Sigma_{\mathsf{enc}}[\ell]}$	$C_i \leftarrow \mathsf{Enc}(pk_B, b_i),\ \forall i \in [\ell]$
$D_i = C_i^{a_i} \cdot \mathsf{Enc}(pk_B, \alpha_i),\ \forall i \in [\ell]$	$\xrightarrow{\{D_i\}_{i\in[\ell]}, \Sigma_{\mathsf{aff}}[\ell]}$	$\beta_i \leftarrow \mathsf{Dec}(sk_B, D_i),\ \forall i \in [\ell]$
$\alpha_i = -\alpha_i \bmod q,\ \forall i \in [\ell]$		$\beta_i = \beta_i \bmod q,\ \forall i \in [\ell]$

the two-party MtA protocol, the parties can securely split k^{-1} and $k^{-1} \cdot x$ into additive shares, and thus generate σ securely.

Multiplication-to-Addition (MtA) Protocols. Efficient constructions of MtA fall into two main categories: OT-based [31,32,33] and homomorphic encryption (HE) based, where HE-based solutions include Paillier encryption (e.g. [39,16,50,51,38,23]), Castagnos-Laguillaumie encryption (CL, e.g. [25,26,60]) and Joye-Libert encryption (JL, e.g. [61]). Threshold ECDSA schemes based on different methods involve different tradeoffs between computation and communication complexity. Those based on OT are excellent in terms of computation cost, but the bandwidth (≈ 100 KB for 128-bit security) is the main bottleneck in real-world deployment scenario. CL-based schemes enjoy the lowest bandwidth, while it is computationally heavy. Those based on Paillier might be the most popular ones and have been employed by numerous fintech companies (Fireblocks, ZenGo, Coinbase, etc.) due to its favorable tradeoff between computational and communicational costs. Recently, JL-based threshold ECDSA also received significant attention due to its better performance compared to Paillier under certain parameter settings [61].

To ensure the security of HE-based MtA protocols like Paillier [54] and JL [47] encryptions, the ciphertexts of HE are always equipped with range-related zero-knowledge (ZK) proofs, to prove that the underlying plaintexts/coefficients are within given ranges [59]. This is mainly due to the mismatch between the plaintext space (e.g., 3072 bits for Paillier under 128-bit security level) and the ECDSA signature space (e.g., 256 bits). Let us take MtA protocol based on Paillier [38,51] as an example: Bob with private input b computes $C = \mathsf{Enc}(b)$ under his public key, and sends it to Alice along with a range ZK proof of proving the underlying plaintext b is within a given range; Alice with private input a picks a random α, computes $D = C^a \cdot \mathsf{Enc}(\alpha) = \mathsf{Enc}(ab + \alpha)$ homomorphically and sends it to Bob along with a range-affine ZK proof of proving her private values a, α are respectively within given ranges. Then by decrypting D, Bob gets $\beta := ab + \alpha$, which together with the $-\alpha$ knowing by Alice constitute additive shares of ab, i.e., $-\alpha + \beta = ab$. Similarly, replacing Paillier with JL yields an MtA protocol based on JL, and it also requires those range proofs mentioned above.

Presigning/Online. Since the work of Canetti et al. [23], almost all efficient threshold ECDSA protocols are in non-interactive mode: before seeing the message msg a presigning phase can be performed, followed by a one-round online phase after given msg. The online phase, which typically requires only a few simple field operations, is already nearly optimal, so the focus for improving the efficiency of threshold ECDSA is on the presigning protocol, whose main overhead is in turn dominated by the generation of additive shares of $k^{-1}, k^{-1}x$ (or called pre-signature) using MtA.

One natural idea is to perform batch presigning, by generating the additive shares in batches of multiple pre-signatures $\{(k^{(i)})^{-1}, (k^{(i)})^{-1}x\}_{i\in[\ell]}$, where $k^{(i)}$ denotes the secret nonce required for one signature. Those shares of pre-signatures can then be consumed by an online protocol to generate ℓ ECDSA signatures in total. Thus the cost of presigning is in an "amortized" way. From the above analysis, the presigning phase heavily relies on the MtA protocol, therefore an efficient batch MtA is in demand.

Now we consider the possible ways for achieving batch MtAs (excluding the trivial repetitions of MtAs). Given multiple inputs $\{a_i, b_i\}_{i\in[\ell]}$, Alice and Bob both need to operate on different ciphertexts for different inputs, so the ciphertexts operations of HE cannot be cost-efficiently aggregated. Recently, Xue et al. [61] presented a JL-based MtA construction, where a commitment scheme is required. Furthermore, they explored batch JL MtA by utilizing vector commitment to reduce amortized costs. However, the effectiveness of this batch technique is limited, since the commitment overhead accounts for only a small fraction. Moreover, it is not clear how to extend this technique into the Paillier MtA, that is probably the most widely used by numerous companies.

There might be some hope to realize efficient batch MtAs via designing efficient batch range proofs, since their overheads account for a large portion of the total overhead ($> 75\%$), for both Paillier and JL MtA. However, it is still an open problem how to efficiently generate and verify Paillier or JL range proofs *in a batched manner*.

Batch Range Proofs for Paillier and JL. While there have been advances in batch techniques for certain types of range proofs, such as aggregate range proof for Pedersen commitments in Bulletproofs [21], extending these techniques to specifically support Paillier or JL range proofs requires careful consideration of the underlying mathematical properties and operations. Recently, Gong et al. [42] adapted the approach from Bulletproofs to compile a batch range proof for Paillier. However, the verification time is linear with respect to the bit-length of witness even for a single proof. Thus, if using this for threshold ECDSA, it would undeniably retard the efficiency. As a result, existing threshold ECDSA constructions did not explore Bulletproofs to construct range proofs, for practical purposes.

In the existing threshold ECDSA, range proofs for a single instance are usually designed as Schnorr-style Σ-protocols and they utilize the general forking lemma [12,17] with a *single* small challenge value, e.g., [38,51,24]. When it comes to batched one for multiple ciphertexts, the Verifier is required to sample *multiple* small challenges. Without this requirement, it is hard for the Prover to guarantee that the aggregated response is within the desired range, and thus the range proof cannot be successfully completed. However, the general forking lemma only supports a single small challenge value, and appears to be insufficient for realizing batch range proofs. We will explain this in more details in Subsect. 1.2.

This motivates us to develop a new variant of the general forking lemma to support multiple small challenges for constructing efficient batch range proofs, which are essential for building efficient batch MtA based on Paillier or JL encryptions, and ultimately for improving the efficiency of threshold ECDSA.

1.1 Our Contributions

In this work, we generalize the well-known general forking lemma [12,17] to the multiple instances setting, and propose a new *Multi-Dimension Forking Lemma*. It deals with ZK arguments where the Verifier's challenge is a vector with multiple independent and small values, and allows to construct batch range proofs for Paillier and JL encryptions efficiently.

Based on our new flavor of forking lemma, we construct batch range-related proofs for Paillier and JL encryptions. These include the batch range proof $\Sigma_{\mathsf{enc}}[\ell]$ for proving that the plaintexts $\{b_i\}$ in ℓ Paillier ciphertexts $\{C_i = \mathsf{Enc}(b_i)\}$ are all within a given range, the batch range-affine proof $\Sigma_{\mathsf{aff}}[\ell]$ for proving that the coefficients $\{a_i, \alpha_i\}$ in ℓ homomorphically generated Paillier ciphertexts $\{D_i = C_i^{a_i} \cdot \mathsf{Enc}(\alpha_i) = \mathsf{Enc}(a_i b_i + \alpha_i)\}$ are respectively within given ranges. For each proof, we compare the original costs of the single proof and the amortized costs of the batched one, for Paillier encryption (cf. Table 2 in Sect. 4) and JL encryption (cf. Table 3 in Sect. 5) respectively.

Next, by utilizing the proposed batch range-related proofs, we construct batch MtA protocol for Paillier (cf. Figure 1) and JL (cf. Figure 3 in Appendix D), which improve the efficiency of existing works with a factor about 2 to 3. More precisely, according to theoretical analysis (cf. Table 1) and benchmarking results (cf. Table 4 in Subsect. 6.1), our batch Paillier-based MtA achieves about 2× reduction in both bandwidth and computation compared to the existing ones [38,51], and our batch JL MtA has 3× improvement on Xue et al.'s JL MtA and 2× improvement on their batched one [61].

By replacing the MtA in existing threshold ECDSA with our batch MtA, we can enhance the efficiency of most Paillier and JL based constructions [50,51,38,23,61]. As three typical examples, we demonstrate how our batch MtA protocol can significantly improve the efficiency of prior works, as follows.

- Canetti et al. [23,24] (CGGMP20) presented three versions of non-interactive threshold ECDSA based on Paillier, the critical parts of which are three presigning schemes and two of them are proven to be Universally Composable (UC) secure. According to our theoretical analysis (cf. Table 5 in Subsect. 6.2), we improve all of the three presigning schemes by a factor of > 1.5. We benchmark the implementations of the two UC-secure versions and our batched variants. The results (cf. Table 6 in Sect. 6) show that our bandwidth costs are about 2.1× to 2.4× lower, and our computation times are approximately 1.5× to 1.7× faster than the constructions of CGGMP20 for the two-party settings.
- Xue et al. [61] (XAL$^+$23) exploited JL encryption to construct MtA protocol, and used JL vector commitment to give a batch MtA. We implement threshold ECDSA using their (batch) JL MtA and our batch JL MtA, respectively. The results (cf. Table 6 in Sect. 6) show that our batch technique

Table 1: **Theoretical cost comparisons of MtA and batch MtA protocols.** The costs are amortized over the number ℓ of instances. Here $\boldsymbol{\mu} = \log N, \boldsymbol{\kappa} = \log q$ correspond to the message size of elements in $\mathbb{Z}_N$ and EC group $\mathbb{G}$, respectively. $\mathbf{N}^2, \mathbf{N}$ denote computing exponentiation over the rings $\mathbb{Z}_{N^2}, \mathbb{Z}_N$, respectively. Note that for JL-based MtA, N will be replaced with a different JL modulus N_J.

MtA	Communication		Computation	
	$\boldsymbol{\mu}$	$\boldsymbol{\kappa}$	$\mathbf{N}^2$	$\mathbf{N}$
Paillier [38,51]	19	22	11	23
Our Batch Paillier	$9 + 10/\ell$	$6 + 16/\ell$	$7 + 4/\ell$	$15 + 8/\ell$
JL [61]	9	19	0	24
Batch JL [61]	$6 + 3/\ell$	$17 + 2/\ell$	0	$20 + 4/\ell$
Our Batch JL	$3 + 6/\ell$	$5 + 16/\ell$	0	$11 + 13/\ell$

improves the bandwidth efficiency by a factor of 2.0 to 2.29, and the computational efficiency by a factor of 1.88 to 2.09, compared with theirs.

- Doerner et al. [33] (DKLs24) proposed an OT-based three-round threshold ECDSA, which is currently round-minimal to the best of our knowledge. To improve the bandwidth efficiency, we replace its underlying OT-based MtA with our batch Paillier and JL MtAs, still being round-minimal. The Paillier-based construction improves the bandwidth efficiency by 7.8×, at the cost of 5.7× slower computation than OT-based DKLs24. The JL-based scheme improves the OT-based one by 12× in bandwidth, but is 14× slower in computation. Under DKLs24 construction framework, the scheme with batch Paillier MtA has about 1.8× lower bandwidth and computational overhead than that with the Paillier MtA from [38,51].

Finally, we note that our multi-dimension forking lemma is a general tool for obtaining efficient batch ZK (range) proofs, and it can also be used to efficiently prove the knowledge of multiple Paillier plaintexts, JL plaintexts or other homomorphic encryptions (with or without considering range).

1.2 Our Techniques

In this subsection, we give a high-level overview on our new techniques in constructing batch range-related ZK proofs efficiently, and we will focus on Paillier encryption in this overview. The same ideas can be applied to JL encryption by taking into account the underlying mathematical structure.

Firstly, we briefly recall the range proof for a single Paillier ciphertext in [38,51,24]. To show that the plaintext m of a Paillier encryption $C = \mathsf{Enc}(m)$ is in a given range $[0, B]$, the Prover first picks a random m_0 (from a set to be specified), and returns its ciphertext $C_0 = \mathsf{Enc}(m_0)$ as the "commit" message of Σ-protocol; after receiving a challenge $e \leftarrow\$ [0, 2^t - 1]$ from the Verifier, the Prover outputs $m^* = m_0 + em$ over $\mathbb{Z}$ in response. Note that to achieve zero-knowledge with ε-bit statistical security, m_0 will be chosen from a larger set $[0, 2^{\varepsilon+t}B]$ to mask m. Accordingly, the Verifier will check whether the obtained m^* satisfies $m^* \in [0, 2^{\varepsilon+t}B]$, and determine whether $m^* \stackrel{?}{=} m_0 + em$ indeed holds via homomorphic encryption $\mathsf{Enc}(m^*) \stackrel{?}{=} C_0 \cdot C^e$. The soundness follows from the uniform randomness of $e \leftarrow\$ [0, 2^t - 1]$, which guarantees that $m \in [-2^{\varepsilon+t}B, 2^{\varepsilon+t}B]$ can be inferred from the verification of $m^* \in [0, 2^{\varepsilon+t}B]$ with overwhelming probability of $1 - \frac{1}{2^\varepsilon}$. We note that there is a gap between the initial range $[0, B]$ and the range $[-2^{\varepsilon+t}B, 2^{\varepsilon+t}B]$ guaranteed by the soundness, and the gap is approximately $2^{\varepsilon+t}$. Nevertheless, such a soundness gap ($2^{\varepsilon+t}$) can be tolerated in many scenarios such as threshold ECDSA.

When facing $\ell > 1$ ciphertexts $\{C_i = \mathsf{Enc}(m_i)\}_{i\in[\ell]}$, batch range proof aims to prove that all underlying plaintexts $\{m_i\}_{i\in[\ell]}$ are in $[0, B]$ in an aggregated manner. One natural batching idea is called Reed-Solomon encoding [55], which has been widely applied in aggregating ZK proofs in the discrete logarithm (DL) setting [41,17,21,14,65]. More precisely, the Prover still picks a single random masking value m_0 and sends its Paillier ciphertext $C_0 = \mathsf{Enc}(m_0)$ as the "commit" message, and the Verifier still sends a single challenge $e \leftarrow\$ [0, 2^t - 1]$, similar to the case of single ciphertext. Now, to deal with multiple ciphertexts, the Prover needs to extend the single challenge e to multiple exponents $(e, e^2, \cdots, e^\ell)$ and computes the response

$$m^* = m_0 + \boxed{e \cdot m_1 + e^2 \cdot m_2 + \cdots + e^\ell \cdot m_\ell}, \tag{1}$$

and accordingly, the Verifier checks the range of the obtained m^* and verifies whether (1) holds via

$$\mathsf{Enc}(m^*) \stackrel{?}{=} C_0 \cdot \boxed{(C_1)^e \cdot (C_2)^{e^2} \cdots (C_\ell)^{e^\ell}}. \tag{2}$$

Such batching method that extending a single challenge e to multiple exponents $(e, e^2, \cdots, e^\ell)$ is in fact widely applied in the literature. The benefit is that it aggregates the proofs of multiple instances effectively, while still allowing the extraction of witnesses $(m_1, \cdots, m_\ell)$ from successful proofs to achieve Argument of Knowledge (AoK). The later is guaranteed by the celebrated general forking lemma [12,17], and the argument is roughly as follows. According to the general forking lemma [12,17], one can rewind a (possibly malicious) Prover to obtain multiple successful proofs m_j^* (say $j \in [\ell+1]$) under *distinct* challenges e_j ($j \in [\ell+1]$), but w.r.t. a same first-round message C_0. This means that each m_j^* satisfies equation (1) under the corresponding e_j, and this forms a system of linear equations in $(m_0, m_1, \cdots, m_\ell)$, namely

$$\begin{pmatrix} m_1^* \\ m_2^* \\ \vdots \\ m_{\ell+1}^* \end{pmatrix} = \begin{pmatrix} 1 & e_1 & e_1^2 & \dots & e_1^\ell \\ 1 & e_2 & e_2^2 & \dots & e_2^\ell \\ \vdots & \vdots & \cdots & & \vdots \\ 1 & e_{\ell+1} & e_{\ell+1}^2 & \dots & e_{\ell+1}^\ell \end{pmatrix} \cdot \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ \vdots \\ m_\ell \end{pmatrix}.$$

Since the challenges e_j's are all distinct (guaranteed by the general forking lemma), the above Vandermonde matrix formed by e_j's powers is *invertible*, and then one can recover the witness $(m_1, \cdots, m_\ell)$ efficiently by solving the system of linear equations.

However, such batching method suffers from two disadvantages, as explained below.

Disadvantage 1: Bad Efficiency. Note that the verification in (2) involves ℓ modular exponentiations with exponents $(e, e^2, \cdots, e^\ell)$, highlighted by dotted boxes. Thus, the bit-length of challenge e dominates the efficiency of the protocol. According to the general forking lemma [12,17], to show the AoK property of the protocol, e should be of bit-length at least $t \geq \lambda$, where λ denotes the security parameter. Therefore, the exponents $(e, e^2, \cdots, e^\ell)$ of the modular multi-exponentiation in (2) have bit-length at least $(\lambda, 2\lambda, \cdots, \ell\lambda)$, which would be extremely long in large settings. Concretely, suppose the number of ciphertexts is $\ell = 100$ and the security parameter is $\lambda = 128$, then the exponents $(e, e^2, \cdots, e^\ell)$ have bit-length $(128, 256, \cdots, 12800)$, which makes the verification extremely inefficient.

Disadvantage 2: Almost No Range Guarantee. Even worse, the above batch protocol in fact *cannot* support range verification for the multiple $\{m_i\}_{i\in[\ell]}$. The reason is simple: even if all $\{m_i\}_{i\in[\ell]}$ lie in $[0, B]$, the m^* computed by (1) would be as large as $2^{\ell\lambda} \cdot B$ due to the largest exponent e^ℓ. Accordingly, the Verifier can only check the range of m^* by verifying whether $m^* \in [0, 2^{\ell\lambda} \cdot B]$ at best, which in turn can only guarantee that each m_i lies in $[-2^{\ell\lambda} \cdot B, 2^{\ell\lambda} \cdot B]$. The gap between $[0, B]$ and $[-2^{\ell\lambda} \cdot B, 2^{\ell\lambda} \cdot B]$ grows exponentially in the ciphertext number ℓ. Consequently, the protocol provides almost no range guarantee when ℓ is large. For example, suppose that the bound is $B = 2^{128}$, the number of ciphertexts is $\ell = 100$ and the security parameter is $\lambda = 128$, the Verifier can only be convinced that each m_i is in $[-2^{12800}, 2^{12800}]$ while Paillier's message space is of size only 2^{3072}, thus it is totally useless. Even in small settings, say $\ell = 10$, the Verifier can only infer that each m_i belongs to $[-2^{1280}, 2^{1280}]$, whose gap with $[0, 2^{128}]$ is still very large, making it hardly useful.

Our Solution: Multiple Small Challenges & Multi-Dimension Forking Lemma. Based on the above observations, the batching technique by extending a single challenge e to multiple exponents $(e, e^2, \cdots, e^\ell)$ leads to bad efficiency and more importantly does not work well for batch range proof.

To solve the above problems, we generalize the well-known general forking lemma [12,17] to the setting of multiple instances, and propose a new *Multi-Dimension Forking Lemma*. It deals with arguments where the challenges are vectors over a small subset of a large field, and allows us to aggregate multiple instances of range proofs by using multiple small challenges.

We show the usefulness of *Multi-Dimension Forking Lemma* by proposing efficient batch range-related proofs for Paillier and JL encryptions. Let us go back to the above situation where the Prover aims to show that all $\{m_i\}_{i\in[\ell]}$ in multiple ciphertexts $\{C_i\}_{i\in[\ell]}$ are in $[0, B]$. Now our new batching technique works as follows.

- The challenge is a vector $\mathbf{e} = (e_1, e_2, \cdots, e_\ell)$ instead of a single element e, and each entry e_i of the vector $\mathbf{e}$ is chosen from a small subset[3] (e.g. $[0, 2^t - 1]$ for $t \geq \lambda$).
- The Prover computes the response

$$m^* = m_0 + \boxed{e_1 \cdot m_1 + e_2 \cdot m_2 + \cdots + e_\ell \cdot m_\ell}, \tag{3}$$

and the Verifier checks the range of the obtained m^* as well as the equation

$$\mathsf{Enc}(m^*) \stackrel{?}{=} C_0 \cdot \boxed{(C_1)^{e_1} \cdot (C_2)^{e_2} \cdots (C_\ell)^{e_\ell}}.$$

Here, each exponent e_i has bit length of λ. Concretely, under the security parameter $\lambda = 128$, the bit-length of each e_i is just 128 no matter how many ciphertexts there are. Consequently, the efficiency of the verification (and also the proof generation) would be $\frac{\ell}{2}\times$ *faster* than the aforementioned solution which uses large multiple exponents $(e, e^2, \cdots, e^\ell)$. This efficiency improvement is huge in large settings.

Most importantly, our batching technique now can provide range guarantee for multiple ciphertexts. This reason is simple: since the challenge vector $\mathbf{e} = (e_1, e_2, \cdots, e_\ell)$ has small entries (e.g., $e_i \in [0, 2^t - 1]$), suppose that all $\{m_i\}_{i\in[\ell]}$ are in $[0, B]$ and m_0 is chosen from $[0, 2^{\varepsilon+t}B]$ where ε is the statistical parameter with $\varepsilon \geq \lambda$, the response m^* computed by (3) must lie in $[0, 2^{\varepsilon+t}B + \ell \cdot 2^t B]$. Now the bound only grows linearly in the number of instances ℓ, in stark contrast to that of the m^* computed by (1), which lies in $[0, 2^{\ell t} \cdot B]$ and grows exponentially in ℓ. Consequently, after receiving m^*, the Verifier can check whether $m^* \in [0, 2^{\varepsilon+t}B]$. It will hold unless the randomly sampled $m_0 > 2^{\varepsilon+t}B - \ell \cdot 2^t B$, thus $m^* \in [0, 2^{\varepsilon+t}B]$ happens with overwhelming probability of $1 - \frac{\ell}{2^\varepsilon} \geq 1 - \frac{\ell}{2^\lambda}$ for a valid statement. Then the success of the check would imply that each m_i lies in $[-2^{\varepsilon+t}B, 2^{\varepsilon+t}B]$, whose gap with the initial range $[0, B]$ is roughly $2^{\varepsilon+t}$. This shows that the range guarantee provided by our batching technique is approximately the same as that provided by the range proof for a single ciphertext, even in large settings (e.g., $\ell = 1000$).

However, now we can no longer resort to the general forking lemma to show the Argument of Knowledge (AoK) of this protocol. The reason is as follows. To apply the general forking lemma, we have to view the challenge vector $\mathbf{e} = (e_1, e_2, \cdots, e_\ell)$ as a whole (e.g., as an element in $[0, 2^t - 1]^\ell$), and by rewinding, one can obtain multiple successful proofs m_j^* (say $j \in [q]$ for some q) under distinct challenges $\mathbf{e}_j = (e_{j,1}, e_{j,2}, \cdots, e_{j,\ell})$ $(j \in [q])$ w.r.t. a same first-round message C_0. This gives us the following system of linear equations in $(m_0, m_1, \cdots, m_\ell)$:

$$\begin{pmatrix} m_1^* \\ m_2^* \\ \vdots \\ m_q^* \end{pmatrix} = \begin{pmatrix} 1 & e_{1,1} & e_{1,2} & \ldots & e_{1,\ell} \\ 1 & e_{2,1} & e_{2,2} & \ldots & e_{2,\ell} \\ \vdots & \vdots & \cdots & & \vdots \\ 1 & e_{q,1} & e_{q,2} & \ldots & e_{q,\ell} \end{pmatrix} \cdot \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ \vdots \\ m_\ell \end{pmatrix}. \tag{4}$$

Although each rows $(1, e_{j,1}, e_{j,2}, \cdots, e_{j,\ell})$ of the coefficient matrix are *distinct* with each other (guaranteed by the general forking lemma), the coefficient matrix formed by the $e_{i,j}$'s *is not necessarily of full column rank.* Consequently, the above system of equations may not have a unique solution and one can hardly recover the witness $(m_1, \cdots, m_\ell)$ correctly.

To see this, let us consider a concrete example that, suppose by rewinding, one gets multiple successful proofs m_j^* under challenge vectors $\mathbf{e}_1 = (1, 1, \cdots, 1)$, $\mathbf{e}_2 = (2, 2, \cdots, 2)$, $\cdots$, $\mathbf{e}_q = (q, q, \cdots, q)$. These challenges are distinct, but even if the number q of successful proofs is exponentially large, the coefficient matrix of (4) is of only rank 2, far from full column rank. Nevertheless, one may think that if we choose q sufficiently large, e.g., $q = 2^{t\ell}$, then all vectors in $[0, ..., 2^t - 1]^\ell$ are taken, and the vectors must form a coefficient matrix of full column rank. However, we stress that these vectors *may not all lead to successful proofs*, and only those vectors associated with successful proofs can be used to build the system of linear equations in (4) and contribute to the coefficient matrix. Still, there is no guarantee that the resulting coefficient matrix formed by those vectors leading to successful proofs is of full column rank.

[3] Here "small" means that the values in the subset should be small functions *in the number ℓ of ciphertexts*, and this is used to provide meaningful range guarantee for our batch protocol. Recall that the existing method via the original forking lemma involves values like $(e, e^2, \cdots, e^\ell)$, which grow exponentially in ℓ and thus cannot be used to provide a meaningful range guarantee for large setting (e.g., $\ell = 100$). Instead, our method involves values $(e_1, e_2, \cdots, e_\ell)$ with each entry being small-size independently of ℓ, and thus we say they are "small".

This is where our new *Multi-Dimension Forking Lemma* comes in. More precisely, we generalize the general forking lemma by allowing the challenge to be vectors, i.e., $\mathbf{e} = (e_1, e_2, \cdots, e_\ell)$ for some dimension ℓ. Clearly, the original general forking lemma can be viewed as a special case of our new lemma with dimension $\ell = 1$. For a challenge $\mathbf{e}$, we also associated it with an extended form $(1, \mathbf{e}) = (1, e_1, e_2, \cdots, e_\ell)$, named *extended challenge*. Then we present a carefully-designed rewinding process to output multiple successful proofs under multiple challenge vectors w.r.t. a same first-round message, such that

– the multiple challenge vectors $\mathbf{e}_j = (e_{j,1}, e_{j,2}, \cdots, e_{j,\ell})$ $(j \in [q])$ are not only distinct,
– but the associated extended challenges $(1, \mathbf{e}_j) = (1, e_{j,1}, e_{j,2}, \cdots, e_{j,\ell})$ are also *linearly independent*.

We call such multiple successful transcripts (consisting of the multiple successful proofs and corresponding challenge vectors $\mathbf{e}_j$) as *Accepting Transcripts with Linearly-Independent Extended Challenges (AT-LIEC)*. Of course, the number q of such extended challenges is at most $1 + \ell$ with ℓ the dimension, so does the number of AT-LIEC. Then we give a careful analysis of the (expected) running time and successful probability of our rewinding process, showing that it indeed works as pre-described efficiently.

Now we can use our new *Multi-Dimension Forking Lemma* to show the AoK property of our batch range proof. With our new rewinding process, one can get multiple successful proofs m_j^* $(j \in [\ell+1])$ under challenges $\mathbf{e}_j$ $(j \in [\ell+1])$ w.r.t. a same first-round message C_0, and the extend challenges $(1, \mathbf{e}_j)$ $(j \in [\ell+1])$ are now guaranteed to be linearly independent by our new lemma. This gives us a system of linear equations similar to (4), where the rows of the coefficient matrix are exactly the extend challenges $(1, \mathbf{e}_j)$. Consequently, the linear independency of the extended challenges implies that the coefficient matrix of (4) is now of full column rank, and then one can solve the system of linear equations to obtain the witness $(m_1, \cdots, m_\ell)$ successfully. In this way, our new forking lemma rescues the AoK of our batch range proof for Paillier while maintaining its good efficiency.

