# UTRA: Universal Token Reusability Attack and Token Unforgeable Delegatable Order-Revealing Encryption

Jaehwan Park[1⋆], Hyeonbum Lee[2⋆], Junbeom Hur[3], Jae Hong Seo[2⋆⋆], and Doowon Kim[1⋆⋆]

[1] University of Tennessee, Knoxville
{jpark127,doowon}@utk.edu
[2] Hanyang University, Seoul
{leehb3706,jaehongseo}@hanyang.ac.kr
[3] Korea University, Seoul
jbhur@isslab.korea.ac.kr

**Abstract.** As datasets grow, users increasingly rely on cloud services for data storage and processing. Consequently, concerns regarding data protection and the practical use of encrypted data have emerged as significant challenges. One promising solution is order-revealing encryption (ORE), which enables efficient operations on encrypted numerical data. To support distributed environments with different users, delegatable ORE (DORE) extends this functionality to multi-client settings, enabling order comparisons between ciphertexts encrypted under different secret keys. However, Hahn et al. proposed a token forgery attack against DORE with a threat model and introduced the secure DORE (SEDORE) scheme as a countermeasure. Despite this enhancement, we claim that SEDORE remains vulnerable under the same threat model.
In this paper, we present a novel Universal Token Reusability Attack, which exposes a critical vulnerability in SEDORE with the identical threat model. To mitigate this, we introduce the concept of verifiable delegatable order-revealing encryption (VDORE), along with a formal definition of token unforgeability. Building on this, we design a new scheme, Token Unforgeable DORE (TUDORE), which ensures token unforgeability. Moreover, TUDORE achieves 1.5× faster token generation than SEDORE with enhanced security.

**Keywords:** Order-revealing encryption · Cross-database system · Token-based authentication

## 1 Introduction

With the increasing size of datasets, processing tasks on local machines are becoming impractical, thereby driving the demand for cloud-based services. How-

⋆ Equal contribution
⋆⋆ Co-corresponding authors

ever, uploading data to cloud services without protections raises privacy concerns. Clients encrypt their data before outsourcing it to servers to address these issues. However, even after encryption, there remains a need to support efficient queries, particularly range queries, which are fundamental in applications such as health service [36,22], finance [33], and location-based service (LBS) [10,35]. For instance, doctors may need to determine whether a patient's HIV viral load exceeds a clinical threshold, financial systems may compare encrypted bids to identify those above the current maximum, and location-based services may retrieve nearby points of interest within a specified distance for route planning.

Since standard encryption schemes do not preserve the order of values, specialized cryptographic constructions are required to support range queries. Typical solutions include fully homomorphic encryption (FHE), order-preserving encryption (OPE), and order-revealing encryption (ORE). FHE provides strong privacy by allowing computations over encrypted data without leakage, but its high computational cost limits practical use [13]. OPE is a symmetric scheme that allows direct ciphertext comparison by preserving plaintext order with low overhead. However, OPE leaks more information than ideal ORE schemes [6].

To strike a better balance between efficiency and security, ORE has been proposed as a means to support such secure operations [11,8,18,17,14,27,5,24]. ORE is a method that reveals only the order by using a publicly disclosed comparison function, without leaking any information about the numerical datasets. For instance, ORE takes two ciphertexts as input and returns the order associated with the underlying plaintexts. With the advancement of ORE, Li et al. [17] proposed a delegatable ORE (DORE) scheme, which supports comparisons across different encryption keys. DORE enables data owners to issue one-time authorization tokens to users based on their secret keys.

However, Hahn et al. [14] identified vulnerabilities in DORE, demonstrating that authorization tokens could be forged by unauthorized users under a practical threat model, which was later adopted by [27]. In these attacks, an authorized user (traitor) who receives an access token from the data owner (victim) collaborates with an unauthorized user (attacker) to enable illegal query execution on the victim's database. These attacks result not only in unauthorized data access but also in financial losses for victims. Because many modern cloud service providers (CSPs) [9,20] adopt a pay-per-query model. Furthermore, such attacks are stealthy since the data owner cannot identify the traitor. In response to these risks, Hahn et al. proposed secure DORE (SEDORE), which strengthens token unforgeability. Building on this, Xu et al. [34] introduced efficient DORE (EDORE), which enhances performance compared to SEDORE.

### 1.1 Our contribution

Despite the improvements in SEDORE and EDORE to mitigate practical forgery attacks, we discover that the same vulnerability exists in DORE [17], SEDORE [14], and EDORE [34] under the identical threat model proposed by [14]. We term this vulnerability as *universal token reusability*. This attack remains highly threatening, as it adheres to the practical and reasonable threat model defined

by [14], even though the traitor provides the attacker with one additional piece of information in our scenario compared to that of [14].

Given the limitations of prior work, we propose a revised DORE scheme incorporating a verification algorithm, termed verifiable DORE (VDORE). We formally define the token unforgeability of the VDORE scheme for provable security and also explain the limitations of [14]'s security analysis. Moreover, we propose TUDORE, a novel token-unforgeable DORE scheme, built upon VDORE. Like SEDORE [14], our scheme retains the original algorithms of DORE [17] for setup, key generation, encryption, and test, while modifying the token generation algorithm to prevent attacks. This ensures minimal computational and storage overhead. Additionally, we incorporate digital signature schemes [29,7] into the token generation process to guarantee token unforgeability. We provide security analysis and experimental results showing that TUDORE achieves competitive efficiency compared to previous works [17,14,34]. In summary, we make the following main contributions:

1. **Universal Token Reusability Attack.** We first highlight that token-based DORE schemes [17,14,34] remain vulnerable to token forgery attacks. Specifically, under the same threat model described in [14], we introduce a new attack called *universal token reusability attack* in Section 4.
2. **Token Verification and Security Definition.** To counter the security threats revealed by the attack scenario, we revise the token-based DORE scheme by integrating a token verification algorithm. We explain the necessity of verification and shortcomings of [14]'s security analysis in Section 4. This revision introduces verifiable DORE (VDORE), which ensures provable security through three properties: *correctness*, *data privacy*, and *token unforgeability* in Section 5.
3. **Token Unforgeable VDORE Scheme.** We present a new secure VDORE scheme, named token unforgeable DORE (TUDORE), which leverages digital signature schemes [29,7]. Especially, we prove that TUDORE satisfies the properties of *correctness*, *data privacy*, and *token unforgeability* in Section 6.
4. **Implementation of TUDORE.** We provide implementation results to compare existing schemes [17,14,34] with our proposed method (TUDORE). Our experimental evaluation demonstrates that TUDORE is not only practical and feasible but also enhances security compared to prior schemes, as detailed in Section 7.

### 1.2 Related Works

**Homomorphic Comparison from Fully Homomorphic Encryption.** Cheon et al. [13] introduced homomorphic comparison for private queries based on fully homomorphic encryption (FHE), which has been further improved in subsequent works [31,12]. These approaches encode the comparison function as an arithmetic circuit and evaluate it homomorphically using FHE. Owing to the strong cryptographic guarantees of FHE, the protocols achieve post-quantum security. However, evaluating even a single 64-bit comparison typically takes several seconds [13,31,12], which is impractical in real-world scenarios. Moreover, to the

best of our knowledge, no existing FHE-based construction has been proposed that supports delegatability. This limitation renders such methods impractical in multi-client scenarios where secret key sharing is undesirable or infeasible.

**Order-preserving Encryption.** Order-preserving Encryption (OPE) was initially introduced by Agrawal et al. [4]. Subsequently, Boldyreva et al. [6] introduced the notion of "best possible" security, referred to as indistinguishability under ordered chosen-plaintext attack (IND-OCPA). This property ensures that two ciphertexts reveal no information about plaintexts other than their order. However, Boldyreva also pointed out that achieving this ideal security is not feasible for stateless and immutable OPE schemes. Therefore, as an effort to resolve these limitations, Popa et al. [26] proposed an interactive order-preserving encoding method via server state. However, their scheme is interactive, requiring $\mathcal{O}(\log N)$ communication rounds for both database updates and query execution. To address this, alternative solutions impose a trade-off by incurring client-side storage costs ranging from $\mathcal{O}(N)$ [16] to $\mathcal{O}(N^{\delta})$ [28], where $0 < \delta < 1$. Nonetheless, despite improvements, OPE remains limited by interaction and trade-offs between performance and security.

**Order-revealing Encryption.** Order-revealing encryption (ORE) is a technique that encrypts numerical data without preserving the order of the plaintext, allowing the comparison of two ciphertexts using a public function to determine their order. To improve efficiency, Chenette et al. [11] proposed a practical ORE scheme that supports bitwise encryption and comparison. Cash et al. [8] introduced a parameter-hiding ORE (pORE) that reveals only the equality pattern of the most significant differing bit (msdb). However, these schemes are limited to a single-user environment. Subsequently, various schemes have been proposed to support multi-client settings [18,24,17,27,34,5]. Especially, [17] proposed delegatable ORE (DORE), which supports comparisons over ciphertexts encrypted under different keys. Moreover, recent research [5] on ORE extends beyond traditional database applications and explores its use in machine learning algorithms.

**Organization.** This paper is organized as follows. In Section 2, we present the background of cross-database systems, delegatable ORE, and our system and threat model. Section 3 reviews existing token-based DORE schemes. In Section 4, we introduce a new attack called the universal token reusability attack targeting these schemes and the weaknesses in [14]'s security analysis. To address this, we propose the concept of verifiable DORE along with a formal security definition in Section 5. Finally, Section 6 and Section 7 describe the construction of a novel token-unforgeable DORE scheme and its experimental evaluation.

## 2 Background and Models

### 2.1 Background

**Cross-database systems.** In our system, similar to DORE [17], SEDORE [14], and EDORE [34], we consider a cross-database scenario. The cross-database system allows multiple users to upload their encrypted databases onto the server, based on their raw data. Users who want to collaborate and share datasets can perform relevant operations by sending queries to each other's databases.

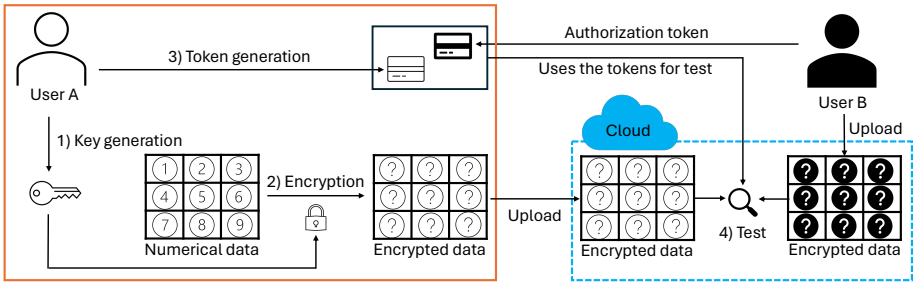



Fig. 1: The description of delegatable order-revealing encryption. The orange box with a solid line indicates the operations executed by User A, whereas the blue box with a dotted line represents the processes handled by the cloud.

However, it is essential to note that not all users on the cloud server can access all databases; only those users authorized by the database owner can utilize specific databases. From this, the database owner distributes authorization tokens to grant authorized users access.

**Delegatable Order-Revealing Encryption.** DORE scheme has been introduced in a multi-client environment [17]. This scheme allows the data owner to grant authorization tokens to other users, enabling them to perform operations on each other's databases based on different secret keys.

In Figure 1, we show the process of delegatable order-revealing encryption and explain it as follows: 1) User A generates their secret key using a key generation algorithm. 2) Afterward, they encrypt their numerical data with the key and upload it to the cloud. 3) If User A wishes to perform computations on User B's dataset, they obtain an authorization token from User B and then generate a token related to their dataset using the token generation algorithm. 4) When the server receives the tokens, it compares the encrypted data of User A and B with the tokens using a test algorithm. Finally, it determines the orders. Note that, the token is not issued per query but only once and remains valid thereafter.

### 2.2 System model

We consider a scenario involving cross-database environments with encrypted databases. In this context, there are three entities with the following roles:

- **The data owner**: Encrypts data using the secret key and uploads it to the server. The data owner also provides authorization tokens to users who are authorized to access its databases.
- **The user**: Requests an authorization token from the data owner. Upon receiving the token, the user can use it to perform computations involving their own database and the data owner's database.
- **The server**: Acts as a storage system for encrypted data uploaded by multiple data owners and processes incoming range queries from users.

Note that the entities uploading data to the server can all become data owners. Moreover, entities obtaining authorization tokens from different data owners can also access other databases.

### 2.3 Threat Model

Following the attack scenario described in [14], we consider two threat models related to data privacy violations and token forgeability, as follows:

- **Data privacy violation**: The server might attempt to disclose the content of the stored data, along with trying to acquire not only the ordering information and the index of the first differing bit between the two ciphertexts but also to recover the data.
- **Token forgeability**: The server and unauthorized users may attempt to access the victim's database by creating forged tokens.

From this perspective, we focus on the notion of *token forgeability*. There are three entities involved in token forgery attacks: a victim ($\mathcal{V}$), who owns the database; an unauthorized user ($\mathcal{A}$) who tried to access victim's database without any permission; an authorized user ($\mathcal{M}$), who may illegally assist the unauthorized user ($\mathcal{A}$) in generating forged tokens.

Importantly, we assume that $\mathcal{M}$ never shares its secret key with $\mathcal{A}$, as such disclosure would allow $\mathcal{A}$ to exploit not only $\mathcal{V}$'s database, but also $\mathcal{M}$'s database by enabling the generation of tokens derived from $\mathcal{M}$'s secret key.

## 3 Overview of Token-Based DORE Schemes

### 3.1 Basic Notation

We first define some notations before revisiting the schemes. We denote $\mathbb{Z}_p$ as a prime field that is isomorphic to the integers mod $p$. Uniform sampling is denoted by $\xleftarrow{\$}$. For instance, $a \xleftarrow{\$} \mathbb{Z}_p$ indicates that $a$ is uniformly chosen from $\mathbb{Z}_p$. $H$ and $F$ denote a cryptographic hash function whose range will be specified from the context. To describe bilinear groups, we denote $\langle p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e \rangle$, that stands for prime $p$ and cyclic groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of order $p$, generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, and bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ , which is non-degenerate and a computable function satisfies $e(P^a, K^b) = e(P, K)^{ab}$ for all $a, b \in Z_p$. Hereafter, we assume that the prime $p$ is sufficiently large because the security of DORE schemes relies on the discrete logarithm assumption.

