Computer Standards & Interfaces 97 (2026) 104107

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

MExpm: Fair computation offloading for batch modular exponentiation with improved privacy and checkability in IoV

Sipeng Shen[1], Qiang Wang *,[1], Fucai Zhou, Jian Xu, Mingxing Jin

Software College, Northeastern University, China

ARTICLE INFO

Keywords:
Internet of Vehicles
Modular exponentiation
Computation offloading
Smart contract

ABSTRACT

Modular exponentiation is a fundamental cryptographic operation extensively applied in the Internet of Vehicles (IoV). However, its computational intensity imposes significant resource and time demands on intelligent vehicles. Offloading such computations to Mobile Edge Computing (MEC) servers has emerged as a promising approach. Nonetheless, existing schemes are generally impractical, as they either fail to ensure fairness between intelligent vehicles and MEC servers, lack privacy protection for the bases and exponents, or cannot guarantee the correctness of results with overwhelming probability due to potential misbehavior by MEC servers. To address these limitations, we propose MExpm, a fair and efficient computation offloading scheme for batch modular exponentiation under a single untrusted server model. Our scheme leverages blockchain technology to ensure fairness through publicly verifiable results. Furthermore, MExpm achieves high checkability, offering a near-perfect probability of checkability. To enhance privacy, we introduce secure obfuscation and logical split techniques, effectively protecting both the bases and the exponents. Extensive theoretical analysis and experimental results demonstrate that our scheme is not only efficient in terms of computation, communication, and storage overheads but also significantly improves privacy protection and checkability.

1. Introduction

1.1. Motivation

Batch modular exponentiation, a fundamental mathematical operation, denoted as $\prod_{i=1}^{n} u_i^{a_i} \mod N$, which is widely used in the Internet of Vehicles (IoV) (i.e., key exchange, digital signatures, and identity authentication) and is assumed as one of the most resource-intensive operations. Considering limited computation resources in intelligent vehicles, locally executing the above task is unviable, which cannot meet both computation resources and time latency requirements [1]. To tackle this challenge, computation offloading (CO) is proposed to undertake resource-intensive computation tasks for intelligent vehicles [2]. However, current cloud computation paradigm for modular exponentiation offloading in [3–8] fails to meet the requirements of low latency, location awareness, and mobility support [9], since the cloud servers are far from the vehicles, it is a challenge for network transfer latency. To overcome the limitations of cloud computation, offloading computational tasks from intelligent vehicles to MEC servers, being closer to intelligent vehicles than cloud servers, can provide adequate computation resources for offloaded tasks while meeting the latency requirements of intelligent vehicles [10,11]. Despite these benefits, it still suffers from some security challenges. Once the computation tasks are offloaded, it will lose control over them. As a result, the MEC server may forge the outcome of the computation. To address this issue, verifiable CO was first proposed by [12] to ensure the integrity of the results. A fundamental requirement for verifiable CO is that the total time invested in the verification process should be less than the time spent performing the computation by himself. Otherwise, the intelligent vehicle would not prefer to offload its computation.

1.2. Limitations of prior art

In this paper, we mainly focus on verifiable computation offloading for batch modular exponentiation with MEC servers. However, to the best of our knowledge, none of the existing prior schemes are practical enough, as demonstrated in Fig. 1. They suffer from the following challenges.

Fairness. Most verifiable CO schemes for batch modular exponentiation make sure the results are correct for the client before paying but often disregard the cloud's interests. As a result, the client might

* Corresponding author.
E-mail address: wangqiang1@mail.neu.edu.cn (Q. Wang).
[1] Equal contribution.

https://doi.org/10.1016/j.csi.2025.104107
Received 3 June 2025; Received in revised form 12 August 2025; Accepted 28 November 2025
Available online 3 December 2025

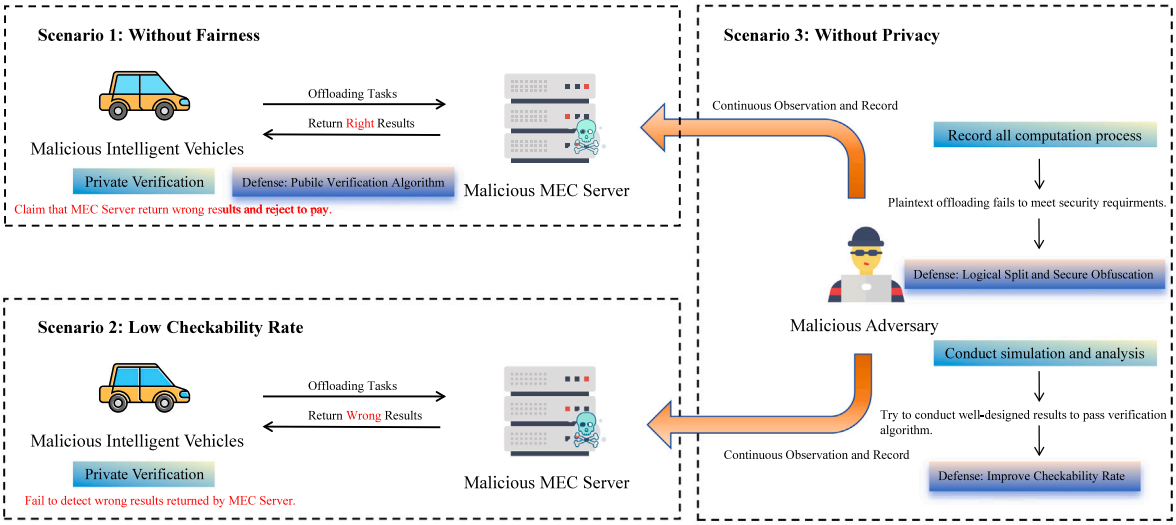

Fig. 1. Limitations of Prior Art and Our Defenses: Scenario 1: Previous works adopt private verification algorithm. Under this assumption, the greedy intelligent vehicle may reject the correct computation results and refuse to pay for the MEC server's work. Scenario 2: Previous works with low checkability fail to detect MEC server's misbehavior. Scenario 3: Previous works with plaintext offloading strategy fail to protect the confidentiality of inputs and outputs.

refuse to pay by deliberately claiming that the MEC server returns an incorrect result even when executed faithfully. Furthermore, the cloud may intentionally manipulate the computation outcome for some economic incentives. When a dispute occurs between them, a fully trusted third party (TTP), such as a judge, has to be involved to deduce which party is wrong. As an ex-post measure, the dispute can be finally handled, but it is unfriendly for time-sensitive IoV applications [13]. Therefore, it is essential to find an immediate resolution without TTP to guarantee fairness between the MEC server and the intelligent vehicle. Due to transparency, accountability, and immutability, blockchain can be used to establish trust among untrusted parties. A naive solution is to delegate the entire computation to the blockchain. It is inefficient and imposes financial burdens on intelligent vehicles, such as significant gas fees for modular exponentiation in Ethereum. Besides, this approach seriously deviates from the original intent of computation offloading.

Checkability Rate. The existing schemes employ verification mechanisms to ensure computation correctness against malicious MEC servers [14]. However, the achieved checkability rate often falls short of expectations, failing to reach 100%. For example, in [3,4], the checkability rate is only 97.5% when the batch size $n = 1000$. In other words, intelligent vehicles may fail to detect misbehavior by malicious MEC servers with a 2.5% probability. Besides, the intelligent vehicle may make misjudgments even if the MEC servers return correct results. When disputes arise between the intelligent vehicle and the MEC servers, as previously mentioned, a complex procedure involving TTP is imperative. This ex-post measure is valid, but it is unsuitable for IoV time-sensitive applications. Furthermore, there is no such a fully trusted entity in the real world.

Privacy. Most of the existing schemes offload modular exponentiation $\prod_{i=1}^{n} u_i^{a_i} \mod N$ in a plaintext way [6,7]. If we directly apply them into IoV, this inevitably comes with the privacy concern. Modular exponentiation plays a critical role in secure cryptography algorithms (i.e., key exchange, digital signatures, identity authentication). In this case, the MEC server knows the base u_i, exponent a_i, and output which is the result of $u^a(\mod N)$, so it will increase the risks of privacy leakage and attacks. From this point, it is essential to protect the privacy of the bases, exponents, and results. To tackle this challenge, some researchers [3–5] utilize the logical split technique to protect privacy. The security relies on a strong assumption that the auxiliary information cannot be known by the malicious adversary. Therefore, the verification algorithm can only be executed by the data owner. For a fair computation offloading task, the verification algorithm should be public. To tackle this challenge, a straightforward approach is to process the computation using fully homomorphic encryption (FHE). Specifically, the bases u_i are encrypted using the data owner's public key and outsourced to the MEC server. The intelligent vehicle encodes the queries a_i under the same public key. To recover the final result returned by the MEC server, the private key of the data owner should be shared with the intelligent vehicle. If the private key is leaked, it will cause serious privacy issues [15,16]. Furthermore, the intelligent vehicle cannot afford this heavy computation owing to the limitation of resources.

