Cluster Computing (2024) 27:9679–9692
https://doi.org/10.1007/s10586-024-04472-w

Securing multi-client range queries over encrypted data

Jae Hwan Park[1] · Zeinab Rezaeifar[2] · Changhee Hahn[1]

Received: 26 November 2023 / Revised: 24 March 2024 / Accepted: 26 March 2024 / Published online: 26 April 2024

Abstract
Order-revealing encryption (ORE) allows secure range query processing over encrypted databases through a publicly accessible comparison function, while keeping other details concealed. Since parameter-hiding ORE (ASIACRYPT 2018) demonstrated improved privacy preservation at the cost of $\mathcal{O}(n^2)$ comparison operations, where n is the bit length of plaintexts, Lv et al. (ESORICS 2021) introduced an efficient ORE scheme that reduced the comparison operations to $\mathcal{O}(n)$, all while accommodating multiple clients. In this paper, we identify a vulnerability in Lv et al.'s ORE scheme, which we refer to as "*Query Reusability*." Exploiting this vulnerability, we develop an optimal query recovery attack. According to our experiment on the real-world datasets, our attack can recover a 64-bit plaintext query within a mere 83ms. We then propose msq-ORE, a multi-client secure range query ORE scheme that effectively mitigates the vulnerability while maintaining computational costs comparable to the state-of-the-art ORE scheme. Lastly, our performance analysis results show that the proposed scheme achieves efficacy.

Keywords Order-revealing encryption · Property-preserving hash · Secure query · Multi-client searchable encryption

1 Introduction

The concept of cloud computing offers numerous benefits to users, industries, and companies by enabling the storage and management of vast amounts of data in the cloud, which is accessible from anywhere in the world. While this outsourcing approach provides many advantages, it also raises concerns about data confidentiality [20, 23, 25]. With data breaches on the rise, such as the healthcare data breaches in the US, which affected over 40 million victims in 2022 and saw a 50% increase in 2023 [10], it is imperative to develop secure methods for managing data in the cloud, such as secure key management [21, 24] and more importantly encrypting data [20, 22, 23, 25]. Local encryption can help mitigate the risks associated with data breaches, albeit at the cost of certain functionalities, such as searching within encrypted data. Therefore, extensive research is being conducted to develop encryption methods that preserve the functionality of searching within encrypted databases [16]. One of solution is Order-Revealing encryption (ORE) which is a scheme that reveals nothing except messages' ordering by a public comparison function. ORE is a generalization of Order-Preserving encryption (OPE) which utilizes simple integer comparison as a comparison algorithm. While OPE does offer a lower computational cost compared to ORE, it relies on a deterministic encryption algorithm that is susceptible to various interference attacks, potentially revealing portions of the plaintext [17].

Currently, a number of ORE notations have been constructed. Boneh et al. [5] proposed an ideal ORE scheme based on multilinear maps that takes two ciphertexts and only discloses the order of two corresponding plaintexts and no additional information. However, multilinear maps are not practical for implementation in the real world. To overcome this drawback, Chenette et al. [8] designed a practical ORE from a symmetric encryption scheme which reveals the most significant different bit (msdb). Thereafter, Lewi et al. [13] suggested a notation that leaks the most significant different block. Cash et al. [6] constructed a parameter-hiding ORE based on property-

✉ Changhee Hahn
chahn@seoultech.ac.kr

Zeinab Rezaeifar
Zeinab.Rezaeifar@uwe.ac.uk

[1] Seoul National University of Science and Technology, Seoul, Republic of Korea

[2] University of the West of England, Bristol, UK

preserving hash (PPH). This scheme leaks the least amount of plaintext information out of all existing schemes, called equality pattern. However, the comparison time of it is inefficient, and it only provides a single-user environment. To solve these problems, Lv et al. [15] introduced an efficient and multi-user support ORE scheme (m-ORE). It reduced computational cost from $O(n^2)$ pairings in [6] to $O(n)$ pairings for the comparison stage while supporting multi-user scenarios.

In this paper, we reexamine m-ORE [15] and conducts a comprehensive analysis of its security vulnerabilities. Our investigation focuses on the vulnerability that arises in multi-user query scenarios. Specifically, we have identified a flaw in m-ORE where a malicious client, who has obtained a secret key shared among all clients from a data owner, can execute a query recovery attack on other clients' queries without requiring cooperation from the server. To formally capture such an issue, we introduce the concept of *Query Reusability*, which refers to the scheme's ability to expand a *Query Comparator* for generating the remaining queries. Further details regarding these vulnerabilities are presented in Sect. 5.

Furthermore, we introduce the two query recovery attack methods and launch our attacks against m-ORE [15], the state-of-the-art ORE construction. According to our experiment, it takes an average of 83ms to recover 64-bit plaintexts by our *bit-by-bit brute force algorithm*. Then we propose msq-ORE, a multi-client range query scheme that is secure against the *Query Reusability*.

Contribution We make the following main contributions:

- We first propose the vulnerability of m-ORE, *Query Reusability*, and the optimal attack technique, *bit-by-bit brute force algorithm*, in multi-client scenarios if the malicious client has the shared secret key from the data owner. This attack method can recover one query from another client within approximately 83ms.
- We propose the msq-ORE scheme, in which an adversary who has a shared secret key can not recover other clients' queries. In our msq-ORE scheme, the data owner generates the test key *tk* which can effectively deter the *Query Reusability* property in [15].
- The computational costs and data size of the proposed msq-ORE scheme are about the same as those of m-ORE [15], while achieving a higher level of security. We prove our claims through both theoretical and experimental results.

The rest of the paper is organized as follows. In Sect. 2, we provide related works. We then describe preliminaries in Sect. 3, followed by the description of property-preserving hash in Section 4. We examine the vulnerability of prior work in Sect. 5 and propose a security-wise enhanced ORE scheme in Section 6. We evaluate the performance of the proposed scheme in Sect. 7 and conclude the paper in Sect. 8.

2 Related Work

The notation of OPE was first designed by Agrawal et al. [2]. And then, Boldyreva et al. [4] proposed the"best possible"security, called indistinguishability under ordered chosen-plaintext attack (IND-OCPA). It means that two ciphertexts do not reveal any information about plaintexts except for their order. However, Boldyreva also suggested that this ideal security cannot be achievable if the OPE scheme is stateless and immutable. Popa et al. [18] presented the first IND-OCPA OPE notation which adopts stateful and interactive techniques by using the B-tree structure. Furthermore, Kerschbaum et al. [12] proposed the Frequency-hiding OPE (FH-OPE) which can hide the frequency about plaintexts to resist a number of inference attacks [7, 14, 17]. However, it cannot completely prevent the attacks and causes a lot of data overhead.

To overcome these problems and achieve better security without an interactive and stateful method, many ORE notations have been suggested. As opposed to OPE, ORE ciphertexts do not preserve the numerical value of plaintext. Instead, ORE adopts a publicly comparison function that uses two ciphertexts and returns the order based on the plaintexts. Boneh et al. [5] first proposed the notion of ORE. It was implemented by using multilinear maps [19]. However, currently, it is an inefficient primitive. To enhance the efficiency, Chenette et al. [8] designed the practical ORE notation that relied on the pseudorandom function. It showed a dramatic increase for practical, but it leaks the most significant different bit (msdb). Therefore, it does not have enough security guarantees. Thereafter, Lewi et al. [13] suggested the improved ORE notation that leaks only the most significant different block. However, both schemes [8, 13] have a deterministic property that always generates the same ciphertext, whenever they encrypt the same plaintext. Cash et al. [6] introduced the parameter-hiding ORE with probabilistic characteristics for the single-user scenario to overcome this. It only leaks the equality pattern of msdb. Subsequently, Lv et al. [15] suggested the m-ORE which reduces the computational overhead from $O(n^2)$ pairings in [6] to $O(n)$ pairings for the comparison stage, and it supports multi-user environments. However, in this paper, we show the vulnerability of m-ORE where the attacker can recover the query from a client by having shared secret key of another client within 83ms. Then, to overcome this problem, we purpose msq-ORE notation. Lastly, Lv et al. further extended m-ORE by

proposing a new data representation method that employs not only bit-wise representation of plaintexts but also the bit-length for comparison [16], while employing m-ORE as is. Since m-ORE and [16] are essentially equivalent, we omit [16] in our experiment.

3 Preliminaries

In this section, we begin by explaining the notations and parameters utilized in this work. Following that, we present some fundamental definitions and the complexity assumptions applied in this study. We provide notations and its description in Table 1.

3.1 Bilinear Maps

A bilinear maps e: $G_1 \times G_2 \rightarrow G_T$, where G_1,G_2 and G_T are cyclic multiplicative groups with prime order p, satisfies the following properties:

1. *Bilinearity*
   - $e(P^a, Q^b) = e(P, Q)^{ab}$ for all $P \in G_1$ and $Q \in G_2$ and all $a, b \in \mathbb{Z}_p$.
2. *Non-degeneracy*
   - $e(g_1, g_2) \neq 1$ if g_1 and g_2 are generators of G_1 and G_2, respectively.
3. *Computability*
   - $e(P, Q)$ can be efficiently computed for any $P \in G_1$ and $Q \in G_2$.

A pairing is called Type-3 if it is asymmetric without an efficiently computable isomorphism between G_1 and G_2. We use this bilinear map in our msq-ORE scheme because it is the most efficient type [19].