### 3.2 Token-based DORE schemes

Li et al. [17] first proposed a token-based delegatable ORE scheme. The data owner delegates the management of data, but can manage authorization through the tokens. DORE scheme consists of 5 algorithms as follows:

- $\mathsf{pp} \leftarrow \mathsf{DORE.Setup}(1^\lambda)$: It takes the security parameter $1^\lambda$ as input and returns the public parameter $\mathsf{pp}$.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DORE.Keygen}(\mathsf{pp})$ : It receives a public parameter $\mathsf{pp}$ as input and returns a pair of public key and secret key $(\mathsf{pk}, \mathsf{sk})$
- $\mathsf{ct} \leftarrow \mathsf{DORE.Enc}(\mathsf{pp}, m, \mathsf{sk})$: It takes a message $m \in \{0,1\}^*$ and $\mathsf{sk}$ as input and returns a ciphertext $\mathsf{ct}$.
- $\mathsf{tok}_{(v\to u)} \leftarrow \mathsf{DORE.Token}(\mathsf{pp}, \mathsf{pk}_{(v)}, \mathsf{sk}_{(u)})$: It takes the public key $\mathsf{pk}_{(v)}$ of user $v$ and the secret key $\mathsf{sk}_{(u)}$ of user $u$ as input and returns an authorization token $\mathsf{tok}_{(v\to u)}$, indicating that user $v$ is authorized by user $u$.
- $res \leftarrow \mathsf{DORE.Test}(\mathsf{pp}, \mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: It takes the two ciphertext $\mathsf{ct}_{(u)}$ and $\mathsf{ct}_{(v)}$, along with two tokens, $\mathsf{tok}_{(v\to u)}$ and $\mathsf{tok}_{(u\to v)}$, as input and returns the comparison result $res \in \{-1, 0, 1\}$. The output values represent the following: 1 indicates $m_u > m_v$, 0 indicates $m_u = m_v$, and $-1$ indicates $m_u < m_v$, where $m_\square$ denotes the plaintext of $\mathsf{ct}_{(\square)}$ for $\square = u, v$.

Li et al.[17] introduced two security properties for DORE: correctness and IND-OCPA. Furthermore, Hahn et al.[14] emphasized that token forgeability is a critical security concern in the DORE scheme. Building on these works, token-based DORE schemes should satisfy three essential properties: correctness, data privacy, and soundness. We formalize and refine these properties in Section 5.

**DERE-to-DORE framework.** In [17], Li et al. proposed a framework for constructing a DORE scheme from a token-based Delegatable Equality-Revealing Encoding (DERE). The primary difference lies in the test algorithm: the DERE test algorithm is restricted to checking only the equality between two ciphertexts. From a DERE scheme, DORE leverages the key generation, encryption, and testing algorithms of DERE and constructs its own encryption and test algorithms by iteratively running the encryption and test algorithms of DERE. Subsequent researchs [14,34] follow this framework. For this reason, we now focus on the DERE scheme rather than the DORE scheme itself.

Li et al. constructed the DERE scheme based on the bilinear setting under the generic group model (GGM) [17]. To prevent a token forging attack, Hahn et al. revised the token generation of DERE and then proposed a new DERE scheme, SEDERE [14]. We describe the DERE and SEDERE schemes in Figure 2.

**One-sided token is sufficient.** In the $\mathsf{Test}$ algorithm, it currently requires two-sided tokens, $\mathsf{tok}_{(v\to u)}$ and $\mathsf{tok}_{(u\to v)}$. When a user $u$ queries a ciphertext $\mathsf{ct}_{(v)}$ stored in $v$'s database, $u$ first generates $\mathsf{tok}_{(v\to u)}$ locally. Since the public key $\mathsf{pk}_{(v)}$ is openly available, $u$ can compute $\mathsf{tok}_{(v\to u)}$ independently, without needing $v$'s secret key. Subsequently, $u$ uses the pair of tokens, $\mathsf{tok}_{(v\to u)}$ (generated locally) and $\mathsf{tok}_{(u\to v)}$ (received from user $v$), to perform the $\mathsf{Test}$ algorithm.

## 4 Concrete Attack and Security Analysis of SEDORE

In this section, we present a concrete attack, named the Universal Token Reusability Attack, on the SEDORE scheme, demonstrating its vulnerability under the threat model described in Section 2.3. We then discuss how this attack relates to the soundness guarantees of SEDORE as proven in [14].

- $\mathsf{pp} \leftarrow \mathsf{DERE.Setup}(1^\lambda)$: It takes the security parameter $1^\lambda$ as input and returns the public parameter $\mathsf{pp} = (\langle p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e\rangle, H, F)$.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DERE.Keygen}(\mathsf{pp})$ : It takes a public parameter $\mathsf{pp}$ as input and uniformly chooses $a, b \xleftarrow{\$} \mathbb{Z}_p$. After that, it returns a pair of public key and secret key $(\mathsf{pk}, \mathsf{sk}) = (g_2^a, (a, b))$. Additionally, We denote a key pair of user $u$ as $(\mathsf{pk}_{(u)}, \mathsf{sk}_{(u)}) = (g_2^{a_{(u)}}, (a_{(u)}, b_{(u)}))$.
- $\mathsf{ct} \leftarrow \mathsf{DERE.Enc}(\mathsf{pp}, m, \mathsf{sk})$: It takes a message $m \in \{0,1\}^*$ and $\mathsf{sk}$ as input. It randomly picks $r \xleftarrow{\$} \mathbb{Z}_p$ and return $\mathsf{ct} := (c^0, c^1) = ((g_1^r H(m))^a, c^1 = g_1^r)$. For user $u$, we rewrite $\mathsf{ct}$ as $\mathsf{ct}_{(u)} = (c^0_{(u)}, c^1_{(u)})$.
- $\mathsf{tok}_{(v\to u)} \leftarrow \mathsf{DERE.Token}(\mathsf{pp}, \mathsf{pk}_{(v)}, \mathsf{sk}_{(u)})$: It takes the public key $\mathsf{pk}_{(v)} = g_2^{a_{(v)}}$ of user $v$ and the secret key $\mathsf{sk}_{(u)} = (a_{(u)}, b_{(u)})$ of user $u$ and returns an authorization token $\mathsf{tok}_{(v\to u)}$. $(v \to u)$ from $\mathsf{tok}_{(v\to u)}$ means that the user $u$ sends the authorization token to user $v$ and $\mathsf{tok}_{(v\to u)}$ consists of $t^0_{(v\to u)}$ and $t^1_{(v\to u)}$.
  - (type-1, DERE [17]): $t^0_{(v\to u)} = \mathsf{pk}_{(v)}, \quad t^1_{(v\to u)} = \mathsf{pk}_{(v)}^{a_{(u)} b_{(u)}}$
  - (type-2, SEDERE [14]):$t^0_{(v\to u)} = F\left(\mathsf{pk}_{(v)}^{a_{(u)}}\right)^{a_{(u)}^{-1}}, \quad t^1_{(v\to u)} = F\left(\mathsf{pk}_{(v)}^{a_{(v)}}\right)^{b_{(v)}}$

  Finally, it returns $\mathsf{tok}_{(v\to u)} := (t^0_{(v\to u)}, t^1_{(v\to u)})$.
- $0\backslash 1 \leftarrow \mathsf{DERE.Test}(\mathsf{pp}, \mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: It takes the ciphertexts from user $v$ and $u$, $\mathsf{ct}_{(v)}$ and $\mathsf{ct}_{(u)}$, and the tokens, $\mathsf{tok}_{(v\to u)}$ and $\mathsf{tok}_{(u\to v)}$ as input. After that, it computes $d_0 = \frac{e(c^0_{(u)}, t^0_{(v\to u)})}{e(c^1_{(u)}, t^1_{(v\to u)})}$ , $d_1 = \frac{e(c^0_{(v)}, t^0_{(u\to v)})}{e(c^1_{(v)}, t^1_{(u\to v)})}$. It then compares them: if $d_0 = d_1$, it returns 1; otherwise, it returns 0.

Fig. 2: DERE and SEDERE Scheme

### 4.1 Universal Token Reusability Attack

Following the DERE-to-DORE framework, the vulnerability in SEDERE also leads to SEDORE. For this reason, we present an attack on the SEDERE scheme in Figure 2, to validate the existence of the vulnerability in SEDORE. Before explaining the attack, we propose the notion of a universal forged token in the bilinear setting. A universal forged token $\mathsf{uft}_{(\mathcal{V}),h_2}$ is a forged token to access $\mathcal{V}$ based on group element $h_2$. Using $\mathsf{uft}_{(\mathcal{V}),h_2}$ and $h_2$, any adversary can query the database of $\mathcal{V}$ without authorized token from $\mathcal{V}$. We define a universal forged token as $\mathsf{uft}_{(\mathcal{V}),h_2} = (h_2^{a_{(\mathcal{V})}^{-1}}, h_2^{b_{(\mathcal{V})}})$ in our attack. We introduce it as follows:
**Step 1**: The user $\mathcal{V}$ creates authorization token $\mathsf{tok}_{(\mathcal{M}\to\mathcal{V})}$ by using (type-2) $\mathsf{DERE.Token}$ algorithm [4] in Figure 2 and sends it to user $\mathcal{M}$ as below:

$$\mathsf{tok}_{(\mathcal{M}\to\mathcal{V})} = \left(F\left(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}}\right)^{a_{(\mathcal{V})}^{-1}}, F\left(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}}\right)^{b_{(\mathcal{V})}}\right)$$

[4] If the map $F$ is the identity map on $\mathbb{G}_2$, type-2 $\mathsf{DERE}$ scheme becomes identical to the type-1 $\mathsf{DERE}$ scheme. Thus, the following attack against the type-2 $\mathsf{DERE}$ scheme can naturally be applied to the type-1 $\mathsf{DERE}$ scheme.

**Step 2**: After $\mathcal{M}$ receives it, $\mathcal{M}$ randomly picks $r \xleftarrow{\$} \mathbb{Z}_p$ and sets a group element $h_2 = F(\mathsf{pk}_{(\mathcal{V})}^{a_{(\mathcal{M})}})^r$. And then $\mathcal{M}$ computes a universal forged token $\mathsf{uft}_{(\mathcal{V}),h_2}$ as following:

$$\mathsf{uft}_{(\mathcal{V}),h_2} = \mathsf{tok}^r_{(\mathcal{M}\to\mathcal{V})} = \left(\left(F(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}})^r\right)^{a_{(\mathcal{V})}^{-1}}, \left(F(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}})^r\right)^{b_{(\mathcal{V})}}\right)$$

After then, $\mathcal{M}$ sends $h_2$ and $\mathsf{uft}_{(\mathcal{V}),h_2}$ to $\mathcal{A}$. Note that $\mathcal{M}$ can compute $\mathsf{uft}_{(\mathcal{V}),h_2}$ by symmetric property $\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}} = g_2^{a_{\mathcal{V}} a_{\mathcal{M}}} = \mathsf{pk}_{(\mathcal{V})}^{a_{(\mathcal{M})}}$. Since $h_2$ is randomized by $\mathcal{M}$'s randomness $r$, $(h_2, \mathsf{uft}_{(\mathcal{V}),h_2}) \in \mathbb{G}_2^3$ look like uniformly random in the view of $\mathcal{A}$. By DL assumption, it is interactable for the $\mathcal{A}$ to find $\mathcal{M}$'s secret key $(a_{(\mathcal{M})}, b_{(\mathcal{M})})$ from $h_2$ and $\mathsf{uft}_{(\mathcal{V}),h_2}$. For this reason, $\mathcal{M}$ may help adversary $\mathcal{A}$ without concern about leaking $\mathcal{M}$'s secret.

**Step 3**: After receiving $(h_2, \mathsf{uft}_{(\mathcal{V}),h_2})$, $\mathcal{A}$ samples $\mathsf{sk}_{(\mathcal{A})} = (a_{(\mathcal{A})}, b_{(\mathcal{A})}) \xleftarrow{\$} \mathbb{Z}_p^2$. And it computes the counterpart forged token $\mathsf{uft}_{(\mathcal{A}),h_2}$ as follows:

$$\mathsf{uft}_{(\mathcal{A}),h_2} = (h_2^{a_{(\mathcal{A})}^{-1}}, h_2^{b_{(\mathcal{A})}})$$

For the query, $\mathcal{A}$ generates $\mathsf{ct}_{(\mathcal{A})} \leftarrow \mathsf{DERE.Enc}(\mathsf{pp}, m, \mathsf{sk}_{(\mathcal{A})})$ using her secret key $(a_{(\mathcal{A})}, b_{(\mathcal{A})})$ and then use a pair of forged tokens $\mathsf{uft}_{(\mathcal{A}),h_2}$ and $\mathsf{uft}_{(\mathcal{V}),h_2}$.

For a given message $m$, let us denote the victim's ciphertext as $\mathsf{ct}_{(\mathcal{V})} = ((g_1^{b_{(\mathcal{V})} r_{(\mathcal{V})}} H(m))^{a_{(\mathcal{V})}}, g_1^{r_{(\mathcal{V})}})$. Then we can get $\mathsf{DERE.Test}(\mathsf{ct}_{(\mathcal{V})}, \mathsf{ct}_{(\mathcal{A})}, \mathsf{uft}_{(\mathcal{V}),h_2}, \mathsf{uft}_{(\mathcal{A}),h_2}) = 1$ by the following equations: For $i = 0, 1$, $\alpha_0 = \mathcal{V}$ and $\alpha_1 = \mathcal{A}$,

$$\begin{aligned} d_i = \frac{e(c^0_{(\alpha_i)}, \mathsf{uft}^0_{(\alpha_i),h_2})}{e(c^1_{(\alpha_i)}, \mathsf{uft}^1_{(\alpha_i),h_2})} &= \frac{e((g_1^{b_{(\alpha_i)} r_{(\alpha_i)}} H(m))^{a_{(\alpha_i)}}, h_2^{a_{(\alpha_i)}^{-1}})}{e(g_1^{r_{(\alpha_i)}}, h_2^{b_{(\alpha_i)}})} \\ &= \frac{e(g_1^{b_{(\alpha_i)} r_{(\alpha_i)}} H(m), h_2)}{e(g_1^{b_{(\alpha_i)} r_{(\alpha_i)}}, h_2)} = e(H(m), h_2). \end{aligned}$$

In other words, $\mathcal{A}$ can be identified as equality between $\mathsf{ct}_{(\mathcal{V})}$ and $\mathsf{ct}_{(\mathcal{A})}$ plaintexts by the $\mathsf{Test}$ algorithm without the authorized token. The universal token reusability attack can also be applied to EDORE, but we defer the attack to the full version [25] due to space limitations.

### 4.2 Limitation of security analysis in [14].

In [14], the authors proposed a soundness game to evaluate the resilience of SEDORE against token forgery attacks. In this game, the adversary aims to construct a *forged token* $\mathsf{tok}^*_{(\mathcal{C}\to\mathcal{A})}$ which, when used with a valid token $\mathsf{tok}_{(\mathcal{A}\to\mathcal{C})}$ generated by $\mathsf{DERE.Token}$, yields a valid result under $\mathsf{DERE.Test}$ in Figure 2. They incorporated cryptographic hash functions into the token generation to ensure soundness, making it computationally infeasible to construct $\mathsf{tok}^*_{(\mathcal{C}\to\mathcal{A})}$.