Table 1
Comparison of properties.

Scheme	Batch size	Privacy	Checkability rate	Verification	Fairness
MExp [3]	1	×	$\frac{119}{120}$	Private	×
SMCExp [4]	1	×	$\frac{119}{120}$	Private	×
SoRSA [6]	1	×	$\frac{119}{120}$	Private	×
EPExp [7]	1	×	0	Private	×
MExpm(ours)	1	✓	≈ 1	Public	✓
MExp [3]	n	×	$1-\frac{n^2}{10(4n^2+6n+2)}$	Private	×
SMCExp [4]	n	×	$1-\frac{n^2}{10(4n^2+6n+2)}$	Private	×
GExp [5]	n	×	$\frac{1}{n+1}$	Private	×
MExpm(ours)	n	✓	$1-\frac{n^2}{(4n^2+6n+2)(N-2)}$	Public	✓

Batch Size: The number of bases in one offloading;
Privacy: Whether it can protect privacy of bases and exponents;
Checkability Rate: The Checkability of Offloading scheme;
Verification: Verification method for offloading;
Fairness: The fairness for both service provider and intelligent vehicles;
✓: means the scheme achieves this property; ×: means it does not.

Compared with existing works in Table 1, MExpm supports privacy and fairness both for service providers and intelligent vehicles. Our contributions can be summarized as follows.

1. To the best of our knowledge, we are the first ones to attempt fair computation offloading of batch modular exponentiation under a single untrusted server model, which is more appropriate for practical applications.
2. We integrate smart contracts into the verification process to ensure fairness and correctness. Compared with existing schemes, our approach incurs lower gas consumption.

3. We employ a logical split method and secure obfuscation techniques to conceal the bases, exponents, and modulus before offloading computation. Consequently, MExpm achieves near-perfect checkability rate.

2. Related work

2.1. Computation offloading

Intelligent vehicles, with limited computation resources and an increasing number of in-car applications, struggle to efficiently execute computation-intensive tasks. To address the challenges faced by intelligent vehicles, computation offloading has been proposed. It transfers communication, computation, and storage tasks to MEC servers situated around intelligent vehicles [17]. Existing computation offloading schemes mainly focus on computation efficiency [17–19], resource allocation [20,21], or decision-making optimization [11] task. While these schemes lay the foundation for offloading computationally intensive tasks to MEC servers, they often lack adequate security considerations, leading to a gap in verifiable computation offloading for batch modular exponentiation.

2.2. Secure outsourcing algorithm for modular exponentiation

The secure outsourcing algorithm for modular exponentiation can be categorized into single and dual-server models. Dual-server model assumes that there is no complicity risk between cloud servers [22–25]. This assumption, complex to implement in real-world applications, is vulnerable to collusion attacks between servers. Therefore, we mainly consider the single-server model, which is first proposed in 2006 by Dijk et al. [26]. Recently, numerous algorithms have been proposed to improve checkability rate [5,27,28]. In 2016, Ding et al. [3] proposed a modular exponentiation outsourcing scheme with checkability rate close to $\frac{119}{120}$, especially when batch size $n = 1$, which is rather higher than before. Thereafter, Su et al. [4], in 2020, expanded Ding's method, optimized the logical split, and changed the modulus of the algorithm to a composite number. Recent schemes including SoRSA [6] and EP-Exp [7] assume that bases in computation tasks are ciphertext and lack the consideration of security of bases. The checkability rate of these methods is still far from 1, and it can result in certain security risks. Meanwhile, many of these schemes concurrently present outsourcing algorithms for u^a. Nevertheless, a single modular exponentiation outsourcing algorithm represents a specific instance of batch modular exponentiation outsourcing with batch size $n = 1$.

2.3. Fair computation

Recently, blockchain and smart contracts have been proposed to address these fairness issues [29]. Smart contracts can provide a secure solution for participants to execute contracts on Ethereum, essentially being executable code with correctness, transparency, and immutability [30]. Although there are some studies utilizing smart contract to fulfill fair computation, they either rely on the assumption that the client and the cloud are honest [31], or utilize smart contract conducting complex computation tasks [29,32,33]. However, in standard blockchain systems such as Ethereum, users are typically charged gas fees based on the complexity of the computational task running in the smart contract. The gas fees for smart contracts are recorded in the fee table EIP150 [34]. Generally, the cost of Ethereum is high, and considering that modular exponentiation is an expensive computation task, Existing schemes may increase the financial burden on users.

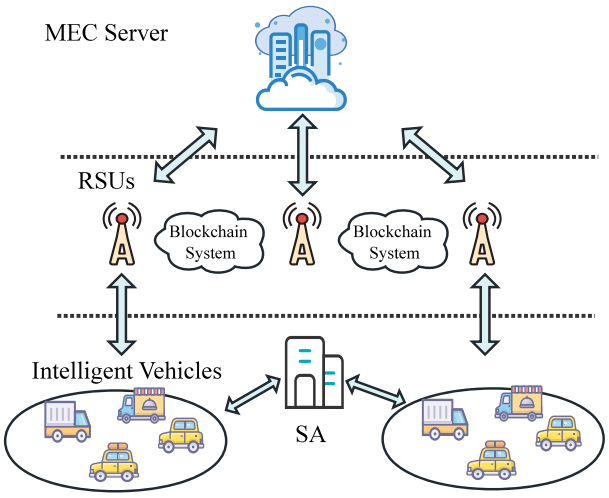

Fig. 2. The architecture of system model.

3. Fair computation offloading for batch modular exponentiation scheme in IoV

3.1. System model

As illustrated in Fig. 2, the fair computation offloading for batch modular exponentiation in IoV mainly comprises four entities: Service Agency (SA), Intelligent Vehicle (IV), Roadside Unit (RSU), and MEC Server.

SA: It is an honest entity. It provides the intelligent vehicle with the initialized bases u_i and modulus N of the batch modular exponentiation task $\prod_{i=1}^{n} u_i^{a_i} \bmod N$, and its communication with intelligent vehicles is based on secure channels.

IV: It is a resource-limited entity. It does not trust the MEC server, but it wants to offload some requests a_i to the MEC server, where $i \in \{1, \ldots, n\}$. Furthermore, it may try to get the result without paying by intentionally saying that the cloud's computation result is wrong.

RSU: It is an untrusted entity, which serves as a full node of the blockchain. It provides verifiable services to guarantee the integrity of the result.

MEC Server: It is a powerful entity deployed at the network's edge with adequate computation resources, which is responsible for performing the computation offloading tasks for the intelligent vehicle. Similar to the intelligent vehicle, it is also a profit-driven entity. It would like to get the reward from the intelligent vehicle without performing the computation.

A fair computation offloading for batch modular exponentiation (MExpm) in IoV consists of the following algorithms.

$(Params, RK) \leftarrow Setup(1^{\lambda}, u_1, \ldots, u_n, N)$. Given a security parameter λ, the bases $u_1, \ldots, u_n$ and N, SA invokes this algorithm to generate the public parameters $Params$ and the recovery key RK, where u_i and N are the base and modulus for modular exponentiation tasks.

$(TK, VK, Aux) \leftarrow KeyGen(a_1, \ldots, a_n, Params)$. On inputting the exponents $a_1, \ldots, a_n$ and the public parameters $Params$, IV runs this algorithm to generate the evaluation key TK for performing the computation task, witness generation key VK and auxiliary information Aux. It is worth noting that this algorithm can be carried out entirely *offline* before the online phase, so it does not introduce additional latency during computation outsourcing. The input a_i is the exponent of u_i, where $i \in \{1, \ldots, n\}$.

$(\sigma_{\mathrm{E}}, \pi_E) \leftarrow Compute(TK, VK)$. On inputting the evaluation key TK and witness generation key VK, the MEC server performs this algorithm to produce the encoding result σ_{E} and witness π_E.

Table 2
Notations.

Symbols	Descriptions
$\{u_1, u_2, \ldots, u_n\}$	Computation bases
λ	Security Parameter
p	512-bit prime integer
N	512-bit prime integer
L	A composite integer $L = pN$
k	A random integer
τ	A composite integer $\tau = kN$
$\{y_1, y_2, \ldots, y_n\}$	Bases after secure obfuscation
$(k_i, g^{k_i}), i \in \{1, 2, 3, 4\}$	Random pairs generated by RandN algorithm
$\{a_1, a_2, \ldots, a_n\}$	Computation exponents
$\phi(\cdot)$	Euler's function
$\{w_i, z_1, \delta_1, m_i\}, i \in \{1, \ldots, n\}$	Computation tasks after logical division
$r \in \{2, \ldots, N\}$	Random integer
$\xi \in \{1, \ldots, n\}$	Random index
d	Modular Multiplicative Inverse of a_ξ
$\{w'_i, z_2, \delta_2, m'_i\}, i \in \{1, \ldots, n\}$	Verification tasks after logical division
$\{\sigma_E, \pi_E\}$	Computation results returned by MEC server

$\{0/1, \sigma_{\mathrm{E}}\} \leftarrow Verify(\sigma_{\mathrm{E}}, \pi_E, Aux)$. On inputting the encoding result σ_{E}, the witness π_E and auxiliary information Aux, the RSU runs this algorithm to check whether the MEC server returns a correct result utilizing smart contract. If not, it outputs 0, 1 and σ_{E} otherwise.