Overall, our batching technique is to use multiple small challenges $(e_1, e_2, \cdots, e_\ell)$ to aggregate the multiple instances to be proved, and our new *Multi-Dimension Forking Lemma* helps us to analyze its Argument of Knowledge (AoK) by linearly-independent extended challenges. The use of multiple small challenges not only improves the efficiency of the batch range proof, but also provides range guarantees close to proving the multiple instances individually. Based on our batching technique and multi-dimension forking lemma, we apply the same ideas to propose more batch range-related proofs for more complex relations that are needed in building batch MtA and threshold ECDSA protocols, to improve their efficiency.

1.3 More Discussions

In this part, we delve further into the motivations behind our multi-dimension forking lemma, by drawing comparisons with a recent work by Attema et al. [7]. Then, we show more application scenarios of our batch MtA protocols. Finally, we give more detailed discussions of the soundness gap of range proofs.

Comparison with A Recent Work [7]. Recently, Attema et al. [7] have dealt with a situation which seems to be similar as that considered in our work. However, we note that their techniques are not applicable to build efficient batch range-related ZK proofs for Paillier or JL encryption, as ours do. The reasons are as follows.

- Their techniques can be used to reduce the running-time of knowledge extractor. However, the problems we encountered and resolved are not about the running-time of the extractor. In fact, as we elaborated in Subsect. 1.2, even if the extractor can take all vectors in $\{0, ..., 2^t - 1\}^\ell$, only those vectors *leading to successful proofs* contribute to the coefficient matrix of (4). Consequently, even if we apply the ideas in [7] to reduce the running-time of the extractor, there is still no guarantee that the extractor can find a coefficient matrix of full rank and extract the witness successfully.
- Example 3 on page 12 in [7] considers a situation that might be closest to ours. In their example, it requires the relation to be defined over a *finite field* $\mathbb{F}$ and the challenges to be *uniformly sampled from the whole field* $\mathbb{F}$. However, for the Paillier or JL encryption, the underlying domain is only a ring (like $\mathbb{Z}_N$), and if the challenges are sampled from the whole domain ($\mathbb{Z}_N$), it will provide no range guarantee. In contrast, our new forking lemma does not require the underlying domain to be finite field, and allows the challenge vector sampled from a small subset to provide meaningful range guarantee.

Besides, we note that their work [7] focuses on *"knowledge soundness"*, while ours achieves *"witness-extended emulation"*. As motivated in [49,44,17,21], ZKAoKs are often used as subprotocols within larger protocols, and the notion of "witness-extended emulation" for ZKAoK will make the security analysis of the larger protocols easier, when compared with using "knowledge soundness". In our case, our efficient batch ZK range proofs are used to construct efficient batch MtA protocols, which in turn are used to build the presigning protocols of threshold ECDSA. So the adoption of "witness-extended emulation" is more suitable for the application of threshold ECDSA.

More Application Scenarios. Our batch MtA protocol can be used to generate a batch of random Beaver triples [11] in the offline phase of a general-purpose MPC protocols. These Beaver triples are used later to handle multiplication gates in the online phase. Threshold ECDSA is just an instance of MPC protocol. As another instance, our efficient batch MtA protocol could also be used to construct threshold BBS+ signatures [34], for building an anonymous credential scheme with threshold issuance.

Furthermore, at a high level, the MtA protocol can be interpreted as an Oblivious Linear Evaluation (OLE), which is a secure two-party protocol allowing a receiver to learn a secret linear combination of a pair of field elements held by a sender. Thus our batch MtA can realize the functionality of vector-OLE, with a_i being the same for all $i \in [\ell]$. Indeed, it is needed in many realistic two-party computation settings, including privacy-preserving machine learning [28,46], VOLE-based Private Set Intersection (PSI) [56], etc.

Range Proofs. Threshold ECDSA constructions [38,51,23,61] always pursue a simple Schnorr-type range proofs for the sake of efficiency. Even though it is in fact quite challenging to avoid a soundness gap as analyzed in [62], it is sufficient to ensure the security of threshold ECDSA. Gennaro and Goldfeder [38] first discussed how, in the absence of range proofs, Alice or Bob may maliciously choose a large input (e.g. Alice runs with input $a + q^7$) and then observe the success or failure of threshold ECDSA (whether the reduction $\bmod N$ took place or not) to obtain the bit information of the other party's private input. To solve this problem, they use range proofs to prove that the input belongs to $\mathbb{Z}_q$, followed by all Paillier and JL based constructions. Although the range proofs have soundness gap, it can ensure that $\bmod N$ is determined not to occur for inputs within $[-2^{\varepsilon+t}q, 2^{\varepsilon+t}q]$. Thus the security of threshold ECDSA can be guaranteed, which has been proven in all prior works.

Bartoli et al. [10] focused on batch Σ-protocols to devise proofs of knowledge, including the batched plaintext proofs of JL encryption, but without providing range guarantee. Assuming that their method is indeed amenable to range proofs for JL encryption, our batch construction is more efficient compared to their approach. More precisely, their computation cost contains $O(\ell^2)$ exponentiations over the JL ring $\mathbb{Z}_N$ where ℓ is the number of ciphertexts, while ours involves only $O(\ell)$ exponentiations.

2 Preliminaries

Notations. Let $\mathbb{N}$ and $\mathbb{R}$ denote the set of natural numbers and the field of real numbers, respectively. Let $\lambda \in \mathbb{N}$ denote the security parameter throughout the paper. For $i, j \in \mathbb{N}$ with $i < j$, define $[i, j] = \{i, i+1, \cdots, j\}$ and $[j] = \{1, 2, \cdots, j\}$. Denote by $x \leftarrow\$ \mathcal{X}$ the operation of sampling x uniformly at random from the set $\mathcal{X}$. For an algorithm $\mathcal{A}$, denote by $y \leftarrow \mathcal{A}(x; r)$, or simply $y \leftarrow \mathcal{A}(x)$, the operation of running $\mathcal{A}$ with input x and randomness r and assigning the output to y. "PPT" is short for probabilistic polynomial-time and "DPT" is short for deterministic polynomial-time. Denote by poly some polynomial function, and negl some negligible function. For a primitive XX and a security notion YY, we typically denote the advantage of a PPT adversary $\mathcal{A}$ by $\mathsf{Adv}^{\mathrm{YY}}_{\mathrm{XX},\mathcal{A}}(\lambda)$.

For an integer $N \in \mathbb{N}$, denote by $\mathbb{Z}_N$ the integral ring $\{0, 1, \cdots, N - 1\}$ and $\mathbb{Z}_N^*$ the group that contains all the elements in $\mathbb{Z}_N$ that are co-prime to N, i.e., $\mathbb{Z}_N^* = \{i \in \mathbb{Z}_N | \gcd(i, N) = 1\}$, where "gcd" stands for greatest common divisor. Let $\mathsf{lcm} : \mathbb{N}^2 \to \mathbb{N}$ denote the operation of computing the least common multiple, and $\phi : \mathbb{N} \to \mathbb{N}$ denote Euler's phi function, for instance, $\phi(N)$ is the size of $\mathbb{Z}_N^*$.

2.1 Zero-Knowledge Arguments of Knowledge

We recall the definitions of zero-knowledge arguments of knowledge from [43,21].

Language. Our proof system will work in the plain setting. Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a PPT decidable binary relation. We call w a witness for an instance u if $(u, w) \in \mathcal{R}$, and define the language $\mathcal{L} = \{u | \exists\ w,\ \text{s.t. } (u, w) \in \mathcal{R}\}$ as the set of instances u that have a witness w in the relation $\mathcal{R}$.

Proof System. A proof system Π is made up of two PPT algorithms: a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. By $tr \leftarrow \langle \mathcal{P}(s), \mathcal{V}(e) \rangle(u)$, we denote the transcript produced by $\mathcal{P}$ and $\mathcal{V}$ when interacting on common input

u, $\mathcal{P}$'s private input s and $\mathcal{V}$'s randomness e. We write $\langle \mathcal{P}(w), \mathcal{V}\rangle(u) = b$ to indicate whether $\mathcal{V}$ accepts, $b = 1$, or rejects, $b = 0$.

Definition 1 (Completeness). *$\Pi = (\mathcal{P}, \mathcal{V})$ has completeness, if for any $(u, w) \in \mathcal{R}$, it holds* $\Pr[\langle \mathcal{P}(w), \mathcal{V}\rangle(u) = 1] = 1 - \mathsf{negl}(\lambda)$.

Definition 2 (Witness-Extended Emulation). *$\Pi = (\mathcal{P}, \mathcal{V})$ has (statistical) witness-extended emulation, if for any DPT prover $\mathcal{P}^*$, there exists an expected polynomial time emulator $\mathcal{E}$, such that for any (possibly unbounded) stateful adversary $\mathcal{A}$, $\mathsf{Adv}^{ext}_{\mathcal{P}^*,\mathcal{E},\mathcal{A}}(\lambda) \le \mathsf{negl}(\lambda)$. The advantage $\mathsf{Adv}^{ext}_{\mathcal{P}^*,\mathcal{E},\mathcal{A}}(\lambda)$ is defined as*

$$\left| \begin{array}{l} \Pr\left[\mathcal{A}(tr) = 1 \,\middle|\, (u, s) \leftarrow \mathcal{A},\ tr \leftarrow \langle \mathcal{P}^*(s), \mathcal{V}\rangle(u)\right] - \\ \Pr\left[\begin{array}{c} \mathcal{A}(tr) = 1\ \wedge \\ (tr\ is\ acc. \Rightarrow (u, w) \in \mathcal{R}) \end{array} \,\middle|\, \begin{array}{c} (u, s) \leftarrow \mathcal{A}, \\ (tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u) \end{array}\right] \end{array} \right| \tag{5}$$

where the oracle is given by $\mathcal{O} = \langle \mathcal{P}^*(s), \mathcal{V}\rangle(u)$, and permits rewinding to a specific point and resuming with fresh randomness for $\mathcal{V}$ from this point onwards.

We can also define computational witness-extended emulation by requiring $\mathsf{Adv}^{ext}_{\mathcal{P}^*,\mathcal{E},\mathcal{A}}(\lambda)$ to be negligible only for PPT $\mathcal{A}$.

Definition 3 (Public Coin). *$\Pi = (\mathcal{P}, \mathcal{V})$ is called public coin, if all challenges sent from $\mathcal{V}$ to $\mathcal{P}$ are chosen uniformly at random and independently of $\mathcal{P}$'s messages, i.e., the challenges correspond to $\mathcal{V}$'s randomness.*

Definition 4 (Honest-Verifier Zero-Knowledge). *$\Pi = (\mathcal{P}, \mathcal{V})$ has honest-verifier zero-knowledge (HVZK), if there exists a PPT simulator $\mathcal{S}$, such that for any $(u, w) \in \mathcal{R}$ and any string e, the following two distributions are statistically indistinguishable:*

$$\Big(tr \leftarrow \langle \mathcal{P}(w), \mathcal{V}(e)\rangle(u) : tr\Big) \overset{s}{\approx} \Big(tr \leftarrow \mathcal{S}(u, e) : tr\Big),$$

where e is the public coin randomness used by $\mathcal{V}$.

Definition 5 (Zero-Knowledge Argument of Knowledge (ZKAoK)). *$\Pi = (\mathcal{P}, \mathcal{V})$ is a zero-knowledge argument of knowledge (ZKAoK) for relation $\mathcal{R}$, if it has completeness, witness-extended emulation and honest-verifier zero-knowledge.*

By the Fiat-Shamir transform [35], a public coin interactive ZKAoK can be easily converted to a non-interactive, zero-knowledge argument in the random oracle model [13]. The idea is replacing the challenges output by $\mathcal{V}$ with the output of a cryptographic hash function.

2.2 A General Forking Lemma

We recall the general forking lemma from [17].

Tree of Accepting Transcripts (AT-Tree). Suppose that we have a $(2\mu + 1)$-move argument with μ challenges $e_1, \cdots, e_\mu$ and $(\mu + 1)$ responses $z_0, z_1, \cdots, z_\mu$. Let $n_i \in \mathbb{N}$ for $1 \le i \le \mu$. Consider $\prod_{i=1}^{\mu} n_i$ accepting transcripts with challenges in the following tree format. The tree has depth μ and $\prod_{i=1}^{\mu} n_i$ leaves. The root of the tree is labelled with the statement u. Each node of depth $i < \mu$ has exactly n_i children, each labelled with a *distinct* value for the i-th challenge e_i. We refer to this as an $(n_1, \cdots, n_\mu)$-tree of accepting transcripts, or an $(n_1, \cdots, n_\mu)$-AT-tree for short (cf. the left tree of Figure 2).

Lemma 1 (General Forking Lemma [17]). *Let $\Pi = (\mathcal{P}, \mathcal{V})$ be a $(2\mu+1)$-move, public coin interactive protocol. For $i \in [\mu]$, let $n_i \in \mathbb{N}$, and let $\mathcal{X}_i$ be the domain from which the i-th challenge e_i is uniformly chosen. Assume that*

(i) there is a PPT witness extraction algorithm χ that succeeds in extracting a witness from an $(n_1, \cdots, n_\mu)$-AT-tree with overwhelming probability,
(ii) $\prod_{i=1}^{\mu} n_i \le \mathsf{poly}(\lambda)$, and
(iii) for any $1 \le i \le \mu$, the domain $\mathcal{X}_i$ is of size super polynomial in λ, i.e., $|\mathcal{X}_i| \ge 2^{\omega(\log \lambda)}$.

Then $\Pi = (\mathcal{P}, \mathcal{V})$ has witness-extended emulation.

Concretely, the advantage related to the witness-extended emulation of $\Pi = (\mathcal{P}, \mathcal{V})$ satisfies

$$\mathsf{Adv}^{ext}_{\mathcal{P}^*,\mathcal{E},\mathcal{A}}(\lambda) \le (\textstyle\prod_{i=1}^{\mu} n_i) \cdot \mathsf{poly}(\lambda) / \sqrt[3]{\min_{i=1}^{\mu} |\mathcal{X}_i|},$$

with $\mathsf{poly}(\lambda)$ some polynomial function in λ. The advantage is negligible when conditions (ii) and (iii) hold.

2.3 Paillier & Joye-Libert Encryptions

Definition 6 (Paillier Encryption [54]). *Define the Paillier cryptosystem as the tuple* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *below.*

1. $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$: *choose* $\kappa/2$*-long primes* P, Q*, and set* $N = PQ$*. Write* $pk = N$ *and* $sk = \mathsf{lcm}(P-1, Q-1)$*, where* κ *is determined according to the security level* λ*.*
2. $C \leftarrow \mathsf{Enc}(pk, m)$: *choose* $\rho \leftarrow\!\!\$\ \mathbb{Z}_N^*$*, then compute* $C = (1+N)^m \cdot \rho^N = (1+mN)\cdot \rho^N \pmod{N^2}$*.*
3. $m \leftarrow \mathsf{Dec}(sk, C)$: *compute* $m = \dfrac{[C^{sk} \pmod{N^2} - 1]}{N} \cdot sk^{-1} \pmod N$*.*

Definition 7 (Joye-Libert (JL) Modulus [47,15]). *Let* $N_J = \bar{P}\bar{Q} = (2^k p' + 1)(2q' + 1)$ *where* $\bar{P}, \bar{Q}$ *are primes,* $k > 1$*, and* p', q' *are prime numbers. In the following, we denote such special RSA modulus* N_J *as JL modulus. Under JL modulus, the Jacobi symbol is* $J_{N_J}(a) = (a^{\frac{\bar{P}-1}{2}} \pmod{\bar{P}}) \cdot (a^{\frac{\bar{Q}-1}{2}} \pmod{\bar{Q}})$*. Define the following sets*

$$\mathbb{J}_{N_J} = \{a \in \mathbb{Z}_{N_J}^* | J_{N_J}(a) = 1\},\ \mathbb{QNR} = \mathbb{J}_{N_J} \setminus \mathbb{QR}$$
$$\mathbb{QR} = \{a \in \mathbb{Z}_{N_J}^* | \exists x \in \mathbb{Z}_{N_J}^*, a = x^2 \pmod{N_J}\},$$
$$\mathbb{QR}_{2^k} = \{a \in \mathbb{Z}_{N_J}^* | \exists x \in \mathbb{Z}_{N_J}^*, a = x^{2^k} \pmod{N_J}\}.$$

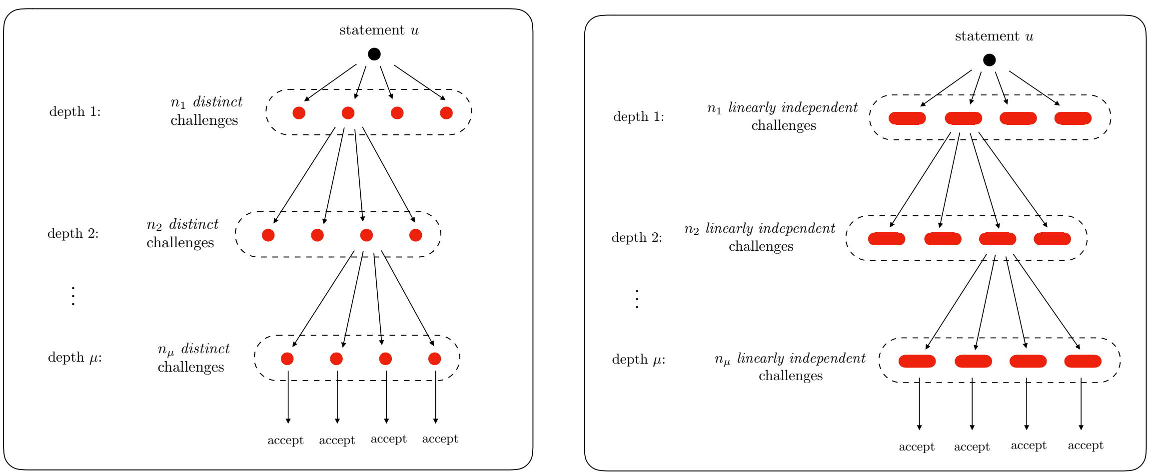
Definition 8 (JL Encryption [61]). *Define the JL cryptosystem as the tuple* $(\mathsf{JL.gen}, \mathsf{JL.enc}, \mathsf{JL.dec})$ *as follows.*

1. $(pk_J, sk_J) \leftarrow \mathsf{JL.gen}(1^\lambda)$: *define a proper integer* k *and choose* $\kappa/2$*-long primes* $\bar{P} = 2^k p' + 1, \bar{Q} = 2q' + 1$ *where* p', q' *are odd numbers, set* $N_J = \bar{P}\bar{Q}$*. It also picks* $y \leftarrow\!\!\$\ \mathbb{QNR}$ *and* $h \leftarrow\!\!\$\ \mathbb{QR}_{2^k}$*. Write* $pk_J = (N_J, h, y, k)$ *and* $sk_J = \bar{P}$*.*
2. $C \leftarrow \mathsf{JL.enc}(pk_J, m)$: *pick* $\rho \leftarrow\!\!\$\ \mathbb{Z}_{N_J}$ *and compute ciphertext* $C = y^m h^\rho \pmod{N_J}$*.*
3. $m \leftarrow \mathsf{JL.dec}(sk_J, C)$: *compute* $z = C^{\frac{\bar{P}-1}{2^k}} \pmod{\bar{P}}$*, find* $m \in \mathbb{Z}_{2^k}$ *such that* $z = \left(y^{\frac{\bar{P}-1}{2^k}}\right)^m \pmod{\bar{P}}$*.*

Based on the prior one in [15] (an extension of [47]), the above scheme is a modified JL encryption proposed in [61].

JL Commitment. The modified JL encryption can be easily converted into a commitment scheme. The public parameter $pp_{\mathsf{J}} = (\widetilde{N}_{\mathsf{J}}, \tilde{h}, \tilde{y}, k)$ can be generated via the key generation algorithm of the JL encryption scheme. For a message $m \in \mathbb{Z}$, its commitment $P \leftarrow \mathsf{JL.commit}(pp_{\mathsf{J}}, m)$ is $P = \tilde{y}^{2^k m} \cdot \tilde{h}^{2^k r} \pmod{\widetilde{N}_{\mathsf{J}}}$ with $r \leftarrow\!\!\$\ \mathbb{Z}_{\widetilde{N}_{\mathsf{J}}}$ as its randomness.

Fig. 2: Left: AT-Tree (Tree of Accepting Transcripts) in the General Forking Lemma; Right: AT-LIEC-Tree (Tree of Accepting Transcripts with Linearly-Independent Extended Challenges) in Our Multi-Dimension Forking Lemma



3 Multi-Dimension Forking Lemma

In this section, we propose our main technical tool — a new version of the forking lemma called *Multi-Dimension Forking Lemma*, which enables an efficient aggregation of multiple instances of (homomorphic) Σ-protocol by using only *small* challenges.

Tree of Accepting Transcripts with Linearly-Independent Extended Challenges (AT-LIEC-Tree). Suppose that we have a $(2\mu+1)$-move argument with μ challenges $\mathbf{e}_1, \cdots, \mathbf{e}_\mu$ and $(\mu+1)$ responses $z_0, z_1, \cdots, z_\mu$. For $i \in [\mu]$, suppose that the i-th challenge $\mathbf{e}_i = (e_{i,1}, \cdots, e_{i,d_i})$ is a vector uniformly chosen from $(\mathcal{X}_i)^{d_i}$, where $\mathcal{X}_i$ is a finite subset of $\mathbb{R}$ and $d_i \in \mathbb{N}$ denotes the dimension of the vector space. For simplicity, we assume that $\mathcal{X}_i$ is the same for all $1 \le i \le \mu$, denoted by $\mathcal{X}$. We stress that $\mathcal{X}$ does not necessarily have any algebraic structure (e.g., being closed under multiplication). Typically, we will set $\mathcal{X} = [0, 2^t - 1]$. The transcript in sequence is as follows:

$$\begin{array}{ll} \mathcal{P} \to \mathcal{V}: & z_0 \\ \mathcal{P} \leftarrow \mathcal{V}: & \mathbf{e}_1 = (e_{1,1}, \cdots, e_{1,d_1}) \in (\mathcal{X})^{d_1} \\ \mathcal{P} \to \mathcal{V}: & z_1 \\ \vdots & \\ \mathcal{P} \leftarrow \mathcal{V}: & \mathbf{e}_\mu = (e_{\mu,1}, \cdots, e_{\mu,d_\mu}) \in (\mathcal{X})^{d_\mu} \\ \mathcal{P} \to \mathcal{V}: & z_\mu. \end{array}$$

For the i-th challenge vector $\mathbf{e}_i$, we call

$$(1, \mathbf{e}_i) = (1, e_{i,1}, \cdots, e_{i,d_i}) \in \{1\} \times (\mathcal{X})^{d_i} \subseteq (\mathbb{R})^{d_i+1}$$

the associated i-th *extended* challenge. Let $n_i \in \mathbb{N}$ for $i \in [\mu]$. Consider $\prod_{i=1}^{\mu} n_i$ accepting transcripts with *extended* challenges in the following tree format. The tree has depth μ and $\prod_{i=1}^{\mu} n_i$ leaves. The root of the tree is labelled with the statement u. Each node of depth $i < \mu$ has exactly n_i children, each labelled with a value for the i-th extended challenge $(1, \mathbf{e}_i) = (1, e_{i,1}, \cdots, e_{i,d_i})$. Here we require that, for each node of depth i, the extended challenges $(1, \mathbf{e}_i^{(\eta)}) = (1, e_{i,1}^{(\eta)}, \cdots, e_{i,d_i}^{(\eta)})$ $(\eta \in [n_i]\})$ related to the n_i children are not only distinct, but also *linearly independent*. Clearly, it is necessary to require that $n_i \le 1 + d_i$ for $i \in [\mu]$. We refer to this as an $(n_1, \cdots, n_\mu)$-tree of accepting transcripts with linearly independent extended challenges, or an $(n_1, \cdots, n_\mu)$-AT-LIEC-tree (cf. the right tree of Figure 2). We note that an AT-LIEC-tree (as defined above) is always an AT-tree (as defined in Subsect. 2.2), but *not* vice versa.

In the following, we establish a new flavor of forking lemma, called *Multi-Dimension Forking Lemma*, which is able to *boost* a witness extractor χ who succeeds in extracting witnesses from AT-LIEC-trees *into* a witness-extended emulator.

Lemma 2 (Multi-Dimension Forking Lemma). *Let $\Pi = (\mathcal{P}, \mathcal{V})$ be a $(2\mu+1)$-move, public coin interactive protocol where the challenges are vectors over set $\mathcal{X}$ chosen uniformly. For $i \in [\mu]$, let d_i denote the dimension of the i-th challenge vector, and let $n_i \in \mathbb{N}$ with $n_i \le 1 + d_i$. Assume that*

(i) there is a PPT witness extraction algorithm χ that succeeds in extracting a witness from an $(n_1, \cdots, n_\mu)$-AT-LIEC-tree overwhelmingly. In formula, for any statement u and any $(n_1, \cdots, n_\mu)$-AT-LIEC-tree related to u, denoted by $\mathsf{AT\text{-}LIEC\text{-}tree}$, it holds $\Pr[w \leftarrow \chi(u, \mathsf{AT\text{-}LIEC\text{-}tree}) : (u, w) \in \mathcal{R}] = 1 - \mathsf{negl}(\lambda)$.
(ii) $\prod_{i=1}^{\mu} n_i \le \mathsf{poly}(\lambda)$, and
(iii) the set $\mathcal{X}$ is of size super polynomial in λ, i.e., $|\mathcal{X}| \ge 2^{\omega(\log \lambda)}$.

Then $\Pi = (\mathcal{P}, \mathcal{V})$ has witness-extended emulation.

Concretely, the advantage related to the witness-extended emulation of $\Pi = (\mathcal{P}, \mathcal{V})$ satisfies

$$\mathsf{Adv}^{ext}_{\mathcal{P}^*, \mathcal{E}, \mathcal{A}}(\lambda) \le (\mu+1) \cdot (\textstyle\prod_{i=1}^{\mu} n_i) \cdot \mathsf{poly}(\lambda) / \sqrt{|\mathcal{X}|}, \tag{6}$$

with $\mathsf{poly}(\lambda)$ some polynomial function in λ. The advantage is negligible when conditions (ii) and (iii) hold.

Before presenting the formal proof, we first give a brief overview of the intuitions behind the proof. The core of the proof is to construct an emulator to collect accepting transcripts whose extended challenge vectors $(1, \mathbf{e}_i)$ form a full rank matrix (for all rounds $i \in [\mu]$), and we need to ensure that this can be done in expected polynomial-time and with high probability.

Algorithm 1: The Emulator $\mathcal{E}$

```
  E^{⟨P*(s),V⟩}(u) → (tr, w):
1 initialize tree = ∅,   is-AT-LIEC = true;
2 run (tr, b, tree) ← T_1^{⟨P*(s),V⟩}(u, tree);
3 if b = 1 then
4 |   if is-AT-LIEC = true then
5 |   |   run w ← χ(u, tree) ;
6 |   |   return (tr, w);
7 |   else
8 |   |_  return (tr, ⊥);
9 else
10|_  return (tr, ⊥);
```

The strategy of the emulator is to execute $\langle \mathcal{P}^*, \mathcal{V}\rangle$ repeatedly, ensure that the extended challenges in the *last* round form a full rank matrix, then ensure that the extended challenges in the *second last* round form a full rank matrix, and so forth. This will be captured by a series of tree-finders $\{\mathcal{T}_i\}$ in our proof.

For the i-th tree finder (i.e., $\mathcal{T}_i$), it essentially samples challenge vectors $\mathbf{e}_i$ freshly, checks whether $\mathbf{e}_i$ leads to an accepting transcript, and checks whether the extended challenge vector $(1, \mathbf{e}_i)$ lies in the span of those it collected so far (denoted by S_i).

- To ensure $\mathcal{T}_i$ run in expected polynomial-time, we use two tricks:
  - If the first attempt fails, $\mathcal{T}_i$ will terminate directly.
  - If $(1, \mathbf{e}_i)$ lies in the span of S_i, $\mathcal{T}_i$ essentially terminates as well since it does not lead to an AT-LIEC-tree.