However, the structure of the soundness game limits the capabilities of the adversary. In practice, an adversary may generate a pair of forged tokens $(\mathsf{tok}^*_{(\mathcal{V}\to\mathcal{A})},$

$\mathsf{tok}^*_{(\mathcal{A}\to\mathcal{V})}$), that deviate from the expected structure in Figure 2. Although the protocol assumes that only properly generated token pairs can be used to query the database, our findings from the attack described above demonstrate that even malformed tokens can still produce valid results under the DERE.Test algorithm. Therefore, the soundness definition in [14] should be revised to account for such adversarial behavior.

The main reason for this vulnerability is that the DERE.Test algorithm does not verify the validity of tokens by itself. Therefore, before executing the test, the server (i.e., the tester) must ensure that the tokens were genuinely issued by authorized users and have not been forged.

## 5 Verifiable DORE (VDORE) and Token Unforgeability

As we mentioned in Section 4, the tester should check the validity of the tokens to prevent universal token reusability attacks. To give a verifiability on DORE scheme (Section 3.2), we revise the Keygen algorithm to output verification key vk and we newly propose a verification algorithm Vfy. We define a verifiable DORE scheme VDORE := (Setup, Keygen, Enc, Token, Test, Vfy) as follows:

- $\mathsf{pp} \leftarrow \mathsf{VDORE.Setup}(1^\lambda)$: It takes the security parameter $1^\lambda$ as input and returns the public parameter pp.
- $(\mathsf{pk}, \boxed{\mathsf{vk}}, \mathsf{sk}) \leftarrow \mathsf{VDORE.Keygen}(\mathsf{pp})$: It takes a public parameter as input and returns a key tuple of public key, verification key, and secret key, $(\mathsf{pk}, \mathsf{vk}, \mathsf{sk})$. The verification key is used to verify the validity of tokens. Both pk and vk may be managed publicly, but sk should be managed privately.
- $\mathsf{ct} \leftarrow \mathsf{VDORE.Enc}(\mathsf{pp}, m, \mathsf{sk})$: It takes a message $m \in \{0,1\}^*$ and sk as input and returns a ciphertext ct.
- $\mathsf{tok}_{(v\to u)} \leftarrow \mathsf{VDORE.Token}(\mathsf{pp}, \mathsf{pk}_{(v)}, \mathsf{sk}_{(u)})$: It takes the public key $\mathsf{pk}_{(v)}$ of user $v$ and the secret key $\mathsf{sk}_{(u)}$ of user $u$ as input and returns an authorization token $\mathsf{tok}_{(v\to u)}$, indicating that user $v$ is authorized by user $u$.
- $res \leftarrow \mathsf{VDORE.Test}(\mathsf{pp}, \mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: It takes the two ciphertext $\mathsf{ct}_{(u)}$ and $\mathsf{ct}_{(v)}$, along with two tokens, $\mathsf{tok}_{(v\to u)}$ and $\mathsf{tok}_{(u\to v)}$, as input and returns the comparison result $res \in \{-1, 0, 1\}$. The output values represent the following: 1 indicates $m_u > m_v$, 0 indicates $m_u = m_v$, and $-1$ indicates $m_u < m_v$, where $m_\square$ denotes the plaintext of $\mathsf{ct}_{(\square)}$ for $\square = u, v$.
- $0\backslash 1 \leftarrow \boxed{\mathsf{VDORE.Vfy}(\mathsf{pp}, \mathsf{vk}_{(u)}, \mathsf{vk}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})}$: It takes the two verification keys $\mathsf{vk}_{(u)}$ and $\mathsf{vk}_{(v)}$, and two tokens $\mathsf{tok}_{(v\to u)}$ and $\mathsf{tok}_{(u\to v)}$ as input. If both tokens $\mathsf{tok}_{(v\to u)}$ and $\mathsf{tok}_{(u\to v)}$ go through (token validity) verification, it returns 1 (accept); otherwise, it returns 0 (reject).

To ensure a secure VDORE scheme, we consider three properties: *correctness*, *data privacy*, and *token unforgeability*.

**Correctness.** The correctness of VDORE ensures that the test algorithm accurately discerns the sequence of two ciphertexts provided by two mutually authenticated users. Let $(m_{(u)}, m_{(v)})$ be a pair of messages, $(\mathsf{pk}_{(u)}, \mathsf{vk}_{(u)}, \mathsf{sk}_{(u)})$,

1. **Setting Phase**: $\mathcal{C}$ runs the setup algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and the key generation algorithm $(\mathsf{sk}_{(\mathcal{C})}, \mathsf{vk}_{(\mathcal{C})}, \mathsf{pk}_{(\mathcal{C})}) \leftarrow \mathsf{Keygen}(\mathsf{pp})$. It then sends $(\mathsf{pp}, \mathsf{vk}_{(\mathcal{C})}, \mathsf{pk}_{(\mathcal{C})})$ to $\mathcal{A}$ and keeps $\mathsf{sk}_{(\mathcal{C})}$ local storage.
2. **Query Phase**: $\mathcal{A}$ can query to $\mathcal{C}$:
   (a) **Key Query**: $\mathcal{A}$ sends a query with index $i$. Upon receiving the query, $\mathcal{C}$ checks whether $(i, \mathsf{pk}_{(i)}, \mathsf{vk}_{(i)}) \in S_{\mathsf{key}}$. If so, it returns $(i, \mathsf{pk}_{(i)}, \mathsf{vk}_{(i)})$ to $\mathcal{A}$. Otherwise, it generates a new key triple by running $(\mathsf{pk}_{(i)}, \mathsf{vk}_{(i)}, \mathsf{sk}_{(i)}) \leftarrow \mathsf{Keygen}(\mathsf{pp})$, returns $(\mathsf{pk}_{(i)}, \mathsf{vk}_{(i)})$ to $\mathcal{A}$, and adds the tuple $(i, \mathsf{pk}_{(i)}, \mathsf{vk}_{(i)})$ to the key query set $S_{\mathsf{key}}$.
   (b) **Token Query**: If $\mathcal{A}$ sends a query with a received public key $\mathsf{pk}_{(i)} \in S_{\mathsf{key}}$, $\mathcal{C}$ generates an authorized token $\mathsf{tok}_{(i\to\mathcal{C})} \leftarrow \mathsf{Token}(\mathsf{pp}, \mathsf{pk}_{(i)}, \mathsf{sk}_{(\mathcal{C})})$ and then sends $\mathsf{tok}_{(i\to\mathcal{C})}$ to $\mathcal{A}$ and adds $\mathsf{tok}_{(i\to\mathcal{C})}$ to token query set $S_{\mathsf{tok}}$.
   (c) **One-way function Query**: If $\mathcal{A}$ sends a query with index $i$ and one-way function $f$, $\mathcal{C}$ output $f(\mathsf{sk}_{(i)})$.
3. **Challenge Phase**: $\mathcal{A}$ outputs a verification key $\mathsf{vk}^*_{(\mathcal{A})}$ and a pair of tokens $(\mathsf{tok}^*_{(\mathcal{C}\to\mathcal{A})}, \mathsf{tok}^*_{(\mathcal{A}\to\mathcal{C})})$. The $\mathcal{A}$ wins if $\mathsf{Vfy}(\mathsf{pp}, \mathsf{vk}_{(\mathcal{C})}, \mathsf{vk}^*_{(\mathcal{A})}, \mathsf{tok}^*_{(\mathcal{A}\to\mathcal{C})}, \mathsf{tok}^*_{(\mathcal{C}\to\mathcal{A})}) = 1$.

Fig. 3: Token forging Game

$(\mathsf{pk}_{(v)}, \mathsf{vk}_{(v)}, \mathsf{sk}_{(v)})$ be a pair of keys generated by $\mathsf{Keygen}$ algorithm, and $\mathsf{ct}_{(u)}$, $\mathsf{ct}_{(v)}$ be a ciphertext of $m_{(u)}$ and $m_{(v)}$ with key $\mathsf{sk}_{(u)}$ and $\mathsf{sk}_{(v)}$ respectively. We say $\mathsf{VDORE}$ scheme is correct if for any pair of messages $(m_{(u)}, m_{(v)})$ and keys $(\mathsf{pk}_{(u)}, \mathsf{vk}_{(u)}, \mathsf{sk}_{(u)})$, $(\mathsf{pk}_{(v)}, \mathsf{vk}_{(v)}, \mathsf{sk}_{(v)})$, the following holds:

- $\mathsf{VDORE.Vfy}(\mathsf{pp}, \mathsf{vk}_{(u)}, \mathsf{vk}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)}) = 1$
- $\mathsf{VDORE.Test}(\mathsf{pp}, \mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)}) = res$
  - If $m_{(u)} > m_{(v)}$, then $res = 1$
  - If $m_{(u)} < m_{(v)}$, then $res = -1$
  - Otherwise, $res = 0$

**Data Privacy.** The data privacy of $\mathsf{VDORE}$ ensures that the ciphertexts $\mathsf{ct}$ generated by $\mathsf{Enc}$ algorithm do not leak information except order. $\mathsf{VDORE}$ provides data privacy if $\mathsf{Enc}$ algorithm satisfies indistinguishability under an ordered chosen plaintext attack (IND-OCPA) [17].

**Token Unforgeability.** $\mathsf{Vfy}$ algorithm of the VDORE scheme should detect forged tokens. To give provable security, we propose a new definition of token unforgeability. First, we construct a token forging game to give a game-based security. We describe the roles of adversary and challenger in Figure 3.

We denote the $\mathcal{A}$'s advantage to the token forging game of VDORE scheme(or Vdere scheme) $\mathsf{Adv}^{\mathsf{TF}}[\mathcal{A}, \mathsf{VDORE}]$(or $\mathsf{Adv}^{\mathsf{TF}}[\mathcal{A}, \mathsf{VDERE}]$), which is a probability that $\mathcal{A}$ wins the token forging game under the $\mathsf{VDORE}$(or $\mathsf{VDERE}$) scheme.

**Definition 1 (Token Unforgeability).** *Let* $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Test}, \mathsf{Vfy})$ *be a* $\mathsf{VDORE}$ *(or* $\mathsf{VDERE}$*) scheme. We say that* $\mathsf{VDORE}$ *(or* $\mathsf{VDERE}$*) satisfies token unforgeability if for any PPT adversary* $\mathcal{A}$ *against token forging game in*

*Figure 3, the $\mathcal{A}$'s advantage to the game* $\mathsf{Adv}^{\mathsf{TF}}[\mathcal{A}, \mathsf{VDORE}]$ *(or* $\mathsf{Adv}^{\mathsf{TF}}[\mathcal{A}, \mathsf{VDERE}]$*) is less than* $\mathsf{negl}(\lambda)$.

**Token Forging Game and Attack Scenario.** In this paragraph, we explain the relationship between the token forging game and the attack scenario. The key pair $(\mathsf{pk}_{(\mathcal{C})}, \mathsf{vk}_{(\mathcal{C})})$ sent by $\mathcal{C}$ represents the public and verification keys of the victim $\mathcal{V}$. Note that other users, including the adversary, may know the public key $\mathsf{pk}_{(\mathcal{V})}$ and verification key $\mathsf{vk}_{(\mathcal{V})}$ of $\mathcal{V}$.

After that, we allow the adversary $\mathcal{A}$ to make three types of queries: key queries, token queries, and one-way function queries. Through key and token queries, $\mathcal{A}$ can obtain multiple public and verification keys along with their corresponding authorized tokens—representing resources obtained from colluding users. Additionally, we allow $\mathcal{A}$ to make one-way function queries to obtain specific values derived from the secret keys of colluding users. Due to the one-wayness of the function, it remains computationally hard for $\mathcal{A}$ to recover the secret key from the output, thereby modeling feasible collusion scenarios without revealing the users' secret keys.

The goal of $\mathcal{A}$ is to find a pair of forged tokens that successfully pass the verification algorithm $\mathsf{Vfy}$. The difficulty of producing such a pair directly corresponds to the difficulty of forging tokens. Therefore, our definition of token unforgeability captures the security of the scheme against token forgery attacks.

**Comparison with the attack game in [14].** As discussed in Section 4, the security game should account for a broader class of adversarial strategies. To this end, our token-forging game in Figure 3 allows the adversary to output an arbitrary pair of tokens, $(\mathsf{tok}^*_{(\mathcal{A}\to\mathcal{C})}, \mathsf{tok}^*_{(\mathcal{C}\to\mathcal{A})})$. In contrast, the soundness game in [14] only allows the adversary to output a forged token $\mathsf{tok}^*_{(\mathcal{A}\to\mathcal{C})}$ corresponding to a given valid token $\mathsf{tok}_{(\mathcal{C}\to\mathcal{A})}$, which fails to capture the UTRA attack.

Furthermore, by incorporating a verification algorithm $\mathsf{Vfy}$, we revise the adversary's winning condition—from obtaining a valid output from the test algorithm $\mathsf{Test}$ to receiving an acceptance from $\mathsf{Vfy}$. To analyze previous schemes such as DORE and SEDORE within our framework, one can adapt the winning condition in the token-forging game accordingly: replacing acceptance from $\mathsf{Vfy}$ with the ability to obtain a valid output from $\mathsf{Test}$. For further details, please refer to the full version [25].

## 6 TUDORE: Token Unforgeable DORE

In this section, we first construct our novel VDERE scheme $\mathsf{TUDERE}$. And then, following the framework in [17], we complete the $\mathsf{TUDORE}$ scheme by using the $\mathsf{TUDERE}$ scheme. One of the main concerns is how to construct a verification algorithm. To guarantee token unforgeability, we apply digital signature schemes.

### 6.1 Signature Schemes compatible with DERE scheme.

A digital signature is a cryptographic scheme for the authenticity of digital messages. A valid digital signature on a message gives a recipient confidence

that the message came from a sender known to the recipient. In our scenario, a signature scheme can give a token validity due to its unforgeability property. Signature schemes consists of four algorithms $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify})$ as follows:

- $\mathsf{pp}_{sig} \leftarrow \mathsf{Sig.Setup}(\lambda)$: This algorithm takes the security parameter $\lambda$ as input and returns the public parameter $\mathsf{pp}_{sig}$.
- $(\mathsf{vk}_{sig}, \mathsf{sk}_{sig}) \leftarrow \mathsf{Sig.Keygen}(\mathsf{pp}_{sig})$: It takes a public parameter as input and outputs a key tuple of signing key $\mathsf{sk}_{sig}$ and verifying key $\mathsf{vk}_{sig}$.
- $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{pp}_{sig}, \mathsf{sk}_{sig}, m)$: It takes public parameter $\mathsf{pp}_{sig}$, signing key $\mathsf{sk}_{sig}$ and message as input and outputs signature $\sigma$.
- $0 \backslash 1 \leftarrow \mathsf{Sig.Verify}(\mathsf{pp}_{sig}, \mathsf{vk}_{sig}, m, \sigma)$: It takes public parameter $\mathsf{pp}_{sig}$, the verifying key $\mathsf{vk}_{sig}$ , message $m$, and signature $\sigma$ as input. If the signature is valid, it returns 1; otherwise, it returns 0.