$Result \leftarrow Recovery(\sigma_{\mathrm{E}}, RK)$. On inputting σ_{E} and recovery key RK, the intelligent vehicle runs this algorithm to decode the true result $Result$.

3.2. Overview of construction and notations

Similar to [3], the bases and exponents are protected using logical split. A recovery algorithm is also involved to protect the confidentiality of the final result. At the setup phase, we utilize the secure obfuscation technique to hide the modulus N and bases u_i. In *Setup* step (a), only the masked modulus $L = p \cdot N$ is sent to MEC server, so the MEC server cannot get any information about N without the mask factor p chosen and kept privately by the User. To prevent the MEC server from learning the original bases u_i, we apply a modular obfuscation technique by embedding each base into a larger modular space (i.e., Eq. (1)) in *Setup* step (b). Since k and p are sampled uniformly, the adversarial MEC server cannot recover them. The original computation offloading task $\prod_{i=1}^{n} u_i^{a_i} \mod N$ is converted into $\prod_{i=1}^{n} y_i^{a_i} \mod L$. The privacy of the exponent a_i is ensured by the logical split, where $a_i = \delta_1 \cdot z_i + m_i \bmod \phi(L)$. Since the standard integer factorization assumption holds, the adversary cannot derive their factors p and N from L. Without factors p and N, it is infeasible to compute $\phi(L) = (p-1)(N-1).\phi(L) = (p-1) \cdot (N-1)$. As a result, the reduction modulo $\phi(L)$ effectively hides the underlying value, which makes it infeasible to recover a_i from $\delta_1 \cdot z_i + m_i \bmod \phi(L)$. Furthermore, the malicious adversary learns nothing about the final computation result without the recovery key. A detailed description of the notations used in MExpm can be found in Table 2.

3.3. Detailed construction

1. $\boldsymbol{Setup}(\mathbf{1}^\lambda, \boldsymbol{u_1}, \ldots, \boldsymbol{u_n}, \boldsymbol{N})$: This algorithm is run by SA. Given a security parameter λ and a 512-bit prime integer N, SA works as follows:

(a) SA generates a 512-bit prime integer p, and computes $L = pN$.

(b) SA uniformly chooses k from Z_N and computes $\tau = kN$. For any $i \in \{1, 2, \ldots, n\}$, SA sets y_i as follows:

$$y_i = u_i + \tau \mod L \tag{1}$$

(c) SA sets $Params = \{L, y_i\}$ and $RK = \{N\}$, where RK is transmitted via a secure channel between SA and IV.

2. $\boldsymbol{KeyGen}(\boldsymbol{a_1}, \ldots, \boldsymbol{a_n}, \boldsymbol{Params})$: This algorithm is executed by the IV to construct the evaluation key TK for performing the computation task, the witness generation key VK, and auxiliary information Aux. Notably, the IV can execute this procedure in an *offline* manner, thereby avoiding additional delays in the online authentication or verification phase. This algorithm works as follows:

(a) IV parses $Params$ as $\{L, y_i\}$ and the input exponents $a_i \in \mathbb{Z}^*_{\phi(L)}$.

(b) IV runs RandN program [35] four times to generate four blinding pairs (k_1, g^{k_1}), (k_2, g^{k_2}), (k_3, g^{k_3}), (k_4, g^{k_4}) and sets:

$$\begin{aligned} &v_1 = g^{k_1} \bmod L, v_2 = g^{k_2} \bmod L, \\ &v_3 = g^{k_3} \bmod L, v_4 = g^{k_4} \bmod L. \end{aligned} \tag{2}$$

where $g \in \mathbb{Z}^*_L$ and its order is $\phi(L)$.

(c) IV performs logical split to compute w_i, z_1, δ_1, and m_i such that

$$\begin{aligned} w_i &= y_i / v_1 \pmod L, \\ k_1 \left(a_1 + a_2 + \cdots + a_n\right) &= k_3 + \delta_1 z_1 \pmod{\phi(L)}, \\ a_i &= \delta_1 z_1 + m_i \pmod{\phi(L)}. \end{aligned} \tag{3}$$

(d) IV chooses two random integers $r \in \{2, \ldots, N\}$ and $\xi \in \{1, \ldots, n\}$ and computes d, where $a_\xi d \equiv 1 \pmod{\phi(L)}$.

(e) IV computes w'_i, z_2, δ_2, and m'_i such that

$$\begin{aligned} w'_i &= y_i / v_2 \pmod L, \\ k_2 \left(a_1 + a_2 + \cdots + a_n\right) &= k_4 + \delta_2 z_2 \pmod{\phi(L)}, \\ a_i &= \delta_2 z_2 + m'_i \pmod{\phi(L)}. \end{aligned} \tag{4}$$

Especially when $i = \xi$, we have $y'_\xi = y_\xi r^d \pmod L$ and $w'_\xi = y'_\xi / v_2 \pmod L$, where $\xi \in [1, n]$ is a random integer.

(f) IV sets $TK = \{(g \prod_{i=1}^{n} w_i, z_1), (w_i, m_i)_{i \in [n]}\}$, $VK = \{(g \prod_{i=1}^{n} w'_i, z_2), (w'_i, m'_i)_{i \in [n]}\}$ and $Aux = \{rv_3, v_4, \delta_1, \delta_2\}$, where $\delta_1, \delta_2 \in \mathbb{Z}^*_{\phi(L)}$. The pseudo code of the key generation procedure can be found in Algorithm 1.

Algorithm 1: KeyGen Algorithm

Input: Exponents $a_1, \cdots, a_n \in \mathbb{Z}^*_{\phi(L)}$, public parameters $Params = \{L, y_i\}$
Output: Evaluation key TK, verification key VK, auxiliary info Aux

```
Parse Params as {L, y_i} ;                                   // Step (a)
Run RandN algorithm four times to get
  (k_1, g^{k_1}), (k_2, g^{k_2}), (k_3, g^{k_3}), (k_4, g^{k_4});   // Step (b)
Compute v_1 = g^{k_1} mod L, v_2 = g^{k_2} mod L, v_3 = g^{k_3} mod L,
  v_4 = g^{k_4} mod L;                                       // Compute Equation 2
Compute k_1(a_1 + ··· + a_n) = k_3 + δ_1 z_1 mod φ(L) ;      // Equation 3
for i ← 1 to n do
    Compute w_i = y_i / v_1 mod L ;                          // Equation 3
    Compute a_i = δ_1 z_1 + m_i mod φ(L);                    // Equation 3
Sample r ∈ {2, ··· , N} and ξ ∈ {1, ··· , n} randomly ;      // Step (d)
Compute d = a_ξ^{-1} mod φ(L);                               // Step (d)
Compute k_2(a_1 + ··· + a_n) = k_4 + δ_2 z_2 mod φ(L);       // Equation 3
for i ← 1 to n do
    Compute w'_i = y_i / v_2 mod L ;                         // Equation 3
    Compute a_i = δ_2 z_2 + m'_i mod φ(L);                   // Equation 3
Randomly Select ξ ∈ [1, n], and Update y'_ξ = y_ξ · r^d mod L,
  w'_ξ = y'_ξ / v_2 mod L ;                                  // Step (e)
Set TK = {(g · ∏_{i=1}^n w_i, z_1), (w_i, m_i)_{i∈[n]}};     // Step (f)
Set VK = {(g · ∏_{i=1}^n w'_i, z_2), (w'_i, m'_i)_{i∈[n]}};  // Step (f)
Set Aux = {rv_3, v_4, δ_1, δ_2};                             // Step (f)
return TK, VK, Aux;
```

3. $\boldsymbol{Compute(TK, VK)}$: This algorithm is run by MEC server to generate encoding result σ_E and witness result π_E. The MEC server works as follows:

(a) MEC server parses TK as $\{(g\prod_{i=1}^{n} w_i, z_1), (w_i, m_i)_{i\in[n]}\}$ and VK as $\{(g\prod_{i=1}^{n} w_i', z_2), (w_i', m_i')_{i\in[n]}\}$, and then sets $\gamma_i = (w_i)^{m_i}$ and $\gamma_i' = (w_i')^{m_i'}$ for any $i \in \{1, \ldots, n\}$, respectively.