  These ensure that $\mathcal{T}_i$ would not repeat the steps too much times, and a careful analysis shows that it runs in expected polynomial-time.
- To ensure $\mathcal{T}_i$ succeed in collecting accepting transcripts whose extended challenge vectors $(1, \mathbf{e}_i)$ form a full rank matrix, we give a careful analysis:
  - Since $\mathcal{T}_i$ samples $\mathbf{e}_i$ independently in each step, the extended challenge vector $(1, \mathbf{e}_i)$ lies in the span of S_i with probability at most $1/|\mathcal{X}|$ (cf. (12) for more details).
  - Note that $\mathcal{T}_i$ succeeds as long as the chosen vectors $\{\mathbf{e}_i\}$ all do not lie in the span of S_i and are all linearly independent.

    Since $\mathcal{T}_i$ runs in expected polynomial-time, by Markov inequality, it stops in $\kappa \cdot \mathsf{poly}(\lambda)$ time except with probability at most $1/\kappa$ for some $\mathsf{poly}(\lambda)$ and κ, and in the case it runs in $\kappa \cdot \mathsf{poly}(\lambda)$ time, the bad event that there is some $\mathbf{e}_i$ chosen by $\mathcal{T}_i$ lying in the span of S_i can happen with probability at most $\kappa \cdot \mathsf{poly}(\lambda)/|\mathcal{X}|$.

  Overall, $\mathcal{T}_i$ is able to collect enough accepting transcripts except with probability $1/\kappa + \kappa \cdot \mathsf{poly}(\lambda)/|\mathcal{X}|$, which is negligibly small by choosing $\kappa = \sqrt{|\mathcal{X}|}$ and by requiring $|\mathcal{X}| \geq 2^{\omega(\log \lambda)}$.

Proof: We construct an expected polynomial time emulator $\mathcal{E}$, which has access to a rewindable transcript oracle $\mathcal{O} = \langle \mathcal{P}^*(s), \mathcal{V}\rangle(u)$ and produces a witness for statement u. See Algorithm 1 for the pseudo-code of $\mathcal{E}$. Intuitively, $\mathcal{E}$ proceeds in two main steps:

- Step 1 (cf. line 2 in Algorithm 1): it invokes a tree-finder $\mathcal{T}_1$, by which it obtains tree and a boolean variable is-AT-LIEC indicating whether tree is an AT-LIEC-tree or not[4];
- Step 2 (cf. lines 3-10 in Algorithm 1): if tree is indeed an $(n_1, ..., n_\mu)$-AT-LIEC-tree (which happens with overwhelming probability by our analysis below), it is able to extract a witness w from tree, using the efficient algorithm χ that exists by assumption.

A key technical tool in Step 1 is the so-called "tree-finder" $\mathcal{T}_1$. It is constructed by recursive calls to a series of $\{\mathcal{T}_i\}_{1 \leq i \leq \mu+1}$ that deal with the protocol $\langle \mathcal{P}^*(s), \mathcal{V}\rangle(u)$ after the first few challenges are already fixed. See Algorithm 2 for the pseudo-code of $\{\mathcal{T}_i\}_{1 \leq i \leq \mu+1}$. The i-th tree-finder $\mathcal{T}_i$ takes the

[4] For simplicity, we let is-AT-LIEC be a global variable that is implicitly given as input to (hence could be modified by) the other parts of the algorithms (e.g., $\mathcal{T}_1$ and $\{\mathcal{T}_i\}_{1 \leq i \leq \mu+1}$).

Algorithm 2: Tree Finder $\mathcal{T}_i$ ($i \in [\mu]$)

$\mathcal{T}_i^{\langle \mathcal{P}^*(s),\mathcal{V}\rangle}(u, \mathbf{e}_1, ..., \mathbf{e}_{i-1}, \mathsf{tree}) \to (tr, b, \mathsf{tree})$:

1 $\mathbf{e}_i \leftarrow\$ (\mathcal{X})^{d_i}$;
2 $(tr, b, \mathsf{tree}) \leftarrow \mathcal{T}_{i+1}^{\langle \mathcal{P}^*(s),\mathcal{V}\rangle}(u, \mathbf{e}_1, ..., \mathbf{e}_{i-1}, \mathbf{e}_i, \mathsf{tree})$;
3 **if** $b = 1$ **then**
4   **initialize** $S_i = (1, \mathbf{e}_i)$, $\mathsf{ctr}_i = 1$;
5   **while** $\mathit{ctr}_i < n_i$ **do**
6         $\mathbf{e}_i \leftarrow\$ (\mathcal{X})^{d_i}$;
7         $(tr', b, \mathsf{tree}) \leftarrow \mathcal{T}_{i+1}^{\langle \mathcal{P}^*(s),\mathcal{V}\rangle}(u, \mathbf{e}_1, ..., \mathbf{e}_{i-1}, \mathbf{e}_i, \mathsf{tree})$;
8   **if** $(1, \mathbf{e}_i) \in \mathit{span}(S_i)$ **then**
9   **set** is-AT-LIEC $\leftarrow$ false;
10   **if** $b = 1$ **then**
11             $S_i = S_i \cup \{(1, \mathbf{e}_i)\}$, $\mathsf{ctr}_i = \mathsf{ctr}_i + 1$;
12   **return** $(tr, 1, \mathit{tree})$;
13 **else**
14   **return** $(tr, 0, \perp)$;

$\mathcal{T}_{\mu+1}^{\langle \mathcal{P}^*(s),\mathcal{V}\rangle}(u, \mathbf{e}_1, ..., \mathbf{e}_\mu, \mathsf{tree}) \to (tr, b, \mathsf{tree})$:

15 **run** $tr \leftarrow \langle \mathcal{P}^*(s), \mathcal{V}(\mathbf{e}_1, ..., \mathbf{e}_\mu)\rangle(u)$ using $\mathbf{e}_1, ..., \mathbf{e}_\mu$ as challenges and $b \leftarrow \mathcal{V}(tr)$;
16 **if** $b = 1$ **then**
17     $\mathsf{tree} = \mathsf{tree} \cup \{(1, \mathbf{e}_1) \to \cdots \to (1, \mathbf{e}_\mu)\}$;
18   **return** $(tr, 1, \mathit{tree})$;
19 **else**
20   **return** $(tr, 0, \perp)$;

previous challenges $(\mathbf{e}_1, ..., \mathbf{e}_{i-1})$ given to it as input, picks random values for $\mathbf{e}_i$, and hands these values to the next tree-finder. If the transcript with $(\mathbf{e}_1, ..., \mathbf{e}_\mu)$ as challenges is accepting, $\mathcal{T}_{\mu+1}$ will add the corresponding extended challenges $\{(1, \mathbf{e}_1) \to \cdots \to (1, \mathbf{e}_\mu)\}$ to tree (cf. lines 16-20 in Algorithm 2).

For $1 \leq i \leq \mu$, each tree finder $\mathcal{T}_i$ may fail on the first value of $\mathbf{e}_i$, ensuring that the whole process runs in expected polynomial time. If the first value of $\mathbf{e}_i$ leads to an accepting transcript, $\mathcal{T}_i$ will proceed until it successfully collects n_i accepting transcripts with the given $(\mathbf{e}_1, ..., \mathbf{e}_{i-1})$ as the first $i-1$ challenges but having different values for the i-th challenges $\mathbf{e}_i$ (cf. lines 4-12 in Algorithm 2). The counter ctr_i records the number of accepting transcripts that $\mathcal{T}_i$ has collected so far, and the set S_i consists of the i-th extended challenges $(1, \mathbf{e}_i)$ that leading to accepting transcripts. If the i-th extended challenge $(1, \mathbf{e}_i)$ of a newly-chosen $\mathbf{e}_i$ lies in the span of S_i, the boolean variable is-AT-LIEC will be set to false (cf. lines 8-9 in Algorithm 2), indicating that the extended challenges might be linearly dependent, thus tree might not be an AT-LIEC one. Put differently, if is-AT-LIEC has never been set to false (i.e., is-AT-LIEC still equals true), the resulting tree must be an AT-LIEC-tree.

Due to the limited space, we provide the analyses of the expected running time of $\mathcal{E}$ and its advantage $\mathsf{Adv}^{ext}_{\mathcal{P}^*,\mathcal{E},\mathcal{A}}(\lambda)$ in Appendix A.2. ∎

4 Batch Range Proofs for Paillier Encryption

In this section, we present the efficient batch range-related proofs for Paillier. For instance, we construct Σ-protocols to prove that all plaintexts and coefficients in multiple ciphertexts are within given ranges. Based on our multi-dimension forking lemma (i.e., Lemma 2) for the special case of $\mu = 1$, we show that the proposed batch range proofs have witness-extended emulation.

4.1 Languages

For Paillier-based threshold ECDSA [51,38,23,24], it may involve the following relations in batch pre-signing phase, where all ZK range proofs can be generated and verified in a batched manner with our new version of the forking lemma.

Let $\ell \in \mathbb{N}$ denote the number of ciphertexts and N be the public key of Paillier encryption. Let $B, B_1, B_2 > 0$ denote the bounds of range. The relation for the range proof of ℓ Paillier's ciphertexts is

$$\mathcal{R}_{\mathsf{enc}}[\ell] = \left\{ \begin{matrix} ((B, \{C_i\}_{i\in[\ell]}), \\ (m_i, \rho_i)_{i=1}^{\ell}) \end{matrix} \middle| \begin{matrix} \forall\, i \in [\ell], m_i \in [0, B] \\ C_i = (1+N)^{m_i} \cdot (\rho_i)^N \bmod N^2 \end{matrix} \right\}.$$

Let $(\mathbb{G}, g, q)$ denote the group-generator-order tuple associated with the curve of ECDSA signatures. A variant of the above language related to the group $(\mathbb{G}, g, q)$ is

$$\mathcal{R}_{\mathsf{log}^*}[\ell] = \left\{ \begin{matrix} ((B, \{X_i, C_i\}_{i\in[\ell]}), \\ (m_i, \rho_i)_{i=1}^{\ell}) \end{matrix} \middle| \begin{matrix} \forall\, i \in [\ell], m_i \in [0, B] \\ C_i = (1+N)^{m_i} \cdot (\rho_i)^N \bmod N^2 \\ X_i = g^{m_i} \in \mathbb{G} \end{matrix} \right\}.$$

Let $(g_1, g_2) \in \mathbb{G}^2$ be the public parameters of ElGamal commitment over $\mathbb{G}$. Another variant requires that the plaintext m_i is also the opening of ElGamal commitment $\mathbf{c}_i = (A_i, B_i)$ with $A_i = g_1^{a_i}, B_i = g_1^{m_i} g_2^{a_i}$, i.e.,

$$\mathcal{R}_{\mathsf{enc\text{-}elg}}[\ell] = \left\{ \begin{matrix} ((B, \{\mathbf{c}_i, C_i\}_{i\in[\ell]}), \\ (m_i, \rho_i, a_i)_{i=1}^{\ell}) \end{matrix} \middle| \begin{matrix} \forall\, i \in [\ell], m_i \in [0, B] \\ C_i = (1+N)^{m_i} \cdot (\rho_i)^N \bmod N^2 \\ A_i = g_1^{a_i}, B_i = g_1^{m_i} g_2^{a_i} \in \mathbb{G} \end{matrix} \right\}.$$

For ℓ pairs of ciphertexts $(C_i, D_i) \in \mathbb{Z}_{N^2}^2$, we consider the following relation

$$\mathcal{R}_{\mathsf{aff}}[\ell] = \left\{ \begin{matrix} ((B_1, B_2, \\ \{C_i, D_i\}_{i\in[\ell]}), \\ \{m_i, \bar{m}_i, \rho_i\}_{i\in[\ell]}) \end{matrix} \middle| \begin{matrix} \forall\, i \in [\ell], m_i \in [0, B_1], \bar{m}_i \in [0, B_2] \\ D_i = C_i^{m_i} (1+N)^{\bar{m}_i} (\rho_i)^N \bmod N^2 \end{matrix} \right\}.$$

Given additional ℓ group elements $\{X_i\}_{i\in[\ell]}$, a variant of the above relation is

$$\mathcal{R}_{\mathsf{afflog}}[\ell] = \left\{ \begin{matrix} ((B_1, B_2, \\ \{X_i, C_i, D_i\}_{i\in[\ell]}), \\ \{m_i, \bar{m}_i, \rho_i\}_{i\in[\ell]}) \end{matrix} \middle| \begin{matrix} \forall\, i \in [\ell], m_i \in [0, B_1], \bar{m}_i \in [0, B_2] \\ D_i = C_i^{m_i} (1+N)^{\bar{m}_i} (\rho_i)^N \bmod N^2 \\ X_i = g^{m_i} \in \mathbb{G} \end{matrix} \right\}.$$

Given additional ℓ Paillier's ciphertexts $\{F_i\}_{i\in[\ell]}$, another more complex relation requires that each $\bar{m}_i$ is also the plaintext of F_i under a different public key N_1, i.e.,

$$\mathcal{R}_{\mathsf{aff\text{-}g}}[\ell] = \left\{ \begin{matrix} ((B_1, B_2, \\ \{X_i, C_i, D_i, F_i\}_{i\in[\ell]}), \\ \{m_i, \bar{m}_i, \rho_i, \bar{\rho}_i\}_{i\in[\ell]}) \end{matrix} \middle| \begin{matrix} \forall\, i \in [\ell], m_i \in [0, B_1], \bar{m}_i \in [0, B_2] \\ D_i = C_i^{m_i} (1+N)^{\bar{m}_i} (\rho_i)^N \bmod N^2 \\ X_i = g^{m_i} \in \mathbb{G} \\ F_i = (1+N_1)^{\bar{m}_i} (\bar{\rho}_i)^{N_1} \bmod N_1^2 \end{matrix} \right\}.$$

Also, if each secret value m_i is the plaintext of another Paillier's ciphertext E_i under N_1, there is a similar relation as follows

$$\mathcal{R}_{\mathsf{aff\text{-}p}}[\ell] = \left\{ \begin{matrix} ((B_1, B_2, \\ \{C_i, D_i, E_i, F_i\}_{i\in[\ell]}), \\ \{m_i, \bar{m}_i, \rho_i, \tilde{\rho}_i, \bar{\rho}_i\}_{i\in[\ell]}) \end{matrix} \middle| \begin{matrix} \forall\, i \in [\ell], m_i \in [0, B_1], \bar{m}_i \in [0, B_2] \\ D_i = C_i^{m_i} (1+N)^{\bar{m}_i} \rho_i^N \bmod N^2 \\ E_i = (1+N_1)^{m_i} \tilde{\rho}_i^{N_1} \bmod N_1^2 \\ F_i = (1+N_1)^{\bar{m}_i} \bar{\rho}_i^{N_1} \bmod N_1^2 \end{matrix} \right\}.$$

4.2 Constructions of Batch Proofs

Auxiliary Parameter. In range proofs for Paillier, the construction of the witness extraction algorithm relies on the strong RSA assumption, corresponding to an RSA modulus with unknown order for the Prover. Thus we instruct the Verifier to generate an auxiliary public parameter $pp = (\widetilde{N}, h_1, h_2)$ of the Ring-Pedersen commitment [36] in advance, where $\widetilde{N}$ is the product of two safe primes $(2\tilde{p}+1)$ and $(2\tilde{q}+1)$ with $\tilde{p}, \tilde{q}$ primes and h_1, h_2 are random squares in $\mathbb{Z}_{\widetilde{N}}^*$. The commitment of message m is $P = h_1^m h_2^r \pmod{\widetilde{N}}$ with $r \leftarrow\$\, \mathbb{Z}_{\widetilde{N}}$.

Batch Proof $\Sigma_{\mathsf{enc}}[\ell]$ for Language $\mathcal{R}_{\mathsf{enc}}[\ell]$. Let $t > 0$ represent the bit-length of challenge set $\mathcal{X}$, i.e., $\mathcal{X} = [0, 2^t - 1]$. Let ε be the statistical parameter.

- **Input:** The common input is $(B, \{C_i\}_{i\in[\ell]})$, Paillier public key N and Ring-Pedersen parameter $pp = (\widetilde{N}, h_1, h_2)$ generated by $\mathcal{V}_{\mathsf{enc}}$. $\mathcal{P}_{\mathsf{enc}}$ holds the witness $(m_i \in [0, B], \rho_i)_{i=1}^{\ell}$.

- The Σ-protocol $\Sigma_{\mathsf{enc}}[\ell]$ for $\mathcal{R}_{\mathsf{enc}}[\ell]$ is described as follows.
  1) $\mathcal{P}_{\mathsf{enc}} \to \mathcal{V}_{\mathsf{enc}}$: $\mathcal{P}_{\mathsf{enc}}$ picks

$$\begin{cases} m_0 \leftarrow\$ [0, 2^{\varepsilon+t}B], \rho_0 \leftarrow\$ \mathbb{Z}_N^* \\ r_0 \leftarrow\$ [0, 2^{\varepsilon+t}B\widetilde{N}], r_i \leftarrow\$ [0, B\widetilde{N}] \; \forall i \in [\ell], \end{cases}$$

  and computes $\begin{cases} C_0 = (1+N)^{m_0} \cdot (\rho_0)^N \pmod{N^2} \\ P_i = h_1^{m_i} h_2^{r_i} \pmod{\widetilde{N}} \; \forall i \in [0,\ell]. \end{cases}$ $\mathcal{P}_{\mathsf{enc}}$ sends $(C_0, \{P_i\}_{i\in[0,\ell]})$ to $\mathcal{V}_{\mathsf{enc}}$.
  2) $\mathcal{V}_{\mathsf{enc}} \to \mathcal{P}_{\mathsf{enc}}$: $\mathcal{V}_{\mathsf{enc}}$ picks $\mathbf{e} = (e_1, \cdots, e_\ell) \leftarrow\$ ([0, 2^t - 1])^\ell$ uniformly at random and sends it to $\mathcal{P}_{\mathsf{enc}}$.
  3) $\mathcal{P}_{\mathsf{enc}} \to \mathcal{V}_{\mathsf{enc}}$: $\mathcal{P}_{\mathsf{enc}}$ computes $\begin{cases} m^* = m_0 + \sum_{i\in[\ell]} e_i m_i \\ r^* = r_0 + \sum_{i\in[\ell]} e_i r_i \\ \rho^* = \rho_0 \cdot \prod_{i\in[\ell]} \rho_i^{e_i} \pmod{N}. \end{cases}$ $\mathcal{P}_{\mathsf{enc}}$ sends (m^*, r^*, ρ^*) to $\mathcal{V}_{\mathsf{enc}}$.
- **Verification:** $\mathcal{V}_{\mathsf{enc}}$ receives $(C_0, \{P_i\}_{i\in[0,\ell]}, m^*, r^*, \rho^*)$ from $\mathcal{P}_{\mathsf{enc}}$, and accepts if $m^* \in [0, 2^{\varepsilon+t}B]$ and the following equations hold:

$$\begin{cases} C_0 \cdot \prod_{i\in[\ell]} C_i^{e_i} \stackrel{?}{=} (1+N)^{m^*} \cdot (\rho^*)^N \pmod{N^2} \\ P_0 \cdot \prod_{i\in[\ell]} P_i^{e_i} \stackrel{?}{=} h_1^{m^*} h_2^{r^*} \pmod{\widetilde{N}}. \end{cases} \tag{7}$$

  If the verification succeeds, $\mathcal{V}_{\mathsf{enc}}$ is convinced that $m_i \in [-2^{\varepsilon+t}B, \, 2^{\varepsilon+t}B]$ holds for each $i \in [\ell]$.

Theorem 1 (ZKAoK for $\mathcal{R}_{enc}[\ell]$). *Let $\ell \leq \mathsf{poly}(\lambda)$ be the number of ciphertexts we considered. Let $\varepsilon, t \geq \lambda$ be chosen according to the security level. Then the proposed protocol $\Sigma_{\mathsf{enc}}[\ell]$ for relation $\mathcal{R}_{\mathsf{enc}}[\ell]$ has completeness and honest verifier zero-knowledge (HVZK). Moreover, it has computational witness-extended emulation under the strong RSA assumption.*

Due to the space limitation, the proof is presented in Appendix B.2.

Batch Proof $\Sigma_{\mathsf{aff}}[\ell]$ for Language $\mathcal{R}_{\mathsf{aff}}[\ell]$. We give a Σ-protocol for $\mathcal{R}_{\mathsf{aff}}[\ell]$, in which the prover $\mathcal{P}_{\mathsf{aff}}$ claims that it knows $m_i \in [0, B_1], \bar{m}_i \in [0, B_2]$ in range, such that the two Paillier ciphertexts C_i, D_i satisfy the affine operation, i.e., $D_i = C_i^{m_i}(1+N)^{\bar{m}_i}\rho_i^N \pmod{N^2}$ for some $\rho_i \in \mathbb{Z}_N^*$, for each $i \in [\ell]$.

- **Input:** The common input is $(B_1, B_2, \{C_i, D_i\}_{i\in[\ell]})$, Paillier public key N and Ring-Pedersen parameter $pp = (\widetilde{N}, h_1, h_2)$. $\mathcal{P}_{\mathsf{aff}}$ holds the witness $\{m_i \in [0, B_1], \bar{m}_i \in [0, B_2], \rho_i\}_{i\in[\ell]}$.
- The Σ-protocol $\Sigma_{\mathsf{aff}}[\ell]$ for $\mathcal{R}_{\mathsf{aff}}[\ell]$ is described as follows.
  1) $\mathcal{P}_{\mathsf{aff}} \to \mathcal{V}_{\mathsf{aff}}$: $\mathcal{P}_{\mathsf{aff}}$ picks

$$\begin{cases} r_i \leftarrow\$ [0, B_1\widetilde{N}], \bar{r}_i \leftarrow\$ [0, B_2\widetilde{N}] \; \forall i \in [\ell] \\ m_i' \leftarrow\$ [0, 2^{\varepsilon+t}B_1], r_i' \leftarrow\$ [0, 2^{\varepsilon+t}B_1\widetilde{N}] \; \forall i \in [\ell] \\ \bar{m}_0 \leftarrow\$ [0, 2^{\varepsilon+t}B_2], \rho_0 \leftarrow\$ \mathbb{Z}_N^*, \bar{r}_0 \leftarrow\$ [0, 2^{\varepsilon+t}B_2\widetilde{N}], \end{cases}$$

  and computes

$$\begin{cases} D_0 = \prod_{i\in[\ell]} C_i^{m_i'} \cdot (1+N)^{\bar{m}_0} \cdot (\rho_0)^N \pmod{N^2} \\ P_i = h_1^{m_i} h_2^{r_i}, \quad P_i' = h_1^{m_i'} h_2^{r_i'} \pmod{\widetilde{N}} \; \forall i \in [\ell] \\ \bar{P}_0 = h_1^{\bar{m}_0} h_2^{\bar{r}_0}, \quad \bar{P}_i = h_1^{\bar{m}_i} h_2^{\bar{r}_i} \pmod{\widetilde{N}} \; \forall i \in [\ell]. \end{cases}$$

  $\mathcal{P}_{\mathsf{aff}}$ sends $(D_0, \bar{P}_0, \{P_i, P_i', \bar{P}_i\}_{i\in[\ell]})$ to $\mathcal{V}_{\mathsf{aff}}$.
  2) $\mathcal{V}_{\mathsf{aff}} \to \mathcal{P}_{\mathsf{aff}}$: $\mathcal{V}_{\mathsf{aff}}$ picks $\mathbf{e} = (e_1, \cdots, e_\ell) \leftarrow\$ ([0, 2^t - 1])^\ell$ uniformly at random and sends it to $\mathcal{P}_{\mathsf{aff}}$.
  3) $\mathcal{P}_{\mathsf{aff}} \to \mathcal{V}_{\mathsf{aff}}$: $\mathcal{P}_{\mathsf{aff}}$ computes

$$\begin{cases} m_i^* = m_i' + e_i m_i, r_i^* = r_i' + e_i r_i \; \forall i \in [\ell] \\ \bar{m}^* = \bar{m}_0 + \sum_{i\in[\ell]} e_i \bar{m}_i, \bar{r}^* = \bar{r}_0 + \sum_{i\in[\ell]} e_i \bar{r}_i \\ \rho^* = \rho_0 \cdot \prod_{i\in[\ell]} \rho_i^{e_i} \pmod{N}. \end{cases}$$

  $\mathcal{P}_{\mathsf{aff}}$ sends $(\{m_i^*, r_i^*\}_{i\in[\ell]}, \bar{m}^*, \bar{r}^*, \rho^*)$ to $\mathcal{V}_{\mathsf{aff}}$.

- **Verification:** $\mathcal{V}_{\mathsf{aff}}$ receives from $\mathcal{P}_{\mathsf{aff}}$ the proof

$$\Pi = (D_0, \bar{P}_0, \{P_i, P_i', \bar{P}_i\}_{i\in[\ell]}, \{m_i^*, r_i^*\}_{i\in[\ell]}, \bar{m}^*, \bar{r}^*, \rho^*),$$

and accepts if $m_i^* \in [0, 2^{\varepsilon+t}B_1], \forall i \in [\ell]$, $\bar{m}^* \in [0, 2^{\varepsilon+t}B_2]$ and the following equations hold:

$$\begin{cases} D_0 \prod_{i\in[\ell]} D_i^{e_i} \stackrel{?}{=} \prod_{i\in[\ell]} C_i^{m_i^*}(1+N)^{\bar{m}^*}(\rho^*)^N \bmod N^2 \\ P_i' \cdot P_i^{e_i} \stackrel{?}{=} h_1^{m_i^*} h_2^{r_i^*} \bmod \widetilde{N}\ \forall i \in [\ell] \\ \bar{P}_0 \cdot \prod_{i\in[\ell]} \bar{P}_i^{e_i} \stackrel{?}{=} h_1^{\bar{m}^*} h_2^{\bar{r}^*} \bmod \widetilde{N}. \end{cases} \quad (8)$$

If the verification succeeds, the verifier $\mathcal{V}_{\mathsf{aff}}$ is convinced that $m_i \in [-2^{\varepsilon+t}B_1, 2^{\varepsilon+t}B_1]$ and $\bar{m}_i \in [-2^{\varepsilon+t}B_2, 2^{\varepsilon+t}B_2]$ hold for all $i \in [\ell]$.

Theorem 2 (ZKAoK for $\mathcal{R}_{\mathsf{aff}}[\ell]$). *Let $\ell \leq \mathsf{poly}(\lambda)$ be the number of ciphertexts we considered. Let $\varepsilon, t \geq \lambda$ be chosen according to the security level. Then the proposed protocol $\Sigma_{\mathsf{aff}}[\ell]$ for relation $\mathcal{R}_{\mathsf{aff}}[\ell]$ has completeness and HVZK. Moreover, it has computational witness-extended emulation under the strong RSA assumptions.*

Due to the space limitation, the proof is presented in Appendix B.3.

Extensions and Cost Comparisons. The above batch range proofs can be extended to other languages of Subsect. 4.1, including $\mathcal{R}_{\mathsf{log}^*}[\ell]$, $\mathcal{R}_{\mathsf{enc\text{-}elg}}[\ell]$, $\mathcal{R}_{\mathsf{afflog}}[\ell]$, $\mathcal{R}_{\mathsf{aff\text{-}g}}[\ell]$ and $\mathcal{R}_{\mathsf{aff\text{-}p}}[\ell]$. Due to the space limitation, we place the extensions in Appendix B.4.

In Table 2, we show the theoretical costs of the existing range-related ZK proofs for Paillier, in which the data is derived from [24, Table 2] as well as our batched variants. For all ZK proofs of Paillier used in threshold ECDSA, our batching technique reduces the proof size, Prover and Verifier computations, with a portion of the overheads being amortized. Taking $\Sigma_{\mathsf{enc}}[\ell]$ as an instance, more than 5/6 proof size and more than 1/2 computation can be amortized over ℓ instances, that is the amortized size is about 1/6 and the computational overhead is approximately 1/2 of the original data when considering $\ell \geq 10$.

Table 2: **Cost comparisons of the existing range-related ZK proofs in [24, Table 2] and our batched ones for Paillier (with gray row-color).** Let ℓ be the number of aggregated statements to be proven. $\boldsymbol{\mu} = \log N, \boldsymbol{\kappa} = \log q$ correspond to the message size of elements in $\mathbb{Z}_N$ and EC group $\mathbb{G}$, respectively. $\mathbf{N}^2, \mathbf{N}, \mathbf{G}$ denote computing exponentiation over the rings $\mathbb{Z}_{N^2}, \mathbb{Z}_N$ and the EC group $\mathbb{G}$, respectively.