DERE scheme in Figure 2 utilize discrete logarithmic (DL)-related keys: $\mathsf{pk} = g_2^a$ and $\mathsf{sk} = (\mathsf{a}, \mathsf{b})$. To utilize the secret key as a signing key, we consider signature schemes with DL-related keys, e.g. Schnorr's signature [29] and BLS signature [7]. Concretely, both signature schemes utilize the signing key $\mathsf{sk}_{sig} = b$ and verification key $\mathsf{vk}_{sig} = g_1^b$. Furthermore, both schemes satisfy existential unforgeability under chosen message attack (EUF-CMA). For more details about both signature schemes, please refer to the full version [25].

### 6.2 Construct TUDORE using Signature Scheme

By combining type-1 DERE scheme in Figure 2 with signature schemes $\mathsf{Sig}$, we construct a token unforgeable DORE scheme: $\mathsf{TUDERE} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Test}, \mathsf{Vfy})$. The main difference between DERE and TUDERE lie in Keygen, Token, and Vfy. In Keygen, an additional token verification key $\mathsf{vk} = g_1^b$ is generated. The Token algorithm outputs a token that includes a signature. Finally, the Vfy algorithm checks the token using the verification algorithm of the signature scheme Sig. We describe TUDERE in Figure 4.

In a similar way in the DERE-to-DORE framework [17], we construct TUDORE scheme using TUDERE scheme in Figure 4; iteratively running Enc and Test algorithms but keeping other algorithms of TUDERE. We describe the scheme TUDORE in Figure 6 in the Appendix.

### 6.3 Security Analysis

In [17], the authors provided a security proof for the correctness and IND-OCPA for the DORE scheme in the GGM. With the proof in [17] and the EUF-CMA signature scheme, we can conclude the following theorem.

**Theorem 1.** *Assume $H$ and $T$ are modeled as a random oracle and* Sig *is an EUF-CMA signature scheme. Then, the* TUDORE *scheme (Figure 6) under the* TUDERE *scheme (Figure 4) satisfies the correctness, data privacy, and token unforgeability under the generic group model.*

- $\mathsf{pp} \leftarrow \mathsf{TUDERE.Setup}(1^\lambda)$: This algorithm takes the security parameter $1^\lambda$ as input and returns the public parameter $\mathsf{pp} = (\langle p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e\rangle, H, T)$. Additionally, it sets $\mathsf{pp}_{sig} := (\langle p, \mathbb{G}_1, g_1\rangle, T)$.
- $(\mathsf{pk}, \boxed{\mathsf{vk}}, \mathsf{sk}) \leftarrow \mathsf{TUDERE.Keygen}(\mathsf{pp})$: This algorithm takes the public parameter $\mathsf{pp}$ as input and randomly chooses $a, b \xleftarrow{\$} \mathbb{Z}_p$. After that, it returns a tuple of keys as $\mathsf{sk} = (a, b)$, $\mathsf{pk} = g_2^a$, and $\mathsf{vk} = g_1^b$. Additionally, it sets signature keys as $\mathsf{sk}_{sig} := b$, and $\mathsf{vk}_{sig} := \mathsf{vk}$.
- $\mathsf{ct} \leftarrow \mathsf{TUDERE.Enc}(\mathsf{pp}, m, \mathsf{sk})$: This algorithm takes a public parameter $\mathsf{pp}$, a message $m \in \{0,1\}^*$, and $\mathsf{sk} = (a, b) \in \mathbb{Z}_p^2$ as input and randomly picks $r \xleftarrow{\$} \mathbb{Z}_p$ and computes $c^0$ and $c^1$ as below:

$$c^0 = \left(g_1^{rb} H(m)\right)^a, \quad c^1 = g_1^r.$$

  Finally, it returns $ct = (c^0, c^1)$.
- $\mathsf{tok}_{(u\to v)} \leftarrow \mathsf{TUDERE.Token}(\mathsf{pp}, \mathsf{pk}_{(u)}, \mathsf{sk}_{(v)})$: This algorithm takes the public key $\mathsf{pk}_{(u)} \in \mathbb{G}_2$ of $u$ and the secret key $\mathsf{sk}_{(v)} = (a_{(v)}, b_{(v)}) \in \mathbb{Z}_p^2$ of $v$ as input and returns the token $\mathsf{tok}_{(u\to v)} = (t^0_{(u\to v)}, t^1_{(u\to v)}, \boxed{\sigma_v})$ as follows:

$$t^0_{(u\to v)} = \mathsf{pk}_{(u)}, \quad t^1_{(u\to v)} = \mathsf{pk}_{(u)}^{a_{(v)} b_{(v)}}$$
$$\sigma_v \leftarrow \mathsf{Sig.Sign}(\mathsf{pp}_{sig}, \mathsf{sk}_{sig,(v)}, (t^0_{(u\to v)}, t^1_{(u\to v)}))$$

- $0\backslash 1 \leftarrow \mathsf{TUDERE.Test}(\mathsf{pp}, \mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$:
  This algorithm takes the two ciphertexts and tokens for $u$ and $v$ and computes

$$d_0 = \frac{e\left(c^0_{(u)}, t^0_{(v\to u)}\right)}{e\left(c^1_{(u)}, t^1_{(v\to u)}\right)}, \quad d_1 = \frac{e\left(c^0_{(v)}, t^0_{(u\to v)}\right)}{e\left(c^1_{(v)}, t^1_{(u\to v)}\right)}$$

  Finally, it compares $d_0$ and $d_1$, and if $d_0 = d_1$, it returns 1 and 0 otherwise.
- $0\backslash 1 \leftarrow \mathsf{TUDERE.Vfy}(\mathsf{pp}, \mathsf{vk}_{(u)}, \mathsf{vk}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: This algorithm takes a public parameter $\mathsf{pp}$, a pair of verification keys $\mathsf{vk}_{(u)}, \mathsf{vk}_{(v)}$ and tokens $\mathsf{tok}_{(u\to v)}, \mathsf{tok}_{(v\to u)}$. It parses $\mathsf{tok}_{(u\to v)}$ and $\mathsf{tok}_{(v\to u)}$ to $((t^0_{(u\to v)}, t^1_{(u\to v)}), \sigma_v)$ and $((t^0_{(v\to u)}, t^1_{(v\to u)}), \sigma_u)$ respectively. Then, run verification algorithms as follows:
  - $res_v \leftarrow \mathsf{Sig.Vfy}(\mathsf{pp}_{sig}, \mathsf{vk}_{(v)}, (t^0_{(u\to v)}, t^1_{(u\to v)}), \sigma_v)$
  - $res_u \leftarrow \mathsf{Sig.Vfy}(\mathsf{pp}_{sig}, \mathsf{vk}_{(u)}, (t^0_{(v\to u)}, t^1_{(v\to u)}), \sigma_u)$

  If $res_v = res_u = 1$, then it outputs 1. Otherwise, outputs 0.

Fig. 4: TUDERE Scheme

**Proof Sketch.** *(Correctness)* The correctness of TUDERE holds in the similar way of [17]. Concretely, for a valid token and ciphertext, the intermediate values $d_0$ and $d_1$ of Test should be equal by the bilinearity of the pairing operation. Additionally, from the correctness of the signature scheme, Vfy should output 1

Table 1: A comparative analysis for element and $n$-bit comparison

| Schemes | Storage cost | | Computational Cost | | | |
|---|---|---|---|---|---|---|
| | Enc | Token | Enc | Test | Token | Vfy |
| DORE [17] | $4n\|\mathbb{G}_1\|$ | $2\|\mathbb{G}_2\|$ | $6nE_{\mathbb{G}_1}$ | $8nP_{\mathbb{G}_T}$ | $E_{\mathbb{G}_2}$ | $\times$ |
| SEDORE [14] | $4n\|\mathbb{G}_1\|$ | $2\|\mathbb{G}_2\|$ | $6nE_{\mathbb{G}_1}$ | $8nP_{\mathbb{G}_T}$ | $3E_{\mathbb{G}_2}$ | $\times$ |
| EDORE [34] | $4n\|\mathbb{Z}_p\| + 2n\|\mathbb{G}_2\|$ | $2\|\mathbb{G}_1\|$ | $2nE_{\mathbb{G}_2}$ | $8nE_{\mathbb{G}_1} + 4nP_{\mathbb{G}_T}$ | $3E_{\mathbb{G}_1}$ | $\times$ |
| TUDORE (Schnorr) | $4n\|\mathbb{G}_1\|$ | $1\|\mathbb{G}_1\| + 2\|\mathbb{G}_2\| + 1\|\mathbb{Z}_p\|$ | $6nE_{\mathbb{G}_1}$ | $8nP_{\mathbb{G}_T}$ | $E_{\mathbb{G}_1} + E_{\mathbb{G}_2}$ | $4E_{\mathbb{G}_1}$ |
| TUDORE (BLS) | $4n\|\mathbb{G}_1\|$ | $1\|\mathbb{G}_1\| + 2\|\mathbb{G}_2\|$ | $6nE_{\mathbb{G}_1}$ | $8nP_{\mathbb{G}_T}$ | $E_{\mathbb{G}_1} + E_{\mathbb{G}_2}$ | $4P_{\mathbb{G}_T}$ |

$|\mathbb{G}_i|$: size of a group element in $\mathbb{G}_i$, $|\mathbb{Z}_p|$: size of a field element in $\mathbb{Z}_p$,
$E_{\mathbb{G}_i}$: group exponentiation on $\mathbb{G}_i$, $P_{\mathbb{G}_T}$: bilinear pairing operation,
$\times$ indicates the Verification algorithm is not available.

so that TUDERE satisfies the correctness. Since TUDORE.Test consists of several correct TUDERE.Test algorithms, it follows that TUDORE satisfies correctness.
*(Data privacy)* Since our underlying encryption algorithm is the same as DERE [17], we can prove IND-OCPA in a similar way in [17]. Since DORE encryption underlying the DERE scheme satisfies IND-OCPA in the generic group model, our TUDORE encryption underlying the TUDERE scheme satisfies IND-OCPA. Therefore, our TUDORE scheme satisfies Data privacy.
*(Token Unforgeability)* Because the Vfy algorithm contains sign verification, TUDERE satisfies the token unforgeability of the EUF-CMA signature scheme.

Concretely, to complete the proof, we construct an EUF-CMA adversary $\mathcal{B}$ using the token forgeability adversary $\mathcal{A}$. To simulate the token query of $\mathcal{A}$, we should restrict $\mathcal{A}$ not get the public key locally. Since the public keys consist of group elements and the adversary does not get the group element itself in GGM, we ensure $\mathcal{A}$ does not get an arbitrary token $\mathsf{tok}_{(i\to\mathcal{C})}$ without the corresponding key query for $\mathsf{pk}_{(i)}$. Therefore, we claim that TUDERE satisfies token unforgeability if the underlying signature scheme satisfies EUF-CMA. For the complete proof, please refer to the full version [25]. □

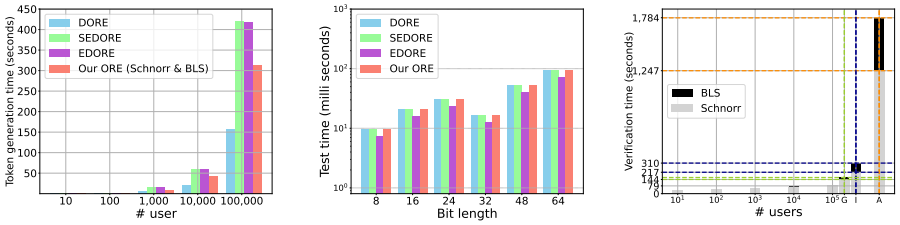
## 7 Performance

### 7.1 Theoretical Performance

In Table 1, we present a comparative analysis of storage and computational costs, where the left side (group elements) shows storage costs and the right side (group/pairings) shows computational costs. Since the Enc and Test algorithms in SEDORE and TUDORE are identical to those in DORE, their performance remains the same. EDORE improves efficiency by reducing the number of group elements and pairing operations. TUDORE increases the token size due to the use of Schnorr or BLS signatures, reflecting a trade-off between security and efficiency. For the Token algorithm, DORE requires only a single group exponentiation, while SEDORE and EDORE require three, and TUDORE requires two. Consequently, TUDORE is approximately twice as slow as DORE, but 1.5× faster than both SEDORE and EDORE.

Notably, only TUDORE includes a verification algorithm that ensures token unforgeability. Verifying Schnorr and BLS signatures requires four exponentiations and four pairings, respectively. Based on their priorities, users can choose the scheme that best balances token size and computational efficiency.

### 7.2 Implementation and Performance.




(a) Token generation time (b) Test time (c) Verification time

In Figure 5b, 8–24 bit settings use the age dataset [32], while 32–64 bit settings use the stock dataset [23]. In Figure 5c's x-axis, "G" indicates Google, "I" indicates IBM, and "A" indicates Amazon, respectively.

Fig. 5: The performance graphs for DORE, SEDORE, and our TUDORE.

In this section, we present the implementation results of token-based DORE schemes: DORE, SEDORE, EDORE, and our proposed TUDORE.

**Environment.** To ensure realistic evaluation, we conducted experiments in two environments: a server (Linux desktop with a 5.20 GHz Intel i9-12900K CPU and 64GB RAM for testing and verification) and a user device (Linux laptop with a 1.4 GHz AMD Ryzen 7 4700U CPU and 16GB RAM for token generation). We implemented TUDORE and the baseline schemes for comparison. Our implementation uses OpenSSL for hashing, the PBC library (in C) with the MNT224 curve for bilinear maps [19], and the GMP library for large integer arithmetic. For the verification algorithm, we adopt both Schnorr and BLS signatures [29,7], allowing users to select based on their computational environment.

**Dataset.** We utilize the dataset from the United Nations [32], which provides population estimates across five-year age groups. However, due to its limited range, we also incorporate real stock volume data from FAANG companies [23]. We use 10,000 samples for our experiments, comprising 5,000 from each dataset.

**Token generation time.** We compare the token generation time for issuing tokens to different numbers of users (10, 100, 1,000, 10,000, and 100,000). As shown in Figure 5a, SEDORE and EDORE take approximately 3 times longer than DORE, while TUDORE is about 1.5 times faster than SEDORE and EDORE. Therefore, our method improves both security and token generation efficiency.