3.2 Complexity Assumption

We say the symmetric external Diffie-Hellman (SXDH) assumption holds with respect to these groups and pairing if all polynomial-time adversary $\mathcal{A}$ has the negligible advantage $\epsilon = \{\epsilon_a, \epsilon_b\}$ for p,q,r,s $\in \mathbb{Z}_p$ and T_1, $T_2 \in G_T$, where

$$\epsilon_a = |Pr[\mathcal{A}(g_1, g_1^p, g_1^q, g_1^{pq})] - Pr[\mathcal{A}(g_1, g_1^p, g_1^q, T_1)]|$$
$$\epsilon_b = |Pr[\mathcal{A}(g_2, g_2^r, g_2^s, g_2^{rs})] - Pr[\mathcal{A}(g_2, g_2^r, g_2^s, T_2)]|.$$

4 Property Preserving Hash

In this section, we first recollect the definitions of Property Preserving Hash (PPH) [6, 15]. Then, we suggest a concrete PPH from Bilinear Maps for m-ORE and our msq-ORE.

Definition 2. A property-preserving hash (PPH) has three algorithms Γ=($\mathcal{PPH}.\mathcal{K}$, $\mathcal{PPH}.\mathcal{H}$, $\mathcal{PPH}.\mathcal{T}$):

- $\mathcal{PPH}.\mathcal{K}(1^\lambda)$: *The key generation algorithm takes a security parameter* λ *as an input, and returns* (pp,hk,tk) *as outputs that represents the public parameter, hash key and test key, respectively. These implicitly define a domain D and range R for the hash.*
- $\mathcal{PPH}.\mathcal{H}(\mathsf{hk}, x)$:*The hash evaluation algorithm takes the hash key and* $x \in D$ *as inputs, and returns a single output* $h \in R$ *that we refer to as the hash of x.*
- $\mathcal{PPH}.\mathcal{T}(\mathsf{tk}, h_1, h_2)$:*The test algorithm takes the test key and two hashes* h_1, h_2 *as inputs and returns a bit* $b \in \{0, 1\}$.

Correctness. Let P be a predicate. We assume that PPH Γ is computationally correct with respect to P if $Pr[IND_\Gamma^P(\mathcal{A})]$ is a negligible function of λ for all efficient adversary $\mathcal{A}$, where the game $IND_\Gamma^P(\mathcal{A})$ is defined as follows: It first generates pp, hk and tk by running $\mathcal{PPH}.\mathcal{K}(1^\lambda)$ and sends tk to $\mathcal{A}$. Then, $\mathcal{A}$ gives x and y to the game. Finally, the game computes $h \leftarrow \mathcal{PPH}.\mathcal{H}(\mathsf{hk}, x)$,

Table 1 Notations

λ	Security parameter
$b_1, b_2, \cdots, b_n$	A binary form of integer m, where b_1 is the most significant bit and b_n is the least significant bit
$x\|\|y$	Concatenation of string x and y
PPT	Probabilistic polynomial time
$x \leftarrow \$\mathbb{Z}_p$	A group element x which is randomly and uniformly sampled from group $\mathbb{Z}_p$ of prime number p
PRF	A pseudorandom function
$[n]$	A set of integers $\{1, 2, \cdots, n\}$

$h' \leftarrow \mathcal{PPH}.\mathcal{H}(\mathsf{hk}, y)$ and returns 1 if $\mathcal{PPH}.\mathcal{T}(\mathsf{tk}, h, h') \neq P(x, y)$.

Security. we recall the PPH security game [15] that is more restricted than the method defined in [6]. Let P be some predicate and $\Gamma = (\mathcal{PPH}.\mathcal{K}, \mathcal{PPH}.\mathcal{H}, \mathcal{PPH}.\mathcal{T})$ be a PPH scheme with respect to P. For an adversary $\mathcal{A}$, we define the game $IND_{\Gamma}^{P}(\mathcal{A})$ and its restricted-chosen-input advantage as $Adv_{\Gamma}^{P,A}(\lambda) = 2Pr[IND_{\Gamma}^{P}(\mathcal{A}) = 1] - 1$. If $Adv_{\Gamma}^{P,A}(\lambda)$ is negligible for all adversary $\mathcal{A}$, we say that PPH Γ is restricted-chosen-input secure.

Game $IND_{\Gamma}^{P}(\mathcal{A})$:
$(\mathsf{pp}, \mathsf{hk}, \mathsf{tk}) \leftarrow \mathcal{PPH}.\mathcal{K}(1^{\lambda})$; $x' \leftarrow \mathcal{A}(\mathsf{tk})$
$h_0 \leftarrow \mathcal{PPH}.\mathcal{H}(\mathsf{hk}, x')$; $h_1 \leftarrow \$\ R$; $b \leftarrow \$\ \{0,1\}$; $b' \leftarrow \mathcal{A}^{Hash}(\mathsf{tk}, x', h_b)$
Return $(b \stackrel{?}{=} b')$
Hash(x) :
If $P(x, x') = 1$ or $x = x'$, then $h \leftarrow \perp$, Else $h \leftarrow \mathcal{PPH}.\mathcal{H}(\mathsf{hk}, x)$
Return(h)

4.1 PPH for m-ORE from Bilinear Maps

We introduce a concrete PPH scheme for m-ORE [15]. Let $P(x, y) = 1$ be the predicate if and only if $x = y \pm 1$, and let H: $\{0,1\}^{\lambda} \times \{0,1\}^{\lambda}$ be a secure PRF. Let Γ = ($\mathcal{PPH}.\mathcal{K}$, $\mathcal{PPH}.\mathcal{H}$, $\mathcal{PPH}.\mathcal{T}$) be a PPH scheme with respect to P as follows:

- $\mathcal{PPH}.\mathcal{K}(1^{\lambda})$: It takes a security parameter λ as an input and selects $k_1, k_{2,1}, k_{2,2} \leftarrow \mathbb{Z}_p$ and sets the hash key $\mathsf{hk} = (k_1, (k_{2,1}, k_{2,2}))$. It samples groups G_1, G_2 and G_T with prime order p and a related bilinear map $e : G_1 \times G_2 \rightarrow G_T$. Subsequently, it randomly picks generators $g_1 \in G_1$ and $g_2 \in G_2$. Following that, it sets the test key $\mathsf{tk} = (g_1^{k_{2,1}}, g_2^{k_{2,2}})$ and $\mathsf{pp} = (G_1, G_2, G_T, e)$. Finally, it returns $(\mathsf{pp}, \mathsf{hk}, \mathsf{tk})$.
- $\mathcal{PPH}.\mathcal{H}(\mathsf{hk}, x)$: It takes the hash key hk and a message x as inputs, then returns the follow hash value:
$$\vec{h} = (h_1, h_2, h_3) = (g_1^{H(k_1,x)\cdot k_{2,1}}, g_2^{H(k_1,x+1)\cdot k_{2,2}}, g_2^{H(k_1,x-1)\cdot k_{2,2}}).$$
- $\mathcal{PPH}.\mathcal{T}(\mathsf{tk}, \vec{h}, \vec{h'})$: It takes the test key tk and two hash values $\vec{h}, \vec{h'}$ as inputs and computes $e(h_1, g_2^{k_{2,2}})$, $e(g_1^{k_{2,1}}, h'_2)$ and $e(g_1^{k_{2,1}}, h'_3)$. Then, it outputs 1 if
$$e(h_1, g_2^{k_{2,2}}) = e(g_1^{k_{2,1}}, h'_2) \text{ or } e(h_1, g_2^{k_{2,2}}) = e(g_1^{k_{2,1}}, h'_3).$$
It returns 0 otherwise.

4.2 PPH for msq-ORE from Bilinear Maps

In this section, we propose the PPH notation for msq-ORE.

- $\mathcal{PPH}.\mathcal{K}(1^{\lambda})$: It takes a security parameter λ as an input and randomly picks $k_1, k_a, k_b, k_s \leftarrow \mathbb{Z}_p$ and computes $\overline{k_{a,s}} \leftarrow k_a \cdot k_s$, $\overline{k_{b,s}} \leftarrow k_b \cdot k_s$. Following that, it sets the hash key $\mathsf{hk} = (k_1, (k_b, \overline{k_{b,s}}))$. It samples groups G_1, G_2 and G_T with prime order p and randomly selects generators $g_1 \in G_1$ and $g_2 \in G_2$, an associated bilinear map $e : G_1 \times G_2 \rightarrow G_T$. After that, it sets $\mathsf{pp} = (G_1, G_2, G_T, e)$ and the test key $\mathsf{tk} = (g_1^{k_a}, g_2^{\overline{k_{a,s}}})$. Finally, it outputs $(\mathsf{pp}, \mathsf{hk}, \mathsf{tk})$.
- $\mathcal{PPH}.\mathcal{H}(\mathsf{hk}, x)$: It takes the hash key hk and a message x as inputs, then returns the follow hash value:
$$\vec{h} = (h_1, h_2, h_3) = (g_1^{H(k_1,x)\cdot k_b}, g_2^{H(k_1,x+1)\cdot \overline{k_{b,s}}}, g_2^{H(k_1,x-1)\cdot \overline{k_{b,s}}}).$$
- $\mathcal{PPH}.\mathcal{T}(\mathsf{tk}, \vec{h}, \vec{h'})$: It takes the test key tk and two hash values $\vec{h}, \vec{h'}$ as inputs and computes $e(h_1, g_2^{\overline{k_{a,s}}})$, $e(g_1^{k_a}, h'_2)$ and $e(g_1^{k_a}, h'_3)$. Then it outputs 1 if
$$e(h_1, g_2^{\overline{k_{a,s}}}) = e(g_1^{k_a}, h'_2) \text{ or } e(h_1, g_2^{\overline{k_{a,s}}}) = e(g_1^{k_a}, h'_3).$$
It returns 0 otherwise.