(b) MEC server sets $Q_0 = \left(g\prod_{i=1}^{n} w_i\right)^{z_1}$ and $Q_1 = \left(g\prod_{i=1}^{n} w_i'\right)^{z_2}$.

(c) MEC server sets $\sigma_E = \{Q_0, (\gamma_i)_{i\in[n]}\}$ and $\pi_E = \{Q_1, (\gamma_i')_{i\in[n]}\}$.

4. $\boldsymbol{Verify(\sigma_E, \pi_E, Aux)}$: The algorithm is run by RSU to check the correctness of the result returned by the MEC. The RSU works as follows:

(a) Upon receiving the encoding result σ_E and witness π_E, it first parses them as $\{Q_0, (\gamma_i)_{i\in[n]}\}$ and $\{Q_1, (\gamma_i')_{i\in[n]}\}$, respectively. (b) it parses auxiliary information Aux as $\{rv_3, v_4, \delta_1, \delta_2\}$.

(c) RSU utilizes smart contract to compute $\eta = \left(Q_0\right)^{\delta_1}$ and then check whether the following equation holds:

$$rv_3 \cdot \eta \cdot \prod_{i=1}^{n} \gamma_i = v_4 \cdot \left(Q_1\right)^{\delta_2} \cdot \prod_{i=1}^{n} \gamma_i' \pmod{L} \tag{5}$$

If not, the smart contract outputs 0 and aborts. Otherwise, outputs 1 and sets $\sigma_E = \{rv_3\eta\prod_{i=1}^{n}\gamma_i\}$. The verification logic of the smart contract can be found in Algorithm 2.

Algorithm 2: Verification Logic of Smart Contract

Input:
$Q_0, Q_1 \in \mathbb{Z}_L$; $(\gamma_i)_{i\in[n]}, (\gamma_i')_{i\in[n]} \in \mathbb{Z}_L$
Scalars: $rv_3, v_4, \delta_1, \delta_2 \in \mathbb{Z}_L$
Output: Boolean flag indicating verification result

```
η ← Q_0^{δ_1} mod L ;                                  // Compute η
prodGamma ← 1;
for i ← 1 to n do
    prodGamma ← prodGamma · γ_i mod L;                 // Accumulate product of γ_i
prodGammaPrime ← 1;
for i ← 1 to n do
    prodGammaPrime ← prodGammaPrime · γ_i' mod L;      // Accumulate product of proofs γ_i'
lhs ← rv_3 · η · prodGamma mod L;                      // Left-hand side of the equality
rhs ← v_4 · Q_1^{δ_2} · prodGammaPrime mod L;          // Right-hand side of the equality
return (lhs == rhs);                                   // Return true if verification passes
```

5. $\boldsymbol{Recovery(\sigma_E, RK)}$: The algorithm is run by IV to recover the encoding result σ_E to the true result $Result$.

(a) IV parses RK as $\{N\}$ and σ_E as $\{rv_3\eta\prod_{i=1}^{n}\gamma_i\}$, where $\eta = \left(g\prod_{i=1}^{n} w_i\right)^{z_1\delta_1}$.

(b) IV recovers the final computation result $Result$ as follows:

$$\begin{aligned} Result &= rv_3\eta\prod_{i=1}^{n}\gamma_i \cdot r^{-1} \pmod{L} \\ &= g^{k_3}\left(g\prod_{i=1}^{n} w_i\right)^{z_1\delta_1}\prod_{i=1}^{n} w_i^{m_i} \pmod{N} \end{aligned} \tag{6}$$

3.4. An illustrative

We now provide a toy example to further illustrate MExpm. MExpm performs the following procedures. The original modular exponentiation performs the following procedures.

1. *Setup*: Generate two distinct secure primes $p = 13, N = 11$. Compute $L = p \cdot N = 143$. Then generate $u = 128, a = 79$.
2. *Compute*: Locally compute result $Result = 128^{79} \mod 43 = 8$.

The proposed MExpm consists of the following procedures.

1. *Setup*: SA generates $p = 13, N = 11$ and computes $L = p \cdot N = 143$. Then SA generates base $u = 128$ and utilizes random integer $k = 5$ to compute $y = u + k \cdot N \mod L = 40$.
2. *KeyGen*: Intelligent Vehicle runs $RandN$ algorithm to obtain $(63, 125), (42, 25), (52, 113), (82, 69)$, $g = 71$ and compute ${v_1}^{-1} = 125^{-1} \mod L = 135, {v_2}^{-1} = 25^{-1} \mod L = 103$. Then it generates a computation task $a = 79$. Thereafter, intelligent vehicle generate random integers $\delta_1 = 11, \delta_2 = 109, r = 7$ and compute $d = a^{-1} \mod \phi(L) = 79, r^d \mod L = 19$. Finally, intelligent vehicle utilizes Eqs. (3) and (4) to conduct logical split, then obtain $w = 109, gw = 17, w' = 59, gw' = 42, \delta_1 z_1 = 57, z_1 = 55, \delta_2 z_2 = 78, z_2 = 44, m = 74, m' = 83, rv_3 = 76$.
3. *Compute*: MEC server receives the offloading tasks and compute $(gw)^{z_1} = 59^{55} \mod 143 = 43, (gw')^{z_2} = 42^{44} \mod 143 = 126, w^m = 12, {w'}^{m'} = 119$.
4. *Verify*: Smart contract is called to verify $Left = 76 \cdot 12 \cdot 43^{11} \mod 143 = 111$ and $Right = 69 \cdot 126^{109} \cdot 119 \mod 143 = 111$.
5. *Recovery*: Intelligent Vehicle computes $r^{-1} \mod L = 41$, then obtain $Result = 111 \cdot 41 \mod 11 = 8$.

4. Theoretical analysis

4.1. Correctness

To prove the correctness, we need to argue that the returned results by the MEC server can pass the verification algorithm and the intelligent vehicle can recover the final result if all entities involved are honest.

For the first part, we mainly argue it based on Eq. (5). That is, we prove that the σ_E and π_E can pass the $verify$ algorithm when MEC server is honest and follows all algorithms mentioned above.

Based on Eq. (4), the right-hand side (RHS) of Eq. (5) can be expressed as:

$$\begin{aligned} RHS &= v_4 \cdot \left(Q_1\right)^{\delta_2} \cdot \prod_{i=1}^{n}\gamma_i' \pmod{L} \\ &= g^{k_4}\left(g\prod_{i=1}^{n} w_i'\right)^{z_2\delta_2}\prod_{i=1}^{n} {w_i'}^{m_i'} \pmod{L} \\ &= g^{k_4+z_2\delta_2}\left(\prod_{i=1}^{n} w_i'\right)^{z_2\delta_2}\prod_{i=1}^{n} {w_i'}^{m_i'} \pmod{L} \\ &= g^{k_2(a_1+a_2+\cdots+a_n)}\prod_{i=1}^{n} {w_i'}^{m_i'+\delta_2 z_2} \pmod{L} \\ &= g^{k_2(a_1+a_2+\cdots+a_n)}\prod_{i=1}^{n} {w_i'}^{a_i} \pmod{L} \\ &= \prod_{i=1}^{n} g^{k_2 a_i}{w_i'}^{a_i} \pmod{L} \\ &= \prod_{i=1}^{n} v_2^{a_i}{w_i'}^{a_i} \pmod{L} \\ &= \prod_{i=1}^{n} {y_i'}^{a_i} \pmod{L} \\ &= \prod_{i=1}^{\xi-1} y_i^{a_i} \cdot y_\xi' r^{d a_\xi} \cdot \prod_{i=\xi+1}^{n} y_i^{a_i} \pmod{L} \end{aligned} \tag{7}$$

Since $a_\xi d \equiv 1 (\bmod\ \phi(L))$, we always have $y'^{r^{da_\xi}}_{\xi} = r y_\xi \bmod L$. Based on Eq. (3), we can get:

$$\begin{aligned} RHS &= r\prod_{i=1}^{n} y_i^{a_i} \ (\bmod\ L) \\ &= r\prod_{i=1}^{n} g^{k_1 a_i} w_i^{a_i} \ (\bmod\ L) \\ &= r g^{k_1(a_1+a_2+\cdots+a_n)} \prod_{i=1}^{n} w_i^{m_i+\delta_1 z_1} \ (\bmod\ L) \\ &= r g^{k_3+z_1\delta_1} \left(\prod_{i=1}^{n} w_i\right)^{z_1\delta_1} \prod_{i=1}^{n} w_i^{m_i} \ (\bmod\ L) \\ &= r g^{k_3} \left(g\prod_{i=1}^{n} w_i\right)^{z_1\delta_1} \prod_{i=1}^{n} w_i^{m_i} \ (\bmod\ L) \\ &= r v_3 \left(Q_0\right)^{\delta_1} \prod_{i=1}^{n} \gamma_i \ (\bmod\ L) \\ &= r v_3 \cdot \eta \cdot \prod_{i=1}^{n} \gamma_i \ (\bmod\ L) \end{aligned} \tag{8}$$

Obviously, according to Eq. (8), if the MEC server and intelligent vehicle IV are honest and follow all procedures described above, the encoding result σ_E and witness result π_E can always pass the $Verify$ algorithm.