ZK-Proof	Proof Size		Prover Computation			Verifier Computation		
	$\boldsymbol{\mu}$	$\boldsymbol{\kappa}$	$\mathbf{N}^2$	$\mathbf{N}$	$\mathbf{G}$	$\mathbf{N}^2$	$\mathbf{N}$	$\mathbf{G}$
Σ_{enc}	6	6	1	5	0	2	3	0
$\Sigma_{\mathsf{enc}}[\ell]$	$1+5/\ell$	$6/\ell$	$1/\ell$	$3+2/\ell$	0	$1+1/\ell$	$1+2/\ell$	0
Σ_{log^*}	6	7	1	5	1	2	3	2
$\Sigma_{\mathsf{log}^*}[\ell]$	$1+5/\ell$	$7/\ell$	$1/\ell$	$3+2/\ell$	$1/\ell$	$1+1/\ell$	$1+2/\ell$	$1+1/\ell$
$\Sigma_{\mathsf{enc\text{-}elg}}$	6	9	1	5	3	2	3	5
$\Sigma_{\mathsf{enc\text{-}elg}}[\ell]$	$1+5/\ell$	$9/\ell$	$1/\ell$	$3+2/\ell$	$3/\ell$	$1+1/\ell$	$1+2/\ell$	$2+3/\ell$
Σ_{aff}	9	16	2	9	0	3	6	0
$\Sigma_{\mathsf{aff}}[\ell]$	$4+5/\ell$	$6+10/\ell$	$1+1/\ell$	$7+2/\ell$	0	$2+1/\ell$	$4+2/\ell$	0
Σ_{afflog}	9	17	2	9	1	3	6	2
$\Sigma_{\mathsf{afflog}}[\ell]$	$4+5/\ell$	$7+10/\ell$	$1+1/\ell$	$7+2/\ell$	1	$2+1/\ell$	$4+2/\ell$	2
$\Sigma_{\mathsf{aff\text{-}g}}$	12	17	3	10	1	5	6	2
$\Sigma_{\mathsf{aff\text{-}g}}[\ell]$	$4+8/\ell$	$7+10/\ell$	$1+2/\ell$	$8+2/\ell$	1	$3+2/\ell$	$4+2/\ell$	2
$\Sigma_{\mathsf{aff\text{-}p}}$	15	16	4	11	0	7	6	0
$\Sigma_{\mathsf{aff\text{-}p}}[\ell]$	$7+8/\ell$	$6+10/\ell$	$2+2/\ell$	$9+2/\ell$	0	$5+2/\ell$	$4+2/\ell$	0

5 Batch Range Proofs for JL Encryption

In this section, we focus on the batch range proofs for JL encryption. It is worth noting that JL-based MtA involves two range-related relations [61]. As their batched variants, we care about the following two relations for the purpose of batch JL MtA.

Let $pk_{\mathsf{J}} = (N_{\mathsf{J}}, h, y, k)$ be the public key of the modified JL encryption, $pp_{\mathsf{J}} = (\widetilde{N}_{\mathsf{J}}, \tilde{h}, \tilde{y}, k)$ be the public parameter of the JL commitment. Let $\ell \in \mathbb{N}$ denote the number of ciphertext-commitment pairs.

Table 3: **Cost comparisons of the existing range-related ZK proofs in [61] and our batched ones for JL (with gray row-color).**

ZK-Proof	Proof Size		Prover Comp.		Verifier Comp.	
	$\boldsymbol{\mu}$	$\boldsymbol{\kappa}$	**N**	**G**	**N**	**G**
Σ_{equ}	4	7	4	0	6	0
Σ_{equ}-vector	$2+2/\ell$	$5+2/\ell$	$3+1/\ell$	0	$4+2/\ell$	0
$\Sigma_{\mathsf{equ}}[\ell]$	$4/\ell$	$7/\ell$	$4/\ell$	0	$2+4/\ell$	0
$\Sigma_{\mathsf{JL_{aff}}}$	2	13	3	1	4	2
$\Sigma_{\mathsf{JL_{aff}}}[\ell]$	$2/\ell$	$4+9/\ell$	$3/\ell$	1	$2+2/\ell$	2

For ℓ pairs $(C_i, P_i) \in \mathbb{Z}_{N_{\mathsf{J}}} \times \mathbb{Z}_{\widetilde{N}_{\mathsf{J}}}$, we consider the relation

$$\mathcal{R}_{\mathsf{equ}}[\ell] = \left\{ \begin{array}{c} ((B, \{C_i, P_i\}_{i\in[\ell]}), \\ \{m_i\}_{i\in[\ell]}) \end{array} \middle| \begin{array}{c} \forall\ i \in [\ell], m_i \in [0, B] \\ C_i = y^{m_i} \cdot h^{\rho_i} \bmod N_{\mathsf{J}} \\ P_i = \tilde{y}^{2^k m_i} \cdot \tilde{h}^{2^k r_i} \bmod \widetilde{N}_{\mathsf{J}} \end{array} \right\}.$$

Let $(\mathbb{G}, g, q)$ denote the group-generator-order tuple associated with the curve of ECDSA signatures. For ℓ pairs of JL ciphertexts (C_i, D_i)
$\in \mathbb{Z}^2_{N_{\mathsf{J}}}$ and ℓ group elements $X_i \in \mathbb{G}$, we also consider the relation

$$\mathcal{R}_{\mathsf{JL_{aff}}}[\ell] = \left\{ \begin{array}{c} ((B_1, B_2, \\ \{X_i, C_i, D_i\}_{i\in[\ell]}), \\ \{m_i, \bar{m}_i, \rho_i\}_{i\in[\ell]}) \end{array} \middle| \begin{array}{c} \forall\ i \in [\ell], m_i \in [0, B_1], \bar{m}_i \in [0, B_2] \\ D_i = C_i^{m_i} \cdot y^{\bar{m}_i} h^{\rho_i} \bmod N_{\mathsf{J}} \\ X_i = g^{m_i} \end{array} \right\}.$$

The constructions and security analysis of ZK proofs for JL are placed in Appendix C. In Table 3, we compare the theoretical costs of prior JL proofs and our batched variants. Specifically, the proof Σ_{equ}-vector proposed by [61] is also a batched version of proof Σ_{equ}, and compared with it, our scheme has better performance in both proof size and computations.

6 Applications in Threshold ECDSA

Building upon existing constructions of multi-party threshold ECDSA [38,51,23,32,33], we outline the design framework of such schemes by employing the two-party MtA as a fundamental component.

Given $m \in \mathbb{Z}_q$ as a hashed message to be signed, an ECDSA signature involves computing $\sigma = k^{-1} \cdot (m + r \cdot x)$, with the signing key x, a secret nonce k and a public nonce r. With another secret nonce γ, the signing equation can be rewritten as

$$\sigma = \frac{\gamma(m + rx)}{k\gamma}.$$

For designing multi-party threshold ECDSA, the parties secretly share the values x, k, γ, with each party holding an additive share (x_i, k_i, γ_i). In the presigning phase, their main goal is securely splitting $k\gamma$ and $x\gamma$ into additive shares with the help of MtA functionality, i.e., generating $(k\gamma)_i, (x\gamma)_i$ for each party s.t. $k\gamma = \sum_i (k\gamma)_i, x\gamma = \sum_i (x\gamma)_i$. Based on this, the online phase is extremely simple, since an ECDSA signature can be assembled easily with each party's revealed shares $(k\gamma)_i$ and $m\gamma_i + r(x\gamma)_i$. The illustration of the above construction framework is depicted in Figure 5 of Appendix E.

Table 4: **Cost comparisons of MtA under various security levels $\lambda = 128, 192, 256$.**

	MtA Schemes	Communication (KB)				Computation (ms)			
		$\ell = 1$	$\ell = 10$	$\ell = 50$	$\ell = 100$	$\ell = 1$	$\ell = 10$	$\ell = 50$	$\ell = 100$
$\lambda = 128$	Paillier [38,51]	7.81	78.12	390.63	781.25	250	2556	12653	24583
	Our Batch Paillier	7.81	39.88	182.38	360.5	260	1453	6825	13625
	JL [61]	9.03	90.31	451.56	903.13	1280	12832	64180	128325
	Batch JL [61]	9.03	64.44	310.69	618.5	1280	9332	46678	93246
	Our Batch JL	9.03	35.47	152.97	299.84	1280	5269	22561	44142
$\lambda = 192$	Paillier [38,51]	18.77	187.68	938.38	1876.75	3366	33944	168355	336710
	Our Batch Paillier	18.77	97.23	445.93	881.8	3394	19725	91013	180336
	JL [61]	13.55	135.47	677.34	1354.69	7272	67931	363807	727985
	Batch JL [61]	13.55	96.66	466.03	927.75	7272	49419	264587	529443
	Our Batch JL	13.55	53.20	229.45	449.77	7272	29141	132293	265397
$\lambda = 256$	Paillier [38,51]	37	370	1850	3700	24740	241360	1206811	2413599
	Our Batch Paillier	37	192.25	882.25	1744.75	24760	139989	651677	1279207
	JL [61]	18.06	180.63	903.13	1806.25	14639	145638	728190	1456383
	Batch JL [61]	18.06	128.88	621.38	1237	14639	106671	529668	1059188
	Our Batch JL	18.06	70.94	305.94	599.69	14639	53718	265585	529615

Taking $x\gamma$ as an example, we show how to securely split it into additive shares among all participating parties with two-party MtA as a tool. Assume the number of signing parties $\{\mathcal{P}_i\}$ is n. Denote a call of two-party MtA as $\mathsf{MtA}(a; b) \to (\alpha; \beta)$, where one party takes a as input and obtains α as output, and the other party takes b as input and obtains β as output. The values a, b, α, β are all secret, cannot be leaked to the other party, s.t. $\alpha + \beta = ab$. For n signatories of threshold ECDSA, the task is the conversion of multiplication into addition, in a form of

$$(x_1 + \cdots + x_n) \cdot (\gamma_1 + \cdots + \gamma_n) \to (x\gamma)_1 + \cdots + (x\gamma)_n.$$

It can be solved via two-party MtA as follows: each pair of parties $\mathcal{P}_i, \mathcal{P}_j$ invoke two-party MtAs on the cross-terms $x_i\gamma_j, x_j\gamma_i$, i.e.,

$$\mathsf{MtA}(x_i; \gamma_j) \to (\alpha_{i,j}; \beta_{j,i}), \quad \text{and} \quad \mathsf{MtA}(\gamma_i; x_j) \to (\beta_{i,j}; \alpha_{j,i}).$$

Finally, it yields the desired n-party multiplication-to-addition, with $\mathcal{P}_i$ holding the additive share $(x\gamma)_i = x_i\gamma_i + \sum_{j \neq i}(\alpha_{i,j} + \beta_{i,j}) \mod q$ for $x\gamma$. Similar operations for generating $(k\gamma)_i$ can be done simultaneously.

Overall, in multi-party threshold ECDSA, each pair of participants needs to invoke the two-party MtA protocol in the presigning phase. It is clear that MtA protocol is a core building block, and thus an efficient batch MtA protocol will be the key to improving the efficiency of threshold ECDSA.

In the following parts, we first present our batch MtA constructions, based on our batch range proofs. Then we demonstrate the improvement of our scheme compared with the prior works, taking UC-secure CGGMP20 [23,24], JL-based XAL$^+$23 [61] and round-minimal DKLs24 [33] as three typical examples.

6.1 Batch MtA Protocols

We first show batch Paillier-based MtA, which runs between Alice and Bob, and it can be easily extended to a JL-based variant. Indeed, in many existing threshold ECDSA [38,23,24], MtA with checking is required to prevent malicious parties from contributing incorrect inputs. Thus we also give the method to realize checking.

Batch Paillier-based MtA Let (pk_B, sk_B) be Bob's key pair of Paillier encryption. Alice and Bob invoke the following protocol with their inputs $\{a_i \in \mathbb{Z}_q\}_{i\in[\ell]}$ and $\{b_i \in \mathbb{Z}_q\}_{i\in[\ell]}$, and receive $\{\alpha_i\}_{i\in[\ell]}, \{\beta_i\}_{i\in[\ell]}$ respectively s.t. $\alpha_i + \beta_i = a_i \cdot b_i \mod q$ for each $i \in [\ell]$.

1) Bob generates $C_i \leftarrow \mathsf{Enc}(pk_B, b_i)$ for each $i \in [\ell]$, then computes the batch proof $\Sigma_{\mathsf{enc}}[\ell]$ for proving each plaintext b_i of C_i is within $\mathbb{Z}_q$, which may be replaced by $\Sigma_{\mathsf{log}^*}[\ell]$ or $\Sigma_{\mathsf{enc\text{-}elg}}[\ell]$ for checking requirement, e.g., [24]. Bob sends $(\{C_i\}_{i\in[\ell]}, \Sigma_{\mathsf{enc}}[\ell])$ to Alice.
2) Alice verifies the batch proof $\Sigma_{\mathsf{enc}}[\ell]$. Then she picks $\alpha_i \leftarrow\!\!\$\ [0, q^5]$ and computes $D_i = C_i^{a_i} \cdot \mathsf{Enc}(pk_B, \alpha_i)$ for each $i \in [\ell]$. Finally Alice generates the batch proof $\Sigma_{\mathsf{aff}}[\ell]$ for proving $a_i \in \mathbb{Z}_q$ and $\alpha_i \in \mathbb{Z}_{q^5}$ hold of each D_i, that may be replaced by $\Sigma_{\mathsf{afflog}}[\ell], \Sigma_{\mathsf{aff\text{-}g}}[\ell]$ or $\Sigma_{\mathsf{aff\text{-}p}}[\ell]$ for checking, e.g., [24]. Alice sends $(\{D_i\}_{i\in[\ell]}, \Sigma_{\mathsf{aff}}[\ell])$ to Bob, and outputs $-\alpha_i \bmod q,\ \forall i \in [\ell]$.
3) Bob verifies the proof $\Sigma_{\mathsf{aff}}[\ell]$ and generates $\beta_i \leftarrow \mathsf{Dec}(sk_B, D_i)$. Finally, Bob outputs $\beta_i \bmod q,\ \forall i \in [\ell]$.

The illustration of the above protocol is shown in Figure 1.

Batch JL-based MtA Replacing the Paillier encryption with JL encryption and utilizing our batch range proofs for languages $\mathcal{R}_{\mathsf{equ}}[\ell], \mathcal{R}_{\mathsf{JL_{aff}}}[\ell]$ from Appendix C, we can construct another batch MtA protocol from JL. Due to space constraints, we defer the construction and illustration (cf. Figure 3) of our batch JL MtA to Appendix D.1.

Benchmarking Results In Table 4, we give comparisons of Paillier-based MtA and our batched one when computing $\ell = 1, 10, 50, 100$ instances, as well as comparisons of JL-based MtA, batched variant proposed by [61] and our batched one. About the batched construction of [61], we will present more details in Subsect. 6.3.

Experimental Environment. Our benchmark is done using C language on CentOS Linux release 7.9.2009 (Core) with Intel(R) Xeon(R) CPU E5-2682 v4 @ 2.50GHz and 16GB of RAM. We utilize the BIGNUM type and relevant functions provided by OpenSSL to implement optimized Paillier encryption [48] and modified JL encryption [61].

Parameters. We benchmark the implementations under three parameter settings with security parameter $\lambda = 128, 192, 256$ respectively. For Paillier modulus N, it has $\log N = 3072$, 7680, 15360 bits to achieve the three security levels respectively, as recommended by NIST [9]. For the EC element bit-length, it has $\log q = 256$, 384 and 512 respectively when $\lambda = 128$, 192 and 256. We use SHA256, SHA384, SHA512 to instantiate hash functions for $\lambda = 128, 192, 256$ respectively. For JL parameters, we set $(k, \log N_{\mathsf{J}}) = (1792, 7680)$, $(2688, 11520)$ and $(3584, 15360)$ respectively when $\lambda = 128, 192, 256$.

In the (batch) range-related ZK proofs, we set $\varepsilon = t = \log q$, and the bounds B, B_1, B_2 are respectively $B = B_1 = q$, $B_2 = q^5$. These parameter settings are the same as those of [38,51,24] for fair comparisons. The JL modulus N_{J} is set according to the requirement $\varepsilon + t + \log B_2 \leq k \leq 1/4 \log N_{\mathsf{J}} - \lambda$ [15,61]. We set larger JL modulus than [61] because we have larger statistical parameter ε and soundness parameter t. For instance, $\varepsilon = t = 256$ for $\lambda = 128$ but $\varepsilon = t = 40$ in [61].

Comparison Results. When $\lambda = 128$, our batch Paillier improves Paillier-based MtA in bandwidth by a factor of 1.96 to 2.17, in computation by a factor of 1.75 to 1.85 when $\ell \geq 10$. Our batch JL improves JL-based MtA in bandwidth by a factor of 2.55 to 3.01, in computation by a factor of 2.44 to 2.91 when $\ell \geq 10$. Moreover, our batch JL improves Xue et al's batch JL [61] in bandwidth by a factor of 1.8 to 2.06, in computation by a factor of 1.8 to 2.11 when $\ell \geq 10$. When $\lambda = 192$ or 256, our batch schemes enjoy similar improving factors as those of $\lambda = 128$.

In conclusion, our batch Paillier improves both the computational and communicational efficiency by about 2×, our batch JL improves by about 3×. Compared with the existing batch JL-based MtA [61], our batch technique improves by about 2×.

6.2 Improvement on UC-secure CGGMP20 [24]

Canetti et al. [24] (an extension of [23]) proposed the first non-interactive threshold ECDSA scheme with presigning / online mode, utilizing a variant with checking of Paillier-based MtA from [38,51]. More precisely, they presented three versions of presigning, we call them Presigning - V1, V2 and V3 respectively. The V1 and V2 versions are proven secure in the universally composable (UC) framework [22], while the V3 version is lightweight and proven to be UC-secure when only logarithmically many signatures are generated concurrently.

For each Paillier range proof utilized in the three versions, we have proposed the batched construction in Subsect. 4.2. By utilizing them to realize batch presigning, the communicational and computational costs are both amortized over multiple pre-signatures. In Table 5, we compare the theoretical costs of the existing single presigning and our batched variant for three versions. Specifically, the data of single

Table 5: **Theoretical cost comparisons of the prior presigning protocols in [24, Table 1] and our batched variants (with gray row-color).** Let ℓ be the number of aggregated presignings, and n be the number of participating parties. $\boldsymbol{\mu} = \log N, \boldsymbol{\kappa} = \log q$. $\mathbf{N}^2, \mathbf{N}, \mathbf{G}$ denote computing exponentiation over $\mathbb{Z}_{N^2}, \mathbb{Z}_N, \mathbb{G}$, respectively.

Schemes	Communication Cost		Computation Cost		
	$\boldsymbol{\mu}$	$\boldsymbol{\kappa}$	$\mathbf{N}^2$	$\mathbf{N}$	$\mathbf{G}$
Presigning - V1	$54n$	$57n$	$33n$	$56n$	$12n$
Batch Presigning - V1	$(23+31/\ell)n$	$(17+40/\ell)n$	$(19+14/\ell)n$	$(36+20/\ell)n$	$(8+4/\ell)n$
Presigning - V2	$51n$	$67n$	$33n$	$49n$	$29n$
Batch Presigning - V2	$(25+26/\ell)n$	$(31+36/\ell)n$	$(21+12/\ell)n$	$(33+16/\ell)n$	$(21+8/\ell)n$
Presigning - V3	$30n$	$67n$	$19n$	$38n$	$26n$
Batch Presigning - V3	$(15+15/\ell)n$	$(38+29/\ell)n$	$(13+6/\ell)n$	$(26+12/\ell)n$	$(20+6/\ell)n$

Table 6: **Cost comparisons of threshold ECDSA in the two-party case with $\lambda = 128$.**

Threshold ECDSA	Type	Communication (KB)			Computation (ms)		
		$\ell=1$	$\ell=10$	$\ell=100$	$\ell=1$	$\ell=10$	$\ell=100$
CGGMP20 - V1 [24]	Paillier	44.06	440.63	4406.25	1522	15132	151319
Batch CGGMP20 - V1	Batch Paillier	44.06	208.88	1857	1537	9533	90791
CGGMP20 - V2 [24]	Paillier	42.44	424.38	4243.75	1419	14189	141876
Batch CGGMP20 - V2	Batch Paillier	42.44	228.63	2090.5	1419	9790	93638
DKLs24 [33]	OT	99.4	994	9940	106.13	867.2	8671.13
DKLs24 with Paillier MtA	Paillier [38,51]	25.13	251.25	2512.5	912	9119	91107
	Batch Paillier	25.13	138.75	1275	922	5260	49462
DKLs24 with JL MtA	JL [61]	24.69	246.88	2468.75	3512	34769	344211
	Batch JL [61]	24.69	195.13	1899.5	3520	26946	259244
	Our Batch JL	24.69	97.81	829.06	3520	14296	124040

presigning for each version comes from [24, Table 1], and the amortized data for the corresponding batched variant is calculated based on our Table 2. It shows that our amortization approach significantly reduces the theoretical costs for all three presigning versions of [24]. At the same time, our batched constructions merely replace the range proofs in the original constructions, thus maintaining the UC security.

We benchmark the implementations of the two UC-secure versions (V1 and V2), as well as our corresponding batched variants under the parameter settings of $\lambda = 128$. Table 6 shows that our batched protocol improves V1 by a factor of 2.4 and 1.7, in bandwidth and computation respectively. And our batched V2 improves V2 by a factor of 2.1 and 1.5, in bandwidth and computation respectively, for the two-party case.

6.3 Improvement on JL-based XAL$^+$23 [61]

Xue et al. [61] constructed MtA protocol from JL encryption. And they explored the JL vector commitment to design a batch JL MtA. From Table 4 in Subsect. 6.1, we have compared their (batch) JL MtA and ours. The result is we achieve an approximate improvement factor of 3× compared to their JL MtA, and 2× compared to their batch JL MtA.

Furthermore, we also have an advantage in the setup phase. Their batch technique uses the vector commitment, which increases the setup costs, since more $\tilde{y}_i$s are needed to generate. Instead, our batch technique uses batch range proofs, without increasing the setup costs. Due to the limited space, we defer the comparisons of theoretical costs for both JL MtA (cf. Table 7) and setup phase (cf. Table 8) to Appendix D.2. Remark that the data of computational overhead for JL MtA in Table 1 and Table 7 is different, because we use Xue et al.'s counting method for Table 7, which will be explained in more details in Appendix D.2.

When implementing threshold ECDSA using their batch JL MtA and ours, Table 6 shows that our batch technique improves the bandwidth efficiency by a factor of 2.0 to 2.29, and the computational efficiency by a factor of 1.88 to 2.09 when $\ell \geq 10$, compared with their batch technique. Remark that we utilize the construction framework of threshold ECDSA derived from [33], which we will introduce in Subsect. 6.4.

6.4 Improvement on bandwidth efficiency of round-minimal DKLs24 [33]

To the best of our knowledge, threshold ECDSA from Doerner et al. [33] is currently round-minimal without honest-majority assumption. It is a three-round signing protocol, with the first two rounds dedicated to the presigning phase. In Appendix E, we present Figure 5 to show their basic framework. More precisely, they realized MtA via the OT-extension protocol of Roy [57], with base OTs supplied by the endemic OT protocol of Masny and Rindal [53]. It requires 49.7n KB total incoming communication for each party, where n is the number of participating parties. To improve the bandwidth efficiency, we replace its underlying MtA by our batch Paillier or JL MtA. By its modularity, the new threshold ECDSA is still round-minimal, and requires significantly less communicational cost.

We instantiate the DKLs24 framework with (batch) Paillier and JL MtAs. The benchmarks were taken over EC curve secp256k1 for 128 bits security. Latency is not taken into account, as these protocols have the same communication rounds. The benchmarking results are presented in Table 6. The comparisons of the JL-based constructions under DKLs24 framework have been analyzed in Subsect. 6.3. Compared with OT-based scheme, when $\ell = 100$, DKLs24 with our batch Paillier improves bandwidth efficiency by 7.8×, but is 5.7× slower in computation time. DKLs24 with our batch JL improves OT-based one by 12× in bandwidth, but is 14× slower in computation. Table 6 also shows that under DKLs24 framework, the construction with our batch Paillier MtA has about 1.8× lower bandwidth and computation overhead than that with the original Paillier MtA from [38,51]. As analyzed in Appendix E, the storage cost for each participant incurred by our batch technique is indeed very small (e.g., 12.5 KB for $\ell = 100$ and $\lambda = 128$).

ACKNOWLEDGMENTS

We would like to thank the reviewers for their valuable comments. We are deeply grateful to Yu Chen for the insightful discussions and suggestions. We also thank Haiyang Xue, Huanyu Ma and Hao Lei for their suggestions on the paper. Shuai Han was partially supported by National Natural Science Foundation of China (Grant No. 62372292) and the National Key R&D Program of China under Grant 2022YFB2701500.

References

1. Binance. `https://www.binance.com`.
2. BitGo. `https://www.bitgo.com`.
3. Coinbase. `https://www.coinbase.com`.
4. Fireblocks. `https://www.fireblocks.com`.
5. Zengo. `https://zengo.com`.
6. Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *SP*, pages 2554–2572, 2022.
7. Thomas Attema, Serge Fehr, and Nicolas Resch. Generalized special-sound interactive proofs and their knowledge soundness. In *TCC*, pages 424–454, 2023.
8. Niko Baric and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, pages 480–494, 1997.
9. Elaine Barker, William Barker, William Burr, W. Polk, and Miles Smid. Recommendation for key management: Part 1: General. *National Institute of Standard and Technology*, 2006.
10. Claudia Bartoli and Ignacio Cascudo. On sigma-protocols and (packed) black-box secret sharing schemes. In *PKC*, pages 426–457, 2024.
11. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, pages 420–432, 1991.
12. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS*, pages 390–399, 2006.
13. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73, 1993.
14. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. Aurora: Transparent succinct arguments for r1cs. In *EUROCRYPT*, pages 103–128, 2019.
15. Fabrice Benhamouda, Javier Herranz, Marc Joye, and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. *J. Cryptol.*, 30(2):519–549, 2017.
16. Dan Boneh, Rosario Gennaro, and Steven Goldfeder. Using level-1 homomorphic encryption to improve threshold DSA signatures for bitcoin wallet security. In *LATINCRYPT*, pages 352–377, 2017.
17. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT*, pages 327–357, 2016.
18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent ot extension and more. In *CRYPTO*, pages 489–518, 2019.
19. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-lpn. In *CRYPTO*, pages 387–416, 2020.
20. Luís TAN Brandão, Michael Davidson, Apostol Vassilev, et al. Nist roadmap toward criteria for threshold schemes for cryptographic primitives. *National Institute of Standard and Technology*, 2020.
21. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *SP*, pages 315–334, 2018.
22. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
23. Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *CCS*, pages 1769–1787, 2020.
24. Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. *IACR Cryptol. ePrint Arch.*, page 60, 2021.
25. Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In *PKC*, pages 266–296, 2020.
26. Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts proactive and adaptive security. *Theor. Comput. Sci.*, pages 78–104, 2023.
27. Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In *Cryptographers' Track at the RSA Conference*, pages 487–505, 2015.
28. Chaochao Chen, Jun Zhou, Li Wang, Xibin Wu, Wenjing Fang, Jin Tan, Lei Wang, Alex X. Liu, Hao Wang, and Cheng Hong. When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control. In *KDD*, pages 2652–2662, 2021.
29. William M Daley and Raymond G Kammer. Digital signature standard (DSS). *BOOZ-ALLEN AND HAMILTON INC MCLEAN VA*, 2000.
30. Yi Deng, Shunli Ma, Xinxuan Zhang, Hailong Wang, Xuyang Song, and Xiang Xie. Promise sigma-protocol: How to construct efficient threshold ECDSA from encryptions based on class groups. In *ASIACRYPT*, pages 557–586, 2021.
31. Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *SP*, pages 980–997, 2018.
32. Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *SP*, pages 1051–1066, 2019.

33. Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA in three rounds. In *SP*, 2024.
34. Jack Doerner, Yashvanth Kondi, Eysa Lee, Abhi Shelat, and LaKyah Tyner. Threshold BBS+ signatures for distributed anonymous credential issuance. In *SP*, pages 773–789, 2023.
35. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
36. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, pages 16–30, 1997.
37. Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. In *CCS*, pages 1187–1201, 2022.
38. Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *CCS*, pages 1179–1194, 2018.
39. Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *ACNS*, pages 156–174, 2016.
40. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In *EUROCRYPT*, pages 354–371, 1996.
41. Rosario Gennaro, Darren Leigh, Ravi Sundaram, and William S. Yerazunis. Batching schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In *ASIACRYPT*, pages 276–292, 2004.
42. B. Gong, W. Lau, M. Au, R. Yang, H. Xue, and L. Li. Efficient zero-knowledge arguments for paillier cryptosystem. In *SP*, 2024.
43. Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, pages 444–459, 2006.
44. Jens Groth and Yuval Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT*, pages 379–396, 2008.
45. Yotam Harchol, Ittai Abraham, and Benny Pinkas. Distributed ssh key management with proactive rsa threshold signatures. In *ACNS*, pages 22–43, 2018.
46. Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Wen-jie Lu, Cheng Hong, and Kui Ren. Ciphergpt: Secure two-party GPT inference. *IACR Cryptol. ePrint Arch.*, page 1147, 2023.
47. Marc Joye and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. pages 76–92, 2013.
48. Mads Jurik. Extensions to the paillier cryptosystem with applications to cryptological protocols. *PhD Thesis*, 2003.
49. Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology*, 16(3):143–184, 2003.
50. Yehuda Lindell. Fast secure two-party ECDSA signing. In *CRYPTO*, pages 613–644, 2017.
51. Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *CCS*, pages 1837–1854, 2018.
52. Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *SP*, pages 1348–1366, 2021.
53. Daniel Masny and Peter Rindal. Endemic oblivious transfer. In *CCS*, pages 309–326, 2019.
54. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.
55. Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.
56. Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In *EUROCRYPT*, pages 901–930, 2021.
57. Lawrence Roy. Softspokenot: Communication-computation tradeoffs in OT extension. *IACR Cryptol. ePrint Arch.*, page 192, 2022.
58. Victor Shoup. Practical threshold signatures. In *EUROCRYPT*, pages 207–220, 2000.
59. Dmytro Tymokhanov and Omer Shlomovits. Alpha-rays: Key extraction attacks on threshold ECDSA implementations. *IACR Cryptol. ePrint Arch.*, page 1621, 2021.
60. Harry W. H. Wong, Jack P. K. Ma, Hoover H. F. Yin, and Sherman S. M. Chow. Real threshold ECDSA. In *NDSS*, 2023.
61. Haiyang Xue, Man Ho Au, Mengling Liu, Kwan Yin Chan, Handong Cui, Xiang Xie, Tsz Hon Yuen, and Chengru Zhang. Efficient multiplicative-to-additive function from joye-libert cryptosystem and its application to threshold ECDSA. 2023.
62. Rupeng Yang, Man Ho Au, Zhenfei Zhang, Qiuliang Xu, Zuoxia Yu, and William Whyte. Efficient lattice-based zero-knowledge arguments with standard soundness: Construction and applications. In *CRYPTO*, pages 147–175, 2019.
63. Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *PODC*, pages 347–356, 2019.
64. Tsz Hon Yuen, Handong Cui, and Xiang Xie. Compact zero-knowledge proofs for threshold ECDSA with trustless setup. In *PKC*, pages 481–511, 2021.
65. Tsz Hon Yuen, Muhammed F Esgin, Joseph K Liu, Man Ho Au, and Zhimin Ding. Dualring: generic construction of ring signatures with efficient instantiations. In *CRYPTO*, pages 251–281, 2021.

A Supplementary Materials of Lemma 2

In this section, we first give an illustration of the AT-LIEC-Tree of our multi-dimension forking lemma in Appendix A.1, then provide the supplementary proofs of Lemma 2 in Appendix A.2.

A.1 The AT-LIEC-Tree of Lemma 2

To better illustrate the difference between the general forking lemma [12,17] and our multi-dimension forking lemma, Figure 2 shows the the AT-Tree (Tree of Accepting Transcripts) in the general forking lemma and the AT-LIEC-Tree (Tree of Accepting Transcripts with Linearly-Independent Extended Challenges) in our multi-dimension forking lemma.

The main difference with the general forking lemma [12,17] is that we require a certain number of challenges to be linearly independent vectors, rather than just distinct ones. Generally, for a protocol of $(2\mu+1)$ rounds with μ challenge vectors $\mathbf{e}_1, \cdots, \mathbf{e}_\mu$ and $(\mu+1)$ responses, our multi-dimension forking lemma enables the extracting of AT-LIEC-tree, where each node of depth i $(<\mu)$ in the tree has children that are associated with *linearly independent* i-th extended challenges $\mathbf{e}_i$, and each path in the tree denotes an accepting transcript. The extraction of such AT-LIEC-tree relies on the rewinding process we devised in Algorithm 1 and Algorithm 2. In the next subsection, we provide the careful analysis of its running time and successful probability.

A.2 Supplementary Proofs of Lemma 2

In this part, we analyze the expected running time of the emulator $\mathcal{E}$ proposed in Algorithm 1 and its advantage $\mathsf{Adv}^{ext}_{\mathcal{P}^*,\mathcal{E},\mathcal{A}}(\lambda)$.

Analysis of $\mathcal{E}$'s running time. Firstly, we analyze the running time of $\mathcal{E}$. For an algorithm $\mathcal{A}$, denote by $\mathbf{T}(\mathcal{A})$ the running time of $\mathcal{A}$, and denote by $\mathbb{E}(\mathbf{T}(\mathcal{A}))$ the expected running time of $\mathcal{A}$. Clearly, by Algorithm 1, we have $\mathbb{E}(\mathbf{T}(\mathcal{E})) \ \le\ \mathbb{E}(\mathbf{T}(\mathcal{T}_1)) + \mathbb{E}(\mathbf{T}(\chi)) + \mathsf{poly}(\lambda)$, where $\mathsf{poly}(\lambda)$ is some polynomial function in λ. Since χ is an efficient witness extraction algorithm, $\mathbb{E}(\mathbf{T}(\chi)) \le \mathsf{poly}(\lambda)$ for some polynomial function $\mathsf{poly}(\lambda)$. Thus,

$$\mathbb{E}(\mathbf{T}(\mathcal{E})) \ \le\ \mathbb{E}(\mathbf{T}(\mathcal{T}_1)) + \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda). \tag{9}$$

It remains to give an upper bound for $\mathbb{E}(\mathbf{T}(\mathcal{T}_1))$.

Fix $i \in [\mu]$, and fix $\mathbf{e}_1, ..., \mathbf{e}_{i-1}$. By Algorithm 2, for a freshly-chosen $\mathbf{e}_i \leftarrow\!\$\ (\mathcal{X})^{d_i}$, the i-th tree-finder $\mathcal{T}_i$ goes to lines 4-12 with the probability

$$\epsilon_i = \Pr[b = 1 | (tr, b, \mathsf{tree}) \leftarrow \mathcal{T}_{i+1}^{\langle \mathcal{P}^*(s), \mathcal{V}\rangle}(u, \mathbf{e}_1, ..., \mathbf{e}_{i-1}, \mathbf{e}_i, \mathsf{tree})], \tag{10}$$

while with the probability $(1-\epsilon_i)$, it directly goes to line 14 in Algorithm 2. In the former case, denote by $\#_i$ the number of times that $\mathcal{T}_i$ runs the while loops in lines 5-11, where $\mathcal{T}_i$ invokes $\mathcal{T}_{i+1}$ once each time. Then we have

$$\mathbb{E}(\mathbf{T}(\mathcal{T}_i)) \le (1 + \epsilon_i \cdot \mathbb{E}(\#_i)) \cdot \big(\mathbb{E}(\mathbf{T}(\mathcal{T}_{i+1})) + \mathsf{poly}(\lambda)\big),$$

for some polynomial function $\mathsf{poly}(\lambda)$.

Next, we give an upper bound for $\mathbb{E}(\#_i)$, the expected number of times that $\mathcal{T}_i$ runs the while loops in lines 5-11 in Algorithm 2. The loops begin with counter $\mathsf{ctr}_i = 1$, increment ctr_i if the condition $(b = 1)$ in line 10 holds, and terminate when $\mathsf{ctr}_i = n_i$. In other words, the while loops terminate when the counter ctr_i has been incremented $n_i - 1$ times. In each loop, ctr_i will be incremented if the condition $(b = 1)$ in line 10 holds, whose probability is exactly the ϵ_i in (10). Therefore, the expected number of times that $\mathcal{T}_i$ runs the while loops is $\mathbb{E}(\#_i) = (n_i - 1)/\epsilon_i$. We stress that the condition $((1, \mathbf{e}_i) \in \mathsf{span}(S_i))$ in line 8 does not determine whether or not ctr_i will be incremented. Then we have $\mathbb{E}(\mathbf{T}(\mathcal{T}_i)) \le n_i \cdot \big(\mathbb{E}(\mathbf{T}(\mathcal{T}_{i+1})) + \mathsf{poly}(\lambda)\big)$. By recursion,

$$\begin{aligned}\mathbb{E}(\mathbf{T}(\mathcal{T}_1)) &\le n_1 \cdot \big(\mathbb{E}(\mathbf{T}(\mathcal{T}_2)) + \mathsf{poly}(\lambda)\big)\\ &\le n_1 \cdot \big(n_2 \cdot \big(\mathbb{E}(\mathbf{T}(\mathcal{T}_3)) + \mathsf{poly}(\lambda)\big) + \mathsf{poly}(\lambda)\big)\\ &\le \cdots \le \textstyle\prod_{i=1}^{\mu} n_i \cdot \mathbb{E}(\mathbf{T}(\mathcal{T}_{\mu+1})) + \mu \cdot \prod_{i=1}^{\mu} n_i \cdot \mathsf{poly}(\lambda).\end{aligned}$$

Finally, by Algorithm 2, $\mathcal{T}_{\mu+1}$ runs the protocol $\langle \mathcal{P}^*(s), \mathcal{V}\rangle(u)$ once through oracle access and runs the efficient verification algorithm $\mathcal{V}(tr)$ once. Thus, $\mathbb{E}(\mathbf{T}(\mathcal{T}_{\mu+1})) \ \le\ \mathsf{poly}(\lambda)$, for some polynomial function $\mathsf{poly}(\lambda)$. So, we have

$$\mathbb{E}(\mathbf{T}(\mathcal{T}_1)) \le (\mu + 1) \cdot (\textstyle\prod_{i=1}^{\mu} n_i) \cdot \mathsf{poly}(\lambda). \tag{11}$$

Assuming that $\prod_{i=1}^{\mu} n_i \leq \mathsf{poly}(\lambda)$, $\mathbb{E}(\mathbf{T}(\mathcal{T}_1))$ is a polynomial function in λ. This, together with (9), shows that the proposed emulator $\mathcal{E}$ runs in expected polynomial time.

Analysis of $\mathcal{E}$'s advantage. Now we analyze the upper bound of $\mathsf{Adv}^{ext}_{\mathcal{P}^*,\mathcal{E},\mathcal{A}}(\lambda)$. The first tree-finder $\mathcal{T}_1$ outputs (tr, b, tree) with $b = 1$ so that tr is an accepting transcript if the very first set of challenges generated by all of the tree-finders produces an accepting transcript. This is exactly the probability that $\mathcal{P}^*$ successfully produces an accepting transcript in one run when interacting with $\mathcal{V}$. Note that the emulator $\mathcal{E}$ always outputs the tr output by $\mathcal{T}_1$ (no matter $\mathsf{is\text{-}AT\text{-}LIEC}$ is true or false). Therefore, we have

$$\begin{aligned}
&\Pr\left[\mathcal{A}(tr) = 1 \wedge tr \text{ is acc.} \,|tr \leftarrow \langle \mathcal{P}^*(s), \mathcal{V}\rangle(u)\right] \\
=&\Pr\left[\mathcal{A}(tr) = 1 \wedge tr \text{ is acc.} \,\middle|(tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u)\right], \text{ and} \\
&\Pr\left[\mathcal{A}(tr) = 1 \wedge tr \text{ is non-acc.} \,|tr \leftarrow \langle \mathcal{P}^*(s), \mathcal{V}\rangle(u)\right] \\
=&\Pr\left[\mathcal{A}(tr) = 1 \wedge tr \text{ is non-acc.} \,\middle|(tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u)\right].
\end{aligned}$$

Following (5), it implies that $\mathsf{Adv}^{ext}_{\mathcal{P}^*,\mathcal{E},\mathcal{A}}(\lambda)$ is equal to

$$\left|\begin{aligned}
&\Pr\left[\mathcal{A}(tr) = 1 \wedge tr \text{ is acc.} \,|tr \leftarrow \langle \mathcal{P}^*(s), \mathcal{V}\rangle(u)\right] \\
&+\Pr\left[\mathcal{A}(tr) = 1 \wedge tr \text{ is non-acc.} \,|tr \leftarrow \langle \mathcal{P}^*(s), \mathcal{V}\rangle(u)\right] \\
&-\Pr\left[\mathcal{A}(tr) = 1 \wedge tr \text{ is non-acc.} \,\middle|(tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u)\right] \\
&-\Pr\left[\begin{matrix}\mathcal{A}(tr) = 1 \wedge \\ (tr \text{ is acc. } \wedge (u, w) \in \mathcal{R})\end{matrix} \,\middle|(tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u)\right]
\end{aligned}\right|.$$

The above expression can be further simplified as

$$\begin{aligned}
&\Pr\left[\mathcal{A}(tr) = 1 \wedge tr \text{ is acc.} \,\middle|(tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u)\right] \\
&\cdot \left|1 - \Pr\left[(u, w) \in \mathcal{R} \,\middle|\, \begin{matrix}(tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u) \\ \wedge\ \mathcal{A}(tr) = 1 \ \wedge \ tr \text{ is acc.}\end{matrix}\right]\right|.
\end{aligned}$$

In the case of that tr is an accepting transcript (hence $b = 1$), for $i \in [\mu]$, $\mathcal{T}_i$ successfully collects n_i extended challenges $(1, \mathbf{e}_i)$ associated with a common predecessor $(\mathbf{e}_1, ..., \mathbf{e}_{i-1})$ (cf. lines 4-12 in Algorithm 2). Moreover, if the boolean variable $\mathsf{is\text{-}AT\text{-}LIEC}$ has never been set to false (i.e., $\mathsf{is\text{-}AT\text{-}LIEC}$ still equals true), these extended challenges are linearly independent. Consequently, in this case, the tree output by the first tree-finder $\mathcal{T}_1$ constitutes an $(n_1, ...n_\mu)$-AT-LIEC-tree. Then, the witness extraction algorithm χ always extracts a valid witness w from tree so that $(u, w) \in \mathcal{R}$. In formula, we have

$$\begin{aligned}
&\Pr\left[(u, w) \in \mathcal{R} \,\middle|\, \begin{matrix}(tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u) \\ \wedge\ \mathcal{A}(tr) = 1 \ \wedge \ tr \text{ is acc.}\end{matrix}\right] \\
=&\Pr\left[\mathsf{is\text{-}AT\text{-}LIEC} = \mathsf{true} \,\middle|\, \begin{matrix}(tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u) \\ \wedge\ \mathcal{A}(tr) = 1 \ \wedge \ tr \text{ is acc.}\end{matrix}\right].
\end{aligned}$$

Putting the above equations together, we get that

$$\begin{aligned}
\mathsf{Adv}^{ext}_{\mathcal{P}^*,\mathcal{E},\mathcal{A}}(\lambda) =& \Pr\left[\mathcal{A}(tr) = 1 \wedge tr \text{ is acc.} \,\middle|(tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u)\right] \\
&\cdot \Pr\left[\mathsf{is\text{-}AT\text{-}LIEC} = \mathsf{false} \,\middle|\, \begin{matrix}(tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u) \\ \wedge\ \mathcal{A}(tr) = 1 \ \wedge \ tr \text{ is acc.}\end{matrix}\right] \\
\leq& \Pr\left[tr \text{ is acc.} \wedge \mathsf{is\text{-}AT\text{-}LIEC} = \mathsf{false} \,\middle|(tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u)\right].
\end{aligned}$$

It remains to give an upper bound for this probability.

Note that the only possible way that $\mathsf{is\text{-}AT\text{-}LIEC}$ could be set to false is through the condition check $((1, \mathbf{e}_i) \in \mathsf{span}(S_i))$ in lines 8-9 in Algorithm 2. For a freshly-chosen $\mathbf{e}_i \leftarrow\!\!\$\ (\mathcal{X})^{d_i}$ in line 6, we estimate the probability $\Pr\left[(1, \mathbf{e}_i) \in \mathsf{span}(S_i)\right]$. Suppose that $S_i = \{(1, \mathbf{e}_i^{(\eta)})\}_{\eta \in [\mathsf{ctr}_i]}$, where the $(1, \mathbf{e}_i^{(\eta)})$'s are the extended challenges leading to accepting transcripts that has been collected so far. We could represent S_i by a matrix whose rows are the $(1, \mathbf{e}_i^{(\eta)})$'s, namely

$$S_i = \begin{pmatrix} 1 & e_{i,1}^{(1)} & \dots & e_{i,d_i}^{(1)} \\ 1 & e_{i,1}^{(2)} & \dots & e_{i,d_i}^{(2)} \\ \vdots & \vdots & & \vdots \\ 1 & e_{i,1}^{(\mathsf{ctr}_i)} & \dots & e_{i,d_i}^{(\mathsf{ctr}_i)} \end{pmatrix} \in (\mathbb{R})^{\mathsf{ctr}_i \times (1+d_i)}.$$

Then $\mathsf{span}(S_i)$ is the subspace generated by the rows of S_i. Denote by rk_i the rank of S_i. Clearly, $\mathsf{rk}_i \leq \mathsf{ctr}_i < n_i \leq 1 + d_i$. Without loss of generality, we assume that matrix S_i is in its row-reduced echelon form. (This means the rk_i leftmost columns of S_i are linearly independent.) Therefore, for each possible $(1, \mathbf{e}_{i,L}) = (1, e_{i,1}, ..., e_{i,\mathsf{rk}_i - 1})$ in $(\mathbb{R})^{\mathsf{rk}_i}$, there is exactly one $\mathbf{e}_{i,R} = (e_{i,\mathsf{rk}_i}, ..., e_{i,d_i})$ in $(\mathbb{R})^{d_i - \mathsf{rk}_i + 1}$, such that $(1, \mathbf{e}_i) = (1, e_{i,1}, ..., e_{i,\mathsf{rk}_i - 1}, e_{i,\mathsf{rk}_i}, ..., e_{i,d_i}) \in \mathsf{span}(S_i)$. In our case, the challenge $\mathbf{e}_i = (e_{i,1}, e_{i,2}, ..., e_{i,d_i})$ is uniformly chosen from the subset $(\mathcal{X})^{d_i}$ of $(\mathbb{R})^{d_i}$, thus for each possible $(1, \mathbf{e}_{i,L}) = (1, e_{i,1}, ..., e_{i,\mathsf{rk}_i - 1})$ in $\{1\} \times (\mathcal{X})^{\mathsf{rk}_i - 1}$, there is at most one $\mathbf{e}_{i,R} = (e_{i,\mathsf{rk}_i}, ..., e_{i,d_i})$ in $(\mathcal{X})^{d_i - \mathsf{rk}_i + 1}$, such that $(1, \mathbf{e}_i) = (1, e_{i,1}, ..., e_{i,\mathsf{rk}_i - 1}, e_{i,\mathsf{rk}_i}, ..., e_{i,d_i}) \in \mathsf{span}(S_i)$. This implies that

$$\Pr\left[(1, \mathbf{e}_i) \in \mathsf{span}(S_i)\right] \leq \frac{|\mathcal{X}|^{\mathsf{rk}_i - 1}}{|\mathcal{X}|^{d_i}} \leq \frac{1}{|\mathcal{X}|}, \tag{12}$$

where the second inequality is due to $\mathsf{rk}_i < 1 + d_i$.

Next, we count the number of times that the condition $((1, \mathbf{e}_i) \in \mathsf{span}(S_i))$ in lines 8-9 in Algorithm 2 might be checked. By (11), $\mathcal{T}_1$ runs in expected polynomial time, i.e., $\mathbb{E}(\mathbf{T}(\mathcal{T}_1)) \leq \widetilde{\mathsf{poly}}(\lambda)$ for polynomial function

$$\widetilde{\mathsf{poly}}(\lambda) = (\mu + 1) \cdot (\textstyle\prod_{i=1}^{\mu} n_i) \cdot \mathsf{poly}(\lambda). \tag{13}$$

Then for any $\kappa = \kappa(\lambda)$ (to be chosen later), by Markov inequality, it holds that

$$\Pr\left[\mathbf{T}(\mathcal{T}_1) \leq \kappa \cdot \widetilde{\mathsf{poly}}(\lambda) \,\middle|\, (tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u)\right] > 1 - 1/\kappa.$$

In the case of $\mathbf{T}(\mathcal{T}_1) \leq \kappa \cdot \widetilde{\mathsf{poly}}(\lambda)$, the number of times that the condition check could be made is at most $\kappa \cdot \widetilde{\mathsf{poly}}(\lambda)$, and by (12) the probability that is-AT-LIEC could be set to false is at most $1/|\mathcal{X}|$ each time. By a union bound, we have

$$\begin{aligned}&\Pr\left[\begin{array}{c} tr \text{ is acc.} \\ \wedge\ \textsf{is-AT-LIEC} = \textsf{false} \end{array} \middle| \begin{array}{c} (tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u) \\ \wedge\ \mathbf{T}(\mathcal{T}_1) \leq \kappa \cdot \widetilde{\mathsf{poly}}(\lambda) \end{array}\right] \\ &\leq \kappa \cdot \widetilde{\mathsf{poly}}(\lambda)/|\mathcal{X}|.\end{aligned}$$

Taking all things together, we obtain

$$\begin{aligned}&\mathsf{Adv}^{ext}_{\mathcal{P}^*,\mathcal{E},\mathcal{A}}(\lambda) \\ \leq &\Pr\left[tr \text{ is acc.} \wedge \textsf{is-AT-LIEC} = \textsf{false} \,\middle|\, (tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u)\right] \\ \leq &\Pr\left[\begin{array}{c} tr \text{ is acc.} \\ \wedge\ \textsf{is-AT-LIEC} = \textsf{false} \end{array} \middle| \begin{array}{c} (tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u) \\ \wedge\ \mathbf{T}(\mathcal{T}_1) \leq \kappa \cdot \widetilde{\mathsf{poly}}(\lambda) \end{array}\right] \\ &+ \Pr\left[\mathbf{T}(\mathcal{T}_1) > \kappa \cdot \widetilde{\mathsf{poly}}(\lambda) \,\middle|\, (tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(u)\right] \\ \leq &\kappa \cdot \widetilde{\mathsf{poly}}(\lambda)/|\mathcal{X}| + 1/\kappa.\end{aligned}$$

To conclude, we choose $\kappa = \sqrt{|\mathcal{X}|}$, then by (13), it yields

$$\begin{aligned}\mathsf{Adv}^{ext}_{\mathcal{P}^*,\mathcal{E},\mathcal{A}}(\lambda) \leq &(\widetilde{\mathsf{poly}}(\lambda) + 1)/\sqrt{|\mathcal{X}|} \\ = &(\mu + 1) \cdot (\textstyle\prod_{i=1}^{\mu} n_i) \cdot \mathsf{poly}(\lambda)/\sqrt{|\mathcal{X}|},\end{aligned}$$

which is negligibly small under the assumption that $\prod_{i=1}^{\mu} n_i \leq \mathsf{poly}(\lambda)$ and $|\mathcal{X}| \geq 2^{\omega(\log \lambda)}$. This completes the analysis of the advantage and the proof of Lemma 2.

B Security Analysis of Batch Range Proofs for Paillier

In this section, we present the security analysis of the batch range proofs for Paillier in Section 4.2 and then extend them into more complex range-related languages. More precisely, we first present some useful facts and claims in Appendix B.1, and then give the formal proofs of Theorem 1 and Theorem 2 in Appendix B.2 and Appendix B.3, respectively. Finally in Appendix B.4, we provide simple extension approaches of constructing batch range proofs for more complex languages introduced in Subsect. 4.1.

B.1 Facts and Claims

Fact 1 *Suppose that* $a^N = b^k \pmod M$*, where* k *and* N *are coprime (*$\exists\, \mu, \nu \in \mathbb{Z}$ *s.t.* $k\mu + N\nu = 1$*) and* $b \in \mathbb{Z}_M^*$*. Then there exists* $b_0 = a^\mu \cdot b^\nu \in \mathbb{Z}_M^*$ *such that* $b_0^k = a \pmod M$*.*

Fact 2 *Let* $c, d \in \mathbb{Z}$ *such that* $c \nmid d$*. There exists a prime power* a^b *such that* $a^{b-1} \mid d$*,* $a^b \nmid d$ *and* $a^b \mid c$*.*

Fact 3 *Let* $a, b \leftarrow\$ [0, R]$ *and* $\delta \leftarrow\$ [0, K]$ *be independently chosen. Then the distributions of* a *and* $b + \delta$ *are* K/R *statistically close.*

Fact 4 *Let* $\widetilde{N}$ *be a product of two distinct primes. Let* $a \leftarrow\$ \mathbb{Z}_{R\cdot\widetilde{N}}$ *and* $b \leftarrow\$ \mathbb{Z}_{\phi(\widetilde{N})}$*. Then the distributions of* $a \pmod{\phi(\widetilde{N})}$ *and* b *are* $1/R$ *statistically close.*

Claim. If we have that $m^* = m_0 + \sum_{i=1}^{\ell} m_i e_i \in [0, 2^{\varepsilon+t}B]$ with each e_i chosen uniformly at random from a set of size 2^t, then it holds that $m_i \in [-2^{\varepsilon+t}B, 2^{\varepsilon+t}B]$ for all $i \in [\ell]$ with the probability at least $1 - \frac{2\ell}{2^t}$.

Proof: If there exists some i^* such that $m_{i^*} \notin [-2^{\varepsilon+t}B, 2^{\varepsilon+t}B]$, we show it cannot happen with overwhelming probability. As $m^* \in [0, 2^{\varepsilon+t}B]$, we have

$$\frac{-m_0 - \sum_{i\neq i^*} m_i e_i}{m_{i^*}} \le e_{i^*} = \frac{m^* - m_0 - \sum_{i\neq i^*} m_i e_i}{m_{i^*}} \le \frac{2^{\varepsilon+t}B - m_0 - \sum_{i\neq i^*} m_i e_i}{m_{i^*}},$$

assuming $m_{i^*} > 0$. That is, $m^* \in [0, 2^{\varepsilon+t}B]$ holds if and only if $e_{i^*} \in S = [s, s + \frac{2^{\varepsilon+t}B}{m_{i^*}}]$ with $s = \frac{-m_0 - \sum_{i\neq i^*} m_i e_i}{m_{i^*}}$. With the condition of $m_{i^*} > 2^{\varepsilon+t}B$, the number of elements e_{i^*} satisfying $m^* \in [0, 2^{\varepsilon+t}B]$ is $|S| = 1$ and thus the probability of $e_{i^*} \in S$ holding for a random choice of e_{i^*} is at most $1/2^t$. The probability of $m_{i^*} < -2^{\varepsilon+t}B$ is also at most $1/2^t$. Thus the overall probability for each $i \in [\ell]$ is at most $\frac{2l}{2^t}$. ∎

We first provide the definition of the strong RSA assumption, as the upcoming claim is based on this assumption.

Definition 9 (Strong RSA Assumption [8]). *Let RSA modulus* $\widetilde{N}$ *be the product of two* $\kappa/2$*-long safe primes* $(2\tilde{p} + 1)$*,* $(2\tilde{q} + 1)$ *with* $\tilde{p}, \tilde{q}$ *primes, where* κ *is determined according to the security level* λ*. Given a random element* h *in* $\mathbb{Z}_{\widetilde{N}}$*, it is computationally hard to find* $x, e \neq 1$ *such that* $x^e = h \pmod{\widetilde{N}}$*.*

Claim. Given the Ring-Pedersen parameter $pp = (\widetilde{N}, h_1, h_2)$, if we have $h_1^{\widehat{m_i}} h_2^{\widehat{r_i}} = P_i^{\Delta_{\mathbf{E}}} \pmod{\widetilde{N}}$, it holds that $\Delta_{\mathbf{E}} \mid \widehat{m_i}$ and $\Delta_{\mathbf{E}} \mid \widehat{r_i}$ with overwhelming probability based on the strong RSA assumption.