Correctness: The correctness relies on whether the following equation holds:

$$H(k_1, x) = H(k_1, y \pm 1).$$

If $x = y \pm 1$, the correctness always holds. Otherwise, we can easily show that finding x, y satisfying this property with non-negligible probability will induce an adversary who can dispute the assumption that H is a PRF.

Remark 1. The msq-ORE PPH is quiet similar to that of [6] and [15]. The major difference is that msq-ORE and m-ORE scheme [15] are deterministic, but that of [6] is probabilistic. Because the PPH notation in [6] always chooses random non-zero $r_1, r_2 \in \mathbb{Z}_p$ and computes these whenever PPH scheme encrypts a message by using the hash algorithm.

4.3 Security Analysis

Theorem 1 *We prove that PPH Γ is restricted-chosen-input secure, assuming that H is a PRF, and SXDH assumption holds.*

Proof The proof is really similar to that of [6]. We proved the theorem via below security games. We started with a real game and ends with a game that perfectly hides the random bit b. Then, we showed that any two adjacent

games could not be distinguished and represent the probability that the Game J_i outputs 1 as $Pr[J_i = 1]$. □

- Game J_{-1}: This game is precisely the real game. The challenger generates the test key tk and the hash values $(h_1,h_2,h_3) \in G_1 \times G_2^2$ for the challenge.
- Game J_0: J_0 is precisely the same as J_{-1} except for that a real random function $H^*(\cdot)$ is replaced by the pseudo-random function $H(k_1, \cdot)$.
- Game J_1: J_1 is precisely the same as J_0 except for that we replace the challenge hash values with (s, h_2, h_3), where $s \leftarrow \$\ G_1$.
- Game J_2: J_2 is precisely the same as J_1 except for that we adapt the challenge hash values to $(s, \bar{s}, h_3)$, where $\bar{s} \leftarrow \$\ G_2$.
- Game J_3: J_3 is exactly the same as J_2 except for that we modify the challenge hash values to $(s, \bar{s}, s^*)$, where $s^* \leftarrow \$\ G_2$.

In the end of the game, we obtain

$$Adv_{\Gamma}^{P,A}(\lambda) = |Pr[J_{-1} = 1] - Pr[J_3 = 1]|$$

by the definition of PPH.

Firstly, we can easily obtain that the game J_{-1} and J_0 are computational indistinguishable based on the PRF security. Secondly, we show that J_0 is indistinguishable from J_1 by the lemma below.

Lemma 1 *$J_0 \approx J_1$ assuming that the SXDH assumption is maintained.*

Proof We assume that an adversary $\mathcal{A}$ can distinguish between game J_0 and J_1 with the advantage

$$\epsilon = |Pr[J_0 = 1] - Pr[J_1 = 1]|.$$

Then, we can induce that an another adversary $\mathcal{B}$ can also prove the problem of SXDH with the same advantage ϵ. The following steps take (g_1, g_2, L, Y) and the challenge term C as inputs. The adversary $\mathcal{B}$ executes the stages as follows:

1. $\mathcal{B}$ randomly selects $k_a, k_s \leftarrow \$\ \mathbb{Z}_p$ and sets the test key $tk = (g_1^{k_a}, g_2^{\overline{k_{a,s}}})$. Following that, $\mathcal{B}$ sends it to $\mathcal{A}$. After that, $\mathcal{A}$ chooses $\bar{x}$ and sends it to $\mathcal{B}$. $\mathcal{B}$ implicitly sets $H^*(\bar{x}) = l$, the discrete logarithm of L, by using the random function H^*. Then, it computes

$$(h_1 = C, h_2 = g_2^{H^*(\bar{x}+1)\cdot\overline{k_{b,s}}}, h_3 = g_2^{H^*(\bar{x}-1)\cdot\overline{k_{b,s}}})$$

and sends these hash values to $\mathcal{A}$. Note that y and Y are k_b and g_1^y, respectively.
2. To answer a query which satisfies $x \neq \bar{x}$ and $x \pm 1 \neq \bar{x}$ from $\mathcal{A}$, $\mathcal{B}$ calculates:

$$(h_1, h_2, h_3) = (Y^{H^*(x)}, g_2^{H^*(x+1)\cdot\overline{k_{b,s}}}, g_2^{H^*(x-1)\cdot\overline{k_{b,s}}}).$$

3. Finally, The output of $\mathcal{B}$ is always the same as that of $\mathcal{A}$.

We denote that $\mathcal{B}$ accurately simulates without querying $\mathcal{A}(\bar{x} \pm 1)$ and $\mathcal{A}(\bar{x})$. If $\mathcal{B}$ uses g_1^{yl} as the challenge term C, it simulates J_0, and if $C \in G_1$, $\mathcal{B}$ simulates J_1. Therefore, $\mathcal{B}$ has the same advantage ϵ with $\mathcal{A}$ to break the SXDH assumption.

Likewise, we also have the following lemmas: □

Lemma 2 *$J_1 \approx J_2$ assuming that the SXDH assumption is maintained.*

Lemma 3 *$J_2 \approx J_3$ assuming that the SXDH assumption is maintained.*

For both lemmas, the proofs are similar to that of Lemma 1. Therefore, we omit details for these. Finally, we complete the proof of Theorem 1 by gathering all of the above lemmas.

5 Vulnerability of m-ORE

In this section, we propose the vulnerability of query in [15]. Firstly, we introduce the recapitulation of m-ORE [15]. Following that, we suggest a *Query Reusability*, a simple attack method, and an optimal attack method, *bit-by-bit brute force algorithm*, for m-ORE. Finally, we show the experimental evaluation for two attack methods.

5.1 Efficient Multi-client Order-Revealing Encryption.

In this section, we show m-ORE [15] in detail.

5.1.1 m-ORE Algorithms.

We introduce m-ORE scheme based on PPH. m-ORE consists of four algorithms: m-ORE.K, m-ORE.E, m-ORE.T, m-ORE.C. Let F: $[n] \times \{0,1\}^n \rightarrow \{0,1\}^{\lambda}$ be a secure PRF. The details of m-ORE algorithms are as follows:

- $(\mathsf{msk}, \mathsf{qk}) \leftarrow \mathbf{m-ORE.K}(1^{\lambda})$: This algorithm takes a security parameter λ as an input and outputs qk and msk. It obtains the public parameter $\mathsf{pp} = (G_1, G_2, G_T, e)$, the hash key $\mathsf{hk} = (k_1, (k_{2,1}, k_{2,2}))$ and the test key $\mathsf{tk} = (g_1^{k_{2,1}}, g_2^{k_{2,2}})$ from $\mathcal{PPH}.\mathcal{K}$. Following that, it sets $\mathsf{qk} = (k_1, g_2^{k_{2,2}})$ and $\mathsf{msk} = \mathsf{hk}$.

– $c \leftarrow \mathbf{m-ORE}.\mathbf{E}(\mathsf{msk}, m)$: This algorithm takes msk and a message m as inputs and outputs a ciphertext c. Firstly, it expresses the message m in binary and picks $r \leftarrow \mathbb{Z}_p$. Then, it computes $c_0 = g_1^{k_{2,1} \cdot r}$ and sets $\overline{k_{2,1}} \leftarrow k_{2,1} \cdot r$ and $\mathsf{hk} = (k_1, (\overline{k_{2,1}}, k_{2,2}))$. For $i = 1, \cdots, n$, this algorithm uses a practical ORE notation [8] and the hash function of m-ORE PPH.

$$u_i = F(i, b_1 b_2 \cdots b_{i-1} || 0^{n-i+1}) + b_i \bmod 2^{\lambda},$$
$$v_i = h_1 \leftarrow \mathcal{PPH}.\mathcal{H}(\mathsf{hk}, u_i)$$

Finally, it selects a random permutation π: $[n] \rightarrow [n]$ and sets $c_i = v_{\pi(i)}$, and returns the ciphertext $c = (c_0, c_1, \cdots, c_n)$.

– $t \leftarrow \mathbf{m-ORE}.\mathbf{T}(\mathsf{qk}, \overline{m})$: This algorithm takes msk and $\overline{m}$ as inputs and outputs a query of $\overline{m}$. Firstly, it expresses the message $\overline{m}$ in binary and picks $r^* \leftarrow \mathbb{Z}_p$. After that, it computes $t_0 = g_2^{k_{2,2} \cdot r^*}$ and sets $\overline{k_{2,2}} \leftarrow k_{2,2} \cdot r^*$. For $i = 1, \cdots, n$, it computes

$$u_i = F(i, b_1 b_2 \cdots b_{i-1} || 0^{n-i+1}) + b_i \bmod 2^{\lambda},$$
$$t_{1,i} = g_2^{\overline{k_{2,2}} \cdot H(k_1, u_i + 1)}, t_{2,i} = g_2^{\overline{k_{2,2}} \cdot H(k_1, u_i - 1)}$$

Finally, it chooses a random permutation π: $[n] \rightarrow [n]$ and sets $t_i = (t_{\pi(1,i)}, t_{\pi(2,i)})$ and returns the query $t = (t_0, (t_{1,1}, t_{2,1}), \cdots, (t_{1,n}, t_{2,n}))$. Here, We define t_0 as a *Query Comparator*.