Second, we will argue that the encoding result σ_E can be decoded to the actual result $Result$. In this section, we mainly rely on Eq. (1), Eq. (2), Eq. (3) and (6). The σ_E can be parsed and computed as follows:

$$\begin{aligned} Result &= r v_3 \eta \prod_{i=1}^{n} \gamma_i \cdot r^{-1} \ (\bmod\ N) \\ &= g^{k_3} \left(g\prod_{i=1}^{n} w_i\right)^{z_1\delta_1} \prod_{i=1}^{n} w_i^{m_i} \ (\bmod\ N) \\ &= g^{k_3+z_1\delta_1} \left(\prod_{i=1}^{n} w_i\right)^{z_1\delta_1} \prod_{i=1}^{n} w_i^{m_i} \ (\bmod\ N) \\ &= g^{k_1(a_1+a_2+\cdots+a_n)} \prod_{i=1}^{n} w_i^{m_i+\delta_1 z_1} \ (\bmod\ N) \\ &= \prod_{i=1}^{n} g^{k_1 a_i} w_i^{a_i} \ (\bmod\ N) \\ &= \prod_{i=1}^{n} y_i^{a_i} (\bmod N) \\ &= \prod_{i=1}^{n} \left(u_i + kN\right)^{a_i} \ (\bmod\ N) \\ &= \prod_{i=1}^{n} u_i^{a_i} \ (\bmod\ N) \end{aligned} \tag{9}$$

Obviously, when Eq. (9) holds, the correctness of the algorithm $Recovery$ is guaranteed and the proof is completed.

4.2. Security analysis

In this section, we demonstrate the privacy for computation offloading results. In MExpm, we firstly convert $\prod_{i=1}^{n} u_i{}^{a_i} (\bmod\ N)$ into $\prod_{i=1}^{n} y_i{}^{a_i} (\bmod\ L)$, then the exponents a_i are transformed into $\delta_1 z_1 + m_{i,i\in[n]}$ and $\delta_2 z_2 + m'_{i,i\in[n]}$. The public information in our scheme are $\{Params, TK, VK, Aux, \sigma_E, \pi_E\}$, and the adversaries cannot obtain any information about secret information $\{u_{i,i\in[n]}, a_{i,i\in[n]}, RK, Result\}$.

Theorem 1. *When the MEC server cheats the client, the misbehavior can be detected with checkability rate* $1 - \frac{n^2}{(4n^2+6n+2)(N-2)}$.

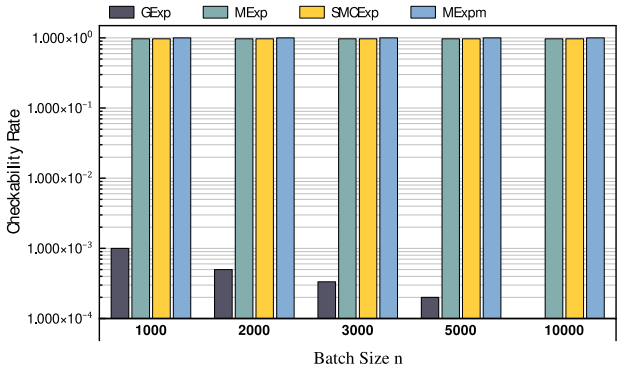

Fig. 3. Comparison of checkability rate.

Proof. If the malicious MEC server deceives the intelligent vehicle IV successfully, the following equation will hold.

$$t \cdot r v_3 \eta \prod_{i=1}^{n} \gamma_i = t \cdot v_4 \left(Q_1\right)^{\delta_2} \prod_{i=1}^{n} \gamma'_i \tag{10}$$

The corresponding encoding result will be decoded by IV as follows:

$$Result = t \cdot v_3 \eta \prod_{i=1}^{n} \gamma_i (\bmod\ N) \tag{11}$$

Since the MEC server could not gain access to the values of δ_1 and δ_2, it cannot obtain the correct values of η and $\left(Q_1\right)^{\delta_2}$. Therefore, the MEC server can only turn to other n pairs to cheat the intelligent vehicle, then it needs to determine the correct meanings of $2n+2$ sub-tasks to obtain pairs: (w_h, m_h) and (w'_j, m'_j). Since the sending order of these $2n+2$ pairs is random, the MEC server is unaware of the specific meanings of each pair. Therefore, it needs to find (w_h, m_h) and (w'_j, m'_j) among the $2n+2$ pairs. The probability of this operation is $\frac{n}{2n+2}\frac{n}{2n+1}$. Additionally, for successful deception, the MEC server needs to determine the value of r, where $r \in \{2, \ldots, N\}$. Thus, the probability of finding the correct r is $\frac{1}{N-2}$. Subsequently, the MEC server generates a random number t and returns $t w_h^{m_h}$ and $t {w'_j}^{m'_j}$. We denote the malicious server successfully determining the correct meaning of (w_h, m_h) and (w'_j, m'_j) as event E_1, and denote the determining the value of r as event E_2. We have: $\Pr(E_1) = \frac{n}{2n+2}\frac{n}{2n+1}$ and $\Pr(E_2) = \frac{1}{N-2}$.

Therefore, the probability of the intelligent vehicle being deceived is: $\Pr(E_1 \cap E_2) = \Pr(E_1)\Pr(E_2) = \frac{n^2}{(4n^2+6n+2)(N-2)}$. Therefore, the checkability rate of our proposed scheme MExpm is: $1 - \frac{n^2}{(4n^2+6n+2)(N-2)}$.

5. Simulation

In this section, we evaluate the performance of our proposed scheme MExpm by comparing it with the most advanced and representative modular exponentiation offloading schemes reported in recent literature. Specifically, we consider MExp [3] and SMCExp [4] for secure batch modular exponentiation, as well as SoRSA [6] and EPExp [7] for single modular exponentiation. These schemes reflect the latest advancements in both batch-oriented and single-operation settings and are widely recognized as benchmarks in the field. Notably, all of these algorithms are incorporated as baselines in our experimental evaluation, covering key performance indicators such as local computation time, end-to-end execution latency, communication overhead, and gas consumption. Since MExpm is designed for batch modular exponentiation, while SoRSA and EPExp are designed solely for single modular exponentiation, for fairness, we also conduct the comparison for the case where the batch size $n = 1$.

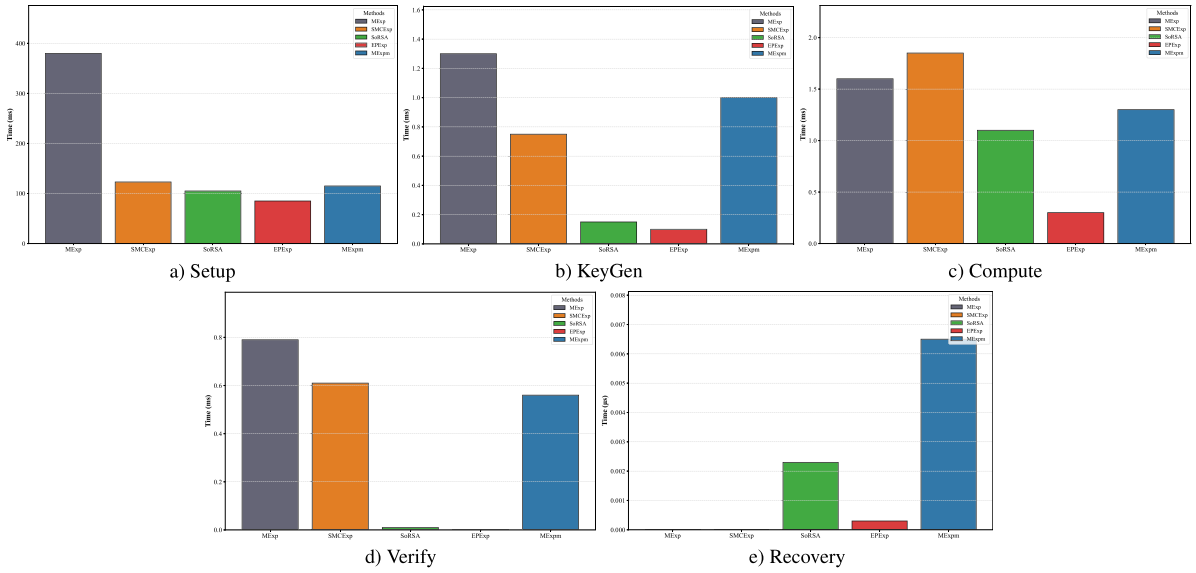

Fig. 4. Comparsion of time cost in $u^a \mod N$.