**Test time.** As shown in Figure 5b and consistent with Table 1, in each dataset, the computational time increases linearly with the bit size due to paddings. The test times for DORE, SEDORE, and TUDORE are identical, while EDORE is the fastest. Moreover, it can be observed that the computational time at 32 bits

is faster than that at 16 and 24 bits. This is because the order in the 32-bit dataset is determined by the bits closer to the most significant bit (MSB).

**Verification time.** We evaluate the verification algorithm using Schnorr and BLS signatures [29,7] across varying user counts (10, 100, 1,000, 10,000, and 100,000). To assess real-world applicability, we simulate cloud service provider (CSP) environments based on employee numbers at Google (182,502) [2], Amazon (1,608,000) [1], and IBM (282,200) [3]. Figure 5c shows that verification times with 100,000 users are 79s / 84s (Schnorr / BLS). In CSP-scale scenarios, the times are 144s / 164s for Google, 217s / 310s for IBM, and 1,247s / 1,783s (20.8min / 29.7min) for Amazon. Despite Amazon's high total time, the amortized cost remains low at 0.77ms / 1.1ms, demonstrating practical efficiency.

**Range query and test algorithm.** From the token-based DORE query via the test algorithm, one can obtain the order information, whether a value is less than or greater than the encrypted data. Leveraging this property, a range query can be implemented by performing the DORE test twice for a single encrypted value, separately comparing the encrypted value against the lower and upper query. Although we do not directly evaluate range queries, their cost can be inferred from our experiments with the DORE scheme. Since each range query involves two comparisons per encrypted data point, the cost per encrypted value is approximately twice that of a single test execution.

## 8 Conclusion

We demonstrate the vulnerability of DORE, SEDORE, and EDORE within the identical threat model of token forgeability, as suggested by [14]. Although SEDORE and EDORE aim to achieve the token unforgeability property, we identify that both schemes are still vulnerable. To illustrate this vulnerability, we propose an attack strategy called the universal token reusability attack under the same threat model. From the attack, SEDORE and EDORE remain vulnerable to token forgery. To address these security concerns, we propose the VDORE with an enhanced security notion. We introduce TUDORE, which leverages the Schnorr and BLS signature schemes to prevent universal token reusability attacks. We provide a formalized definition and proof to clarify the unclear definitions and proofs of token unforgeability in previous work. Moreover, our scheme offers a faster token generation algorithm than SEDORE and EDORE.

Since the ORE scheme itself aims to compare numerical values, the input data is limited to numerical datasets. However, to apply it to various applications with other types, one can consider an appropriate encoding/decoding process to convert the data type to a numerical type. Additionally, considering the significance of range queries, exploring the integration of TUDORE-based range queries into real-world applications remains an important direction. We leave these as future work.

**Acknowledgments.** This research was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP), grant funded by the Korea Government (MSIT) (No.2022-0-00411, 50% & RS-2021-II210727, 40%), in part by Culture, Sports and Tourism R&D Program

through the Korea Creative Content Agency grant funded by the Ministry of Culture, Sports and Tourism (RS-2024-00332210, 10%), the NSF (2335798), and the Tennessee Department of Economic and Community Development.

# A Deffered Figures

## A.1 TUDORE from the TUDERE.

Following the DERE-to-DORE framework, we construct TUDORE from TUDERE as shown in Figure 4. The conversion process is described in the following figure.

- $\mathsf{pp} \leftarrow \mathsf{TUDORE.Setup}(1^\lambda)$: With the inputs, run TUDERE.Setup algorithm and output pp.
- $(\mathsf{pk}, \mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{TUDORE.Keygen}(\mathsf{pp})$: With the inputs, run TUDERE.Keygen(pp) algorithm and output $(\mathsf{pk}, \mathsf{vk}, \mathsf{vk})$.
- $\mathsf{tok}_{(u\to v)} \leftarrow \mathsf{TUDORE.Token}(\mathsf{pp}, \mathsf{pk}_{(u)}, \mathsf{sk}_{(v)})$: With the inputs, run TUDERE.Token algorithm and output $\mathsf{tok}_{(u\to v)}$.
- $0\backslash 1 \leftarrow \mathsf{TUDERE.Vfy}(\mathsf{pp}, \mathsf{vk}_{(u)}, \mathsf{vk}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: With the inputs, run TUDERE.Vfy algorithm and output decision bit $0\backslash 1$.
- $\mathsf{ct} \leftarrow \mathsf{TUDORE.Enc}(\mathsf{pp}, m, \mathsf{sk})$: It takes a message $m \in \{0,1\}^*$ and sk as input. It encrypts message bit encodings $\epsilon(m_i, a)$, which is defined as $\epsilon(m_i, a) = (i, m_1 m_2 \ldots m_i || 0^{n-i}, a)$ for $a \in \{0,1,2\}$, as follows:
  If $m_i = 0$, it computes ciphertexts $\mathsf{ct}[i] = (\mathsf{ct}[i,0], \mathsf{ct}[i,1])$ as follows:

  $$\mathsf{ct}[i,0] = \mathsf{TUDERE.Enc}(\mathsf{pp}, \epsilon(m_i, 0), \mathsf{sk}),\ \mathsf{ct}[i,1] = \mathsf{TUDERE.Enc}(\mathsf{pp}, \epsilon(m_i, 1), \mathsf{sk}).$$

  Else if $m_i = 1$, it computes ciphertexts $\mathsf{ct}[i] = (\mathsf{ct}[i,0], \mathsf{ct}[i,1])$ as follows:

  $$\mathsf{ct}[i,0] = \mathsf{TUDERE.Enc}(\mathsf{pp}, \epsilon(m_i, 1), \mathsf{sk}),\ \mathsf{ct}[i,1] = \mathsf{TUDERE.Enc}(\mathsf{pp}, \epsilon(m_i, 2), \mathsf{sk}).$$

  Finally, this algorithm returns $\mathsf{ct} = (\mathsf{ct}[1], \ldots \mathsf{ct}[n])$.
- $res \leftarrow \mathsf{TUDORE.Test}(\mathsf{pp}, \mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: It takes a pair of ciphertexts $\mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}$ and tokens $\mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)}$ as input. Test algorithm runs TUDERE.Test iteratively. For $i = 1$ to $n$, the algorithm follows it:
  1. **If** $i = n + 1$, then return 0
  2. **Else** Compute $res_u^i$ and $res_v^i$ as follows:
     - $res_u^i \leftarrow \mathsf{TUDERE.Test}(\mathsf{ct}_{(u)}[i,0], \mathsf{ct}_{(v)}[i,1])$
     - $res_v^i \leftarrow \mathsf{TUDERE.Test}(\mathsf{ct}_{(u)}[i,1], \mathsf{ct}_{(v)}[i,0])$

     (a) **If** $res_u^i = 1$, then returns 1
     (b) **Else if** $res_v^i = 1$, then return $-1$
     (c) **Else** $i \leftarrow i + 1$

Fig. 6: TUDORE scheme from TUDERE

# B Universal Token Reusability Attack on efficient DORE

This section shows how to adapt our UTRA method to EDORE [34]. Before describing our attack, we show the EDERE suggested by [34].

## B.1 Efficient DERE

EDERE scheme consists of five algorithms: (Setup, Keygen, Enc, Token, Test) as follows:

- $\mathsf{pp} \leftarrow \mathsf{EDERE.Setup}(1^\lambda)$: It takes the security parameter $1^\lambda$ as input and returns the public parameter $\mathsf{pp} = (\langle p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e\rangle, H, F)$.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{EDERE.Keygen}(\mathsf{pp})$ : This algorithm takes a public parameter ($\mathsf{pp}$) as input and returns a pair of public key and secret key ($\mathsf{pk}, \mathsf{sk}$). From this, it uniformly chooses $a, b, \xi \xleftarrow{\$} \mathbb{Z}_p^*$ and generates $\mathsf{sk}$ and the corresponding $\mathsf{pk}$ as below:
$$\mathsf{pk} = g_2^a, \quad \mathsf{sk} = (a, b, \xi)$$
We denote a key pair of user $u$ as $(\mathsf{pk}_{(u)}, \mathsf{sk}_{(u)}) = (g_2^{a_{(u)}}, (a_{(u)}, b_{(u)}, \xi_{(u)}))$.
- $\mathsf{ct} \leftarrow \mathsf{EDERE.Enc}(\mathsf{pp}, m, \mathsf{sk})$: This algorithm receives a message $m \in \{0,1\}^*$ and $\mathsf{sk}$ as input and returns a ciphertext $\mathsf{ct}$. This algorithm randomly picks $r, \eta \xleftarrow{\$} \mathbb{Z}_p$ and computes $c^1$, $c^2$, and $c^3$ as below:
$$c^1 = r - \xi\eta, \quad c^2 = \eta, \quad c^3 = H(m)^{(br)^{-1}}$$
After that, it returns $\mathsf{ct} = (c^0, c^1, c^2)$. For user $u$, we rewrite $ct$ as $\mathsf{ct}_{(u)} = (c^0_{(u)}, c^1_{(u)}, c^2_{(u)})$.
- $\mathsf{tok}_{(v\to u)} \leftarrow \mathsf{EDERE.Token}(\mathsf{pp}, \mathsf{pk}_{(v)}, \mathsf{sk}_{(u)})$: This algorithm takes the public key $\mathsf{pk}_{(v)} = g_2^{a_{(v)}}$ of user $v$ and the secret key $\mathsf{sk}_{(u)} = (a_{(u)}, b_{(u)}, \xi_{(u)})$ of user $u$ and returns an authorization token $\mathsf{tok}_{(v\to u)}$ consisting of $t^1_{(v\to u)}$ and $t^2_{(v\to u)}$.
$$t^1_{(v\to u)} = F(\mathsf{pk}_{(v)}^{a_{(u)}})^{b_{(u)}}, \quad t^2_{(v\to u)} = F(\mathsf{pk}_{(v)}^{a_{(u)}})^{b_{(u)}\xi_{(u)}}$$
Finally, it returns $\mathsf{tok}_{(v\to u)} := (t^1_{(v\to u)}, t^2_{(v\to u)})$.
- $0\backslash 1 \leftarrow \mathsf{EDERE.Test}(\mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: This algorithm receives the ciphertexts from user $v$ and $u$, $\mathsf{ct}_{(v)}$ and $\mathsf{ct}_{(u)}$, and the tokens $\mathsf{tok}_{(v\to u)}$ and $\mathsf{tok}_{(u\to v)}$ as input. After that, it computes
$$d_0 = e\left(\prod_{k=1}^{2}(t^k_{(v\to u)})^{c^k_{(u)}}, c^3_{(u)}\right), \quad d_1 = e\left(\prod_{k=1}^{2}(t^k_{(u\to v)})^{c^k_{(v)}}, c^3_{(v)}\right).$$
Finally, it compares $d_0$ and $d_1$ and returns 1 if $d_0 = d_1$ and 0 otherwise.

### B.2 UTRA for EDORE.

Xu et al. also follow the DERE-to-DORE framework to construct $\mathsf{EDORE}$ [34]. Due to the equivalence of the $\mathsf{Vfy}$ algorithm between $\mathsf{EDORE}$ and $\mathsf{EDERE}$, we consider the universal token reusability attack on $\mathsf{EDERE}$ scheme. The scenario is as follows:

**Step 1:** The user $\mathcal{V}$ creates authorization token $\mathsf{tok}_{(\mathcal{M}\to\mathcal{V})}$ by using $\mathsf{EDERE.Token}$ algorithm and sends it to user $\mathcal{M}$ as below:

$$\mathsf{tok}_{(\mathcal{M}\to\mathcal{V})} = \left(F(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}})^{b_{(\mathcal{V})}}, \quad F(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}})^{b_{(\mathcal{V})}\xi_{(\mathcal{V})}}\right)$$

**Step 2:** After $\mathcal{M}$ receives it, $\mathcal{M}$ randomly picks $r \xleftarrow{\$} \mathbb{Z}_p$ and sets a group element $h_2 = F(\mathsf{pk}_{(\mathcal{V})}^{a_{(\mathcal{M})}})^r$. And then $\mathcal{M}$ computes a universal forged token $\mathsf{uft}_{(\mathcal{V}),h_2}$ as following:

$$\begin{aligned}\mathsf{uft}_{(\mathcal{V}),h_2} &= \mathsf{tok}_{(\mathcal{M}\to\mathcal{V})}^{r} \\ &= \left(\left(F(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}})^r\right)^{b_{(\mathcal{V})}}, \left(F(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}})^r\right)^{b_{(\mathcal{V})}\xi_{(\mathcal{V})}}\right)\end{aligned}$$

After then, $\mathcal{M}$ sends $\mathsf{uft}_{(\mathcal{V}),h_2}$ and $h_2$ to $\mathcal{A}$. Note that $\mathcal{M}$ can compute $\mathsf{uft}_{(\mathcal{V}),h_2}$ by symmetric property $\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}} = g_2^{a_{\mathcal{V}}a_{\mathcal{M}}} = \mathsf{pk}_{(\mathcal{V})}^{a_{(\mathcal{M})}}$. Since $h_2$ is randomized by $r$, $h_2$ looks like uniform random in the view of $\mathcal{A}$. Furthermore, it is intractable for the $\mathcal{A}$ to find a secret key $(a_{(\mathcal{M})}, b_{(\mathcal{M})})$ of $\mathcal{M}$ from $h_2$ and $\mathsf{uft}_{(\mathcal{V}),h_2}$ due to the discrete logarithm assumption. For this reason, $\mathcal{M}$ may help adversary $\mathcal{A}$ without concern about leaking $\mathcal{M}$'s secret.

**Step 3:** When $\mathcal{A}$ receives $\mathsf{uft}_{(\mathcal{V}),h_2}$ and $h_2$, she samples her secret key $\mathsf{sk}_{(\mathcal{A})} = (a_{(\mathcal{A})}, b_{(\mathcal{A})}, \xi_{(\mathcal{A})}) \xleftarrow{\$} \mathbb{Z}_p^3$ and then computes the counterpart forged token $\mathsf{uft}_{(\mathcal{A}),h_2}$ as follows:

$$\mathsf{uft}_{(\mathcal{A}),h_2} = (h_2^{b_{(\mathcal{A})}}, h_2^{b_{(\mathcal{A})}\xi_{(\mathcal{A})}})$$

For the query, $\mathcal{A}$ generates $\mathsf{ct}_{(\mathcal{A})} \leftarrow \mathsf{EDERE.Enc}(m, \mathsf{sk}_{(\mathcal{A})})$ using her secret key $(a_{(\mathcal{A})}, b_{(\mathcal{A})}, \xi_{(\mathcal{A})})$ and then use a pair of forged tokens $\mathsf{uft}_{(\mathcal{A}),h_2}$ and $\mathsf{uft}_{(\mathcal{V}),h_2}$.