Proof: Denote the predicate $\neg\mathsf{extract} = (\Delta_{\mathbf{E}} \nmid \widehat{m_i}) \vee (\Delta_{\mathbf{E}} \nmid \widehat{r_i})$, and let $\widetilde{N}, h_2$ be our strong RSA challenge, where h_2 is a random quadratic residue in $\mathbb{Z}_{\widetilde{N}}^*$, of which the order is $\phi(\widetilde{N})/4 = \tilde{p}\tilde{q}$ with overwhelming probability. We show that if $\neg\mathsf{extract}$ occurs with noticeable probability, then there is an algorithm that can break the strong RSA challenge with noticeable probability.

Let $h_1 = h_2^\chi \pmod{\widetilde{N}}$ for a random $\chi \leftarrow\$ [0, \widetilde{N}^2]$. It is not hard to see that the distribution of these values is indistinguishable from the real one with sufficiently high probability. It implies

$$h_2^{\widehat{r_i} + \chi\widehat{m_i}} = P_i^{\Delta_{\mathbf{E}}} \pmod{\widetilde{N}}. \tag{14}$$

If $\Delta_{\mathbf{E}} \nmid (\widehat{r_i} + \chi\widehat{m_i})$, let $\delta = \mathsf{gcd}((\widehat{r_i} + \chi\widehat{m_i}), \Delta_{\mathbf{E}})$, and $\delta_0 = (\widehat{r_i} + \chi\widehat{m_i})/\delta, \delta_1 = \Delta_{\mathbf{E}}/\delta > 1$, then we can find μ, ν s.t. $\mu\delta_0 + \nu\delta_1 = 1$ over $\mathbb{Z}$. From equation (14), we have $h_2^{\delta_0} = P_i^{\delta_1} \pmod{\widetilde{N}}$, thus $h_2 = h_2^{\mu\delta_0 + \nu\delta_1} = [h_2^\nu P_i^\mu]^{\delta_1} \pmod{\widetilde{N}}$. Denote $x = h_2^\nu P_i^\mu \pmod{\widetilde{N}}$, (x, δ_1) is a solution to the strong RSA challenge with $x^{\delta_1} = h_2 \pmod{\widetilde{N}}$.

Consider the case that $\Delta_{\mathbf{E}} \mid (\widehat{r_i} + \chi\widehat{m_i})$ but $\Delta_{\mathbf{E}} \nmid \widehat{m_i}$. Write $\chi = \chi_0 + \chi_1\tilde{p}\tilde{q}$, then $\widehat{r_i} + \chi\widehat{m_i} = \widehat{r_i} + \chi_0\widehat{m_i} + \chi_1\widehat{m_i}\tilde{p}\tilde{q}$. Since $\Delta_{\mathbf{E}} \nmid \widehat{m_i}\tilde{p}\tilde{q}$ where $\tilde{p}, \tilde{q}$ are primes, by Fact 2, there exists a prime power $a^b (a \ge 2)$ such that $(a^b \mid \Delta_{\mathbf{E}}) \wedge (a^{b-1} \mid \widehat{m_i}\tilde{p}\tilde{q}) \wedge (a^b \nmid \widehat{m_i}\tilde{p}\tilde{q})$. If $\Delta_{\mathbf{E}} \mid (\widehat{r_i} + \chi\widehat{m_i})$, then $a^b \mid (\widehat{r_i} + \chi\widehat{m_i})$. This also implies $a^{b-1} \mid \widehat{r_i}$. Set $a_0 = (\widehat{r_i} + \chi_0\widehat{m_i})/a^{b-1}$ and $a_1 = \widehat{m_i}\tilde{p}\tilde{q}/a^{b-1}$. We have that $a_0 + \chi_1 a_1 = 0 \pmod a$ but $a_1 \neq 0 \pmod a$, thus χ_1 is uniquely determined modulo a. On the other hand, conditioned on the Prover's view, χ_1 has full entropy, as $h_2^\chi = h_2^{\chi_0} \pmod{\widetilde{N}}$. That is, assuming $\Delta_{\mathbf{E}} \nmid \widehat{m_i}$, the probability of

$\Delta_{\mathbf{E}} \mid (\widehat{r_i} + \chi\widehat{m_i})$ is at most $\frac{1}{a} + \mathsf{negl}(\lambda) \leq \frac{1}{2} + \mathsf{negl}(\lambda)$. Finally we have the probability of $\neg\mathsf{extract}$ is at most the probability of solving the strong RSA challenge divided by $(1/2 - \mathsf{negl}(\lambda))$, which is negligible overall. In more detail, it is

$$\begin{aligned}\Pr[\neg\mathsf{extract}] &= \Pr[\Delta_{\mathbf{E}} \mid (\widehat{r_i} + \chi\widehat{m_i}) \wedge \neg\mathsf{extract}] \\ &\quad + \Pr[\Delta_{\mathbf{E}} \nmid (\widehat{r_i} + \chi\widehat{m_i}) \wedge \neg\mathsf{extract}] \\ &= \Pr[\Delta_{\mathbf{E}} \mid (\widehat{r_i} + \chi\widehat{m_i}) \wedge \Delta_{\mathbf{E}} \nmid \widehat{m_i}] + \Pr[\mathsf{sRSA}] \\ &\leq (1/2 + \mathsf{negl}(\lambda)) \cdot \Pr[\Delta_{\mathbf{E}} \nmid \widehat{m_i}] + \Pr[\mathsf{sRSA}] \\ &\leq (1/2 + \mathsf{negl}(\lambda)) \cdot \Pr[\neg\mathsf{extract}] + \Pr[\mathsf{sRSA}]. \quad \blacksquare\end{aligned}$$

Claim. Given the Ring-Pedersen parameter $pp = (\widetilde{N}, h_1, h_2)$, if we have $h_1^{\widehat{m}_{i,j}} h_2^{\widehat{r}_{i,j}} = 1 \pmod{\widetilde{N}}$, then it holds that $\widehat{m}_{i,j} = \widehat{r}_{i,j} = 0$ with overwhelming probability based on the strong RSA assumption.

Proof: let $\widetilde{N}, h_2$ be our strong RSA challenge, where h_2 is a random quadratic residue in $\mathbb{Z}^*_{\widetilde{N}}$. Let $h_1 = h_2^{\chi}$ for a random χ, then we have $h_2^{\chi\cdot\widehat{m}_{i,j}+\widehat{r}_{i,j}} = 1 \pmod{\widetilde{N}}$. Consider the case of $b = \chi \cdot \widehat{m}_{i,j} + \widehat{r}_{i,j} \neq 0$, let $a > 1$ be any number that is co-prime to b. We have $(h_2^{a^{-1} \pmod b})^a = h_2 \pmod{\widetilde{N}}$, which finds $x = h_2^{a^{-1} \pmod b}$ as the a-th root of h_2, breaking the strong RSA assumption. Thus $b = 0$ holds. If $\widehat{m}_{i,j} \neq 0$, then χ is uniquely determined, while χ has full entropy conditioned on the Prover's view. That is, $\widehat{m}_{i,j} = \widehat{r}_{i,j} = 0$ holds with overwhelming probability, based on the strong RSA assumption. $\blacksquare$

B.2 Proof of Theorem 1

Theorem 1 (ZKAoK for $\mathcal{R}_{enc}[\ell]$). *Let $\ell \leq \mathsf{poly}(\lambda)$ be the number of ciphertexts we considered. Let $\varepsilon, t \geq \lambda$ be chosen according to the security level. Then the proposed protocol $\Sigma_{\mathsf{enc}}[\ell]$ for relation $\mathcal{R}_{\mathsf{enc}}[\ell]$ has completeness and honest verifier zero-knowledge (HVZK). Moreover, it has computational witness-extended emulation under the strong RSA assumption.*

Proof:

- **Completeness.** The protocol may reject a valid statement only if $m_0 > 2^{\varepsilon+t}B - \ell\cdot 2^t B$ which happens with probability at most $\ell/2^{\varepsilon}$. By choosing $\varepsilon \geq \lambda$ and $\ell \leq \mathsf{poly}(\lambda)$, the probability $\ell/2^{\varepsilon}$ is negligible.
- **Honest-verifier zero-knowledge.** It suffices to construct a PPT simulator $\mathcal{S}_{\mathsf{enc}}$ such that, for a given instance $(B, \{C_i\}_{i\in[\ell]})$, it produces a simulated proof which is distributed statistically close to the real one generated by $\mathcal{P}_{\mathsf{enc}}$ interacting with the honest verifier $\mathcal{V}_{\mathsf{enc}}$. Given the randomness $\mathbf{e}$ consumed by $\mathcal{V}_{\mathsf{enc}}$, the simulator $\mathcal{S}_{\mathsf{enc}}$ first picks $\begin{cases} m^* \leftarrow\$ [0, 2^{\varepsilon+t}B] \\ r^* \leftarrow\$ [0, 2^{\varepsilon+t}B\widetilde{N}] \\ \rho^* \leftarrow\$ \mathbb{Z}_N^*, r_i \leftarrow\$ [0, B\widetilde{N}]\ \forall i \in [\ell], \end{cases}$ then computes $P_i = h_2^{r_i} \pmod{\widetilde{N}}$ for each $i \in [\ell]$ and sets C_0, P_0 according to the verification equations in (7), i.e.,

  $$\begin{cases} C_0 = (1+N)^{m^*}(\rho^*)^N \cdot \prod_{i\in[\ell]} C_i^{-e_i} \pmod{N^2} \\ P_0 = h_1^{m^*} h_2^{r^*} \cdot \prod_{i\in[\ell]} P_i^{-e_i} \pmod{\widetilde{N}}. \end{cases}$$

  From Facts 3 and 4, the distribution of m^* or r^* is $\ell/2^{\varepsilon}$ close to the real distribution, and each P_i is $1/B$ close to the real distribution. Overall, the real and simulated distributions are at most $2\cdot\frac{\ell}{2^{\varepsilon}} + \ell\cdot\frac{1}{B} \approx \frac{3\ell}{2^{\varepsilon}}$ far apart, which is negligible.
- **Computational witness-extended emulation.** By our multi-dimension forking lemma (i.e., Lemma 2), it suffices to construct a witness extractor χ_{enc} that succeeds in extracting a witness from an AT-LIEC-tree with overwhelming probability. Here the AT-LIEC-tree we considered has depth $\mu = 1$ and $n_1 = (\ell+1)$ leaves. Given $(B, \{C_i\}_{i\in[\ell]})$ and $(\ell+1)$ accepting transcripts

  $$(C_0, \{P_i\}_{i\in[0,\ell]}, \mathbf{e}^{(\eta)}, m^{*(\eta)}, r^{*(\eta)}, \rho^{*(\eta)})_{\eta=0}^{\ell}$$

  which share the same $(C_0, \{P_i\}_{i\in[0,\ell]})$ but have linearly independent *extended* challenge vectors $(1, \mathbf{e}^{(\eta)}) = (1, e_1^{(\eta)}, ..., e_\ell^{(\eta)})$, χ_{enc} extracts the witness $\{m_i \in [-2^{\varepsilon+t}B, 2^{\varepsilon+t}B], \rho_i\}_{i\in[\ell]}$ as follows.

(1) Since the *extended* challenge vectors $(1, \mathbf{e}^{(\eta)})_{\eta\in[0,\ell]}$, are linearly independent, the following matrix is invertible:

$$\mathbf{E} := \begin{pmatrix} 1 & e_1^{(0)} & \dots & e_\ell^{(0)} \\ 1 & e_1^{(1)} & \dots & e_\ell^{(1)} \\ \vdots & \vdots & & \vdots \\ 1 & e_1^{(\ell)} & \dots & e_\ell^{(\ell)} \end{pmatrix} \in (\{0,1\}^t)^{(\ell+1)\times(\ell+1)}. \quad (15)$$

The extractor χ_{enc} computes the determinant of $\mathbf{E}$, namely $\Delta_{\mathbf{E}} \in \mathbb{Z}$, and the *adjoint matrix* of $\mathbf{E}$, namely $\mathbf{D} := \mathbf{E}^* \in \mathbb{Z}^{(\ell+1)\times(\ell+1)}$. This process could be done efficiently. By the property of adjoint matrix, it holds that $\mathbf{D} \cdot \mathbf{E} = \Delta_{\mathbf{E}} \cdot \mathbf{I}_{\ell+1}$ over $\mathbb{Z}$ where $\mathbf{I}_{\ell+1}$ denotes the identity matrix of dimension $\ell+1$. We note that $\mathbf{D} \in \mathbb{Z}^{(\ell+1)\times(\ell+1)}$, namely, each entry of $\mathbf{D}$ is an integer.

(2) For each $i \in [\ell]$, let $\mathbf{d}_i^\top = (d_{i,0}, \cdots, d_{i,\ell})$ be the i-th row of $\mathbf{D}$ (ignoring its 0-th row), then

$$\mathbf{d}_i^\top \cdot \mathbf{E} = (\underbrace{0, \dots, 0}_{\#=i}, \Delta_{\mathbf{E}}, \underbrace{0, \dots, 0}_{\#=\ell-i}). \quad (16)$$

The extractor χ_{enc} computes $\widehat{m_i} = \sum_{\eta=0}^{\ell} d_{i,\eta} \cdot m^{*(\eta)}$ and $\widehat{r_i} = \sum_{\eta=0}^{\ell} d_{i,\eta} \cdot r^{*(\eta)}$, as well as $\widehat{\rho_i} = \prod_{\eta=0}^{\ell} (\rho^{*(\eta)})^{d_{i,\eta}} \pmod N$. By the verification equations, for each $\eta \in [0,\ell]$, it holds that

$$P_0 \cdot \prod_{j\in[\ell]} P_j^{e_j^{(\eta)}} = h_1^{m^{*(\eta)}} h_2^{r^{*(\eta)}} \bmod \widetilde{N} \text{ and} \quad (17)$$

$$C_0 \cdot \prod_{j\in[\ell]} C_j^{e_j^{(\eta)}} = (1+N)^{m^{*(\eta)}} (\rho^{*(\eta)})^N \bmod N^2. \quad (18)$$

By aggregating (17) using $\mathbf{d}_i^\top$ as exponents, we obtain that

$$\begin{aligned} &\prod_{\eta=0}^{\ell} (P_0 \cdot \prod_{j\in[\ell]} P_j^{e_j^{(\eta)}})^{d_{i,\eta}} \\ =&\prod_{\eta=0}^{\ell} (h_1^{m^{*(\eta)}} h_2^{r^{*(\eta)}})^{d_{i,\eta}} \pmod{\widetilde{N}} \end{aligned} \quad (19)$$

It is clear that the right-hand side of equation (19) is equal to $h_1^{\widehat{m_i}} h_2^{\widehat{r_i}}$. The left-hand side is equal to $P_i^{\Delta_{\mathbf{E}}}$. The above equality is due to equation (16). Thus, it holds that $h_1^{\widehat{m_i}} h_2^{\widehat{r_i}} = P_i^{\Delta_{\mathbf{E}}} \pmod{\widetilde{N}}$. Similarly, by aggregating the equation (18) using $\mathbf{d}_i^\top$ as exponent, we obtain that $(1+N)^{\widehat{m_i}} \widehat{\rho_i}^N = C_i^{\Delta_{\mathbf{E}}} \pmod{N^2}$. That is, we get two equations

$$h_1^{\widehat{m_i}} h_2^{\widehat{r_i}} = P_i^{\Delta_{\mathbf{E}}} \pmod{\widetilde{N}},\ (1+N)^{\widehat{m_i}} \widehat{\rho_i}^N = C_i^{\Delta_{\mathbf{E}}} \pmod{N^2}$$

(3) Following Claim B.1, we have that $\Delta_{\mathbf{E}} \mid \widehat{m_i}$ and $\Delta_{\mathbf{E}} \mid \widehat{r_i}$ based on the strong RSA assumption. Define $m_i = \widehat{m_i}/\Delta_{\mathbf{E}}$, and $r_i = \widehat{r_i}/\Delta_{\mathbf{E}}$. For each $i \in [\ell]$, the extractor χ_{enc} has extracted m_i, r_i s.t. $h_1^{m_i} h_2^{r_i} = P_i \pmod{\widetilde{N}}$. Define $m_0 = m^{*(0)} - \sum_{i=1}^{\ell} m_i e_i^{(0)}, r_0 = r^{*(0)} - \sum_{i=1}^{\ell} r_i e_i^{(0)}$, it is clear that $h_1^{m_0} h_2^{r_0} = P_0 \pmod{\widetilde{N}}$ from equation (17). As $m^{*(0)} \le 2^{\varepsilon+t}B$, we have $m_i \in [-2^{\varepsilon+t}B, 2^{\varepsilon+t}B]$ for each $i \in [\ell]$ with the overwhelming probability of $1 - \frac{2\ell}{2^t}$, following Claim B.1.

As $\Delta_{\mathbf{E}}$ is co-prime to N with overwhelming probability, $m_i = \widehat{m_i} \cdot \Delta_{\mathbf{E}}^{-1} \pmod N$ holds. As we have

$$\widehat{\rho_i}^N = \left(C_i \cdot (1+N)^{-m_i}\right)^{\Delta_{\mathbf{E}}} \pmod{N^2},$$

the extractor χ_{enc} can extract ρ_i s.t. $\rho_i^{\Delta_{\mathbf{E}}} = \widehat{\rho_i} \pmod{N^2}$, following Fact 1. Since $m_i \in [-2^{\varepsilon+t}B, 2^{\varepsilon+t}B]$ holds, the witness (m_i, ρ_i) s.t. $(1+N)^{m_i} \rho_i^N = C_i \pmod{N^2}$ has been extracted for each $i \in [\ell]$.

The extraction is efficient, and valid witnesses $\{m_i \in [-2^{\varepsilon+t}B, 2^{\varepsilon+t}B], \rho_i\}_{i\in[\ell]}$ are successfully extracted from a $(\mu = 1, n_1 = (\ell+1))$-AT-LIEC-tree with overwhelming probability. Moreover, note that $n_1 = \ell + 1 \le \mathsf{poly}(\lambda)$, and the size of $\mathcal{X}$ is 2^t, which is exponentially large in λ with $t \ge \lambda$. Therefore, by Lemma 2, the witness-extended emulation holds. ∎

B.3 Proof of Theorem 2

Theorem 2 (ZKAoK for $\mathcal{R}_{\mathsf{aff}}[\ell]$). *Let $\ell \le \mathsf{poly}(\lambda)$ be the number of ciphertexts we considered. Let $\varepsilon, t \ge \lambda$ be chosen according to the security level. Then the proposed protocol $\Sigma_{\mathsf{aff}}[\ell]$ for relation $\mathcal{R}_{\mathsf{aff}}[\ell]$ has completeness and HVZK. Moreover, it has computational witness-extended emulation under the strong RSA assumptions.*

Proof: The proofs of completeness and HVZK properties are similar with those of Theorem 1, so we omit those. By our multi-dimension forking lemma, it suffices to construct a witness extractor χ_{aff} that succeeds in extracting a witness from an AT-LIEC-tree with overwhelming probability. Here the AT-LIEC-tree we considered has depth $\mu = 1$ and $n_1 = (\ell+1)$ leaves.

Given an instance $(B_1, B_2, \{C_i, D_i \in \mathbb{Z}_{N^2}\}_{i\in[\ell]})$ and $(\ell+1)$ accepting transcripts $\{\Pi_\eta\}_{\eta=0}^{\ell}$, where

$$\Pi_\eta = (D_0, \bar{P}_0, \{P_i, P'_i, \bar{P}_i\}_{i\in[\ell]}, \mathbf{e}^{(\eta)}, \mathbf{m}^{*(\eta)}, \mathbf{r}^{*(\eta)}, \bar{m}^{*(\eta)}, \bar{r}^{*(\eta)}, \rho^{*(\eta)})$$

and $\mathbf{m}^{*(\eta)} = \{m_i^{*(\eta)}\}_{i\in[\ell]}, \mathbf{r}^{*(\eta)} = \{r_i^{*(\eta)}\}_{i\in[\ell]}$, sharing the same $(D_0, \bar{P}_0, \{P_i, P'_i, \bar{P}_i\}_{i\in[\ell]})$ but having linearly independent *extended* challenge vectors $(1, \mathbf{e}^{(\eta)}) = (1, e_1^{(\eta)}, ..., e_\ell^{(\eta)})$, χ_{aff} extracts the witness

$$m_i \in [-2^{\varepsilon+t}B_1, 2^{\varepsilon+t}B_1], \bar{m}_i \in [-2^{\varepsilon+t}B_2, 2^{\varepsilon+t}B_2], \rho_i$$

for each $i \in [\ell]$ as follows.

(1) To compute the matrix $\mathbf{D}$ same as the Step (1) of Theorem 1's proof. That is, for each $i \in [\ell]$, let $\mathbf{d}_i^\top = (d_{i,0}, \cdots, d_{i,\ell})$ be the i-th row of $\mathbf{D}$ (ignoring its 0-th row), then $\mathbf{d}_i^\top \cdot \mathbf{E} = (\underbrace{0,...,0}_{\#=i}, \Delta_{\mathbf{E}}, \underbrace{0,...,0}_{\#=\ell-i})$, where $\Delta_{\mathbf{E}}$ is the determinant of $\mathbf{E}$, as equation (15).

(2) For each $i, j \in [\ell]$, the extractor χ_{aff} computes

$$\begin{cases} \widehat{m}_{i,j} = \sum_{\eta=0}^{\ell} d_{i,\eta} \cdot m_j^{*(\eta)}, \quad \widehat{r}_{i,j} = \sum_{\eta=0}^{\ell} d_{i,\eta} \cdot r_j^{*(\eta)} \\ \widehat{\bar{m}_i} = \sum_{\eta=0}^{\ell} d_{i,\eta} \cdot \bar{m}^{*(\eta)}, \widehat{\bar{r}_i} = \sum_{\eta=0}^{\ell} d_{i,\eta} \cdot \bar{r}^{*(\eta)} \end{cases}$$

as well as $\widehat{\rho}_i = \prod_{\eta=0}^{\ell} (\rho^{*(\eta)})^{d_{i,\eta}} \pmod N$. Next, we will show that the following equations hold for each $i \in [\ell]$

$$\begin{cases} \widehat{m}_{i,j} = \hat{r}_{i,j} = 0, \ \forall j \in [\ell] \wedge j \neq i \\ h_1^{\widehat{m}_{i,i}} h_2^{\widehat{r}_{i,i}} = P_i^{\Delta_{\mathbf{E}}} \pmod{\widetilde{N}}, \quad h_1^{\widehat{\bar{m}_i}} h_2^{\widehat{\bar{r}_i}} = \bar{P}_i^{\Delta_{\mathbf{E}}} \pmod{\widetilde{N}} \\ C_i^{\widehat{m}_{i,i}} \cdot (1+N)^{\widehat{\bar{m}_i}} (\widehat{\rho}_i)^N = D_i^{\Delta_{\mathbf{E}}} \pmod{N^2}. \end{cases} \tag{20}$$

By the verification process of $\Sigma_{\mathsf{aff}}[\ell]$, for each $\eta \in [0, \ell]$, it holds that

$$D_0 \prod_{j\in[\ell]} D_j^{e_j^{(\eta)}} = \left(\prod_{j\in[\ell]} C_j^{m_j^{*(\eta)}}\right) (1+N)^{\bar{m}^{*(\eta)}} (\rho^{*(\eta)})^N \tag{21}$$

modulo N^2 and

$$P'_j \cdot P_j^{e_j^{(\eta)}} = h_1^{m_j^{*(\eta)}} h_2^{r_j^{*(\eta)}} \pmod{\widetilde{N}} \ \forall j \in [\ell] \tag{22}$$

$$\bar{P}_0 \cdot \prod_{j\in[\ell]} \bar{P}_j^{e_j^{(\eta)}} = h_1^{\bar{m}^{*(\eta)}} h_2^{\bar{r}^{*(\eta)}} \pmod{\widetilde{N}}. \tag{23}$$

By aggregating equation (23) using $\mathbf{d}_i^\top$ as exponents, it is easy to obtain that $h_1^{\widehat{\bar{m}_i}} h_2^{\widehat{\bar{r}_i}} = \bar{P}_i^{\Delta_{\mathbf{E}}} \pmod{\widetilde{N}}$. Then, by aggregating equation (22) using $\mathbf{d}_i^\top$ as exponents, we obtain that

$$\prod_{\eta=0}^{\ell} \left(P'_j \cdot P_j^{e_j^{(\eta)}}\right)^{d_{i,\eta}} = \prod_{\eta=0}^{\ell} \left(h_1^{m_j^{*(\eta)}} h_2^{r_j^{*(\eta)}}\right)^{d_{i,\eta}} \pmod{\widetilde{N}} \tag{24}$$

for each $j \in [\ell]$. The right-hand side of equation (24) is $h_1^{\widehat{m}_{i,j}} h_2^{\widehat{r}_{i,j}}$, while the left-hand side is $\begin{cases} P_i^{\Delta_{\mathbf{E}}}, \ j = i \\ 1, \ \forall j \in [\ell] \wedge j \neq i. \end{cases}$ As $h_1^{\widehat{m}_{i,j}} h_2^{\widehat{r}_{i,j}} = 1 \pmod{\widetilde{N}}$, we have $\widehat{m}_{i,j} = \widehat{r}_{i,j} = 0$ for $j \neq i$ following Claim B.1.

By aggregating the equation (21) using $\mathbf{d}_i^\top$ as exponents, we get

$$\begin{aligned} D_i^{\Delta_{\mathbf{E}}} &= \prod_{\eta=0}^{\ell} \left(\prod_{j\in[\ell]} C_j^{m_j^{*(\eta)}}\right)^{d_{i,\eta}} \cdot (1+N)^{\widehat{\bar{m}_i}} (\widehat{\rho}_i)^N \\ &= \left(\prod_{j\in[\ell]} C_j^{\widehat{m}_{i,j}}\right) \cdot (1+N)^{\widehat{\bar{m}_i}} (\widehat{\rho}_i)^N \\ &= C_i^{\widehat{m}_{i,i}} \cdot (1+N)^{\widehat{\bar{m}_i}} (\widehat{\rho}_i)^N \pmod{N^2}. \end{aligned}$$

(3) Combining Claim B.1 and equations in (20), we have that $\Delta_{\mathbf{E}} \mid \widehat{m}_{i,i},\ \Delta_{\mathbf{E}} \mid \widehat{r}_{i,i},\ \Delta_{\mathbf{E}} \mid \widehat{m_i},\ \Delta_{\mathbf{E}} \mid \widehat{r_i}$ with overwhelming probability. Define $m_i = \widehat{m}_{i,i}/\Delta_{\mathbf{E}}$, $r_i = \widehat{r}_{i,i}/\Delta_{\mathbf{E}}$, $\bar{m}_i = \widehat{m_i}/\Delta_{\mathbf{E}}$ and $\bar{r}_i = \widehat{r_i}/\Delta_{\mathbf{E}}$. For each $i \in [\ell]$, the extractor χ_{aff} has extracted $m_i, r_i, \bar{m}_i, \bar{r}_i$ such that $h_1^{m_i} h_2^{r_i} = P_i,\ h_1^{\bar{m}_i} h_2^{\bar{r}_i} = \bar{P}_i \pmod{\widetilde{N}}$. The values $m_i', r_i', \bar{m}_0, \bar{r}_0$ used for masking can also be extracted via

$$\begin{cases} m_i' = m_i^{*(0)} - e_i^{(0)} m_i,\ r_i' = r_i^{*(0)} - e_i^{(0)} r_i \quad \forall i \in [\ell], \\ \bar{m}_0 = \bar{m}^{*(0)} - \sum_{i\in[\ell]} e_i^{(0)} \bar{m}_i, \\ \bar{r}_0 = \bar{r}^{*(0)} - \sum_{i\in[\ell]} e_i^{(0)} \bar{r}_i. \end{cases}$$

They satisfy $h_1^{m_i'} h_2^{r_i'} = P_i'$ and $h_1^{\bar{m}_0} h_2^{\bar{r}_0} = \bar{P}_0$ from equations (22) and (23).