– $b \leftarrow \mathbf{m-ORE}.\mathbf{C}(c, t)$: This algorithm takes a ciphertext c and a query t and outputs a bit b. Firstly, it sets $\mathsf{tk} = (c_0, t_0) = (g_1^{\overline{k_{2,1}}}, g_2^{\overline{k_{2,2}}})$ and computes $\mathcal{PPH}.\mathcal{T}(\mathsf{tk}, c_i, t_{1,j})$ and $\mathcal{PPH}.\mathcal{T}(\mathsf{tk}, c_i, t_{2,j})$ for every $i, j \in [n]$. If there exists a pair (i^*, j^*) which satisfies $\mathcal{PPH}.\mathcal{T}(\mathsf{tk}, c_{i^*}, t_{1,j^*})$, it outputs 1 and stops. It means that $m > \overline{m}$; else if there exists a pair (i^*, j^*) which satisfies $\mathcal{PPH}.\mathcal{T}(\mathsf{tk}, c_{i^*}, t_{2,j^*})$, it outputs 0 and stops. It means that $m < \overline{m}$; otherwise it outputs $\perp$ which means that $m = \overline{m}$.

5.2 Query Reusability

In this section, we suggest the vulnerability of m-ORE's query [15]. We define *Query Reusability* as simply multiplying *Query Comparator* by an exponential to create the rest of the query. For example, in [15], $t_{1,i}$ and $t_{2,i}$ ($g_2^{\overline{k_{2,2}} \cdot H(k_1, u_i + 1)}$, $g_2^{\overline{k_{2,2}} \cdot H(k_1, u_i - 1)}$) are created by multiplying *Query Comparator* ($t_0 = g_2^{\overline{k_{2,2}}}$) by an exponential factor.

5.2.1 Assumption.

Our attack follows the below assumptions.

– An adversary is a client who is authorized by a data owner. Therefore, he receives a pseudorandom key from the data owner.
– The adversary does not collaborate with the server and other clients.
– The adversary can eavesdrop on a channel between a victim client and server.
– The adversary can not get any information about dataset.

Algorithm 1 Query plus algorithm

```
Input : m
Output : query_plus_value
query_plus_value ←⊥
    for i ← 1, ··· , n do
    u_i = F(i, b_1 b_2 ··· b_{i−1} || 0^{n−i+1}) + b_i mod 2^λ
    query_plus_value.append(H(k_1, u_i + 1))
    end
return query_plus_value
```

Algorithm 2 Simple brute-force algorithm

```
Input : query_comparator, victim_query
Output : victim_number
    for i ← 0, ··· , 2^n − 1 do
count ←⊥, i_(2) ← i_(10)
attack_query ← Qeury_plus(i)
    for j ← 0, ··· , n do
        for l ← 0, ··· , n do
            if
        (query_comparator)^{attack_query[j]} ==
        victim_query[l] then
                count = count + 1
                break
            end
        end
        if count == n then
            victim_number = i
        return victim_number
        end
    end
    end
```

Algorithm 3 Bit-by-bit brute force algorithm

Input : $query\_comparator, victim\_query$
Output : $attack\_success\_num$
$attack\_success\_num \leftarrow \perp$
for $i \leftarrow 1, \cdots, n$ **do**
  $condition = Ture$
$try\_num = 0$
$u_i = F(i, attack\_success\_num || 0^{n-i+1}) +$
$try\_num \bmod 2^{\lambda}$
for $j \leftarrow 1, \cdots, n$ **do**
  if $victim\_query[j] ==$
  $(query\_comparator)^{H(k_1, u_i+1)}$ **then**
    $attack\_success\_num.append(try\_num)$
    $condition = False$
    break
  end
end
if $condition == True$ **then**
  $try\_num = 1$
  $attack\_success\_num.append(try\_num)$
end
end

return $attack\_success\_num$

5.3 Attack Methods

In this section, we introduce the two attack methods. Firstly, we propose a simple brute force attack, then we suggest an optimal attack method, *bit-by-bit brute force algorithm*. We only use $t_{1,i}$ for query recovery attacks with *Query comparator* because the pseudorandom function F and H are unique. Therefore, we do not need $t_{2,i}$ in the attack scenario. We call $t_{1,i}$ as a *query plus* value. Furthermore, we assume that the length of the victim number and message m is $n-$bit in binary form.

5.3.1 Simple brute-force algorithm

This algorithm is a simple brute-force algorithm. In Algorithm 2, it takes a *query_comparator* and *victim_query* as inputs and returns a *victim_number*. The *victim_query* is a victim client's token without *query comparator*. Then, a *count* is a parameter that counts how many queries the number i is equal to the victim query value in total. Firstly, it converts the i value from decimal to binary and computes an *attack_query* for i by using the *Query plus algorithm* which generates *query_plus_value* for the input binary number in Algorithm 1. Following that, the *count* counts how many queries the number i is duplicated with the victim query value. If it is equal to the length of the number of the victim number n, this algorithm returns the *victim_number* because it means that all pseudorandom values are equal to the victim number.

5.3.2 Bit-by-bit brute-force algorithm.

Algorithm 2 is inefficient because an adversary is required to scan the plaintext space in its entirety in launching the attack. In this subsection, we provide an optimal attack method that boosts the attack efficiency significantly. Specifically, in Algorithm 3, it takes a *query_comparator* and *victim_query* as inputs and returns a value called *attack_success_num*. This uses the property that the number of binary numbers consists of only"0"and"1". The reason why this attack is possible is that Lv et al.'s ORE scheme [15] is based on Chenette et al.'s ORE [8] which works in bits. This method is an attack approach that recovers queries bit by bit using the *Query Reusability* property. It selects either"0"or"1", and for each bit, it uses a *query comparator* to predict the expected value of the query. If the predicted value exists in the value of the query being attacked, it adds the selected value, otherwise it adds the other value. For example, if"0"is selected, the expected value corresponding to"0"is obtained for each bit, and if the computed value matches a value in the victim query,"0"is added, otherwise "1"is added. An *attack_success_num* is a parameter that stores "0"or"1"that has the same query value in victim query and a *condition* is a value that indicates whether the value computed using the *query comparator* has succeeded or failed in the attack. A *try_num* refers to the value that will be stored for each bit in *attack_success_num*. The process of this algorithm is similar to that of *Simple brtue-force* algorithm. We randomly set the value of"0"to be performed first because the malicious user cannot know which number ("0"or"1") is used more in the plaintext under the assumptions. We can expect that the more the number"1"appears, the longer the attack cost takes. Because if there is no match for the value in"0", it has to compare it with all query values.

5.4 Experimental Evaluation

5.4.1 Setup

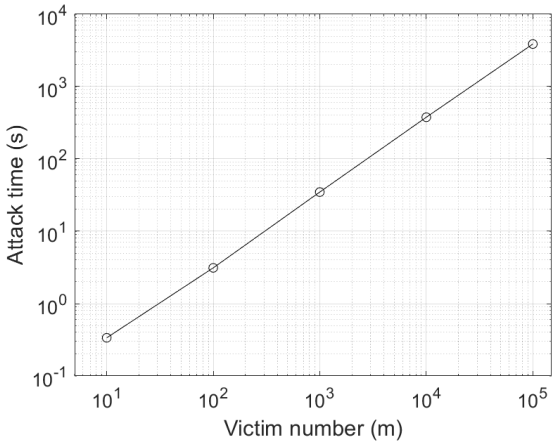
We implement the m-ORE scheme and two attack methods in Python. For a pseudorandom function that maps a given numerical data to a group element, we utilized AES-GCM in our experiment, that takes a secret key and input data as parameters. We use the Bplib library [11] to implement bilinear pairing. We set a 32-bit security parameter λ. To achieve environmentally independent results, we perform our experiments in the Google colaboratory environment [3]. Furthermore, we use three different datasets, the California public employee payroll data [1], Gowalla [9], and Foursquare [26]. Specifically, the California public employee payroll data is a dataset of California's public

pay and pension, where 30,012,803 employee records can be found. Gowalla is a location-based social networking website dataset where users share their locations by checking-in. The dataset consists of 6,442,890 checkins in a numerical format. Lastly, Foursquare also is a location based social networks dataset, where 27,149 checking-in numerical data are stored.

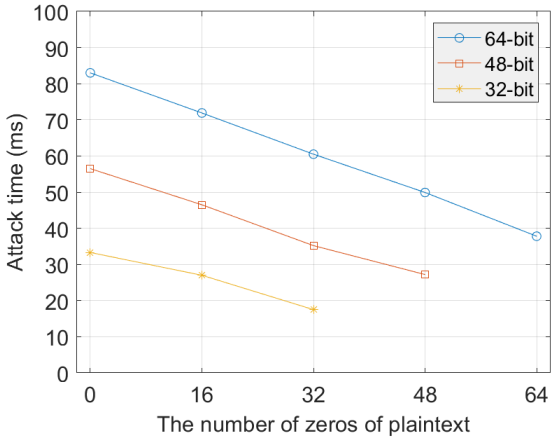
5.4.2 Evaluation.