For a given message $m$, let us denote the victim's ciphertext as $\mathsf{ct}_{(\mathcal{V})} = (r_{(\mathcal{V})} - \xi_{(\mathcal{V})}\eta_{(\mathcal{V})}, \eta_{(\mathcal{V})}, H(m)^{(b_{(\mathcal{V})}r_{(\mathcal{V})})^{-1}})$. Then we can get $\mathsf{DERE.Test}(\mathsf{ct}_{(\mathcal{V})}, \mathsf{ct}_{(\mathcal{A})}, \mathsf{uft}_{(\mathcal{V}),h_2}, \mathsf{uft}_{(\mathcal{A}),h_2}) = 1$ by the following equations.

$$\begin{aligned}d_0 &= e\left(\prod_{k=1}^{2}(\mathsf{uft}_{(\mathcal{V}),h_2}^{k})^{c_{(\mathcal{V})}^{k}}, c_{(\mathcal{V})}^{3}\right) \\ &= e(h_2^{b_{(\mathcal{V})}(r_{(\mathcal{V})}-\xi_{(\mathcal{V})}\eta_{(\mathcal{V})})} \cdot h_2^{b_{(\mathcal{V})}\xi_{(\mathcal{V})}\eta_{(\mathcal{V})}}, H(m)^{(b_{(\mathcal{V})}r_{(\mathcal{V})})^{-1}}) \\ &= e(h_2^{b_{(\mathcal{V})}r_{(\mathcal{V})}}, H(m)^{(b_{(\mathcal{V})}r_{(\mathcal{V})})^{-1}}) = e(h_2, H(m)).\end{aligned}$$

$$
\begin{aligned}
d_1 &= e\left(\prod_{k=1}^{2}(\mathsf{uft}^{k}_{(\mathcal{A}),h_2})^{c^{k}_{(\mathcal{A})}}, c^{3}_{(\mathcal{A})}\right) \\
&= e(h_2^{b_{(\mathcal{A})}(r_{(\mathcal{A})}-\xi_{(\mathcal{A})}\eta_{(\mathcal{A})})} \cdot h_2^{b_{(\mathcal{A})}\xi_{(\mathcal{A})}\eta_{(\mathcal{A})}}, H(m)^{(b_{(\mathcal{A})}r_{(\mathcal{A})})^{-1}}) \\
&= e(h_2^{b_{(\mathcal{A})}r_{(\mathcal{A})}}, H(m)^{(b_{(\mathcal{A})}r_{(\mathcal{A})})^{-1}}) = e(h_2, H(m)).
\end{aligned}
$$

# C Refinement of Security Definition and analysis of UTRA

1. **Setting Phase**: $\mathcal{C}$ runs the setup algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and the key generation algorithm $(\mathsf{sk}_{(\mathcal{C})}, \mathsf{pk}_{(\mathcal{C})}) \leftarrow \mathsf{Keygen}(\mathsf{pp})$. It then sends $(\mathsf{pp}, \mathsf{pk}_{(\mathcal{C})})$ to $\mathcal{A}$ and keeps $\mathsf{sk}_{(\mathcal{C})}$ local storage.
2. **Query Phase**: $\mathcal{A}$ can query to $\mathcal{C}$:
   (a) **Key Query**: $\mathcal{A}$ sends a query with index $i$. Upon receiving the query, $\mathcal{C}$ checks whether $(i, \mathsf{pk}_{(i)}) \in S_{\mathsf{key}}$. If so, it returns $(i, \mathsf{pk}_{(i)})$ to $\mathcal{A}$. Otherwise, it generates a new key triple by running $(\mathsf{pk}_{(i)}, \mathsf{vk}_{(i)}, \mathsf{sk}_{(i)}) \leftarrow \mathsf{Keygen}(\mathsf{pp})$, returns $(\mathsf{pk}_{(i)}, \mathsf{vk}_{(i)})$ to $\mathcal{A}$, and adds the tuple $(i, \mathsf{pk}_{(i)}, \mathsf{vk}_{(i)})$ to the key query set $S_{\mathsf{key}}$.
   (b) **Token Query**: If $\mathcal{A}$ sends a query with a received public key $\mathsf{pk}_{(i)} \in S_{\mathsf{key}}$, $\mathcal{C}$ generates an authorized token $\mathsf{tok}_{(i\to\mathcal{C})} \leftarrow \mathsf{Token}(\mathsf{pp}, \mathsf{pk}_{(i)}, \mathsf{sk}_{(\mathcal{C})})$ and then sends $\mathsf{tok}_{(i\to\mathcal{C})}$ to $\mathcal{A}$ and adds $\mathsf{tok}_{(i\to\mathcal{C})}$ to token query set $S_{\mathsf{tok}}$.
   The number of queries is at most polynomially large at $\lambda$.
3. **Challenge Phase**:
   (a) $\mathcal{C}$ sends a ciphertext $\mathsf{ct}_{(\mathcal{C})} \leftarrow \mathsf{Enc}(\mathsf{pp}, m_{\mathcal{C}}, \mathsf{sk}_{(\mathcal{C})})$ for a message $m_{\mathcal{C}}$ and a challenge resulting bit $res \xleftarrow{\$} \{-1, 0, 1\}$.
   (b) After receiving $\mathsf{ct}_{(\mathcal{C})}$, $\mathcal{A}$ samples its own keys $(\mathsf{pk}_{(\mathcal{A})}, \mathsf{sk}_{(\mathcal{A})}) \leftarrow \mathsf{Keygen}(\mathsf{pp})$ locally. $\mathcal{A}$ picks a message $m_{\mathcal{A}}$ and encrypts it $\mathsf{ct}_{(\mathcal{A})} \leftarrow \mathsf{Enc}(\mathsf{pp}, m_{\mathcal{A}}, \mathsf{sk}_{(\mathcal{A})})$.
   (c) $\boxed{\mathcal{A} \text{ generates a token } \mathsf{tok}_{(\mathcal{C}\to\mathcal{A})} \leftarrow \mathsf{Token}(\mathsf{pp}, \mathsf{pk}_{(\mathcal{C})}, \mathsf{sk}_{(\mathcal{A})}).}$
   (d) $\mathcal{A}$ forges a token $\mathsf{tok}^*_{(\mathcal{A}\to\mathcal{C})}$ and finally outputs $\mathsf{ct}_{(\mathcal{A})}, \mathsf{tok}^*_{(\mathcal{A}\to\mathcal{C})}, \mathsf{tok}_{(\mathcal{C}\to\mathcal{A})}$
   The $\mathcal{A}$ wins if $\mathsf{Test}(\mathsf{pp}, \mathsf{ct}_{(\mathcal{C})}, \mathsf{ct}_{(\mathcal{A})}, \mathsf{tok}^*_{(\mathcal{A}\to\mathcal{C})}, \mathsf{tok}_{(\mathcal{C}\to\mathcal{A})}) = res$.

Fig. 7: Soundness Game in [14]

In this section, we review the soundness game defined in [14] and present a refined version. The original soundness game is illustrated in Figure 7.

In this game, the adversary is only allowed to forge a token $\mathsf{tok}^*_{(\mathcal{A}\to\mathcal{C})}$, which serves as a counterpart to the valid token $\mathsf{tok}_{(\mathcal{C}\to\mathcal{A})}$ generated using the adversary's own secret key. Hahn et al. provided a soundness proof for SEDORE based on this game, relying on the random oracle model and the discrete logarithm assumption to argue that forging such a counterpart token is infeasible.

However, this formulation does not capture our Universal Token Reusability Attack (UTRA), which does not require access to a valid token $\mathsf{tok}_{(\mathcal{C}\to\mathcal{A})}$. To

1. **Setting Phase**: $\mathcal{C}$ runs the setup algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and the key generation algorithm $(\mathsf{sk}_{(\mathcal{C})}, \mathsf{pk}_{(\mathcal{C})}) \leftarrow \mathsf{Keygen}(\mathsf{pp})$. It then sends $(\mathsf{pp}, \mathsf{pk}_{(\mathcal{C})})$ to $\mathcal{A}$ and keeps $\mathsf{sk}_{(\mathcal{C})}$ local storage.
2. **Query Phase**: $\mathcal{A}$ can query to $\mathcal{C}$:
   (a) **Key Query**: $\mathcal{A}$ sends a query with index $i$. Upon receiving the query, $\mathcal{C}$ checks whether $(i, \mathsf{pk}_{(i)}) \in S_{\mathsf{key}}$. If so, it returns $(i, \mathsf{pk}_{(i)})$ to $\mathcal{A}$. Otherwise, it generates a new key triple by running $(\mathsf{pk}_{(i)}, \mathsf{vk}_{(i)}, \mathsf{sk}_{(i)}) \leftarrow \mathsf{Keygen}(\mathsf{pp})$, returns $(\mathsf{pk}_{(i)}, \mathsf{vk}_{(i)})$ to $\mathcal{A}$, and adds the tuple $(i, \mathsf{pk}_{(i)}, \mathsf{vk}_{(i)})$ to the key query set $S_{\mathsf{key}}$.
   (b) **Token Query**: If $\mathcal{A}$ sends a query with a received public key $\mathsf{pk}_{(i)} \in S_{\mathsf{key}}$, $\mathcal{C}$ generates an authorized token $\mathsf{tok}_{(i\to\mathcal{C})} \leftarrow \mathsf{Token}(\mathsf{pp}, \mathsf{pk}_{(i)}, \mathsf{sk}_{(\mathcal{C})})$ and then sends $\mathsf{tok}_{(i\to\mathcal{C})}$ to $\mathcal{A}$ and adds $\mathsf{tok}_{(i\to\mathcal{C})}$ to token query set $S_{\mathsf{tok}}$.
   (c) **One-way function Query**: If $\mathcal{A}$ sends a query with index $i$ and one-way function $f$, $\mathcal{C}$ output $f(\mathsf{sk}_{(i)})$.
   The number of queries is at most polynomially large at $\lambda$.
3. **Challenge Phase**:
   (a) $\mathcal{C}$ sends a ciphertext $\mathsf{ct}_{(\mathcal{C})} \leftarrow \mathsf{Enc}(\mathsf{pp}, m_{\mathcal{C}}, \mathsf{sk}_{(\mathcal{C})})$ for a message $m_{\mathcal{C}}$ and a challenge resulting bit $res \xleftarrow{\$} \{-1, 1\}$.
   (b) After receiving $\mathsf{ct}_{(\mathcal{C})}$, $\mathcal{A}$ samples its own keys $(\mathsf{pk}_{(\mathcal{A})}, \mathsf{sk}_{(\mathcal{A})}) \leftarrow \mathsf{Keygen}(\mathsf{pp})$ locally. $\mathcal{A}$ picks a message $m_{\mathcal{A}}$ and encrypts it $\mathsf{ct}_{(\mathcal{A})} \leftarrow \mathsf{Enc}(\mathsf{pp}, m_{\mathcal{A}}, \mathsf{sk}_{(\mathcal{A})})$.
   (c) $\mathcal{A}$ forges a pair of tokens $(\mathsf{tok}^*_{(\mathcal{A}\to\mathcal{C})}, \mathsf{tok}^*_{(\mathcal{C}\to\mathcal{A})})$ and finally outputs $\mathsf{ct}_{(\mathcal{A})}, \mathsf{tok}^*_{(\mathcal{A}\to\mathcal{C})}, \mathsf{tok}^*_{(\mathcal{C}\to\mathcal{A})}$.
   The $\mathcal{A}$ wins if $\mathsf{Test}(\mathsf{pp}, \mathsf{ct}_{(\mathcal{C})}, \mathsf{ct}_{(\mathcal{A})}, \mathsf{tok}^*_{(\mathcal{A}\to\mathcal{C})}, \mathsf{tok}^*_{(\mathcal{C}\to\mathcal{A})}) = res$.

Fig. 8: Refinement of Soundness Game

address this gap, we propose a refined soundness game that accommodates a broader class of attacks, including UTRA.

To refine the game, we do not require the adversary to generate a valid token $\mathsf{tok}_{(\mathcal{C}\to\mathcal{A})}$ via the $\mathsf{Token}$ algorithm; instead, it may produce a forged token $\mathsf{tok}^*_{(\mathcal{C}\to\mathcal{A})}$ through an arbitrary method. In addition, we allow $\mathcal{A}$ to make one-way function queries to model more general collusion scenarios. Specifically, in the UTRA attack described in Section 4, we present a collusion case where an authorized user $\mathcal{M}$ sends $h_2 = F(\mathsf{pk}_{(\mathcal{V})}^{a_{(\mathcal{M})}})^r$ to the adversary $\mathcal{A}$. As noted, $\mathcal{M}$ can participate in the collusion without the risk of leaking their secret key $\mathsf{sk}_{(\mathcal{M})}$, due to the one-wayness of group exponentiation under the discrete logarithm assumption.

Furthermore, we restrict the challenger's response bit $res$ to be sampled from $\{-1, 1\}$, which indicate whether the left ciphertext is greater or the right ciphertext is greater, respectively. We exclude the case $res = 0$, which corresponds either to equality of the inputs or to a failure caused by invalid tokens. The equality case is semantically meaningful and should not be conflated with invalid inputs. However, if $res = 0$ is allowed as a winning case, an adversary could trivially win by submitting arbitrary invalid tokens. Therefore, we eliminate 0 to prevent such trivial wins.

We denote the $\mathcal{A}$'s advantage to the soundness game of DORE scheme(or DERE scheme) $\mathsf{Adv}^{\mathsf{Sound}}[\mathcal{A}, \mathsf{DORE}]$(or $\mathsf{Adv}^{\mathsf{Sound}}[\mathcal{A}, \mathsf{DERE}]$), which is a probability that $\mathcal{A}$ wins the token forging game under the $\mathsf{DORE}$(or $\mathsf{DERE}$) scheme, respectively.

**Definition 2 (Refined Soundness).** *DORE scheme described in Section 3.2 is sound if the probability inequality holds:*

$$|\mathsf{Adv}^{\mathsf{Sound}}[\mathcal{A}, \mathsf{DORE}] - 1/2| \leq \mathsf{negl}(\lambda)$$

To give a formality of security analysis, we prove the unsoundness of SE-DORE following the refined soundness game in Figure 8.

