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 dataset sizes grow, users increasingly rely on encrypted data and secure range queries on cloud servers, raising privacy concerns about potential data leakage. Order-revealing encryption (ORE) enables efficient operations on numerical datasets, and Delegatable ORE (DORE) extends this functionality to multi-client environments, but it faces risks of token forgery. Secure DORE (SEDORE) and Efficient DORE (EDORE) address some vulnerabilities, with EDORE improving speed and storage efficiency. However, we find that both schemes remain susceptible to token forgery.
To address this issue, we propose the concept of Verifiable Delegatable Order-Revealing Encryption (VDORE) with a formal definition of token unforgeability. We then construct a new VDORE scheme TUDORE (Token Unforgebale DORE), which ensures token unforgeability. Furthermore, our TUDORE achieves about $1.5\times$ speed-up in token generation compared to SEDORE and EDORE.

1 Introduction

With increasing dataset sizes, processing tasks are becoming impractical on local machines, driving demand for cloud services. To address privacy concerns, clients upload encrypted data and use secure range queries, protecting datasets from adversaries while facilitating database operations. Order-revealing encryption (ORE) has been proposed to enable secure operations on numerical data [9,7,13,12,10,21,5,20,4,18].

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 example, ORE takes two ciphertexts as input and returns the order associated with the underlying plaintexts. Furthermore, Li et al. [12] proposed an

* Equal contribution
** Co-corresponding authors

ORE primitive called delegatable order-revealing encryption (DORE), which allows appropriate operations even with different encryption keys. DORE works by having data owners provide authorization tokens, based on their secret keys, to users.

However, Hahn et al. [10] identified vulnerabilities in DORE, demonstrating that unauthorized users could forge authorization tokens under a threat model that is both practical and reasonable [21]. In such attacks, an authorized user (traitor) collaborates with an unauthorized user (attacker) to forge tokens, enabling illegal query execution on the data owner's database. These attacks result not only in unauthorized data access but also financial losses for the victims, as many modern cloud service providers (CSPs) [8,15,26] operate on a pay-per-query model. Furthermore, in this attack scenario, the data owner cannot identify the traitor, making the attack stealthy.

To address these security concerns, Hahn et al. proposed Secure Delegatable Order-Revealing Encryption (SEDORE). Subsequently, Xu et al. [25] introduced Efficient Delegatable Order-Revealing Encryption (EDORE), which improves latency and reduces storage costs compared to SEDORE.

Despite the improvements in SEDORE and EDORE to mitigate practical forgery attacks, we discover that the same vulnerability exists in DORE, SEDORE, and EDORE under the identical threat model proposed by [10]. We term this vulnerability as *universal token reusability.* This attack remains highly threatening because it adheres to the practical threat model defined by [10], even when the traitor provides the attacker with more information in our scenario compared to SEDORE.

To address these issues, we propose a revised DORE scheme incorporating a verification algorithm, termed Verifiable Delegatable Order-Revealing Encryption (VDORE). We also formally define the token unforgeability of the VDORE scheme for provable security.

Furthermore, we propose TUDORE, a novel token-unforgeable DORE scheme, build upon VDORE. Like SEDORE [10], our scheme retains the original algorithms of DORE [12] 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 [22,6] 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 [12,10,25].

In summary, we make the following main contributions:

1. **Universal Token Reusability Attack.** We first highlight that token-based DORE schemes [12,10,25] remain vulnerable to token forgery attacks. Specifically, under the same threat model described in [10], we introduce a new attack called *universal token reusability attack* in Section 5.
2. **Token Verification and Security Definition.** To mitigate the security risks identified in the attack scenario, we revise the token-based DORE scheme by integrating a token verification algorithm. We explain the necessity of verification in Section 5. This revision results in a new concept called

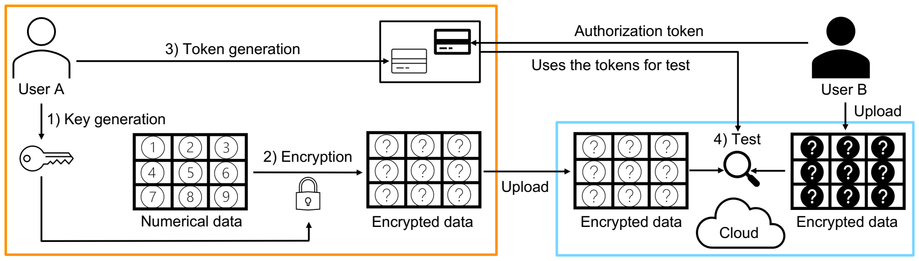


Fig. 1: The description of order-revealing encryption. The orange box indicates the operations executed by User A, whereas the blue box represents the processes handled by the cloud.