We show the result of the simple brute-force algorithm attack method in Fig 1a. We assume that there is no negative integer in the attack simulation and conduct this experiment from 0 to victim number m sequentially because the malicious user can not obtain any features from the dataset by the assumption. The x-axis is the victim number m that adversary wants to recover plaintext value and the y-axis is the attack time in seconds. It requires approximately 3874 s if the adversary wants to recover the query of 10^5. We can calculate the average attack cost for one query by using all attack times. On average, 35.2ms per number is required for attacks. If we want to attack the largest value (166,539,600) in the Foursquare dataset, it needs approximately 68 days.

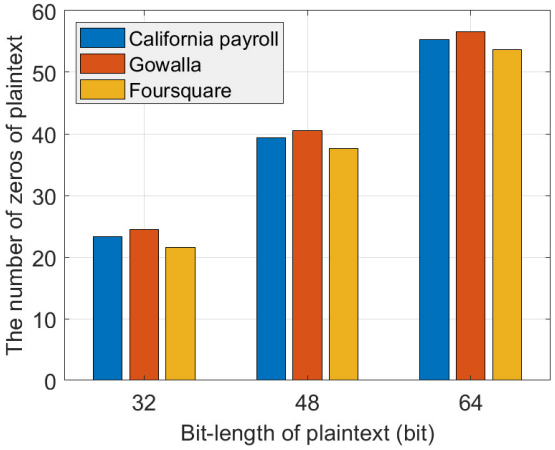
In Fig. 1b, Fig. 1c and Fig. 1d, we show the results of the *bit-by-bit brute-force algorithm*. We performed simulations 1,000 times each. In Fig. 1b, as we expected, the more plaintexts have the number of"1", the more the attack costs are required. In the worst case that all bits are"1" in 64-bit plaintext, the time it takes to attack is only 83ms. Before we show the results of the cost for the optimal attack method for the three datasets, we suggest the number of zeros of plaintexts for three datasets. In Fig 1c, the number of"0"is large in the order of Gowalla, California payroll and Foursquare. Therefore, we can predict that the order of computational costs also follows the order of that. In Fig 1d, we show that the costs are small in the order of Gowalla, California payroll and Foursquare and located within the values of Fig 1b. For example, in Gowalla, there are 56.525 instances of the number"1"and



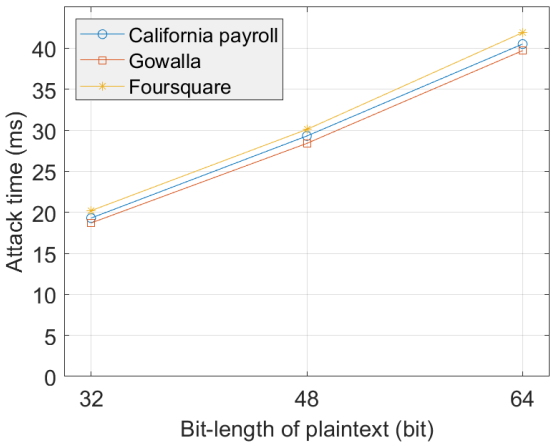
(a) Attack time for simple brute-force



(b) Attack time for the number of zeros



(c) The number of zeros for dataset



(d) Attack time for dataset

Fig. 1 Evaluation for two attack methods

the attack cost is 39.7ms in 64-bit. Its cost falls within a range of 37.8ms to 49.9ms in Fig. 1b. Then, the number of"1"s when representing 166,539,600 as binary in 64-bit is 13, and we can expect that it needs approximately 48ms. This result shows a substantial difference from that of the simple brute force algorithm (68 days).

As a result, we showed that the *bit-by-bit brute-force algorithm* is more efficient than simple brute-force algorithm via experiments and that the client who has a shared pseudorandom key can recover any query within 83ms by using our optimal attack algorithm without any dataset information and cooperation.

6 Multi-client Secure Range Query Order-Revealing Encryption (msq-ORE)

6.1 Definition of multi-client ORE

Definition 3. For a multi-client scheme, say Π_{msq}, it consists of (**msq-ORE.K**, **msq-ORE.E**, **msq-ORE.T**, **msq-ORE.C**).

- **msq-ORE.K**(1^λ): On input the security parameter λ, this key generation algorithm returns the data owner key dk and the query key qk.
- **msq-ORE.E**(dk, m): On input the data owner key dk and a message m, this encryption algorithm returns a ciphertext c.
- **msq-ORE.T**(qk, m^*): On input the query key qk and a message m^* for a query, this token generation algorithm returns a token t.
- **msq-ORE.C**(c, t): On input a ciphertext c and a token t, this comparion algorithm returns a bit $b \in \{0, 1\}$.

Correctness: We say that Π_{msq} is computationally correct if the probability $\Pr[\textbf{msq-ORE.T}(c,t) \neq 1(m_1 > m_2)]$ is negligible of λ with $\mathsf{dk}, \mathsf{qk} \leftarrow \textbf{msq-ORE.K}(1^\lambda)$, $c \leftarrow \textbf{msq-ORE.E}(\mathsf{dk}, m_1)$ and $t \leftarrow \textbf{msq-ORE.T}(\mathsf{qk}, m_2)$.

Security: Our msq-ORE scheme is non-adaptively simulation secure, similar to Cash et al. [6] and Lv et al [15]. Formal definition is as follows:

Definition 4. For an msq-ORE scheme Π_{msq} = (**msq-ORE.K**, **msq-ORE.E**, **msq-ORE.T**, **msq-ORE.C**), a PPT adversary $\mathcal{A}$, a simulator $\mathcal{S}$ and leakage function $\mathcal{L}(\cdot)$, we define the two games, $\text{Real}_{\mathcal{A}}^{msq-ore}(\lambda)$ and $\text{Sim}_{\mathcal{A},\mathcal{L},\mathcal{S}}^{msq-ore}(\lambda)$ as shown in Fig. 2.

Following that, we say that Π_{msq} is a secure msq-ORE scheme with leakage function $\mathcal{L}(\cdot)$ if for every PPT adversary $\mathcal{A}$ there exists an effective polynomial-size simulator $\mathcal{S}$ such that the outputs of the $\text{Real}_{\mathcal{A}}^{msq-ore}(1^\lambda)$ and $\text{Sim}_{\mathcal{A},\mathcal{L},\mathcal{S}}^{msq-ore}(1^\lambda)$ are indistinguishable.

6.2 msq-ORE scheme from PPH

The technical intuition of msq-ORE, in terms of enhanced security on top of m-ORE, is splitting a single query key (i.e., $k_{2,2}$) in m-ORE into two part (i.e., $k_{a,s}, k_{b,s}$) in msq-ORE. While an adversary is able to efficiently recover plaintext queries by abusing $k_{2,2}$ in m-ORE, he can no longer launch the same attack twice in msq-ORE because the key separation.

Let λ be the security parameter. Let $P(x, y) = 1$ if $x = y \pm 1$ be the predicate and we define our msq-ORE scheme Π_{msq} = ($\textbf{msq}-\textbf{ORE.K}$, $\textbf{msq}-\textbf{ORE.E}$, $\textbf{msq}-\textbf{ORE.T}$, $\textbf{msq}-\textbf{ORE.C}$) as follows:

- $\textbf{msq}-\textbf{ORE.K}(1^\lambda)$: It takes λ as an input and returns dk and qk. It obtains $\mathsf{pp} = (G_1, G_2, G_T, e)$, $\mathsf{hk} = (k_1, (k_b, \overline{k_{b,s}}))$ and $\mathsf{tk} = (g_1^{k_a}, g_2^{\overline{k_{a,s}}})$ by using $\mathcal{PPH}.\mathcal{K}(1^\lambda)$. Then, it sets the data owner key $\mathsf{dk} = (k_1, (g_1^{k_a}, k_b))$ and the query key $\mathsf{qk} = (k_1, (g_2^{\overline{k_{a,s}}}, g_2^{\overline{k_{b,s}}}))$. Finally, it returns $(\mathsf{dk}, \mathsf{qk})$.
- $\textbf{msq}-\textbf{ORE.E}(\mathsf{dk}, m)$: It takes dk and m as inputs. It selects $r_d \leftarrow\$\ \mathbb{Z}_p$ and computes the binary form $(b_1, b_2, \cdots, b_n)$ of message m. Then, it sets $qc_e = g_1^{\overline{k_a}}$ and $\mathsf{hk} = (k_1, (\overline{k_b}, \overline{k_{b,s}}))$, $\overline{k_a} \leftarrow r_d \cdot k_a$, $\overline{k_b} \leftarrow r_d \cdot k_b$. For $i = 1, 2, \cdots, n$, it computes:

$$u_i = F(i, b_1 b_2 \cdots b_{i-1} || 0^{n-i+1}) + b_i \bmod 2^\lambda,$$
$$v_i = h_1 \leftarrow \mathcal{PPH}.\mathcal{H}(\mathsf{hk}, u_i).$$

$\underline{\text{Real}_{\mathcal{A}}^{msq-ore}(\lambda)}$:
$\mathsf{dk}, \mathsf{qk} \leftarrow \textbf{msq-ORE.K}(1^\lambda)$
$(m_1, \cdots, m_q) \leftarrow \mathcal{A}(\lambda)$
for $1 \leq i \leq q$:
  $c_i \leftarrow \textbf{msq-ORE.E}(\mathsf{dk}, m_i)$
  $t_i \leftarrow \textbf{msq-ORE.T}(\mathsf{qk}, m_i)$
return $(c_1, \cdots, c_q)$ and $(t_1, \cdots, t_q)$

$\underline{\text{Sim}_{\mathcal{A}}^{msq-ore}(\lambda)}$:
$\mathsf{st}_{\mathcal{S}} \leftarrow \mathcal{S}(\lambda)$
$(m_1, \cdots, m_q) \leftarrow \mathcal{A}(\lambda)$
$((c_1, \cdots, c_q), (t_1, \cdots, t_q)) \leftarrow \mathcal{S}(\mathsf{st}_{\mathcal{S}}, \mathcal{L}(m_1, \cdots, m_q))$
return $(c_1, \cdots, c_q)$ and $(t_1, \cdots, t_q)$

Fig. 2 The real and ideal execution

Finally, it picks a random permutation $\pi : [n] \rightarrow [n]$ and computes $c_i = v_{\pi(i)}$, and returns the ciphertext $c = (qc_e, c_1, \cdots, c_n)$.