**Theorem 2.** *SEDORE scheme in [14] is unsound under the definition of Definition 2*

*Proof.* Since SEDORE scheme follows DERE-to-DORE framework, the unsoundness of SEDERE implies these of SEDORE. In terms of SEDERE forging, the adversary can universal forged token $\mathsf{uft}_{(\mathcal{C}),h_2}$ as follows:

1. $\mathcal{A}$ makes a key and token query and obtains $(\mathsf{pk}_{(i)}, \mathsf{tok}_{(i\to\mathcal{C})})$.
2. $\mathcal{A}$ samples $r \leftarrow \mathbb{Z}$ and makes a one-way function query on the value $F(\mathsf{pk}^{a}_{(\mathcal{C})})^r$, where $(a, b) = \mathsf{sk}$, with input index $i$. From this query, $\mathcal{A}$ obtains $h_2 = F(\mathsf{pk}^{a_{(i)}}_{(\mathcal{C})})^r$, where $a_{(i)}$ is a component of $\mathsf{sk}_{(i)}$.
3. $\mathcal{A}$ sets $\mathsf{uft}_{(\mathcal{C}),h_2} = \mathsf{tok}^{r}_{(i\to\mathcal{C})}$.
4. $\mathcal{A}$ samples $\mathsf{sk}_{(\mathcal{A})} = (a_{(\mathcal{A})}, b_{(\mathcal{A})}) \xleftarrow{\$} \mathbb{Z}_p^2$. And it computes the counterpart forged token $\mathsf{uft}_{(\mathcal{A}),h_2} = (h_2^{a^{-1}_{(\mathcal{A})}}, h_2^{b_{(\mathcal{A})}})$.

As described in Section 4, the adversary $\mathcal{A}$ can obtain a valid result from the $\mathsf{DERE.Test}$ algorithm using the pair of forged tokens $\mathsf{uft}_{(\mathcal{A}),h_2}, \mathsf{uft}_{(\mathcal{C}),h_2}$. With this result, $\mathcal{A}$ can extract the plaintext $m_{\mathcal{C}} = m_1 \cdots m_n$ from the ciphertext $\mathsf{ct}_{(\mathcal{C})}$ by executing bitwise SEDERE. Note that $\mathsf{ct}_{(\mathcal{C})}$ consists of bitwise encryptions $\mathsf{ct}_1 \cdots \mathsf{ct}_n$, which can be individually recovered using the $\mathsf{DERE.Test}$ algorithm with the forged tokens. After extracting the plaintext, depending on the challenge result value $res$, $\mathcal{A}$ can adaptively generate and submit its own ciphertext $\mathsf{ct}_{(\mathcal{A})}$ to win the game. □

From Theorem 2, we argue that the existing soundness definition for the DORE scheme must be revised to the one in Definition 2 to adequately address the UTRA attack. However, before constructing a secure DORE scheme that satisfies this refined soundness notion, we first consider a stronger property: token unforgeability.

**Necessity of Verifying Token Validity.** The $\mathsf{Test}$ algorithm does not directly indicate whether a token is forged. While soundness definition ensures that a forger cannot extract order information without valid tokens, the tester itself is unable to detect the use of invalid tokens. Therefore, incorporating a verification algorithm not only strengthens the security notion by enforcing unforgeability but also empowers the tester to detect forged tokens. For this reason, we propose the VDORE scheme in Section 5, which extends the DORE framework by integrating a verification algorithm with token unforgeability property.

## D Security Definitions

**Definition 3 (DL Assumption).** *Let $\mathcal{G}$ be a group generation algorithm that outputs cyclic group $\mathbb{G}$ with prime order $p \in \mathbb{Z}_p$ and generator $g \in \mathbb{G}$. We say that $\mathbb{G}$ satisfies the discrete logarithm (DL) assumption if, for any PPT adversary $\mathcal{A}$, the following inequality holds:*

$$\Pr\left[ g^x = h \;\middle|\; \begin{array}{l} (p, g, \mathbb{G}) \leftarrow \mathcal{G}(1^\lambda), h \xleftarrow{\$} \mathbb{G}; \\ x \leftarrow \mathcal{A}(p, g, h, \mathbb{G}) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

**Definition 4 (EUF-CMA).** *Let $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{Vfy})$ be a signature scheme. We say that $\mathsf{Sig}$ is Existential Unforgeability under Chosen Message Attack (EUF-CMA) if for any PPT $\mathcal{A}$, the $\mathcal{A}$'s advantage $\mathsf{Adv}^{\mathsf{EUF-CMA}}[\mathcal{A}, \mathsf{Sig}]$ to the game in Figure 9 is $\mathsf{negl}(\lambda)$.*

**Signature Forge game**
$\mathcal{A}(1^\lambda) \rightarrow (m^*, \sigma^*)$

1. **Setting Phase**: $\mathcal{C}_{sig}$ runs setup algorithm $\mathsf{pp}_{sig} \leftarrow \mathsf{Setup}(1^\lambda)$ and key generation algorithm $(\mathsf{vk}_{sig}, \mathsf{sk}_{sig}) \leftarrow \mathsf{Keygen}(\mathsf{pp}_{sig})$. And then sends $(\mathsf{pp}_{sig}, \mathsf{vk}_{sig})$ to $\mathcal{A}$.
2. **Query Phase**: $\mathcal{A}$ sends a query to $\mathcal{C}$ with chosen message $m$. Then, $\mathcal{C}$ generates signature $\sigma_m \leftarrow \mathsf{Sign}(\mathsf{pp}_{sig}, \mathsf{sk}_{sig}, m)$ and return it to $\mathcal{A}$. Additionally, $\mathcal{C}$ adds $m$ in queried message set $\mathcal{M}$. The number of queries is at most polynomially large at $\lambda$.
3. **Challenge Phase**: $\mathcal{A}$ outputs a message $m^*$ with forging signature $\sigma_{m^*}$. The $\mathcal{A}$ wins if $\mathsf{Vfy}(\mathsf{pp}_{sig}, \mathsf{vk}_{sig}, m^*, \sigma_{m^*}) = 1$ and $m^* \notin \mathcal{M}$.

Fig. 9: EUF-CMA Game

## E Signature Schemes

**Schnorr Signature Scheme [29]** Schnorr Signature scheme is a EUF-CMA signature scheme under the DL assumption. The Schnorr signature scheme $\mathsf{Sch}$ consists of four algorithms ($\mathsf{Setup}$, $\mathsf{Keygen}$, $\mathsf{Sign}$, $\mathsf{Verify}$) as follows:

- $\mathsf{pp}_{sig} \leftarrow \mathsf{Sch.Setup}(\lambda)$: This algorithm takes the security parameter $\lambda$ as input and returns the public parameter $\mathsf{pp}_{sig} = (\langle p, \mathbb{G}, g \rangle, T)$. $p$ is a prime order of group $\mathbb{G}$ and $g$ is a generator of $\mathbb{G}$. $T : \{0,1\}^* \rightarrow \mathbb{Z}_p$ is a hash function.
- $(\mathsf{vk}_{sig}, \mathsf{sk}_{sig}) \leftarrow \mathsf{Sch.Keygen}(\mathsf{pp}_{sig})$: It takes a public parameter as input and picks random $a \xleftarrow{\$} \mathbb{Z}_p$. And then, it returns a key tuple of signing key $\mathsf{sk}_{sig} = a$ and verifying key $\mathsf{vk}_{sig} = A = g^a$.

- $\sigma \leftarrow \mathsf{Sch.Sign}(\mathsf{pp}_{sig}, \mathsf{sk}_{sig}, m)$: It takes public parameter $\mathsf{pp}_{sig}$, signing key $\mathsf{sk}_{sig} = a$ and message $m \in \{0,1\}^*$. The signing process is as follows:
  1. Picks random $r \xleftarrow{\$} \mathbb{Z}_p$ and compute $R \leftarrow g^r$
  2. Compute $c \leftarrow T(R \parallel m)$
  3. Compute $s \leftarrow r + ca$

  And then, it returns $\sigma = (R, s) \in \mathbb{G} \times \mathbb{Z}_p$
- $0 \backslash 1 \leftarrow \mathsf{Sch.Verify}(\mathsf{pp}_{sig}, \mathsf{vk}_{sig}, m, \sigma)$: It takes public parameter $\mathsf{pp}_{sig}$, the verifying key $\mathsf{vk}_{sig} = A$, message $m$, and signature $\sigma = (R, s)$. If $g^s = R \cdot A^{T(R \| m)}$ it returns 1; otherwise, it returns 0.

**BLS Signature Scheme [7]** The BLS signature is a pairing-based signature scheme satisfying EUF-CMA under the CDH assumption, which is implied by the DL assumption. The BLS signature scheme $\mathsf{BLS}$ consists of four algorithms ($\mathsf{Setup}$, $\mathsf{Keygen}$, $\mathsf{Sign}$, $\mathsf{Verify}$) as follows:

- $\mathsf{pp}_{sig} \leftarrow \mathsf{BLS.Setup}(\lambda)$: This algorithm takes the security parameter $\lambda$ as input and returns the public parameter $\mathsf{pp}_{sig} = (\langle p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e \rangle, T)$. $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order $p$. $g_1$ and $g_2$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a degenerated bilinear map. $T : \{0,1\}^* \rightarrow \mathbb{G}_2$ is a cryptographic hash function.
- $(\mathsf{vk}_{sig}, \mathsf{sk}_{sig}) \leftarrow \mathsf{BLS.Keygen}(\mathsf{pp}_{sig})$: It takes a public parameter as input and picks random $a \xleftarrow{\$} \mathbb{Z}_p$. And then, it returns a key tuple of signing key $\mathsf{sk}_{sig} = a$ and verifying key $\mathsf{vk}_{sig} = A = g_1^a$.
- $\sigma \leftarrow \mathsf{BLS.Sign}(\mathsf{pp}_{sig}, \mathsf{sk}_{sig}, m)$: It takes public parameter $\mathsf{pp}_{sig}$, signing key $\mathsf{sk}_{sig} = a$ and message $m \in \{0,1\}^*$. And then, it returns $\sigma = T(m)^a$
- $0 \backslash 1 \leftarrow \mathsf{BLS.Verify}(\mathsf{pp}_{sig}, \mathsf{vk}_{sig}, m, \sigma)$: It takes public parameter $\mathsf{pp}_{sig}$, the verifying key $\mathsf{vk}_{sig} = A$, message $m$, and signature $\sigma \in \mathbb{G}_2$. If $e(g_1, \sigma) = e(A, T(m))$ it returns 1; otherwise, it returns 0.

**Generic Group Model [30,21]** A generic group model (GGM) is an idealized model for a group whose operations are carried out by making oracle queries. The GGM is designed to capture the behavior of general algorithms that operate independently of any particular group descriptions. By this restriction, the GGM establishes a lower bound on the cost of solving the discrete logarithm (DL) problem, which is sub-exponential [30,21]. In other words, GGM implies DL assumption, which serves as the underlying assumption of both Schnorr's signature [29] and BLS signatures [7].

Specifically, we consider the bilinear GGM, which additionally simulates a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ as proposed in [15]. The bilinear GGM is defined by the following:

**Definition 5 (Bilinear Generic Group Algorithm [30,15]).** *A bilinear generic group algorithm $\mathcal{A}$ is an algorithm that can access bilinear generic group oracle $\mathcal{O}_{BL}$ to treat group operation. The bilinear generic group oracle runs as follows in Figure 10.*

**Generic Group Oracle $\mathcal{O}_{BL}$**

- **Query format**: two indices with op-type $(i, j, \mathsf{op}) \in \mathbb{Z}_p \times \mathbb{Z}_p \times \{\times_1, \times_2, \times_T, e\}$.
- **Output**: a bitstring $s \in \{0,1\}^*$.
- **Encoding List**: $\mathcal{L} := \{(i, \mathsf{type}), s \in \mathbb{Z}_p \times \{1, 2, T\} \times \{0,1\}^*\}$, the $\mathcal{O}$ manages the list $\mathcal{L}$ locally.
- **Group Operation**: If $\mathcal{O}$ takes a query $(i, j, \times_{\mathsf{type}})$ for $\mathsf{type} \in \{1, 2, T\}$, then $\mathcal{O}$ follows the process.
  1. If the index-type tuple $(i+j, \mathsf{type})$ belongs to the list $\mathcal{L}$, then outputs $(i+j, \mathsf{type})$ corresponding string $s$ where $(i + j, \mathsf{type}, s) \in \mathcal{L}$
  2. Else, sample $s \xleftarrow{\$} \{0,1\}^*$ until $(*, *, s) \notin \mathcal{L}$
  3. Output $s$ and adds $(i + j, \mathsf{type}, s)$ to the list $\mathcal{L}$
- **Bilinear Map**: If $\mathcal{O}$ takes a query $(i, j, e)$, then $\mathcal{O}$ follows the process.
  1. If the index-type tuple $(ij, T)$ belongs to the list $\mathcal{L}$, then outputs $(ij, T)$ corresponding string $s$ where $(ij, T, s) \in \mathcal{L}$
  2. Else, sample $s \xleftarrow{\$} \{0,1\}^*$ until $(*, *, s) \notin \mathcal{L}$
  3. Output $s$ and adds $(ij, T, s)$ to the list $\mathcal{L}$

Fig. 10: Bilinear Generic Group Oracle

## F Token Unforgeability of TUDORE (Proof of Thm. 1)

In this section, we complete token unforgeability of VDORE scheme. As we mentioned, we show that the adventage of token forging game adversary $\mathcal{A}$ should be negligible.

Let $\mathcal{A}$ and $\mathcal{B}$ be adversaries against the token forging game (Figure 3) and EUF-CMA game (Figure 9) respectively. Now we construct $\mathcal{B}$ which exploits $\mathcal{A}$. Note that $\mathcal{B}$ roles challenger in token forging game against $\mathcal{A}$. Additionally, we restrict $\mathcal{A}$ should send key query to get a public key, which is reasonable under GGM. Specifically, we consider the bilinear GGM model to access $\mathcal{O}_{BL}$ in Figure 10.

*Simulation $\mathcal{C}$ against $\mathcal{A}$* As we mentioned, $\mathcal{B}$ should simulate challenger $\mathcal{C}$ for the token forging game in Figure 3. We describe how to simulate $\mathcal{C}$ in Figure 11.

In the setting phase, $\mathcal{B}$ generates $\mathsf{pk}_{(\mathcal{B})}$ using generic group oracle $\mathcal{O}_{BL}$ in Figure 10. And then, sends verificatino key $\mathsf{vk}_{(\mathcal{B})} = \mathsf{vk}_{sig}$ received by $\mathcal{C}_{sig}$ and public key $\mathsf{pk}_{(\mathcal{B})}$ of $\mathcal{B}$ to $\mathcal{A}$.

In the key query phase, $\mathcal{B}$ simulates $\mathcal{C}$ using $\mathcal{O}_{BL}$. Concretely, $\mathcal{B}$ runs Keygen with accessing $\mathcal{O}_{BL}$.