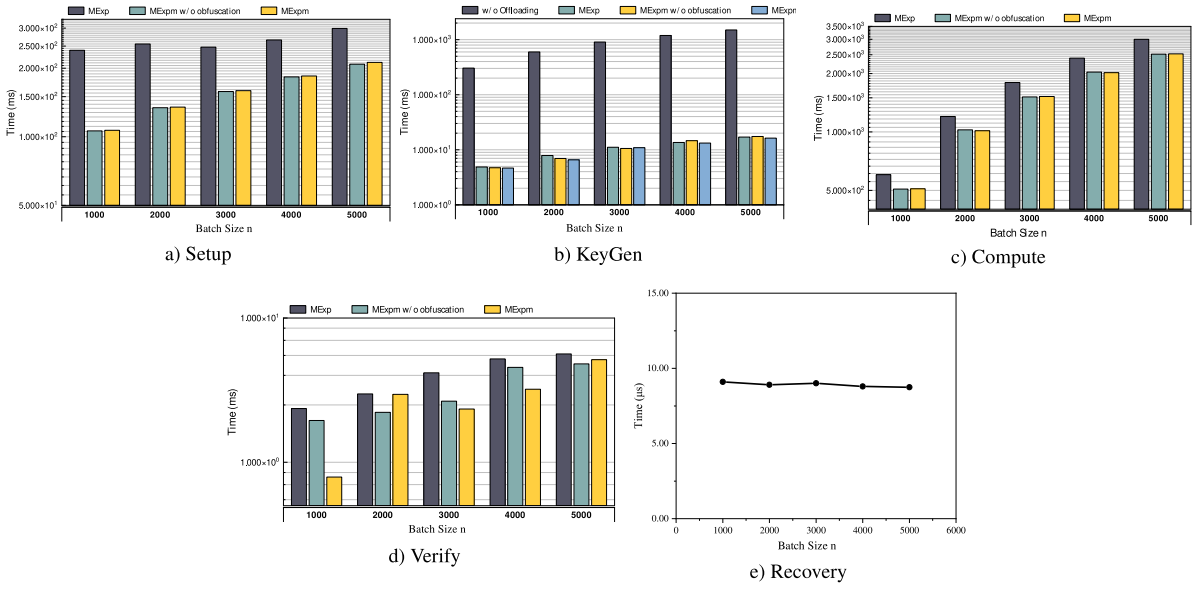

Fig. 5. Comparison of time cost in $\prod_{i=1}^{n} u_i^{a_i} \mod N$.

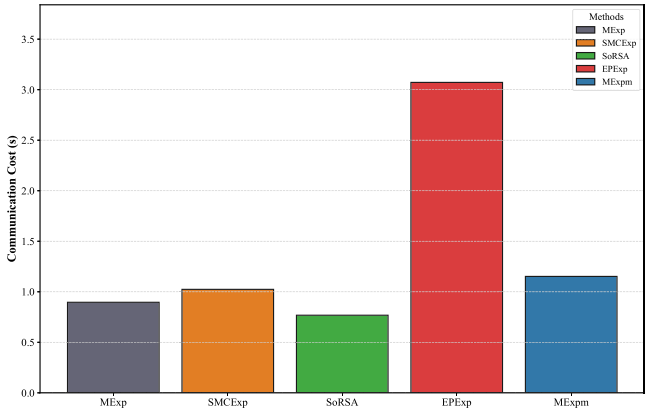

Fig. 6. Comparison of communication cost.

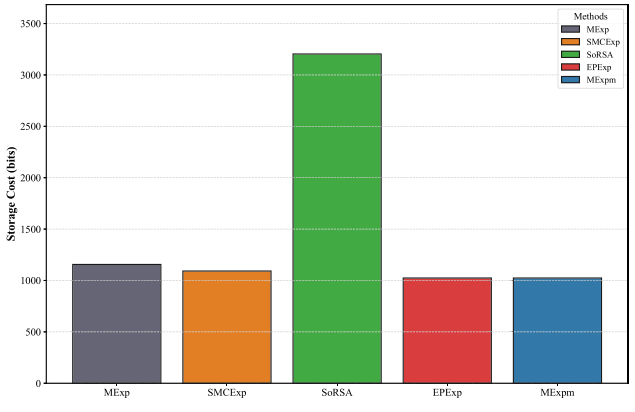

Fig. 7. Comparison of storage cost.

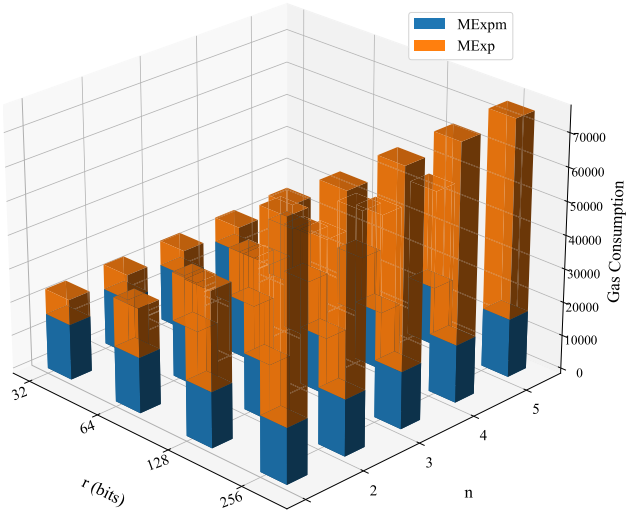

Fig. 8. Comparison of Gas Consumption in Verify Algorithm when the size of r is larger than 32 bits.

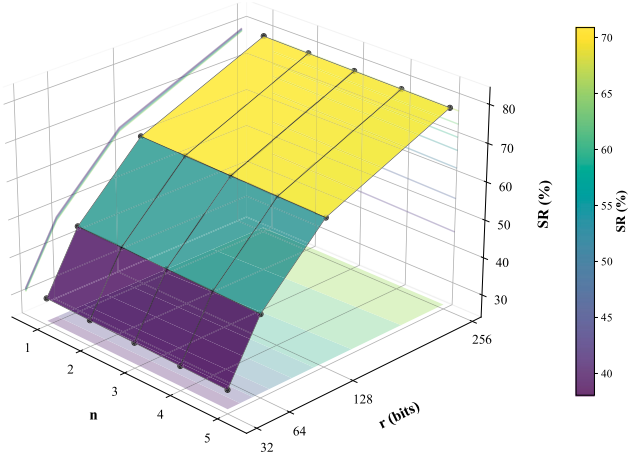

Fig. 9. The relative saving ratio of MExp and MExpm.

5.1. Experimental setting

We implemented MExpm and MExp for batch modular exponentiation offloading, MExp, SMCExp, EPExp, SoRSA and MExpm for single modular exponentiation offloading using Python 3.8, along with the PyCryptodome and GNU Multiple Precision (gmpy2 version 2.1.5) library. All simulation experiments were conducted on the same Windows machine equipped with an Intel Core TM i9-13900HX processor (running at 2.20 GHz) and 16 GB of memory. We perform each algorithm 100 times and then computed the mean of its time cost. The size of prime numbers selected in MExpm are all 512 bits, meaning the number L is 1024 bits. For MExp and methods without offloading, we randomly generate a pair of 1024-bit prime numbers. In our simulation, 'MExpm w/o obfuscation' denotes MExpm without the secure obfuscation operation and 'w/o offloading' indicates the local execution of the modular exponentiation operation.

5.2. Evaluation metrics

To comprehensively evaluate MExpm, we assess its performance across five dimensions: privacy, computation cost, blockchain resource usage, communication overhead, and storage overhead. We quantify privacy using the checkability rate. Computation cost is measured by the execution time (in milliseconds), where longer runtimes correspond to higher resource consumption. We assess blockchain resource usage on the Ethereum simulation platform, Remix, using gas consumption as the metric. Communication overhead is defined as the total data transmitted during offloading. Storage overhead is quantified by the additional storage required on both the client and server sides.

5.3. Checkability

The details of checkability rate comparison of these four schemes are shown in Fig. 3. As n increases, our proposed scheme MExpm always maintains a high checkability rate close to 1, while the other three schemes gradually decrease. When $n = 1000$, the checkability rate of GExp is only $\frac{1}{1001}$. It means that a forged result can pass the verification algorithm with the probability $\frac{1000}{1001}$. Both MExp and SMCExp have the same checkability rate. However, when $n = 5000$, the checkability rate for these two schemes is only 0.975. Since MExpm uses a prime number N with size 512, its checkability rate is higher than 0.999.