– $\mathbf{msq-ORE.T}(\mathsf{qk}, m^*)$: It takes qk and m^* as inputs and outputs a token t. It picks $r_t \leftarrow \$\ \mathbb{Z}_p$ and computes the binary form $(b_1^*, b_2^*, \cdots, b_n^*)$ of message b^* and $qc_t = g_2^{\overline{k_{a,s}} \cdot r_t}$. Following that, for $i = 1, 2, \cdots, n$, it computes

$$u_i = F(i, b_1^* b_2^* \cdots b_{i-1}^* || 0^{n-i+1}) + b_i^* \bmod 2^\lambda,$$
$$t_{i,1} = g_2^{\overline{k_{b,s}} \cdot r_t \cdot H(k_1, u_i+1)}, t_{i,2} = g_2^{\overline{k_{b,s}} \cdot r_t \cdot H(k_1, u_i-1)}.$$

Finally, it sets a random permutation π: $[n] \rightarrow [n]$ and computes $t_i = (t_{\pi(i,1)}, t_{\pi(i,2)})$ and returns the token $t = (qc_t, (t_{1,1}, t_{1,2}), \cdots, (t_{n,1}, t_{n,2}))$. This token is self-generated value without $\mathcal{PPH}.\mathcal{H}$. From this, it only uses the query key qk rather than the hash key hk and the token is available to the $\mathcal{PPH}.\mathcal{T}$.

– $\mathbf{msq-ORE.C}(c, t)$: It takes a ciphertext c of m and a token t of m^* as inputs. Firstly, it sets $\mathsf{tk} = (qc_e, qc_t) = (g_1^{\overline{k_a}}, g_2^{\overline{k_{a,s}} \cdot r_t})$ as the test key and performs

$$\mathcal{PPH}.\mathcal{T}(\mathsf{tk}, c_i, t_{j,1}),\ \mathcal{PPH}.\mathcal{T}(\mathsf{tk}, c_i, t_{j,2})$$

for every $i, j \in [n]$. If there exists a pair (i^*, j^*) such that $\mathcal{PPH}.\mathcal{T}(\mathsf{tk}, c_{i^*}, t_{j^*,1})$, it outputs 1, and stops. It means that $m > m^*$; else if there exists a pair (i^*, j^*) such that $\mathcal{PPH}.\mathcal{T}(\mathsf{tk}, c_{i^*}, t_{j^*,2})$, it outputs 0, and stops. It means that $m < m^*$; otherwise it outputs $\perp$, meaning $m = m^*$.

Correctness: For two binary form $(b_1, \cdots, b_n)$ and $(b_1^*, \cdots, b_n^*)$ of two messages m and m^*, if $m > m^*$, there must exist a proper index $\bar{i} \in [n]$ such that $u_{\bar{i}} = u'_{\bar{i}} + 1$. Therefore the correctness of Π_{msq} is followed by correctness of PPH. For both cases where $m < m^*$ and $m = m^*$, we can use the same argument.

Remark 2. Note that the token of our msq-ORE does not use a query comparator when it makes rest of the query. For example, the following

$$(t_{i,1}, t_{i,2}) = (g_2^{\overline{k_{b,s}} \cdot r_t \cdot H(k_1, u_i+1)}, g_2^{\overline{k_{b,s}} \cdot r_t \cdot H(k_1, u_i-1)})$$

are not based on $qc_t = g_2^{\overline{k_{a,s}} \cdot r_t}$. Therefore, our msq-ORE scheme is secure against *Query Reusability* property.

The leakage, i.e., $\mathcal{L}(m_1, \cdots, m_q)$, of our msq-ORE scheme is equivalent to that of [6] and [15] as follows:

$$(\forall 1 \le i, j, k \le q, 1(\mathbf{msdb}(m_i, m_j) = \mathbf{msdb}(m_i, m_k))).$$

This means that the leakage of $\mathbf{msdb}(m_i, m_j)$ is identical to $\mathbf{msdb}(m_i, m_k)$ for any three messages m_i, m_j, m_k regardless of the order of plaintexts.

Theorem 2 *Assuming that H is a secure PRF and PPH* Γ *is restricted-chosen-input secure, our* $\mathbf{msq-ORE}$ Π_{msq} *is* $\mathcal{L}_f$*-non-adaptively-simulation secure.*

Proof We define a consecutive games to show the security of our msq-ORE as follows:

– Game J_{-1}: This game is the real game $\mathrm{Real}_{\mathcal{A}}^{msq-ore}(\lambda)$.
– Game J_0: This is identical to J_{-1} except the pseudo-random function H has been substituted with a truely random function H^*.
– Game $J_{i \cdot q+j}$: These games are identical to J_0 except u_i^j is replaced by a random string relies on a predicate **Switch**.
– Game J_{qn+1}: Simulation $\mathrm{Sim}_{\mathcal{A},\mathcal{L},\mathcal{S}}^{msq-ore}(\lambda)$.

We show that any successive games are indistinguishable, and then we construct an efficient simulator $\mathcal{S}$ such that the output of J_{qn} and $\mathrm{Sim}_{\mathcal{A},\mathcal{L},\mathcal{S}}^{msq-ore}(\lambda)$ are statistically indistinguishable. We follow the idea of [6] and [15] to define the predicate **Switch**. We say that $\mathbf{Switch}_{i,j} = 1$ if $\forall\ b \in [q]$, $\mathbf{msdb}(m_j, m_b) \neq i$, it means that it is possible to substitute the i-th bit of m_j with a random string. If $\mathbf{Switch} = 0$, there exists $u_i^j = u_i^b \pm 1$, this condition which can be identified by the $\mathcal{PPH}.\mathcal{T}$ algorithm therefore the i-th bit of m_j can not be replaced by a random string.

Lemma 4 *Assuming PPH scheme* Γ *is restricted chosen input secure, we have* $J_{k-1} \approx J_k$ *for any* $k \in [1, qn]$.

Proof For any $k \in [1, qn]$, we argue that it is suitable to show $J_{k-1} \approx J_k$ under the condition $\mathbf{Switch}_{i^* j^*} = 1$ for $k = i^* \cdot q + j^*$ where $i^* \in [0, n-1], j^* \in [1, q]$ to prove. If $\mathbf{Switch}_{i^* j^*} = 0$, it means that $J_{k-1} = J_k$. We demonstrate that if there exists an adversary $\mathcal{A}$ that distinguish J_k from J_{k-1} with significant advantage ϵ, then we can create a simulator $\mathcal{B}$ that wins the restricted-chosen-input game with same advantage with $\mathcal{A}$. $\mathcal{B}$ executes the stage as follows:

1. Firstly, IND_Γ^P is executed by it. Then, it sends the test key tk to $\mathcal{A}$. Upon receiving a list of plaintext $m_1, \cdots, m_q$, it sets $u_{i^*}^{j^*} = F^*(i^*, b_1^{j^*} b_2^{j^*} \cdots b_{i*-1}^{j^*} || 0^{n-i^*+1}) + b_{i^*}^{j^*} \bmod w^\lambda$ as the challenge ciphertext bit by using the truely random function F^* (The binary form $b_{i^*}^{j^*}$ is the i^*-th bit of m_{j^*}).
2. Following that, the challenge bit $u_{i^*}^{j^*}$ is sent to the IND_Γ^P's challenger by $\mathcal{B}$ and after receiving T as the challenge term, it assigns $t_{i^*}^{j^*} = T$.
3. $\mathcal{B}$ executes the following steps for simulating the other bit.
4. If for all ciphertext bits $u_{t'}^{j'}$ that come after $u_{i^*}^{j^*}$ (i.e. $i'q + j' > i^*q + j^*$), initially, for every $j \in [1, q]$, $\mathcal{B}$ chooses q elements $r_1, \cdots, r_q \leftarrow \mathbb{Z}_p$ and executes:

$u_{i'}^{j'} = F^*(i', b_1^{j'} b_2^{j'} \cdots b_{i'-1}^{j'} || 0^{n-i'+1}) + b_{i'}^{j'} \bmod 2^\lambda$
$c_{i'}^{j'} = h_1 \leftarrow \mathcal{PPH}.\mathcal{H}(hk, u_{i'}^{j'})$.

If for all ciphertext bits $u_{i'}^{j'}$ that come before $u_{i^*}^{j^*}$ under the condition $\mathbf{Switch}_{i',j'} = 0$ (i.e., $(i'q + j') < (i^*q + j^*) \cap \mathbf{Switch}_{i',j'} = 0$), then as described earlier, $\mathbf{Switch}_{i',j'} = 1$, $u_{i'}^{j'} \leftarrow \{0.1\}^\lambda$; $c_{i'}^{j'} = h_1 \leftarrow \mathcal{PPH}.\mathcal{H}(hk, u_{i'}^{j'})$.