Verifiable DORE (VDORE). VDORE ensures provable security by satisfying three key properties: *correctness*, *data privacy*, and *token unforgeability*, as detailed in Section 6.

3. **Token Unforgeable VDORE Scheme.** We present a new secure VDORE scheme, named Token Unforgeable DORE (TUDORE), which leverages digital signature schemes [22,6]. Specifically, we prove that TUDORE satisfies the properties of *correctness*, *data privacy*, and *token unforgeability* in Section 7.
4. **Implementation of TUDORE.** We provide implementation results to compare existing schemes [12,10,25] with our proposed method (TUDORE). Our experimental evaluation demonstrates that TUDORE is both practical and feasible, as detailed in Section 8.

2 Background

Cross-database systems. In our system, similar to DORE [12], SEDORE [10], and EDORE [25], 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. 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 [12]. 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.

3 System and Threat Models

3.1 System model

As we mentioned in Section 2, 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.

3.2 Threat Model

Following the attack scenario described in [10], 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, we focus on *token forgeability*. There are three entities involved in forge token attacks: a victim ($\mathcal{V}$) who is the owner of the database, the authorized user ($\mathcal{M}$) who may illegally aid an unauthorized user ($\mathcal{A}$) in creating forge tokens. Furthermore, we assume that $\mathcal{M}$ never shares its secret key with $\mathcal{A}$, as $\mathcal{A}$ can exploit not only $\mathcal{V}$'s database but also $\mathcal{M}$'s database by creating unintentional tokens from $\mathcal{M}$'s secret key.

4 Revisiting token-based DORE Schemes

4.1 Basic Notation

We first define some notations before revisiting the schemes. We denote $\mathbb{Z}_p$ as a prime field that is isomorphic to integers mod p. Uniform sampling is denoted by $\overset{\$}{\leftarrow}$. For instance, $a \overset{\$}{\leftarrow} \mathbb{Z}_p$ 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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, 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.

4.2 Token-based delegatable ORE scheme

Li et al. [12] 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^λ as input and returns the public parameter pp.
– $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DORE.Keygen}(\mathsf{pp})$: It receives a public parameter 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 sk as input and returns a ciphertext 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.[12] introduced two security properties for DORE: correctness and IND-OCPA. Furthermore, Hahn et al.[10] 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 token unforgeability. We formalize these properties in Section 6.
One-side Token is sufficient. In the 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 Test algorithm.

- $\mathsf{pp} \leftarrow \mathsf{DERE.Setup}(1^\lambda)$: It takes the security parameter 1^λ 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 pp as input and uniformly chooses $a, b \xleftarrow{\$} \mathbb{Z}_p$. After then, 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 sk as input. It randomly picks $r \xleftarrow{\$} \mathbb{Z}_p$ and return $\mathsf{ct} := (c^0, c^1) = ((g_1^{rb} H(m))^a, c^1 = g_1^r)$. For user u, we rewrite 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 [12]): $t^0_{(v\to u)} = \mathsf{pk}_{(v)}, \quad t^1_{(v\to u)} = \mathsf{pk}_{(v)}^{a_{(u)} b_{(u)}}$
  - (type-2, SEDERE [10]):$t^0_{(v\to u)} = F\Big(\mathsf{pk}_{(v)}^{a_{(u)}}\Big)^{a^{-1}_{(u)}}, \quad t^1_{(v\to u)} = F\Big(\mathsf{pk}_{(v)}^{a_{(v)}}\Big)^{b_{(v)}}$

  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

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

  Finally, it compares d_0 and d_1 and returns 1 if $d_0 = d_1$ and 0 otherwise.

Fig. 2: DERE and SEDERE Scheme

4.3 Revisiting DORE, SEDORE and EDORE

DERE-to-DORE framework. In [12], 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. We defer to describe the framework in Figure 5.

Subsequent researchs [10,25] follow this framework. For this reason, we now focus on the DERE scheme rather than the DORE scheme itself.

DERE, SEDERE, and EDERE. Li et al. constructed the DERE scheme based on the bilinear setting under generic group model (GGM) [12]. To prevent a token forging attack, Hanh et al. revise the token generation algorithm of DERE and then propose a new DERE scheme, called SEDERE [10]. We describe the DERE and SEDERE schemes in Figure 2. Due to the page limits, we defer the description of EDERE in Appendix A.