5.4. Computation cost

5.4.1. Single modular exponentiation offloading

The comparsion results of single modular exponentiation offloading could be found in Fig. 4. Compared with MExp and SMCExp, MExpm demonstrates better performance either in $KeyGen$ algorithm or in $Setup$ algorithm. Particularly in $Verify$ algorithm, MExpm outperforms these competitors. When it comes to SoRSA and EPExp, whose security assumption is rather simple and cannot applied in real-world scenarios, it seems unfair to compare them with schemes for batch modular exponentiation with higher security standard.

5.4.2. Batch modular exponentiation offloading

Fig. 5 compares the computational cost of batch modular exponentiation offloading. MExpm consistently requires fewer resources than MExp across all phases. Although MExpm adds a recovery phase, it consists of a single modular inversion—incurring a fixed and negligible overhead.

5.5. Communication and storage cost

To simulate a low-bandwidth network environment, we set the transmission rate to 1 Kbps. Fig. 6 shows the communication cost of the all competitors and MExpm in terms of the time cost of transmission. For fair comparison, all schemes employ 1024-bit modulus. For EPExp and SoRSA, whose authors assume that they only offload ciphertext to servers and do not need to take security of bases into consideration, and thus, they have lower communication cost compared with other schemes. Compared with MExp and SMCExp, MExpm shares the same communication cost in $Compute$ and $Verify$ algorithm. SMCExp shows the least communication cost in $KeyGen$ and $Setup$ algorithm. The results in Fig. 6 demonstrate that MExpm can deploy a more secure offloading strategy while with similar communication cost compared with other competitors. Fig. 7 shows the storage cost among all schemes, SoRSA needs to store n, q, p, C, k, t_1, t_2 to conduct verification and recovery, leading to the most demanding storage cost. EPExp demonstrates the best storage performance, while it lacks a consideration of a malicious MEC server. Among MExp and SMCExp, MExpm demonstrates the best performance in storage performance.

5.6. Gas consumption

The results of the gas consumption comparison are demonstrated in Fig. 8. It can be observed that as r increases, the gas consumption for both MExpm and MExp grows steadily. However, the gap between them widens significantly. For instance, when $n = 5$, the gas fee difference rises from 7,504 gas at $r = 32$ bits to 58,981 gas at $r = 256$ bits. Furthermore, the gas cost of MExpm's $Verify$ algorithm scales linearly with n and is largely unaffected by r, whereas MExp's verification cost increases with both r and n. This highlights MExpm's superior efficiency in reducing computational and financial burdens for intelligent vehicles, especially at larger scales. To provide a normalized view of these savings, we evaluate the relative saving ratio (SR) defined as

$$SR = \frac{G^{n,r}_{\text{MExp}} - G^{n,r}_{\text{MExpm}}}{G^{n,r}_{\text{MExp}}}$$

where $G^{n,r}_{\text{MExp}}$, $G^{n,r}_{\text{MExpm}}$ is the gas consumption of MExp and MExpm with same n and r respectively. As illustrated in Fig. 9, MExpm consistently achieves $SR > 0$ across all tested parameter, with observed savings ranging from approximately 30% to 70%. These results confirm that MExpm delivers substantial resource savings over MExp, reinforcing its scalability and economic advantages.

5.7. Economic analysis of gas savings

To further assess the practical impact of our scheme in real-world deployments, we provide an economic estimation of gas savings achieved by the proposed MExpm scheme compared with the representative baseline MExp, particularly in the context of blockchain-based smart contract verification. As shown in Fig. 8, the gas cost of each offloaded batch modular exponentiation result increases with both the batch size n and the bit length of the randomness parameter r. When $n = 1$ and bit length of r is 32 bits, MExp incurs 24,013 gas while MExpm requires only 16,509 gas, leading to a difference of 7504 gas per verification. The average gas price is approximately 30 Gwei (1 Gwei = 10^{-9} ETH), the ETH/USD exchange rate is approximately $\$4,000$, and 1 million gas cost approximately $\$120$ on Ethereum networks. Therefore, the savings can be translated as

$$\text{Gas Savings} = 7{,}504 \text{ gas} \times 0.03 \text{ USD}/1{,}000{,}000 \approx 0.90 \text{ USD}$$

Consider a practical usage scenario where each intelligent vehicle offloads batch modular exponentiation tasks 10 times per day (e.g., for authentication, key negotiation, digital signatures, etc.), the annual number of invocations is 10 (tasks/day)×365 (days/year) = 3,650 tasks/year. Thus, the total annual gas cost saving per vehicle is: 0.90 USD/task × 3,650 tasks/year ≈ **3285USD/year**. This estimation highlights the substantial economic benefits of MExpm when deployed at scale in large IoV systems. For a fleet of 10,000 vehicles, the projected gas savings could exceed 32.8 million USD annually.

5.8. Robustness evaluation against malicious MEC servers

To address potential security risks in practical deployments, we conducted robustness experiments simulating malicious Mobile Edge Computing (MEC) servers that deviate from the prescribed computation protocol. Such adversarial behaviors may include, but are not limited to:

- **Forged Results:** The MEC server deliberately returns computation results that deviate from the prescribed algorithm, thereby attempting to mislead the verifier regarding the correctness of the computation.
- **Partial Omission or Manipulation:** The MEC server selectively omits partial computation results or manipulates intermediate values with the intent of reducing its own computational workload.
- **Witness Tampering:** The MEC server alters or forges witnesses with the objective of deceiving the verifier and illegitimately passing the verification process.

In our simulation, the Intelligent Vehicle (IV) executes the KeyGen algorithm entirely offline to generate the evaluation key TK, witness generation key VK, and auxiliary information Aux. The TK and VK are then transmitted to the MEC server and the Roadside Unit (RSU), respectively. The RSU, acting as a lightweight verifier, executes the verification algorithm upon receiving computation results from the MEC server.

Experimental results demonstrate that our verification mechanism achieves a 100% detection rate for all injected malicious behaviors, with zero false positives under benign conditions. This confirms that the proposed scheme maintains strong security guarantees even in the presence of malicious MEC servers, thereby reinforcing its practicality for real-world V2X deployments.

5.9. Deployment feasibility in real-world V2X environments

The proposed scheme is designed for secure computation outsourcing in resource-constrained vehicular networks. In such scenarios, the primary metric for evaluation is whether the total computational cost incurred locally after outsourcing is significantly lower than that of fully local execution. Therefore, as is common in the literature on computation outsourcing, we perform all experiments — including the computation algorithm, verification algorithm, and a non-outsourced baseline — on the same hardware platform. This ensures a fair and reproducible comparison under identical computational conditions, thereby directly demonstrating the benefits of outsourcing.

In the proposed system, the Service Authority (SA) is responsible for handling the bulk of initialization tasks, such as modulus generation, base obfuscation, and parameter distribution. This design choice aligns with the practical division of labor in vehicular networks, reducing the computational burden on field devices. The Intelligent Vehicle (IV) executes the KeyGen algorithm to generate the evaluation key TK, witness generation key VK, and auxiliary information Aux. Crucially, the KeyGen algorithm can be performed entirely *offline* before the online phase, ensuring that no additional delay is introduced when initiating the outsourced computation. Once TK is generated, the IV transmits TK to the MEC server for performing the computation tasks.

In real deployments, RSUs are lightweight verification devices typically deployed at traffic intersections or along highways, where power supply and network connectivity can be unstable. To emulate these constraints, we configure the RSU role on a lightweight laptop and limit the network transmission rate to **1 Kbps**, thereby simulating a realistic low-bandwidth vehicular environment. Meanwhile, the Service Authority (SA) undertakes the bulk of initialization tasks, ensuring that RSUs and IVs remain computationally efficient during the online phase.

This deployment-oriented design, together with our simulation settings, ensures that the evaluation faithfully reflects real-world limitations while remaining reproducible. Consequently, the proposed scheme is shown to be both practically feasible and robust for secure computation outsourcing in V2X environments.

6. Conclusion and future work

In this paper, we propose MExpm, a secure and efficient computation offloading scheme for batch modular exponentiation in Vehicle-to-Everything (V2X) communications. Our proposed scheme addresses critical challenges in V2X systems, such as computational burden, latency, and privacy concerns, by leveraging Mobile Edge Computing (MEC) servers and blockchain technology. Our scheme achieves several significant improvements over existing methods. It ensures fairness in computation offloading by using smart contracts, provides high checkability to detect any misbehavior by MEC servers, and enhances privacy protection through secure obfuscation and logical split techniques. These features make MExpm particularly well-suited for various real-time applications in V2X systems. These include

- **Real-time Cryptographic Operations**: MExpm can offload resource-intensive cryptographic tasks, such as digital signatures and key exchanges, ensuring secure and efficient communication with reduced latency.
- **Safety-Critical Message Signature**: Offloading the computation of digital signatures (which rely on exponentiation) for emergency braking alerts and collision avoidance warnings, with on-chain smart contracts validating each signature to prevent tampering.
- **Privacy-Preserving Authentication**: The scheme guarantees the privacy of sensitive data, such as cryptographic bases and exponents, while allowing verification of computation results. This is essential for secure authentication in V2X communications, protecting both vehicles and infrastructure from malicious attacks.
- **Traffic Management Systems**: MExpm can be integrated into smart city infrastructures, supporting secure communication for traffic management, tolling systems, and other applications where privacy and computation efficiency are crucial.