5. For all $j^* \in [q]$, after simulating all the ciphertext bits, it sets a random permutation π_{j^*} and outputs $ct_{j^*} = (c_{\pi_{j^*}(1)}^{j^*}, c_{\pi_{j^*}(2)}^{j^*}, \cdots, c_{\pi_{j^*}(n)}^{j^*}; g_1^{r_{j^*}})$ to $\mathcal{A}$. Finally, $\mathcal{B}$ produces a bit value that depends on the output of A.

Algorithm 4 FillMatrix

Input : $i, j, k, \mathcal{M}$
Output : M_F
  if $j = k$ **then**
    $\forall i' \in [i, n], M_F[j][i'] = r$, where $r \leftarrow \{0,1\}^\lambda$;
    return 0;
  end
  else
    $a \leftarrow \mathsf{msdb}(m_j, m_{j^*})$;
    $b \leftarrow \mathsf{msdb}(m_j, m_k)$;
    search for the smallest j^* s.t. $a = b$;
    $r' \leftarrow \{0,1\}^\lambda$;
    $\forall j' \in [j, j^* - 1]$, set $M_F[j'][i] = r'$;
    $\forall j' \in [j^*, k]$, set $M_F[j'][i] = r' - 1$;
    run $\mathbf{FillMatrix}(i+1, j, j^* - 1, \mathcal{M})$;
    run $\mathbf{FillMatrix}(i+1, j^*, k, \mathcal{M})$;
  end
return M_F

We argue that the encryption oracle is correctly simulated by $\mathcal{B}$ because F^* is a random function and, for all $i'q + j' \neq i^*q + j^*$, the probability $\Pr[u_{i^*}^{j^*} = u_{i'}^{j'} \pm 1]$ is negligible, which means that $\mathcal{B}$ fails to simulate the encryption oracle with only negligible probability. Furthermore, When $T = h_1 \leftarrow \mathcal{PPH}.\mathcal{H}(hk, u_{i'}^{j'})$, $\mathcal{B}$ precisely simulates J_{k-1} and if T is random, $\mathcal{B}$ simulates properly, owing to the security of PRF. Hence, if $\mathcal{A}$ can distinguish between G_k and G_{k-1} with a noticeable advantage, then the advantage of $\mathcal{B}$ is also noticeable in the IND_Γ^P game. Considering that we have previously shown the advantage of an adversary in IND_Γ^P game is negligible, for any $k \in [1, qn]$, we have $J_{k-1} \approx J_k$. Since the prove of $J_{k-1} \approx J_k$ with respect to the token is quite similar to that of ciphertexts, we skip it.

Lemma 5 *There exists a simulator S that can efficiently generate outputs from both J_{qn} and J_{qn+1} that are indistinguishable.*

Proof When $\mathbf{Switch}_{i,j} = 1$ for any i-th of m_j, we set u_i^j as a random string because it does not influence leakage profile. Therefore, we only simulate the bit that $\mathbf{Switch}_{i,j} = 0$.

We present a recursive algorithm $\mathbf{FillMatrix}(i, j, k, \mathcal{M})$ in Algorithm 4, before we explain the simulator $\mathcal{S}$. It takes a set of message $\mathcal{M} = \{m_1, \cdots, m_q\}$ without of loss of generality and the set of tuples (i, j, k) where $i \in [n]$, $j \leq k \in [q]$, here, n represents the bit-length of every message and q represents the total number of messages as inputs and outputs a matrix M_F.

Firstly, to obtain $M_F^{q \times n}$, the simulator $\mathcal{S}$ runs the $\mathbf{FillMatrix}(1, 1, q, \mathcal{M})$. After that, $\mathcal{S}$ runs $\mathcal{PPH}.\mathcal{K}$ and obtains hk and tk. Subsequently, $\forall j \in [1, q]$, it selects q random permutation $\pi_1, \cdots, \pi_q$ and q elements $r_1, \cdots, r_q \leftarrow \mathbb{Z}_p$. And then, it sets $qc_e = g_1^{k_a \cdot r_j}$, $\overline{k_a} \leftarrow k_a \cdot r_j$, $\overline{k_b} \leftarrow k_b \cdot r_j$ and $hk = (k_1, (\overline{k_b}, \overline{k_{b,s}}))$. Following that, $\forall i \in [n]$, $\forall j \in [q]$, it computes $c_i^j = h_1 \leftarrow \mathcal{PPH}.\mathcal{H}(hk, M_F[j][i])$. Finally, $\forall j \in [q]$, it returns a ciphertext set $(ct_1, \cdots, ct_q)$, $ct_j = (qc_e, c_{\pi_j(1)}^j, \cdots, c_{\pi_j(n)}^j)$.

The simulator $\mathcal{S}$ also produces the tokens to simulate $\mathbf{msq} - \mathbf{ORE.T}$ through the following process. After the simulation of the ciphertext is complete, $\forall j \in [1, q]$, $\mathcal{S}$ selects new q components $r_1', \cdots, r_q' \leftarrow \mathbb{Z}_p$ and new q random permutations $\pi_1', \cdots, \pi_q'$. Then, $\forall i \in [n]$, $\forall j \in [q]$,it sets $qc_t = g_2^{\overline{k_{a,s}} \cdot r_j}$, $t_{i,1}^j = g_2^{\overline{k_{b,s}} \cdot r_j \cdot H(k_1, \mathcal{M}_F[j][i]+1)}$ and $t_{i,2}^j = g_2^{\overline{k_{b,s}} \cdot r_j \cdot H(k_1, \mathcal{M}_F[j][i]-1)}$. Finally, $\forall j \in [q]$, it returns a token set $(t_1, \cdots, t_q)$, where

$$t_j = (qc_t, (t_{\pi_j(1,1)}^j, t_{\pi_j(1,2)}^j), \cdots, (t_{\pi_j(n,1)}^j, t_{\pi_j(n,2)}^j)).$$

We show that $\mathcal{S}$ accurately simulates the games as follows: Firstly, the simulator $\mathcal{S}$ identifies how many leaked bits for the messages $(m_1, \cdots, m_q)$. Note that, if a set of messages $(m_1, \cdots, m_q)$ share the same prefix of length $(l-1)$-th bit and if there exists the first $m_{\overline{j}}$ such that $\mathbf{msdb}(m_1, m_{\overline{j}}) = \mathbf{msdb}(m_1, m_q)$, we can deduce that messages $\{m_1, \cdots, m_{\overline{j-1}}\}$ have "1"on their l-th bit and the messages $\{m_{\overline{j}}, \cdots, m_q\}$ have"0"on their l-th bit. This means that the l-th bit of messages are leaked. Subsequently, the **FillMatrix** algorithm continues to operate recursively to identify other leaked bits and determine the total number of bits that have been leaked. This information will also be identified in the game J_{qn}. Note that both $\mathcal{S}$ and the J_{qn} game do not only determine the total number of leaked bits but also the specific position of the leaked bits, as the messages are ordered. Therefore, the output of the ciphertext set leaks the **msdb** between any two ciphertexts. However, for both J_{qn} and this game, the random permutation function helps to hide this leakage (index of the

leaked bit) except the total number. Therefore, the simulation is identical to J_{qn}, and we establish the proof for **Lemma** 5.

7 Experimental Evaluation

In this section, we show an experimental evaluation of our msq-ORE scheme. We analyze our ORE scheme by evaluating it in comparison with [6, 8] and [15] and implement these schemes in Python. The experiment environment is identical to that of Sect. 5. We show the evaluation results with respect to ciphertext size, encryption time and comparison time. We randomly select 1,000 data on each from three datasets (California payroll [1], Gowalla [9], Foursquare [26]) and show average value for performance test in Fig. 3. Finally, a theoretical analysis is given in Table 2.

7.1 Evaluation

In Fig. 3a, we show the experimental result for the ciphertext size of ORE schemes. It shows that Chenette et al. [8]'s scheme needs the smallest ciphertext size and that of [6] is the biggest. This is because the other three ORE schemes are based on Chenette et al. [8]'s ORE. The proposed msq-ORE and Lv et al. [15]'s scheme have an identical size, because both use modular operations, there is no change in size even if the calculated value is added to the exponential value. In Fig. 3b, we show the result of the encryption time for four schemes with three datsets. Our scheme's encryption speed is almost identical with Lv et al. [15]'s scheme. Because when our msq-ORE scheme encrypts the message u_i (e.g. $h_1 \leftarrow g_1^{H(k_1,u_i)\cdot k_b \cdot r_d}$), it adds $r_d \cdot H(k_1, u_i)$ from k_b, which is the same as that m-ORE computes the message u_i (e.g. $h_1 \leftarrow g_1^{H(k_1,u_i)\cdot k_{2,1}\cdot r}$) by adding $r \cdot H(k_1, u_i)$ from $k_{2,1}$ in the encryption algorithm. In Fig. 3c, we introduce the computational costs for the comparison algorithm. The results of the comparison time are almost identical, because our msq-ORE and m-ORE processes are identical in the comparison algorithm.