As $m_i^{*(0)} \le 2^{\varepsilon+t} B_1,\ \forall i \in [\ell]$ and $\bar{m}^{*(0)} \le 2^{\varepsilon+t} B_2$, we have

$$m_i \in [-2^{\varepsilon+t} B_1, 2^{\varepsilon+t} B_1],\ \bar{m}_i \in [-2^{\varepsilon+t} B_2, 2^{\varepsilon+t} B_2] \tag{25}$$

for each $i \in [\ell]$ from Claim B.1. As $\Delta_{\mathbf{E}}$ is co-prime to N with overwhelming probability, $m_i = \widehat{m}_{i,i} \cdot \Delta_{\mathbf{E}}^{-1} \pmod N), \bar{m}_i = \widehat{m_i} \cdot \Delta_{\mathbf{E}}^{-1} \pmod N$ hold. From the equation

$$\widehat{\rho_i}^N = \left(D_i \cdot C_i^{-m_i} (1+N)^{-\bar{m}_i}\right)^{\Delta_{\mathbf{E}}} \pmod{N^2}$$

the extractor χ_{aff} can extract ρ_i s.t. $\rho_i^{\Delta_{\mathbf{E}}} = \widehat{\rho_i} \pmod{N^2}$, following Fact 1. Since the range conditions in (25) hold, the witness $(m_i, \bar{m}_i, \rho_i)$ s.t. $C_i^{m_i} \cdot (1+N)^{\bar{m}_i} (\rho_i)^N = D_i \pmod{N^2}$ has been extracted for each $i \in [\ell]$.

The extraction is efficient, and valid witnesses are successfully extracted with overwhelming probability. Therefore, by Lemma 2, the witness-extended emulation holds. ∎

B.4 Extension to other languages of Subsect. 4.1

In this part, we show the following extensions:

- Sigma-protocol $\Sigma_{\mathsf{enc}}[\ell]$ for the language $\mathcal{R}_{\mathsf{enc}}[\ell]$ in Subsect. 4.2 can be extended to the Sigma-protocols $\Sigma_{\mathsf{log}^*}[\ell]$ and $\Sigma_{\mathsf{enc\text{-}elg}}[\ell]$ for the languages $\mathcal{R}_{\mathsf{log}^*}[\ell]$ and $\mathcal{R}_{\mathsf{enc\text{-}elg}}[\ell]$, respectively.
- Sigma-protocol $\Sigma_{\mathsf{aff}}[\ell]$ for the language $\mathcal{R}_{\mathsf{aff}}[\ell]$ in Subsect. 4.2 can be extended to the Sigma-protocols $\Sigma_{\mathsf{afflog}}[\ell]$, $\Sigma_{\mathsf{aff\text{-}g}}[\ell]$ and $\Sigma_{\mathsf{aff\text{-}p}}[\ell]$ for the languages $\mathcal{R}_{\mathsf{afflog}}[\ell]$, $\mathcal{R}_{\mathsf{aff\text{-}g}}[\ell]$ and $\mathcal{R}_{\mathsf{aff\text{-}p}}[\ell]$ respectively.

Based on $\Sigma_{\mathsf{enc}}[\ell]$, the batch range proof $\Sigma_{\mathsf{log}^*}[\ell]$ can be constructed by adding $X_0 = g^{m_0}$ in the first round and verifying $X_0 \cdot \prod_{i\in[\ell]} X_i^{e_i} \stackrel{?}{=} g^{m^*}$. Also, $\Sigma_{\mathsf{enc}}[\ell]$ can be extended to the batch range proof $\Sigma_{\mathsf{enc\text{-}elg}}[\ell]$, by adding

$$A_0 = g_1^{a_0}, B_0 = g_1^{m_0} g_2^{a_0}, a^* = a_0 + \textstyle\sum_{i\in[\ell]} e_i a_i \pmod q$$

to the proof and verifying $A_0 \cdot \prod_{i\in[\ell]} A_i^{e_i} \stackrel{?}{=} g_1^{a^*}$ and $B_0 \cdot \prod_{i\in[\ell]} B_i^{e_i} \stackrel{?}{=} g_1^{m^*} g_2^{a^*}$ in addition.

To extend $\Sigma_{\mathsf{aff}}[\ell]$ into $\Sigma_{\mathsf{afflog}}[\ell]$, we additionally need to prove that m_i is also the discrete logarithm of group element X_i. It is sufficient for the Prover to add $X_i' = g^{m_i'}$ in the first round and for the Verifier to additionally check $X_i' \cdot X_i^{e_i} \stackrel{?}{=} g^{m_i^*},\ \forall i \in [\ell]$. This constitutes the Sigma-protocol $\Sigma_{\mathsf{afflog}}[\ell]$ for $\mathcal{R}_{\mathsf{afflog}}[\ell]$. For the more complex relation $\mathcal{R}_{\mathsf{aff\text{-}g}}[\ell]$ with additional ciphertexts $\{F_i\}_{i\in[\ell]}$, the Prover needs to prove that each $\bar{m}_i$ is also the plaintext of F_i under public key N_1. On the basis of $\Sigma_{\mathsf{afflog}}[\ell]$, the Prover adds a random Paillier ciphertext of $\bar{m}_0$ in the first round, i.e., $F_0 = (1+N_1)^{\bar{m}_0} (\bar{\rho}_0)^{N_1} \pmod{N_1^2}$, and adds $\bar{\rho}^* = \bar{\rho}_0 \cdot \prod_{i\in[\ell]} (\bar{\rho}_i)^{e_i}$
$\pmod{N_1}$ in the third round. Accordingly, the Verifier needs to additionally check that $F_0 \cdot \prod_{i\in[\ell]} F_i^{e_i} \stackrel{?}{=} (1+N_1)^{\bar{m}^*} (\bar{\rho}^*)^{N_1} \pmod{N_1^2}$. In this way, we get a batch proof $\Sigma_{\mathsf{aff\text{-}g}}[\ell]$ for $\mathcal{R}_{\mathsf{aff\text{-}g}}[\ell]$. To construct the batch proof $\Sigma_{\mathsf{aff\text{-}p}}[\ell]$ for the language $\mathcal{R}_{\mathsf{aff\text{-}p}}[\ell]$, the Prover replaces $X_i' = g^{m_i'}$ of $\Sigma_{\mathsf{afflog}}[\ell]$ with a random Paillier ciphertext of m_i', namely $E_i' = (1+N_1)^{m_i'} (\tilde{\rho}_i')^{N_1} \pmod{N_1^2}$. Then replying $\tilde{\rho}_i^* = \tilde{\rho}_i' \cdot (\tilde{\rho}_i)^{e_i} \pmod{N_1}$ allows the Verifier to check $E_i' \cdot E_i^{e_i} \stackrel{?}{=} (1+N_1)^{m_i^*} (\tilde{\rho}_i^*)^{N_1} \pmod{N_1^2}$ for each $i \in [\ell]$.

C More Details on Batch Range Proofs for JL Encryption

In this section, we present the constructions and security analysis of batch range proofs for JL encryption. More precisely, we give two range proofs in Appendix C.1 and Appendix C.2, which respectively correspond to the two relations $\mathcal{R}_{\mathsf{equ}}[\ell]$ and $\mathcal{R}_{\mathsf{JL_{aff}}}[\ell]$ introduced in Section 5.

C.1 Batch Proof $\Sigma_{\mathsf{equ}}[\ell]$ for Language $\mathcal{R}_{\mathsf{equ}}[\ell]$

In this part, we present the concrete construction of $\Sigma_{\mathsf{equ}}[\ell]$ in Appendix C.1, give some useful assumptions, facts and claims in Appendix C.1 and a formal security analysis in Appendix C.1.

Construction The batch proof $\Sigma_{\mathsf{equ}}[\ell]$ allows the Prover to demonstrate that the plaintext of each JL ciphertext is within a given range, simultaneously corresponds to the opening of the corresponding JL commitment.

- **Input:** The common input is $(B, \{C_i, P_i\}_{i\in[\ell]})$ and pk_{J}, pp_{J}. $\mathcal{P}_{\mathsf{equ}}$ holds the witness $\{m_i \in [0, B]\}_{i\in[\ell]}$ with each one satisfying $P_i = \tilde{y}^{2^k m_i} \cdot \tilde{h}^{2^k r_i} \pmod{\widetilde{N}_{\mathsf{J}}}$ and $C_i = y^{m_i} \cdot h^{\rho_i} \pmod{N_{\mathsf{J}}}$ for some $r_i \in \mathbb{Z}_{\widetilde{N}_{\mathsf{J}}}, \rho_i \in \mathbb{Z}_{N_{\mathsf{J}}}$.
- The Σ-protocol $\Sigma_{\mathsf{equ}}[\ell]$ for $\mathcal{R}_{\mathsf{equ}}[\ell]$ is described as follows.
  (1) $\mathcal{P}_{\mathsf{equ}} \to \mathcal{V}_{\mathsf{equ}}$: $\mathcal{P}_{\mathsf{equ}}$ picks $\begin{cases} m_0 \leftarrow\$ [0, 2^{\varepsilon+t}B] \\ \rho_0 \leftarrow\$ [0, 2^{\varepsilon+t}N_{\mathsf{J}}] \\ r_0 \leftarrow\$ [0, 2^{\varepsilon+t}\widetilde{N}_{\mathsf{J}}], \end{cases}$ computes $\begin{cases} C_0 = y^{m_0} \cdot h^{\rho_0} \pmod{N_{\mathsf{J}}} \\ P_0 = \tilde{y}^{2^k m_0} \cdot \tilde{h}^{2^k r_0} \pmod{\widetilde{N}_{\mathsf{J}}}, \end{cases}$ and sends (C_0, P_0) to $\mathcal{V}_{\mathsf{equ}}$.
  (2) $\mathcal{V}_{\mathsf{equ}} \to \mathcal{P}_{\mathsf{equ}}$: $\mathcal{V}_{\mathsf{equ}}$ sends $\mathbf{e} \leftarrow\$ ([0, 2^t - 1])^\ell$ to $\mathcal{P}_{\mathsf{equ}}$.
  (3) $\mathcal{P}_{\mathsf{equ}} \to \mathcal{V}_{\mathsf{equ}}$: $\mathcal{P}_{\mathsf{equ}}$ computes $\begin{cases} m^* = m_0 + \sum_{i\in[\ell]} e_i m_i \\ \rho^* = \rho_0 + \sum_{i\in[\ell]} e_i \rho_i \\ r^* = r_0 + \sum_{i\in[\ell]} e_i r_i, \end{cases}$ and sends (m^*, ρ^*, r^*) to $\mathcal{V}_{\mathsf{equ}}$.
- **Verification:** $\mathcal{V}_{\mathsf{equ}}$ receives $(C_0, P_0, m^*, \rho^*, r^*)$ from $\mathcal{P}_{\mathsf{equ}}$, and accepts if $m^* \in [0, 2^{\varepsilon+t}B]$ and the following equations hold:

$$\begin{cases} C_0 \cdot \prod_{i\in[\ell]} C_i^{e_i} \stackrel{?}{=} y^{m^*} \cdot h^{\rho^*} \pmod{N_{\mathsf{J}}} \\ P_0 \cdot \prod_{i\in[\ell]} P_i^{e_i} \stackrel{?}{=} \tilde{y}^{2^k m^*} \cdot \tilde{h}^{2^k r^*} \pmod{\widetilde{N}_{\mathsf{J}}}. \end{cases} \tag{26}$$

  If the verification succeeds, $\mathcal{V}_{\mathsf{equ}}$ is convinced that $m_i \in [-2^{\varepsilon+t}B,\ 2^{\varepsilon+t}B]$ holds for each $i \in [\ell]$.

Assumptions, Facts and Claims The security of the batch proof $\Sigma_{\mathsf{equ}}[\ell]$ is based on the strong JL and k-QR assumptions, which are also used in the single JL range proof [61]. Here, we give the definitions of the k-QR and strong JL assumptions, and some facts for the JL modulus, as well as a claim that will be used in the proofs of Theorem 3 and 4.

Definition 10 (k-QR Assumption). *Let N_J be the JL modulus. The k-QR assumption asserts that*

$$\mathsf{Adv}_{\mathcal{A}}^{kQR} = |\Pr[\mathcal{A}(N_J, x) = 1 | x \leftarrow\$ \mathbb{QR}_{2^k}] - \Pr[\mathcal{A}(N_J, x) = 1 | x \leftarrow\$ \mathbb{QNR}]|$$

is negligible in λ, for any PPT adversary $\mathcal{A}$.

Definition 11 (Strong JL Assumption [61]). *Let N_J be the JL modulus. Given a random element $h \in \mathbb{QR}_{2^k}$, it is computationally hard to find $x, e \neq 1$ such that $x^e = h \pmod{N_J}$.*

Fact 5 *Let $N_J = \bar{P}\bar{Q} = (2^k p' + 1)(2q' + 1)$ be a JL modulus, we have*

1. *$J_{N_J}(-1) = -1$.*
2. *$\mathbb{QR}_{2^k}$ is the cyclic subgroup of $\mathbb{Z}^*_{N_J}$ of order $p'q'$.*
3. *A random element from $\mathbb{QR}_{2^k}$ is its generator with probability $(1 - 1/p')(1 - 1/q')$.*
4. *Finding a non-trivial square root (i.e., $\neq \pm 1$) is equivalent to factoring the modulus N_J.*

Claim. Given the public parameter $pp_{\mathsf{J}} = (\widetilde{N}_{\mathsf{J}}, \tilde{h}, \tilde{y}, k)$ of the JL commitment scheme, if we have a ring element $P_i \in \widetilde{N}_{\mathsf{J}}$ satisfying $\tilde{y}^{2^k \widehat{m_i}} \cdot \tilde{h}^{2^k \widehat{r_i}} = P_i^{\Delta_{\mathbf{E}}} \pmod{\widetilde{N}_{\mathsf{J}}}$, then it holds that $\Delta_{\mathbf{E}} \mid \widehat{m_i}$ and $\Delta_{\mathbf{E}} \mid \widehat{r_i}$ with overwhelming probability, based on the k-QR (Definition 10) and strong JL assumptions (Definition 11).

Proof: Denote the predicate $\neg\mathsf{extract} = (\Delta_{\mathbf{E}} \nmid \widehat{m_i}) \vee (\Delta_{\mathbf{E}} \nmid \widehat{r_i})$. Let $\widetilde{N}_{\mathsf{J}}, \tilde{h}$ be our challenge of the strong JL problem, where $\widetilde{N}_{\mathsf{J}} = (2^k \tilde{p}' + 1)(2\tilde{q}' + 1)$ is a JL modulus and $\tilde{h} \in \mathbb{QR}_{2^k}$ has the order of $\tilde{p}'\tilde{q}'$ with overwhelming probability. We show that if $\neg\mathsf{extract}$ occurs with noticeable probability, then there is an algorithm that can break the strong JL assumption with noticeable probability, finding a solution $x, e \neq 1$ such that $x^e = \tilde{h} \pmod{\widetilde{N}_\mathsf{J}}$.

Let $\tilde{y} = \tilde{h}^\chi \pmod{\widetilde{N}_\mathsf{J}}$ for a random $\chi \leftarrow\!\!\$ \, [0, \widetilde{N}_\mathsf{J}^2]$. The only difference is the generation of $\tilde{y}$ (i.e., $\tilde{y} \in \mathbb{QNR}$ or $\tilde{y} \in \mathbb{QR}_{2^k}$). The committer could not find this difference due to the k-QR assumption. It implies

$$\tilde{h}^{2^k(\widehat{r_i}+\chi\widehat{m_i})} = P_i^{\Delta_\mathbf{E}} \pmod{\widetilde{N}_\mathsf{J}}. \tag{27}$$

Let $\Delta_\mathbf{E} = 2^v \cdot \rho$ for some $v \geq 0$ and odd number ρ. It distinguishes two cases:

1. $v \geq k$: The equation (27) can be rewritten as
$$[\tilde{h}^{\widehat{r_i}+\chi\widehat{m_i}} \cdot P_i^{-2^{v-k}\rho}]^{2^k} = 1 \pmod{\widetilde{N}_\mathsf{J}}. \tag{28}$$
Let $a = \widehat{r_i} + \chi\widehat{m_i}$, $b = 2^{v-k}\rho$ and $c = k \geq 0$.
2. $v < k$: The equation (27) can be rewritten as
$$[\tilde{h}^{2^{k-v}(\widehat{r_i}+\chi\widehat{m_i})} \cdot P_i^{-\rho}]^{2^v} = 1 \pmod{\widetilde{N}_\mathsf{J}}. \tag{29}$$
Let $a = 2^{k-v}(\widehat{r_i} + \chi\widehat{m_i})$, $b = \rho$ and $c = v \geq 0$.

The above two cases both lead to the following equation

$$[\tilde{h}^a \cdot P_i^{-b}]^{2^c} = 1 \pmod{\widetilde{N}_\mathsf{J}}. \tag{30}$$

It falls into the following cases:

1. $c = 0$: $\tilde{h}^a \cdot P_i^{-b} = 1 \pmod{\widetilde{N}_\mathsf{J}}$.
2. $c = 1$: Based on the Fact 5(4), we have $\tilde{h}^a \cdot P_i^{-b} = \pm 1 \pmod{\widetilde{N}_\mathsf{J}}$. $\tilde{h}^a \cdot P_i^{-b} = -1 \pmod{\widetilde{N}_\mathsf{J}}$ would never happen since $J_{\widetilde{N}_\mathsf{J}}(-1) = -1$ and $J_{\widetilde{N}_\mathsf{J}}(\tilde{h}) = J_{\widetilde{N}_\mathsf{J}}(P_i) = 1$. Thus we have $\tilde{h}^a \cdot P_i^{-b} = 1 \pmod{\widetilde{N}_\mathsf{J}}$.
3. $c \geq 2$: Based on the Fact 5(4), we have $[\tilde{h}^a \cdot P_i^{-b}]^{2^{c-1}} = \pm 1 \pmod{\widetilde{N}_\mathsf{J}}$. $[\tilde{h}^a \cdot P_i^{-b}]^{2^{c-1}} = -1 \pmod{\widetilde{N}_\mathsf{J}}$ would never happen with the same reason as case (2). Thus we have $[\tilde{h}^a \cdot P_i^{-b}]^{2^{c-1}} = 1 \pmod{\widetilde{N}_\mathsf{J}}$, which could be analyzed via recursion.

Overall, all of the above three cases lead to $\tilde{h}^a = P_i^b \pmod{\widetilde{N}_\mathsf{J}}$ from the equation (30). The next analysis is divided into two cases:

- Case 1 - $b \nmid a$: Let $\beta = \gcd(b, a)$. There exists efficient algorithm to find f, g such that $\beta = fb + ga$. Then we have
$$\tilde{h}^\beta = \tilde{h}^{fb+ga} = \tilde{h}^{fb}P_i^{gb} = [\tilde{h}^f P_i^g]^b \pmod{\widetilde{N}_\mathsf{J}}.$$
As $\tilde{p}', \tilde{q}'$ are set to be primes, we have that β is co-prime to $\tilde{p}'\tilde{q}'$ with overwhelming probability. Thus $\tilde{h} = [\tilde{h}^f P_i^g]^{b/\beta} \pmod{\widetilde{N}_\mathsf{J}}$ holds. Due to $b \nmid a$, we have $b > \beta$ and $b/\beta > 1$. Finally, in this case, we find a solution of the strong JL problem, that is $x^e = \tilde{h} \pmod{\widetilde{N}_\mathsf{J}}$ with $x = \tilde{h}^f P_i^g$ and $e = b/\beta > 1$.
- Case 2 - $b \mid a$: For the case of $v \geq k$, we have $a = \widehat{r_i} + \chi\widehat{m_i}$, $b = 2^{v-k}\rho$; for the other case of $v < k$, we have $a = 2^{k-v}(\widehat{r_i} + \chi\widehat{m_i})$, $b = \rho$. When $b \mid a$ holds, these two cases both imply that $2^v\rho \mid 2^k(\widehat{r_i} + \chi\widehat{m_i})$, that is $\Delta_E \mid \widehat{r_i} + \chi\widehat{m_i}$. In this case, the predicate $\neg\mathsf{extract}$ is equal to $\Delta_E \nmid \widehat{m_i}$. The reason is, if $\Delta_E \mid \widehat{m_i}$, then $\Delta_E \mid \widehat{r_i}$ due to $\Delta_E \mid \widehat{r_i} + \chi\widehat{m_i}$, contradictory to the predicate.
Write $\chi = \chi_0 + \chi_1\tilde{p}'\tilde{q}'$, then $\widehat{r_i} + \chi\widehat{m_i} = \widehat{r_i} + \chi_0\widehat{m_i} + \chi_1\widehat{m_i}\tilde{p}'\tilde{q}'$. Since $\Delta_\mathbf{E} \nmid \widehat{m_i}\tilde{p}'\tilde{q}'$ where $\tilde{p}', \tilde{q}'$ are primes, by Fact 2 in Appendix B.1, there exists a prime power ι^j ($\iota \geq 2$) such that
$$(\iota^j \mid \Delta_\mathbf{E}) \wedge (\iota^{j-1} \mid \widehat{m_i}\tilde{p}'\tilde{q}') \wedge (\iota^j \nmid \widehat{m_i}\tilde{p}'\tilde{q}').$$
Due to $\Delta_\mathbf{E} \mid (\widehat{r_i} + \chi\widehat{m_i})$, $\iota^j \mid (\widehat{r_i} + \chi\widehat{m_i})$ holds. Then $\iota^{j-1} \mid \widehat{m_i}$ implies $\iota^{j-1} \mid \widehat{r_i}$. Set $\iota_0 = (\widehat{r_i} + \chi_0\widehat{m_i})/\iota^{j-1}$ and $\iota_1 = \widehat{m_i}\tilde{p}'\tilde{q}'/\iota^{j-1}$. We have that $\iota_0 + \chi_1 \cdot \iota_1 = 0 \pmod{\iota}$ but $\iota_1 \neq 0 \pmod{\iota}$ due to $\iota^j \nmid \widehat{m_i}\tilde{p}'\tilde{q}'$, thus χ_1 is uniquely determined modulo ι. On the other hand, conditioned on the Prover's view, χ_1 has full entropy, as $\tilde{h}^\chi = \tilde{h}^{\chi_0} \pmod{\widetilde{N}_\mathsf{J}}$. That is, assuming the predicate $\neg\mathsf{extract}$ happens, the probability of $\Delta_\mathbf{E} \mid (\widehat{r_i} + \chi\widehat{m_i})$ (i.e., $b \mid a$) is at most $\frac{1}{\iota} + \mathsf{negl}(\lambda) \leq \frac{1}{2} + \mathsf{negl}(\lambda)$.

Finally we have the probability of $\neg\mathsf{extract}$ is at most the probability of solving the strong JL problem or breaking k-QR assumption divided by $(1/2 - \mathsf{negl}(\lambda))$, which is negligible overall. In more detail, it is

$$\begin{aligned}
\Pr[\neg\mathsf{extract}] &= \Pr[b \nmid a \wedge \neg\mathsf{extract}] + \Pr[b \mid a \wedge \neg\mathsf{extract}] \\
&= \Pr[k\text{-}\mathsf{QR}] + \Pr[\mathsf{sJL}] + \Pr[\Delta_\mathbf{E} \mid (\widehat{r_i} + \chi\widehat{m_i}) \wedge \neg\mathsf{extract}] \\
&\leq \Pr[k\text{-}\mathsf{QR}] + \Pr[\mathsf{sJL}] + (1/2 + \mathsf{negl}(\lambda)) \cdot \Pr[\neg\mathsf{extract}].
\end{aligned}$$

∎

Security Analysis

Theorem 3 (ZKAoK for $\mathcal{R}_{equ}[\ell]$). *Let $\ell \le \mathsf{poly}(\lambda)$ be the number of ciphertexts or commitments we considered. Let $\varepsilon, t \ge \lambda$ be chosen according to the security level. Then the proposed protocol $\Sigma_{\mathsf{equ}}[\ell]$ has completeness and HVZK. Moreover, it has computational witness-extended emulation under the strong JL and k-QR assumptions.*

Proof: The analysis of completeness and HVZK is straightforward, so we only show the computational witness-extended emulation. Given an instance $(B, \{C_i, P_i\}_{i\in[\ell]})$ and $(\ell+1)$ accepting transcripts $(C_0, P_0, \mathbf{e}^{(\eta)}, m^{*(\eta)}, r^{*(\eta)}, \rho^{*(\eta)})_{\eta=0}^{\ell}$, χ_{equ} extracts the witness as follows.

(1) To compute the matrix $\mathbf{D}$, that is same as the Step (1) of the security proof in Appendix B.2. For each $i \in [\ell]$, let $\mathbf{d}_i^\top = (d_{i,0}, \cdots, d_{i,\ell})$ be the i-th row of $\mathbf{D}$.

(2) The extractor χ_{equ} computes $\widehat{m_i} = \sum_{\eta=0}^{\ell} d_{i,\eta} \cdot m^{*(\eta)}$, $\widehat{r_i} = \sum_{\eta=0}^{\ell} d_{i,\eta} \cdot r^{*(\eta)}$ and $\widehat{\rho_i} = \sum_{\eta=0}^{\ell} d_{i,\eta} \cdot \rho^{*(\eta)}$. The verification shows that

$$\begin{cases} C_0 \cdot \prod_{j\in[\ell]} C_j^{e_j^{(\eta)}} = y^{m^{*(\eta)}} \cdot h^{\rho^{*(\eta)}} \pmod{N_{\mathsf{J}}} \\ P_0 \cdot \prod_{j\in[\ell]} P_j^{e_j^{(\eta)}} = \tilde{y}^{2^k m^{*(\eta)}} \cdot \tilde{h}^{2^k r^{*(\eta)}} \pmod{\widetilde{N}_{\mathsf{J}}}, \end{cases}$$

for each $\eta \in [0, \ell]$. By aggregating the above equations using $\mathbf{d}_i^\top$ as exponents, it is easy to get

$$\begin{aligned} y^{\widehat{m_i}} \cdot h^{\widehat{\rho_i}} &= C_i^{\Delta_{\mathbf{E}}} \pmod{N_{\mathsf{J}}} \text{ and} \\ \tilde{y}^{2^k \widehat{m_i}} \cdot \tilde{h}^{2^k \widehat{r_i}} &= P_i^{\Delta_{\mathbf{E}}} \pmod{\widetilde{N}_{\mathsf{J}}}. \end{aligned} \tag{31}$$

(3) From Claim C.1 in Appendix C.1 and $\tilde{y}^{2^k \widehat{m_i}} \cdot \tilde{h}^{2^k \widehat{r_i}} = P_i^{\Delta_{\mathbf{E}}} \pmod{\widetilde{N}_{\mathsf{J}}}$, we have that $\Delta_{\mathbf{E}} \mid \widehat{m_i}$ and $\Delta_{\mathbf{E}} \mid \widehat{r_i}$, based on the strong JL and k-QR assumptions. Since $\widetilde{N}_{\mathsf{J}} = (2^k\tilde{p}' + 1)(2\tilde{q}' + 1)$ is a JL modulus, we have that the order of P_i is $\tilde{p}'\tilde{q}'$, from items (2) and (3) of the Fact 5. As Δ_E is co-prime to $\tilde{p}'\tilde{q}'$ with overwhelming probability, we have that

$$\tilde{y}^{2^k m_i} \cdot \tilde{h}^{2^k r_i} = P_i \pmod{\widetilde{N}_{\mathsf{J}}}$$

with $m_i = \widehat{m_i}/\Delta_{\mathbf{E}}$ and $r_i = \widehat{r_i}/\Delta_{\mathbf{E}}$. For each $i \in [\ell]$, the extractor χ_{equ} has extracted m_i, r_i as the opening of the commitment P_i. Define $m_0 = m^{*(0)} - \sum_{i=1}^{\ell} m_i e_i^{(0)}$, $r_0 = r^{*(0)} - \sum_{i=1}^{\ell} r_i e_i^{(0)}$, it is clear that $\tilde{y}^{2^k m_0} \cdot \tilde{h}^{2^k r_0} = P_0 \pmod{\widetilde{N}_{\mathsf{J}}}$ from the verification equations in (26). As $m^{*(0)} \le 2^{\varepsilon+t}B$, we have that $m_i \in [-2^{\varepsilon+t}B, 2^{\varepsilon+t}B]$ for each $i \in [\ell]$ from Claim B.1 (c.f. Appendix B.1).