In the token query phase, by our premise, $\mathcal{B}$ already knows an exponent $a_{(\mathcal{A})}$ of token queried public key $\mathsf{pk}_{(\mathcal{A})}$, which should belong to the key query set $S_{\mathsf{key}}$. Then, $\mathcal{B}$ can generates $(t^0_{(\mathcal{A}\to\mathcal{B})}, t^1_{(\mathcal{A}\to\mathcal{B})}) = (\mathsf{pk}_{(\mathcal{A})}, \mathsf{pk}_{\mathcal{A}}^{a_{(\mathcal{B})}} = g_2^{a_{(\mathcal{A})}a_{(\mathcal{B})}})$ with accessing $\mathcal{O}_{BL}$. After then, $\mathcal{B}$ gets signature $\sigma_{\mathcal{B}}$ from signature query to $\mathcal{C}_{sig}$. Therefore $\mathcal{B}$ can response the token query by getting $\mathsf{tok}_{(\mathcal{A}\to\mathcal{C})}$ from $\mathcal{O}_{BL}$.

$\mathcal{B}^{\mathcal{A}}(1^\lambda) \to (\widehat{m}, \widehat{\sigma})$

1. **Setting Phase**: $\mathcal{C}_{sig}$ runs setup algorithm $\mathsf{pp}_{sig} := (\langle p, [\mathbb{G}_1, g_1]_{\mathcal{O}_{BL}} \rangle, T) \leftarrow \mathsf{Setup}(1^\lambda)$ and key generation algorithm $(\mathsf{vk}_{sig}, \mathsf{sk}_{sig}) \leftarrow \mathsf{Keygen}(\mathsf{pp}_{sig})$. And then sends $(\mathsf{pp}_{sig}, \mathsf{vk}_{sig})$ to $\mathcal{B}$.
2. **Simulation $\mathcal{C}$ against $\mathcal{A}$**: $\mathcal{B}$ roles token forging game challenger $\mathcal{C}$ against $\mathcal{A}$.
   (a) **Setting Phase**: $\mathcal{B}$ construct $\mathsf{pp} = (\langle p, [\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e]_{\mathcal{O}_{BL}} \rangle, H, T)$ using $\mathsf{pp}_{sig}$. And then $\mathcal{B}$ samples $a_{(\mathcal{B})} \xleftarrow{\$} \mathbb{Z}_p$ and access generic group oracle $\mathcal{O}_{BL}$ to get a public key $\mathsf{pk}_{(\mathcal{B})} \leftarrow \mathcal{O}_{BL}(a_{(\mathcal{B})}, 0, \times_2)$. And then $\mathcal{B}$ sends $(\mathsf{pp}, \mathsf{vk}_{(\mathcal{B})} = \mathsf{vk}_{sig}, \mathsf{pk}_{(\mathcal{B})})$ to $\mathcal{A}$.
   (b) **Key Query**: If $\mathcal{A}$ sends key query with index $i$ to $\mathcal{B}$, then $\mathcal{B}$ follows the role of challenger in Figure 3. If $(i, \mathsf{pk}_{(i)}, \mathsf{vk}_{(i)}) \in S_{\mathsf{key}}$, then output $(i, \mathsf{pk}_{(i)}, \mathsf{vk}_{(i)})$. Otherwise, it runs $(\mathsf{sk}, \mathsf{pk}, \mathsf{tk}) \leftarrow \mathsf{Keygen}^{\mathcal{O}_{BL}}(\mathsf{pp})$. And it returns $(\mathsf{pk}, \mathsf{tk})$ and adds the tuple $(i, \mathsf{pk}, \mathsf{tk})$ to key query set $S_{\mathsf{key}}$
   (c) **Token Query**: If $\mathcal{A}$ sends token query with $\mathsf{pk}_{(\mathcal{A})}$, then $\mathcal{B}$ finds $a_{(\mathcal{A})}$ from the key query set $S_{\mathsf{key}}$. And then $\mathcal{B}$ access generic group oracle $\mathcal{O}_{BL}$ to get random string $t^1_{(\mathcal{A}\to\mathcal{B})} \leftarrow \mathcal{O}_{BL}(a_{(\mathcal{A})} a_{(\mathcal{B})}, 0, \times_2)$. After then, $\mathcal{B}$ sends signature query $(t^0_{(\mathcal{A}\to\mathcal{B})} := \mathsf{pk}_{(\mathcal{A})}, t^1_{(\mathcal{A}\to\mathcal{B})})$ to $\mathcal{C}_{sig}$ and gets a signature $\sigma_{\mathcal{B}}$. Finally, $\mathcal{B}$ responses $\mathsf{tok}_{(\mathcal{A}\to\mathcal{B})} = (t^0_{(\mathcal{A}\to\mathcal{B})}, t^1_{(\mathcal{A}\to\mathcal{B})}, \sigma_{\mathcal{B}})$ to $\mathcal{A}$
   (d) **Receive Forged Token**: $\mathcal{B}$ recieves forged token $(\widehat{\mathsf{vk}_{(\mathcal{A})}}, \widehat{\mathsf{tok}_{(\mathcal{B}\to\mathcal{A})}}, \widehat{\mathsf{tok}_{(\mathcal{A}\to\mathcal{B})}})$ from $\mathcal{A}$.
3. **Challenge Phase**: $\mathcal{B}$ answers $\widehat{\mathsf{tok}_{(\mathcal{A}\to\mathcal{B})}} = \left( (\widehat{t^0_{(\mathcal{A}\to\mathcal{B})}}, \widehat{t^1_{(\mathcal{A}\to\mathcal{B})}}), \widehat{\sigma_{\mathcal{B}}} \right) = (\widehat{m}, \widehat{\sigma})$ to $\mathcal{C}_{sig}$.

Fig. 11: Construct $\mathcal{B}$ using $\mathcal{A}$

Finally, $\mathcal{B}$ receives the forged token from $\mathcal{A}$ and then uses it to win the EUF-CMA game (Figure 9).

In the simulation process, $\mathcal{B}$ does not fail to respond to the $\mathcal{A}$'s queries, and the responses follow the same distribution as in the real game. This means that, from the adversary's perspective, the real game in Figure 3 is indistinguishable from the simulated game by $\mathcal{B}$ in Figure 11.

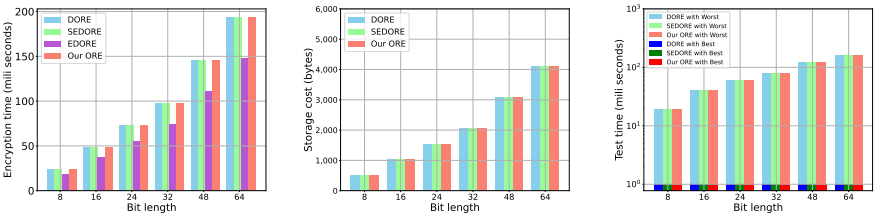
*Probability Analysis* If $\mathcal{A}$ succeeds to forge the token, then the signature parts $\widehat{\sigma}_{\mathcal{B}}$ should be valid signature for the message $\widehat{m} = (\widehat{t^0_{(\mathcal{A}\to\mathcal{B})}}, \widehat{t^1_{(\mathcal{A}\to\mathcal{B})}})$. That means, $\mathcal{B}$ can succeed in the forging signature of adaptively chosen message $\widehat{m}$. Then, we get the following inequality.

$$\mathsf{Adv}^{\mathsf{TF}}[\mathcal{A}, \mathsf{VDERE}] \leq \mathsf{Adv}^{\mathsf{EUF-CMA}}[\mathcal{B}, \mathsf{SSig}] \tag{1}$$

where $\mathsf{Adv}^{\mathsf{TF}}[\mathcal{A}, \mathsf{VDERE}]$ is $\mathcal{A}$'s advantage to the token forging game (Figure 3) and $\mathsf{Adv}^{\mathsf{EUF-CMA}}[\mathcal{B}, \mathsf{SSig}]$ is $\mathcal{B}$'s advantage to the EUF-CMA game Figure 9 under the security parameter $\lambda$.

By our premise, $\mathsf{Adv}^{\mathsf{EUF-CMA}}[\mathcal{B}, \mathsf{SSig}]$ is at most negligible to $\lambda$. Then, Equation (1) implies the following inequality: $\mathsf{Adv}^{\mathsf{TF}}[\mathcal{A}, \mathsf{VDERE}] < \mathsf{negl}(\lambda)$. Thus, we can conclude that VDERE satisfies token unforgeability. □

## G Additioanl Experiments




(a) Encryption time (b) Ciphertext storage cost (c) Test time(B & W)

Fig. 12: The performance graphs for encryption time, ciphertext storage cost, and test time for DORE, SEDORE, and our VDORE. In Figure 12c, B indicates Best and W indicates Worst.

This section introduces the performance for encryption time, ciphertext storage cost, and test time. The encryption is executed by the users, and the test is operated by the servers. As shown in Figure 12, we can identify all values are identical throughout DORE, SEDORE, and VDORE because the three schemes use the same encryption and test algorithm. Furthermore, in Figure 12c, we show the evaluation of the test algorithm for the best and worst scenarios. From this, the best-case scenario involved comparing two plaintexts where the most significant bit (MSB) differed. For instance, comparing $1 \cdots b_6 b_{7(2)}$ and $0 \cdots b'_6 b'_{7(2)}$ in an 8-bit scenario. Therefore, we compare values for this experiment where only the MSB of each bit length is set to 1 and 0. On the other hand, the worst-case scenario involves comparing two identical plaintexts. The light-colored graphs represent results for the worst-case scenario, while the dark-colored ones depict the best-case scenario. In the best-case scenario, regardless of the bit length, each has a fixed cost of about 1 second. In the worst-case scenario, three ORE schemes take approximately 19 seconds and 158 seconds for 8-bit and 64-bit operations, respectively. In Figure 5b, we show that the computational time falls within the range of Figure 12c.

## References

1. Amazon employee. `https://explodingtopics.com/blog/amazon-employees`, (Accessed on 05/30/2024)
2. Google employee. `https://seo.ai/blog/how-many-people-work-at-google`, (Accessed on 05/30/2024)
3. IBM employee. `https://stockanalysis.com/stocks/ibm/employees/`, (Accessed on 05/30/2024)

4. 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. pp. 563–574 (2004)
5. Berger, R., Dörre, F., Koch, A.: Two-party decision tree training from updatable order-revealing encryption. In: International Conference on Applied Cryptography and Network Security. pp. 288–317. Springer (2024)
6. 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)
7. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. Journal of cryptology **17**, 297–319 (2004)
8. 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)
9. Challita, S., Zalila, F., Gourdin, C., Merle, P.: A precise model for google cloud platform. In: 2018 IEEE international conference on cloud engineering (IC2E). pp. 177–183. IEEE (2018)
10. Chen, Z., Nie, J., Li, Z., Susilo, W., Ge, C.: Geometric searchable encryption for privacy-preserving location-based services. IEEE Transactions on Services Computing **16**(4), 2672–2684 (2023)
11. 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 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers 23. pp. 474–493. Springer (2016)
12. Cheon, J.H., Kim, D., Kim, D.: Efficient homomorphic comparison methods with optimal complexity. In: Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26. pp. 221–256. Springer (2020)
13. Cheon, J.H., Kim, M., Kim, M.: Optimized search-and-compute circuits and their application to query evaluation on encrypted data. IEEE Transactions on Information Forensics and Security **11**(1), 188–199 (2015)
14. Hahn, C., Hur, J.: Delegatable order-revealing encryption for reliable cross-database query. IEEE Transactions on Services Computing (2022)
15. Jager, T., Rupp, A.: The semi-generic group model and applications to pairing-based cryptography. In: Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16. pp. 539–556. Springer (2010)
16. Kerschbaum, F., Schröpfer, A.: Optimal average-complexity ideal-security order-preserving encryption. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 275–286 (2014)
17. Li, Y., Wang, H., Zhao, Y.: Delegatable order-revealing encryption. In: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security. pp. 134–147 (2019)
18. 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

2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part II 26. pp. 44–63. Springer (2021)
19. Lynn, B.: Pairing-based cryptography library. https://crypto.stanford.edu/pbc/ (09 2006)
20. Mathew, S., Varia, J.: Overview of amazon web services. Amazon Whitepapers **105**(1), 22 (2014)
21. Maurer, U.: Abstract models of computation in cryptography. In: Cryptography and Coding: 10th IMA International Conference, Cirencester, UK, December 19-21, 2005. Proceedings 10. pp. 1–12. Springer (2005)
22. Miao, Y., Li, F., Li, X., Liu, Z., Ning, J., Li, H., Choo, K.K.R., Deng, R.H.: Time-controllable keyword search scheme with efficient revocation in mobile e-health cloud. IEEE Transactions on Mobile Computing **23**(5), 3650–3665 (2023)
23. MISHRA, A.: Faang- complete stock data. `https://www.kaggle.com/datasets/aayushmishra1512/faang-complete-stock-data` (05 2020), (Accessed on 05/30/2024)
24. Park, J.H., Rezaeifar, Z., Hahn, C.: Securing multi-client range queries over encrypted data. Cluster Computing pp. 1–14 (2024)
25. Park, J., Lee, H., Hur, J., Seo, J.H., Kim, D.: Utra: Universe token reusability attack and verifiable delegatable order-revealing encryption. Cryptology ePrint Archive (2024)
26. Popa, R.A., Li, F.H., Zeldovich, N.: An ideal-security protocol for order-preserving encoding. In: 2013 IEEE Symposium on Security and Privacy. pp. 463–477. IEEE (2013)
27. Qiao, H., Peng, C., Feng, Q., Luo, M., He, D.: Ciphertext range query scheme against agent transfer and permission extension attacks for cloud computing. IEEE Internet of Things Journal (2024)
28. 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. pp. 1131–1142 (2016)
29. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Advances in Cryptology—CRYPTO'89 Proceedings 9. pp. 239–252. Springer (1990)
30. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16. pp. 256–266. Springer (1997)
31. Tan, B.H.M., Lee, H.T., Wang, H., Ren, S., Aung, K.M.M.: Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. IEEE Transactions on Dependable and Secure Computing **18**(6), 2861–2874 (2020)
32. U.Nations: World population prospects - population division - united nations. `https://population.un.org/wpp/` (05 2022), (Accessed on 05/30/2024)
33. Xiao, J., Chang, J., Lin, L., Li, B., Dai, X., Xiong, Z., Choo, K.K.R., Gai, K., Jin, H.: Cloak: Hiding retrieval information in blockchain systems via distributed query requests. IEEE Transactions on Services Computing (2024)
34. Xu, J., Peng, C., Li, R., Fu, J., Luo, M.: An efficient delegatable order-revealing encryption scheme for multi-user range queries. IEEE Transactions on Cloud Computing (2024)
35. Xu, Y., Cheng, H., Liu, X., Jiang, C., Zhang, X., Wang, M.: Pcse: Privacy-preserving collaborative searchable encryption for group data sharing in cloud computing. IEEE Transactions on Mobile Computing (2025)

36. Yu, P., Ni, W., Liu, R.P., Zhang, Z., Zhang, H., Wen, Q.: Efficient encrypted range query on cloud platforms. ACM Transactions on Cyber-Physical Systems (TCPS) **6**(3), 1–23 (2022)