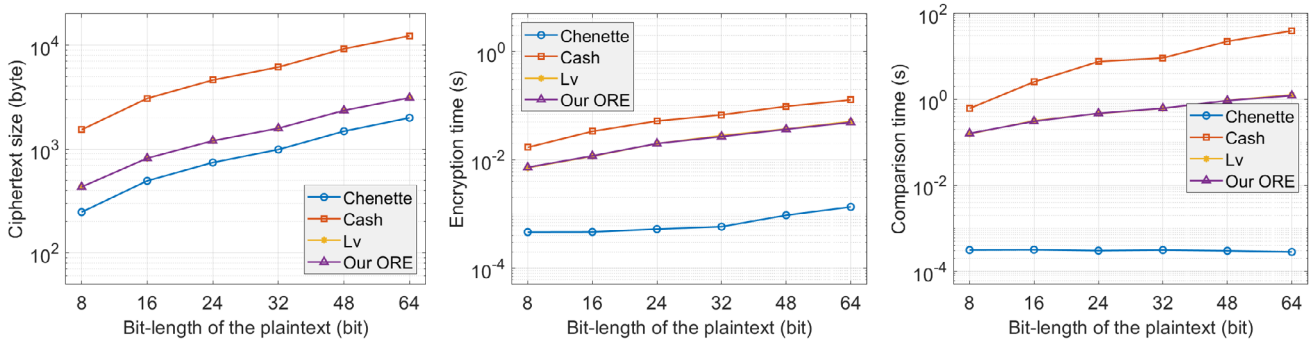
Table 2 Comparison of parameter-hiding ORE schemes. E_1, E_2, and P refer to the exponentiation operation in G_1 and G_2, and the pairing operation, respectively

Scheme	Encrypt	Query	Comparison
Cash et al. [6]	$2n(E_1 + E_2)$	–	$4n^2P$
m-ORE [15]	$(n+1)E_1$	$(2n+1)E_2$	$3nP$
msq-ORE	$(n+1)E_1$	$(2n+1)E_2$	$3nP$

We show that our msq-ORE requires almost identical computational cost with m-ORE [15] by using three datasets. Therefore, our msq-ORE scheme is not only guarantees stronger security in multi-client environments than m-ORE scheme [15], but also needs the similar computational cost.

8 Conclusion

Order-Revealing Encryption (ORE) marks a significant advancement in secure range query processing over encrypted databases. While earlier schemes promised improved privacy preservation, they often came with prohibitive computational costs. In response to these challenges, this paper introduces msq-ORE, a multi-client secure range query ORE scheme that effectively mitigates vulnerabilities while maintaining computational efficiency comparable to the state-of-the-art. The practical significance of this work lies in its ability to provide secure and efficient range query processing over encrypted databases, without compromising on computational performance. This not only enhances privacy preservation in real-world applications but also paves the way for broader adoption of secure computing techniques across various domains.



(a) Ciphertext size (b) Encryption time (c) Comparison time

Fig. 3 Performance comparison

Looking ahead, future research directions may include further refinement and analysis of msq-ORE, extending its applicability to more complex query types and larger datasets, and exploring optimizations to improve performance. Additionally, continued research into vulnerabilities and attack vectors will be crucial for maintaining the efficacy of encryption schemes in safeguarding sensitive data in an evolving threat landscape.

Acknowledgements This study was supported by the Research Program funded by the SeoulTech(Seoul National University of Science and Technology).

Data availibility The datasets generated during and/or analysed during the current study are the California public employee payroll data [1], Gowalla [9] and Foursquare [26].

Declarations

Conflict of interest The authors declare that they have no Conflict of interest as defined by Springer or other interests thatmight be perceived to influence the results and/or discussion reported in this paper.

References

1. California public employee payroll data, 2014. source: transparent California
2. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, 2004, pp. 563–574
3. Bisong,E., Bisong, E.: Google colaboratory, Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners, pp. 59–64 (2019)
4. Boldyreva, A., Chenette, N., Lee, Y., O'neill, A.: Order-preserving symmetric encryption. In: Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26–30, 2009. Proceedings 28, pp. 224–241. Springer (2009)
5. Boneh, D., Lewi, K., Raykova, M., Sahai, A., Zhandry, M., Zimmerman, J.: Semantically secure order-revealing encryption. Multi-input functional encryption without obfuscation. In: Advances in Cryptology-EUROCRYPT, pp. 563–594. Springer. Heidelberg (2015)
6. Cash, D., Liu, F.-H., O'Neill, A., Zhandry, M., Zhang, C.: Parameter-hiding order revealing encryption, in Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part I 24, pp. 181–210. Springer (2018)
7. Chen, X., Li, C., Wang, D., Wen, S., Zhang, J., Nepal, S., Xiang, Y., Ren, K.: Android hiv: a study of repackaging malware for evading machine-learning detection. IEEE Trans. Inf. Forensics Secur. **15**, 987–1001 (2019)
8. Chenette, N., Lewi, K., Weis, S.A., Wu, D.J.: Practical order-revealing encryption with limited leakage. In: Fast Software Encryption, 23rd International Conference, FSE: Bochum, Germany, March 20–23, 2016, Revised Selected Papers 23, pp. 474–493. Springer 2016 (2016)
9. Cho, E., Myers, S.A., Leskovec, J.: Friendship and mobility: user movement in location-based social networks. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2011, pp. 1082–1090
10. Clement, N.: M & a effect on data breaches in hospitals: 2010–2022 (2023)
11. Danezis, G.: A bilinear pairing library
12. Kerschbaum,F.: Frequency-hiding order-preserving encryption. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015, pp. 656–667
13. Lewi, K., Wu, D.J.: Order-revealing encryption: new constructions, applications, and lower bounds. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 1167–1178
14. Lin, G., Wen, S., Han, Q.-L., Zhang, J., Xiang, Y.: Software vulnerability detection using deep neural networks: a survey. Proc. IEEE **108**, 1825–1848 (2020)
15. Lv, C., Wang, J., Sun, S.-F., Wang, Y., Qi, S., Chen, X.: Efficient multi-client order-revealing encryption and its applications. In: Computer Security-ESORICS, 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part II 26, pp. 44–63. Springer 2021 (2021)
16. Lv, C., Wang, J., Sun, S.F., Wang, Y., Qi, S., Chen, X.: Towards practical multi-client order-revealing encryption: improvement and application. IEEE Trans. Depend. Secure Comput. (2023)
17. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015, pp. 644–655
18. Popa, R.A., Li, F.H., Zeldovich, N.: An ideal-security protocol for order-preserving encoding. In: IEEE Symposium on Security and Privacy. IEEE 2013, pp. 463–477 (2013)
19. Roche, D.S., Apon, D., Choi, S.G., Yerukhimovich, A.: Pope: partial order preserving encoding. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 1131–1142
20. Yadav, V.K., Andola, N., Verma, S., Venkatesan, S.: Pscls: provably secure certificateless signature scheme for iot device on cloud. J. Supercomput. **79**, 4962–4982 (2023)
21. Yadav, V.K., Venkatesan, S., Verma, S.: Man in the middle attack on ntru key exchange. In: Communication, Networks and Computing: First International Conference, CNC Gwalior, India, March 22–24, 2018, Revised Selected Papers 1, pp. 251–261. Springer (2019) (2018)
22. Yadav, V.K., Verma, S., Venkatesan, S.: An efficient and light weight polynomial multiplication for ideal lattice-based cryptography. Multim. Tools Appl. **80**, 3089–3120 (2021)
23. Yadav, V.K., Verma, S., Venkatesan, S.: Efficient and privacy-preserving location-based services over the cloud. Clust. Comput. **25**, 3175–3192 (2022)
24. Yadav, V.K., Yadav, R.K., Chaurasia, B.K. Verma, S., Venkatesan, S.: Mitm attack on modification of Diffie-Hellman key exchange algorithm. In: Communication, Networks and Computing: Second International Conference, CNC: Gwalior, India, December 29–31, 2020, Revised Selected Papers 2, pp. 144–155 Springer (2021) (2020)
25. Yadav, V.K., Yadav, R.K., Verma, S., Venkatesan, S.: Cp2eh: a comprehensive privacy-preserving e-health scheme over cloud. J. Supercomput. 1–31 (2022)
26. Yang, D., Zhang, D., Qu, B.: Participatory cultural mapping based on collective behavior data in location-based social networks. ACM Trans. Intell. Syst. Technol. **7**, 1–23 (2016)

Jae Hwan Park is an undergraduate student in the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, Korea, since 2017. His research interests include information security and applied cryptography.

Zeinab Rezaeifar received her B.S. in Communication Engineering, from Shahid Bahonar University of Kerman, Iran in 2008 and an M.S. degree in Network Communication Engineering, from Isfahan University of Technology, Iran in 2012. She received her Ph.D. degree in Computer Science and Engineering from Hanyang University, South Korea in 2018. From 2018 to 2020, she was doing a postdoctoral in Computer Science and Engineering department at Korea University. From April 2020 to Jun 2021, she was working as a Research Associate at BT Ireland Innovation Center (BTIIC) at Ulster University. She is currently working as a lecturer in Computer Science at University of the West of England. Her main research interests include security issues in wireless charging of Electric Vehicle (EV), security and network issues in VANET, security issues in Named Data Networking, and multi-step attacks (cyber attacks) detection and reconstruction.

Changhee Hahn received B.S. and M.S. degrees from Chung-Ang University, Seoul, South Korea, in 2014 and 2016, respectively, both in Computer Science. He received the Ph.D. degree in 2020 in the Department of Computer Science and Engineering, College of Informatics, Korea University, Korea. He was with Korea University as a postdoctoral researcher from 2020 to 2021. He is currently an Assistant Professor with the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, Korea. His research interests include information security and cloud computing security.