Although MExpm significantly reduces computation resources compared to local execution, it introduces additional complexity due to enhanced security features. Future work should aim to design a more generalized verifiable computation offloading framework that optimizes the balance between security and computational efficiency.

CRediT authorship contribution statement

Sipeng Shen: Writing – review & editing, Writing – original draft, Methodology, Conceptualization. **Qiang Wang:** Writing – review & editing, Writing – original draft, Methodology. **Fucai Zhou:** Writing – review & editing, Supervision. **Jian Xu:** Writing – review & editing, Supervision. **Mingxing Jin:** Writing – review & editing.

Declaration of competing interest

The authors declare no competing interests.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants 62202090, 62173101, and 62372069, by Natural Science Foundation of Liaoning Province under Grant 2025-MS-046, by the Fundamental Research Funds for the Central Universities, China under Grant N2417006, and by Liaoning Collaboration Innovation Center for CSLE under Grant XTCX2024-015.

Data availability

Data will be made available on request.

References

[1] R. Sun, Y. Wen, N. Cheng, W. Wang, R. Chai, Y. Hui, Structural knowledge-driven meta-learning for task offloading in vehicular networks with integrated communications, sensing and computing, J. Inf. Intell. (2024).
[2] S. Yuan, Y. Fan, Y. Cai, A survey on computation offloading for vehicular edge computing, in: Proceedings of the 2019 7th International Conference on Information Technology: IoT and Smart City, 2019, pp. 107–112.
[3] Y. Ding, Z. Xu, J. Ye, K.-K.R. Choo, Secure outsourcing of modular exponentiations under single untrusted programme model, J. Comput. System Sci. 90 (2017) 1–13.
[4] Q. Su, R. Zhang, R. Xue, Secure outsourcing algorithms for composite modular exponentiation based on single untrusted cloud, Comput. J. 63 (8) (2020) 1271–1271.
[5] Y. Wang, Q. Wu, D.S. Wong, B. Qin, S.S. Chow, Z. Liu, X. Tan, Securely outsourcing exponentiations with single untrusted program for cloud storage, in: Computer Security-ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 2014 7-11. Proceedings, Part I 19, Springer, 2014, pp. 326–343.
[6] H. Zhang, J. Yu, C. Tian, L. Tong, J. Lin, L. Ge, H. Wang, Efficient and secure outsourcing scheme for rsa decryption in internet of things, IEEE Internet Things J. 7 (8) (2020) 6868–6881.
[7] Q. Hu, M. Duan, Z. Yang, S. Yu, B. Xiao, Efficient parallel secure outsourcing of modular exponentiation to cloud for iot applications, IEEE Internet Things J. 8 (16) (2020) 12782–12791.
[8] K. Zhou, M. Afifi, J. Ren, Expsos: Secure and verifiable outsourcing of exponentiation operations for mobile cloud computing, IEEE Trans. Inf. Forensics Secur. 12 (11) (2017) 2518–2531.
[9] Y. Saleem, F. Salim, M.H. Rehmani, Integration of cognitive radio sensor networks and cloud computing: A recent trend, in: Cognitive Radio Sensor Networks: Applications, Architectures, and Challenges, IGI Global, 2014, pp. 288–312.
[10] J. Wang, Y. Wang, H. Ke, Joint optimization for mec computation offloading and resource allocation in iov based on deep reinforcement learning, Mob. Inf. Syst. 2022 (2022).
[11] L. Yao, X. Xu, M. Bilal, H. Wang, Dynamic edge computation offloading for internet of vehicles with deep reinforcement learning, IEEE Trans. Intell. Transp. Syst. (2022).
[12] R. Gennaro, C. Gentry, B. Parno, Non-interactive verifiable computing: Outsourcing computation to untrusted workers, in: Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 2010 15-19. Proceedings 30, Springer, 2010, pp. 465–482.
[13] Y. Guan, H. Zheng, J. Shao, R. Lu, G. Wei, Fair outsourcing polynomial computation based on the blockchain, IEEE Trans. Serv. Comput. 15 (5) (2021) 2795–2808.
[14] H. Pagnia, F.C. Gärtner, et al., On the Impossibility of Fair Exchange Without a Trusted Third Party, Tech. Rep., Citeseer, 1999.
[15] A. Aloufi, P. Hu, Y. Song, K. Lauter, Computing blindfolded on data homomorphically encrypted under multiple keys: A survey, ACM Comput. Surv. 54 (9) (2021) http://dx.doi.org/10.1145/3477139.
[16] J. Zhang, Z.L. Jiang, P. Li, S.M. Yiu, Privacy-preserving multikey computing framework for encrypted data in the cloud, Inform. Sci. 575 (2021) 217–230, http://dx.doi.org/10.1016/j.ins.2021.06.017, https://www.sciencedirect.com/science/article/pii/S0020025521006083.
[17] M. Cui, S. Zhong, B. Li, X. Chen, K. Huang, Offloading autonomous driving services via edge computing, IEEE Internet Things J. 7 (10) (2020) 10535–10547.
[18] J. Zhou, F. Wu, K. Zhang, Y. Mao, S. Leng, Joint optimization of offloading and resource allocation in vehicular networks with mobile edge computing, in: 2018 10th International Conference on Wireless Communications and Signal Processing, WCSP, IEEE, 2018, pp. 1–6.
[19] S. Yang, A task offloading solution for internet of vehicles using combination auction matching model based on mobile edge computing, IEEE Access 8 (2020) 53261–53273.
[20] H. Xu, W. Huang, Y. Zhou, D. Yang, M. Li, Z. Han, Edge computing resource allocation for unmanned aerial vehicle assisted mobile network with blockchain applications, IEEE Trans. Wirel. Commun. 20 (5) (2021) 3107–3121.
[21] Y. Liu, H. Yu, S. Xie, Y. Zhang, Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks, IEEE Trans. Veh. Technol. 68 (11) (2019) 11158–11168.
[22] J. Kilian, Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12. 2005, Proceedings, vol. 3378, Springer, 2005.
[23] X. Ma, J. Li, F. Zhang, Efficient and secure batch exponentiations outsourcing in cloud computing, in: 2012 Fourth International Conference on Intelligent Networking and Collaborative Systems, IEEE, 2012, pp. 600–605.
[24] X. Chen, J. Li, J. Ma, Q. Tang, W. Lou, New algorithms for secure outsourcing of modular exponentiations, IEEE Trans. Parallel Distrib. Syst. 25 (9) (2013) 2386–2396.
[25] Y. Ren, N. Ding, X. Zhang, H. Lu, D. Gu, Verifiable outsourcing algorithms for modular exponentiations with improved checkability, in: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, 2016, pp. 293–303.
[26] M. Van Dijk, D. Clarke, B. Gassend, G.E. Suh, S. Devadas, Speeding up exponentiation using an untrusted computational resource, Des. Codes Cryptogr. 39 (2006) 253–273.
[27] J. Ye, J. Wang, Secure outsourcing of modular exponentiation with single untrusted server, in: 2015 18th International Conference on Network-Based Information Systems, IEEE, 2015, pp. 643–645.
[28] S. Li, L. Huang, A. Fu, J. Yearwood, Cexp: secure and verifiable outsourcing of composite modular exponentiation with single untrusted server, Digit. Commun. Netw. 3 (4) (2017) 236–241.
[29] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, A. van Moorsel, Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 211–227.
[30] J. Ellul, G.J. Pace, Runtime verification of ethereum smart contracts, in: 2018 14th European Dependable Computing Conference, EDCC, IEEE, 2018, pp. 158–163.

[31] M.R. Dorsala, V. Sastry, S. Chapram, Fair payments for verifiable cloud services using smart contracts, Comput. Secur. 90 (2020) 101712.
[32] S. Avizheh, M. Nabi, R. Safavi-Naini, K. M. Venkateswarlu, Verifiable computation using smart contracts, in: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, 2019, pp. 17–28.
[33] D. Xu, Y. Ren, X. Li, G. Feng, Efficient and secure outsourcing of modular exponentiation based on smart contract, Int. J. Netw. Secur. 22 (6) (2020) 934–944.
[34] G. Wood, et al., Ethereum: a secure decentralised generalised transaction ledger, (2014) 2017.
[35] S. Mahdavi-Hezavehi, Y. Alimardani, R. Rahmani, D. Rosaci, An efficient framework for a third party auditor in cloud computing environments, Comput. J. 63 (1) (2020) 1285–1297.