From $y^{\widehat{m_i}} \cdot h^{\widehat{\rho_i}} = C_i^{\Delta_{\mathbf{E}}} \pmod{N_{\mathsf{J}}}$ and $\widehat{m_i} = m_i \Delta_{\mathbf{E}}$, we get

$$[C_i y^{-m_i}]^{\Delta_{\mathbf{E}}} = [h^{\rho_i}]^{\Delta_{\mathbf{E}}} \pmod{N_{\mathsf{J}}} \quad \text{for some } \rho_i. \tag{32}$$

As we have $h \in \mathbb{QR}_{2^k}$, from items (2) and (3) of the Fact 5 (c.f. Appendix C.1), its order is $p'q'$ with probability $(1 - 1/p')(1 - 1/q')$. As p', q' are set to be primes, we have that $\Delta_{\mathbf{E}}$ is co-prime to $p'q'$ with overwhelming probability. In this case, ρ_i is some value satisfying $\rho_i = \widehat{\rho_i} \cdot \Delta_{\mathbf{E}}^{-1} \pmod{p'q'}$, and $\Delta_{\mathbf{E}}$ in the equation (32) can be eliminated, as $C_i = y^{m_i} h^{\rho_i} \pmod{N_{\mathsf{J}}}$. As a result, the extracted m_i is the plaintext of C_i.

Since $m_i \in [-2^{\varepsilon+t}B, 2^{\varepsilon+t}B]$ holds, the witness m_i as the plaintext of C_i and the opening of P_i has been extracted for each $i \in [\ell]$.

The valid witnesses $\{m_i \in [-2^{\varepsilon+t}B, 2^{\varepsilon+t}B]\}_{i\in[\ell]}$ are successfully extracted, so the witness-extended emulation holds. ∎

C.2 Batch Proof $\Sigma_{\mathsf{JL_{aff}}}[\ell]$ for Language $\mathcal{R}_{\mathsf{JL_{aff}}}[\ell]$

In this part, we present the batch range proof for $\mathcal{R}_{\mathsf{JL_{aff}}}[\ell]$. As introduced in Section 2.3, the modified JL encryption can be easily converted into a JL commitment. If the Prover does not know the secret key information of pk_{J}, then $pk_{\mathsf{J}} = (N_{\mathsf{J}}, h, y, k)$ can be regarded as the public parameters of JL commitment scheme from the Prover's view.

Additionally, given multiple bases $y_1, y_2, \cdots, y_\ell$, the JL commitment scheme can be further extended into a vector commitment scheme, where the commitment of a vector $\mathbf{m} = \{m_1, \cdots, m_\ell\}$ is $P = \prod_{i\in[\ell]} y_i^{2^k m_i} \cdot h^{2^k r} \pmod{\widetilde{N}_{\mathsf{J}}}$.

Intuitively, the homomorphically-generated ciphertext D_i in relation $\mathcal{R}_{\mathsf{JL_{aff}}}[\ell]$ does resemble a vector commitment in form. Taking C_i as another base with $J_{N_\mathsf{J}}(C_i) = 1$, $P_i = D_i^{2^k} \pmod{N_\mathsf{J}}$ is a vector commitment to the vector $(m_i, \bar{m}_i)$ under the bases (C_i, y, h). Thus the relation $\mathcal{R}_{\mathsf{JL_{aff}}}[\ell]$ can be rewritten as

$$\mathcal{R}_{\mathsf{JL_{aff}}}[\ell] = \left\{ \begin{array}{c|c} ((B_1, B_2, & \forall\, i \in [\ell], m_i \in [0, B_1], \bar{m}_i \in [0, B_2] \\ \{X_i, C_i, P_i\}_{i\in[\ell]}), & P_i = C_i^{2^k m_i} \cdot y^{2^k \bar{m}_i} \cdot h^{2^k \rho_i} \bmod N_\mathsf{J} \\ \{m_i, \bar{m}_i, \rho_i\}_{i\in[\ell]}) & X_i = g^{m_i} \end{array} \right\}.$$

We give a Σ-protocol for the relation $\mathcal{R}_{\mathsf{JL_{aff}}}[\ell]$.

- **Input:** The common input is $(B_1, B_2, \{X_i, C_i, P_i\}_{i\in[\ell]})$ and $pp_\mathsf{J} = (N_\mathsf{J}, y, h, k)$. $\mathcal{P}_{\mathsf{JL_{aff}}}$ holds the witness $\{m_i \in [0, B_1], \bar{m}_i \in [0, B_2],$
$\rho_i\}_{i\in[\ell]}$.
- The Σ-protocol $\Sigma_{\mathsf{JL_{aff}}}[\ell]$ for $\mathcal{R}_{\mathsf{JL_{aff}}}[\ell]$ is described as follows.
  (1) $\mathcal{P}_{\mathsf{JL_{aff}}} \to \mathcal{V}_{\mathsf{JL_{aff}}}$: $\mathcal{P}_{\mathsf{JL_{aff}}}$ picks $\begin{cases} m_i' \leftarrow\!\$\ [0, 2^{\varepsilon+t} B_1]\ \forall i \in [\ell] \\ \bar{m}_0 \leftarrow\!\$\ [0, 2^{\varepsilon+t} B_2], \rho_0 \leftarrow\!\$\ [0, 2^{\varepsilon+t} N_\mathsf{J}] \end{cases}$ and computes $X_i' = g^{m_i'}$, $\forall i \in [\ell]$,

  $$P_0 = \left(\prod_{i\in[\ell]} C_i^{2^k m_i'}\right) \cdot y^{2^k \bar{m}_0} \cdot h^{2^k \rho_0} \pmod{N_\mathsf{J}}.$$

  $\mathcal{P}_{\mathsf{JL_{aff}}}$ sends $(P_0, \{X_i'\}_{i\in[\ell]})$ to $\mathcal{V}_{\mathsf{JL_{aff}}}$.
  (2) $\mathcal{V}_{\mathsf{JL_{aff}}} \to \mathcal{P}_{\mathsf{JL_{aff}}}$: $\mathcal{V}_{\mathsf{JL_{aff}}}$ sends $\mathbf{e} \leftarrow\!\$\ ([0, 2^t - 1])^\ell$ to $\mathcal{P}_{\mathsf{JL_{aff}}}$.
  (3) $\mathcal{P}_{\mathsf{JL_{aff}}} \to \mathcal{V}_{\mathsf{JL_{aff}}}$: $\mathcal{P}_{\mathsf{JL_{aff}}}$ computes

  $$\begin{cases} m_i^* = m_i' + e_i m_i,\ \forall i \in [\ell] \\ \bar{m}^* = \bar{m}_0 + \sum_{i\in[\ell]} e_i \bar{m}_i, \rho^* = \rho_0 + \sum_{i\in[\ell]} e_i \rho_i. \end{cases}$$

  $\mathcal{P}_{\mathsf{JL_{aff}}}$ sends $(\{m_i^*\}_{i\in[\ell]}, \bar{m}^*, \rho^*)$ to $\mathcal{V}_{\mathsf{JL_{aff}}}$.
- **Verification:** $\mathcal{V}_{\mathsf{JL_{aff}}}$ receives the proof

  $$\Pi = (P_0, \{X_i'\}_{i\in[\ell]}, \{m_i^*\}_{i\in[\ell]}, \bar{m}^*, \rho^*)$$

  from $\mathcal{P}_{\mathsf{JL_{aff}}}$, and accepts if $m_i^* \in [0, 2^{\varepsilon+t} B_1], \forall i \in [\ell]$, $\bar{m}^* \in [0, 2^{\varepsilon+t} B_2]$ and the following equations hold:

  $$\begin{cases} X_i' \cdot X_i^{e_i} \stackrel{?}{=} g^{m_i^*},\ \forall i \in [\ell] \\ P_0 \cdot \prod_{i\in[\ell]} P_i^{e_i} \stackrel{?}{=} \left(\prod_{i\in[\ell]} C_i^{2^k m_i^*}\right) y^{2^k \bar{m}^*} h^{2^k \rho^*} \bmod N_\mathsf{J} \end{cases} \tag{33}$$

  If the verification succeeds, $\mathcal{V}_{\mathsf{JL_{aff}}}$ is convinced that $m_i \in [-2^{\varepsilon+t} B_1, 2^{\varepsilon+t} B_1]$ and $\bar{m}_i \in [-2^{\varepsilon+t} B_2, 2^{\varepsilon+t} B_2]$ hold for each $i \in [\ell]$.

Theorem 4 (ZKAoK for $\mathcal{R}_{\mathsf{JL_{aff}}}[\ell]$). *The protocol $\Sigma_{\mathsf{JL_{aff}}}[\ell]$ has completeness and HVZK. Moreover, it has computational witness-extended emulation under the strong JL and k-QR assumptions.*

Fig. 3: **Our JL-based batch MtA protocol.**

Alice$(pp_A, \{a_i\}_{i\in[\ell]})$		Bob$(pk_B, sk_B, \{b_i\}_{i\in[\ell]})$
		$C_i \leftarrow \mathsf{JL.enc}(pk_B, b_i),\ \forall i \in [\ell]$
$A_i = g^{a_i},\ \forall i \in [\ell]$	$\xleftarrow{\{C_i, P_i\}_{i\in[\ell]}, \Sigma_{\mathsf{equ}}[\ell]}$	$P_i \leftarrow \mathsf{JL.commit}(pp_A, b_i),\ \forall i \in [\ell]$
$D_i = C_i^{a_i} \cdot \mathsf{JL.enc}(pk_B, \alpha_i),\ \forall i \in [\ell]$	$\xrightarrow{\{D_i, A_i\}_{i\in[\ell]}, \Sigma_{\mathsf{JL_{aff}}}[\ell]}$	$\beta_i \leftarrow \mathsf{JL.dec}(sk_B, D_i),\ \forall i \in [\ell]$
$\alpha_i = -\alpha_i \bmod q,\ \forall i \in [\ell]$		$\beta_i = \beta_i \bmod q,\ \forall i \in [\ell]$

Proof: For completeness and HVZK property, the security proofs are avoided as they are very similar with those in Theorem 1's proof. Next, we show that $\Sigma_{\mathsf{JL_{aff}}}[\ell]$ has computational witness-extended emulation under the strong JL and k-QR assumptions.

Given an instance $(B_1, B_2, \{P_i\}_{i\in[\ell]})$ and $(\ell+1)$ accepting transcripts $(P_0, \{X_i'\}_{i\in[\ell]}, \mathbf{e}^{(\eta)}, \mathbf{m}^{*(\eta)}, \bar{m}^{*(\eta)}, \rho^{*(\eta)})_{\eta=0}^{\ell}$ with $\mathbf{m}^{*(\eta)}$
$= \{m_i^{*(\eta)}\}_{i\in[\ell]}$, $\chi_{\mathsf{JL_{aff}}}$ extracts the witness $(m_i \in [-2^{\varepsilon+t}B_1, 2^{\varepsilon+t}B_1]$,
$\bar{m}_i \in [-2^{\varepsilon+t}B_2, 2^{\varepsilon+t}B_2], \rho_i)$ for each $i \in [\ell]$ as follows.

(1) To compute the matrix $\mathbf{D}$, that is same as the Step (1) of Theorem 1's proof. Let $\mathbf{d}_i^\top = (d_{i,0}, \cdots, d_{i,\ell})$ be the i-th row of $\mathbf{D}$.

(2) The extractor $\chi_{\mathsf{JL_{aff}}}$ computes $\begin{cases}\widehat{m}_{i,j} = \sum_{\eta=0}^{\ell} d_{i,\eta}\cdot m_j^{*(\eta)},\ \forall j\in[\ell]\\ \widehat{\bar{m}_i} = \sum_{\eta=0}^{\ell} d_{i,\eta}\cdot \bar{m}^{*(\eta)}\\ \widehat{\rho_i} = \sum_{\eta=0}^{\ell} d_{i,\eta}\cdot \rho^{*(\eta)}\end{cases}$ without modulus. It holds that

$$P_i^{\Delta_{\mathbf{E}}} = \left(\prod_{j\in[\ell]} C_j^{2^k \widehat{m}_{i,j}}\right)\cdot y^{2^k \widehat{\bar{m}_i}} \cdot h^{2^k \widehat{\rho_i}} \pmod{N_{\mathsf{J}}} \tag{34}$$

and $g^{\widehat{m}_{i,j}} = \begin{cases}1,\ \forall j\in[\ell]\wedge j\neq i\\ X_i^{\Delta_{\mathbf{E}}},\ j=i.\end{cases}$ For each $i\in[\ell]$, the above equations are generated by aggregating the verification equations in (33) using $\mathbf{d}_i^\top$ as exponents, similarly with the Step (2) of Theorem 3's proof.

(3) Assuming that $C_j = y^{s_j}\cdot h^{t_j} \pmod{N_{\mathsf{J}}}$ for random s_j, t_j for each $j\in[\ell]$. Then we have

$$P_i^{\Delta_{\mathbf{E}}} = y^{2^k(\sum_j s_j \widehat{m}_{i,j} + \widehat{\bar{m}_i})}\cdot h^{2^k(\sum_j t_j \widehat{m}_{i,j} + \widehat{\rho_i})} \pmod{N_{\mathsf{J}}}$$

from equation (34). From Claim C.1 in Appendix C.1, $\Delta_{\mathbf{E}}$ must divide $\sum_j s_j\widehat{m}_{i,j} + \widehat{\bar{m}_i}$ and $\sum_j t_j \widehat{m}_{i,j} + \widehat{\rho_i}$. Due to the randomness of s_i, t_i conditioned on the Prover's view, $\Delta_{\mathbf{E}}$ must divide each $\widehat{m}_{i,j}, \widehat{\bar{m}_i}, \widehat{\rho_i}$. Define $m_{i,j} = \widehat{m}_{i,j}/\Delta_{\mathbf{E}}$. As $\Delta_{\mathbf{E}}$ is co-prime to prime q with overwhelming probability, then we have $g^{m_{i,j}} = \begin{cases}1, j\in[\ell], j\neq i\\ X_i, j=i.\end{cases}$

As constructed in the Algorithm 2 of the multi-dimension forking lemma (Lemma 2), a malicious Prover $\mathcal{P}^*$ is invoked after each successful sampling of a challenge vector $\mathbf{e}^{(\eta)}$, expecting it to provide a valid corresponding transcript, e.g., $\mathbf{m}^{*(\eta)}$. That is, the matrices $\mathbf{E}, \mathbf{D}$ are randomly unknown for the Prover $\mathcal{P}^*$'s view before the last invocation (i.e., when $\eta = \ell$). Thus, the probability of $\mathcal{P}^*$'s outputs $m_j^{*(\eta)}$ satisfying $\sum_{\eta=0}^{\ell} d_{i,\eta}\cdot m_j^{*(\eta)}/\Delta_{\mathbf{E}} \neq 0$ over $\mathbb{Z}$ but $\sum_{\eta=0}^{\ell} d_{i,\eta}\cdot m_j^{*(\eta)}/\Delta_{\mathbf{E}} = 0 \pmod q$ for each $i\neq j$ is at most $\frac{1}{q}$. This implies that $m_{i,j} = \sum_{\eta=0}^{\ell} d_{i,\eta}\cdot m_j^{*(\eta)}/\Delta_{\mathbf{E}} = 0$ for $\forall j\in[\ell]\wedge j\neq i$ with overwhelming probability. Define $m_i = m_{i,i}, \bar{m}_i = \widehat{\bar{m}_i}/\Delta_{\mathbf{E}}$ and $\rho_i = \widehat{\rho_i}/\Delta_{\mathbf{E}}$. For each $i\in[\ell]$, $\chi_{\mathsf{JL_{aff}}}$ has extracted $m_i, \bar{m}_i$ in range satisfying $P_i = C_i^{2^k m_i}\cdot y^{2^k \bar{m}_i}\cdot h^{2^k\rho_i}$ and $X_i = g^{m_i}$.

The valid witnesses are successfully extracted, so the witness-extended emulation holds. ∎

D Comparisons of XAL⁺23 [61] and Ours

In this section, we first present our batch JL-based MtA protocol in Appendix D.1. Then we compare Xue et al.'s (batch) MtA constructions [61] and ours under their counting method.

D.1 Our Batch JL-based MtA

Let (pk_B, sk_B) be Bob's key pair of JL encryption and pp_A be the public parameter of JL commitment generated by Alice.

1) Bob generates the ciphertext $C_i \leftarrow \mathsf{JL.enc}(pk_B, b_i)$ and commitment $P_i \leftarrow \mathsf{JL.commit}(pp_A, b_i)$ for each $i\in[\ell]$, then compute the batch proof $\Sigma_{\mathsf{equ}}[\ell]$ to prove that the plaintext of each C_i is within $\mathbb{Z}_q$ and corresponds to the opening of each P_i, Bob sends $(\{C_i, P_i\}_{i\in[\ell]}, \Sigma_{\mathsf{equ}}[\ell])$ to Alice.
2) Alice verifies the batch proof $\Sigma_{\mathsf{equ}}[\ell]$. Then she picks $\alpha_i \leftarrow\!\$\ [0, q^5]$ and computes $D_i = C_i^{a_i}\cdot\mathsf{JL.enc}(pk_B, \alpha_i)$ for each $i\in[\ell]$. For checking the correctness of the input a_i, Alice also generates $A_i = g^{a_i}$ for each $i\in[\ell]$. Finally Alice generates the batch proof $\Sigma_{\mathsf{JL_{aff}}}[\ell]$ to prove $((q, q^5, \{A_i, C_i, D_i^{2^k}\}_{i\in[\ell]}), \{a_i, \alpha_i, \rho_i\}_{i\in[\ell]}) \in \mathcal{R}_{\mathsf{JL_{aff}}}[\ell]$ where ρ_i is the randomness used in encrypting α_i. Alice sends $(\{D_i, A_i\}_{i\in[\ell]}, \Sigma_{\mathsf{JL_{aff}}}[\ell])$ to Bob, and outputs $-\alpha_i \bmod q$,
$\forall i\in[\ell]$.
3) Bob verifies the proof $\Sigma_{\mathsf{JL_{aff}}}[\ell]$ and outputs $\beta_i \leftarrow \mathsf{JL.Dec}(sk_B, D_i)$
$\bmod q,\ \forall i\in[\ell]$.

The illustration of the above protocol is shown in Figure 3.

Table 7: **Cost comparisons of JL-based MtA protocol by utilizing the counting method of [61, Table 2].** $\boldsymbol{\mu}$ and $\mathbf{E}$ correspond to one element and one full exponentiation in $\mathbb{Z}_{N_\mathsf{J}}$, respectively.

MtAs	Communication ($\boldsymbol{\mu}$)		Computation ($\mathbf{E}$)	
	Alice	Bob	Alice	Bob
JL [61]	3.5	6.5	5	6
Batch JL [61]	3.5	$3.5 + 3/\ell$	$4 + 1/\ell$	$4 + 2/\ell$
Our Batch JL	$1 + 2.5/\ell$	$2 + 4.5/\ell$	$1.5 + 3.5/\ell$	$2.5 + 3.5/\ell$

Table 8: **Cost comparisons of setup phase in JL-based MtA.**

Setup	Communication ($\boldsymbol{\mu}$)	Computation ($\mathbf{E}$)
JL [61]	$12n + 4$	$8n + 6$
Batch JL [61]	$6(\ell + 1)n + 2\ell + 2$	$(4\ell + 4)n + 2\ell + 4$
Our Batch JL	$12n + 4$	$8n + 6$

D.2 Comparison

Following Xue et al.'s counting method, the computational cost of JL-based MtA is measured in terms of full exponentiation operations over $\mathbb{Z}_{N_\mathsf{J}}$. For example, the JL encryption algorithm computes $C = y^m h^\rho \pmod{N_\mathsf{J}}$. If we have the plaintext $m \in \mathbb{Z}_q$, this is counted as $(1+\log q/\log N_\mathsf{J})$ full exponentiation operations.

With this counting method, we present Table 7 and Table 8. We note that the data for XAL$^+$23 comes from [61, Table 2, 3]. Table 7 shows that our batch JL scheme improves JL-based MtA of XAL$^+$23 [61] in both communication and computation by a factor of ≈ 3, additionally improves their batch JL by a factor of ≈ 2. These improving factors under this kind of counting are similar with those in Table 1. Table 8 shows that our batch technique does not require to increase the setup costs, has better performance than their batch JL scheme.

Fig. 4: **Threshold ECDSA functionality $\mathcal{F}_{\mathsf{ECDSA}}$**

- KGen: On receiving $\mathsf{KGen}(1^\lambda)$ from all parties $\mathcal{P}_1, \cdots, \mathcal{P}_n$
  1. Pick a random $x \leftarrow\$ \mathbb{Z}_q$ and compute $X = g^x$. Set (x, X) as an ECDSA key pair and store $(\mathbb{G}, g, q, x, X)$.
  2. Send X as the public key to all $\mathcal{P}_1, \cdots, \mathcal{P}_n$ and ignore any further call to KGen.
- Sign: On receiving $\mathsf{Sign}(sid, \mathsf{msg})$ from all parties, if KGen was already called and sid has not been used before:
  1. Sample $k \leftarrow\$ \mathbb{Z}_q$ and compute $R = (r_x, r_y) = g^k$.
  2. Compute $r = r_x \bmod q$, $\sigma = k^{-1}(\mathcal{H}(\mathsf{msg}) + rx)) \bmod q$.
  3. Send (r, σ) to all parties $\mathcal{P}_1, \cdots, \mathcal{P}_n$ and store $(\mathsf{Complete}, sid)$ in the memory.

E Threshold ECDSA and The Framework of DKLs24 [33]

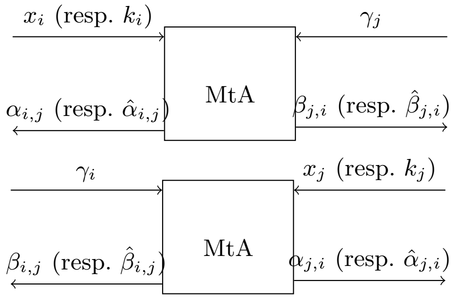
In this section, we first formally introduce the ECDSA algorithm and give the ideal functionality of threshold ECDSA. Then, we present a construction framework proposed by DKLs24 [33] and simply describe the batch presigning based on DKLs24. Finally, we present more related works of threshold ECDSA.

Let $(\mathbb{G}, g, q)$ denote the group-generator-order tuple associated with the curve of ECDSA signatures. ECDSA scheme [29] makes use of the hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_q$ and works as follows.

Fig. 5: **The basic framework of multi-party threshold ECDSA proposed by DKLs24 [33].** Define $x = \sum_{i=1}^{n} x_i \bmod q$ as the secret key of ECDSA, $k = \sum_{i=1}^{n} k_i \bmod q$ as the secret nonce selected for each signature and $\gamma = \sum_{i=1}^{n} \gamma_i \bmod q$ as another secret nonce for masking k. The generated shares $(x\gamma)_i, (k\gamma)_i$ are the additive shares of $x\gamma, k\gamma$ respectively. All MtA calls are done simultaneously, completed within two rounds of communication.

Party $\mathcal{P}_i(x_i, k_i, \gamma_i), i \in [n]$

Each other party $\mathcal{P}_j(x_j, k_j, \gamma_j), j \in [n] \setminus \{i\}$



$(x\gamma)_i = \gamma_i x_i + \sum_{j=1, j\neq i}^{n} (\alpha_{i,j} + \beta_{i,j}) \bmod q$

$(k\gamma)_i = \gamma_i k_i + \sum_{j=1, j\neq i}^{n} (\hat{\alpha}_{i,j} + \hat{\beta}_{i,j}) \bmod q$

$(x\gamma)_j = \gamma_j x_j + \sum_{i=1, i\neq j}^{n} (\beta_{j,i} + \alpha_{j,i}) \bmod q$

$(k\gamma)_j = \gamma_j k_j + \sum_{i=1, i\neq j}^{n} (\hat{\beta}_{j,i} + \hat{\alpha}_{j,i}) \bmod q$

Online phase for a message msg

$s_i = \gamma_i \mathcal{H}(\mathsf{msg}) + (x\gamma)_i \cdot r \bmod q$ ⟵ Broadcast $(k\gamma)_j, s_j$ ⟵ $s_j = \gamma_j \mathcal{H}(\mathsf{msg}) + (x\gamma)_j \cdot r \bmod q$

$$\sigma = \frac{\sum_{j\in[n]} s_j}{\sum_{j\in[n]} (k\gamma)_j} \bmod q$$

1. $\mathsf{KGen}(1^\lambda) \to (x, X)$: choose $x \leftarrow\!\$\ \mathbb{Z}_q$, set x as the private key, then compute $X = g^x$ and set X as the public key.
2. $\mathsf{Sign}(x, \mathsf{msg}) \to (r, \sigma)$: choose $k \leftarrow\!\$\ \mathbb{Z}_q$ and compute $R = (r_x, r_y) = g^k$. Set $r = r_x \bmod q$ and compute $\sigma = k^{-1} \cdot (\mathcal{H}(\mathsf{msg}) + rx) \bmod q$. Output (r, σ) as the signature.
3. $\mathsf{Verify}(X, (r, \sigma)) \to b$: compute $(r'_x, r'_y) = R' = (g^{\mathcal{H}(\mathsf{msg})} \cdot X^r)^{\sigma^{-1}}$, output $b = 1$ if and only if $r = r'_x \bmod q$.

Figure 4 presents the ideal functionality $\mathcal{F}_{\mathsf{ECDSA}}$ for threshold ECDSA.

In Figure 5, we give a construction framework of threshold ECDSA derived from DKLs24 [33], utilizing MtA as a black-box module. In the end of presigning phase, each party $\mathcal{P}_i$ gets $(x\gamma)_i$ and $(k\gamma)_i$, which are respectively the additive shares of $x\gamma$ and $k\gamma$. After receiving the message msg to be signed, each party can compute $s_i = \gamma_i \mathcal{H}(\mathsf{msg}) + (x\gamma)_i \cdot r \bmod q$, where r is assembled by $\prod_{i\in[n]} g^{k_i}$, each of which is disclosed by each party in the presigning phase. Then in the online phase, after each one broadcasts $((k\gamma)_i, s_i)$ via one round of communication, the ECDSA signature is finally generated, $\sigma = \frac{\sum_{i\in[n]} s_i}{\sum_{i\in[n]} (k\gamma)_i} \pmod q$.

The batched version. By replacing each MtA with batch MtA, we can easily obtain a two-round batch presigning protocol. In the end of batch presigning, each party $\mathcal{P}_i$ gets additive shares $\{(\gamma^{(j)} x)_i, (\gamma^{(j)} k^{(j)})_i\}_{j\in[\ell]}$ where ℓ is the batch size, such that

$$\textstyle\sum_{i\in[n]} (\gamma^{(j)} x)_i = \gamma^{(j)} x, \ \ \sum_{i\in[n]} (\gamma^{(j)} k^{(j)})_i = \gamma^{(j)} k^{(j)}.$$

For one instance of online signing, one secret nonce $k^{(j)}$ and masking nonce $\gamma^{(j)}$ are consumed. Therefore, these additive shares can be used to generate ℓ ECDSA signatures.

Storage Cost. For the usage of online phase, each party $\mathcal{P}_i$ is required to store 4ℓ elements in $\mathbb{Z}_q$, which are

$$\{r^{(j)}, \gamma_i^{(j)}, (\gamma^{(j)} x)_i, (\gamma^{(j)} k^{(j)})_i\}_{j\in[\ell]}.$$

When the number of batched presigning instances is $\ell = 100$, the storage cost is 12.5 KB for $\lambda = 128$.

More related works of threshold ECDSA. Castagnos et al. [25] replaced Paillier encryption with Castagnos and Laguillaumie (CL) encryption [27], followed by subsequent works [64,30,26] to improve efficiency. In CL-based threshold ECDSA, range proofs can be avoided since CL's message space matches the ECDSA signature space, however other ZK proofs, such as correctness proofs of CL encryption are required. Abram et al. [6] used pseudorandom correlation generators (PCG) [18,19] to build threshold ECDSA. Their bandwidth compleixty is $1 \sim 2$ orders of magnitude smaller than those based on Paillier, CL or JL encryption, however, their amortized computational cost is expensive. There are schemes that consider cheater identification, e.g., [23,26,60]. Canetti et al. [23] and Castagnos et al. [26] can only abort the threshold signing protocol after identifying a cheater. Wong et al. [60] provided self-healing property: after finding malicious behaviors, it allows continuation as long as a threshold number of honest parties remains. If replacing their MtA with our batch MtA, the properties of cheater identification and self-healing still remain intact.