5 Universal Token Reusability Attack

In this section, we first demonstrate the vulnerability of SEDERE, which also causes the vulnerability of SEDORE, within the threat model described in Section 3.2 by presenting a concrete attack. 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 an 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:

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

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 its secret key $\mathsf{sk}_{(\mathcal{A})}(a_{(\mathcal{A})}, b_{(\mathcal{A})}) \xleftarrow{\$} \mathbb{Z}_p^2$. 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 $\alpha_1 = \mathcal{A}$,

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

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

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

Limitation of SEDORE. Hahn et al. proposed SEDORE [10] to prevent token forging attacks against colluding users $\mathcal{M}$ and $\mathcal{A}$. Concretely, they applied cryptographic hash functions to the token generation algorithm. Using hash functions is helpful to prevent reconstructing new *valid* token $\mathsf{tok}_{(\mathcal{A}\to\mathcal{V})}$ from an authorized token. However, a pair of valid tokens $\mathsf{tok}_{(\mathcal{A}\to\mathcal{V})}$ and $\mathsf{tok}_{(\mathcal{V}\to\mathcal{A})}$ is not a requirement for querying $\mathcal{V}$'s database. Note that we give an attack scenario using $\mathsf{uft}_{(\mathcal{V}),h_2}$ and $\mathsf{uft}_{(\mathcal{A}),h_2}$, that are not valid tokens.

One of the main reasons is that the $\mathsf{DERE}, \mathsf{Test}$ algorithm does not verify who generates the tokens. For this reason, before executing the test algorithm, the tester, corresponding to the server in our scenario, must ensure that the tokens were generated by authorized individuals and not forged.

6 Verifiable DORE (VDORE) and Token Unforgeability

As we mentioned in the above section, the tester should check the validity of the tokens to prevent universal token reusability attacks. To give a verifiability on DORE scheme (Section 4.2), we revise the Keygen algorithm to output verification key vk and we newly propose a verification algorithm Vfy. We define a verifiable DORE scheme $\mathsf{VDORE} := (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Test}, \mathsf{Vfy})$ as follows:

– $\mathsf{pp} \leftarrow \mathsf{VDORE.Setup}(1^\lambda)$: It takes the security parameter 1^λ 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).

Token Forging game
$\mathcal{A}(1^\lambda) \to (\widehat{\mathsf{vk}_{(\mathcal{A})}}, \widehat{\mathsf{tok}_{(\mathcal{C}\to\mathcal{A})}}, \widehat{\mathsf{tok}_{(\mathcal{A}\to\mathcal{C})}})$

1. **Setting Phase**: $\mathcal{C}$ runs setup algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and key generation algorithm $(\mathsf{sk}_{(\mathcal{C})}, \mathsf{vk}_{(\mathcal{C})}, \mathsf{pk}_{(\mathcal{C})}) \leftarrow \mathsf{Keygen}(\mathsf{pp})$. And then sends $(\mathsf{pp}, \mathsf{vk}_{(\mathcal{C})}, \mathsf{pk}_{(\mathcal{C})})$ to $\mathcal{A}$.
2. **Query Phase**: $\mathcal{A}$ can query to $\mathcal{C}$:
   (a) **Key Query**: $\mathcal{A}$ sends a query with index i. If $(i, \mathsf{pk}_{(i)}, \mathsf{vk}_{(i)}) \in S_{\mathsf{key}}$, then output $(i, \mathsf{pk}_{(i)}, \mathsf{vk}_{(i)})$. Otherwise, it runs $(\mathsf{sk}, \mathsf{pk}, \mathsf{tk}) \leftarrow \mathsf{Keygen}(\mathsf{pp})$. And it returns $(\mathsf{pk}, \mathsf{tk})$ and adds the tuple $(i, \mathsf{pk}, \mathsf{tk})$ to key query set S_{key}
   (b) **Token Query**: If $\mathcal{A}$ sends a query with keys $\mathsf{pk}_{(\mathcal{A})}$, $\mathcal{C}$ generates an authorized token $\mathsf{tok}_{(\mathcal{A}\to\mathcal{C})} \leftarrow \mathsf{Token}(\mathsf{pp}, \mathsf{pk}_{(\mathcal{A})}, \mathsf{sk}_{(\mathcal{C})})$ and then sends $\mathsf{tok}_{(\mathcal{A}\to\mathcal{C})}$ to $\mathcal{A}$ and adds $\mathsf{tok}_{(\mathcal{A}\to\mathcal{C})}$ to token query set S_{tok}.
   The number of queries is at most polynomially large at λ.
3. **Challenge Phase**: $\mathcal{A}$ outputs a verification key and a pair of tokens $(\widehat{\mathsf{vk}_{(\mathcal{A})}}, \widehat{\mathsf{tok}_{(\mathcal{C}\to\mathcal{A})}}, \widehat{\mathsf{tok}_{(\mathcal{A}\to\mathcal{C})}})$.
   The $\mathcal{A}$ wins if $\mathsf{Vfy}(\mathsf{pp}, \mathsf{vk}_{(\mathcal{C})}, \widehat{\mathsf{vk}_{(\mathcal{A})}}, \widehat{\mathsf{tok}_{(\mathcal{A}\to\mathcal{C})}}, \widehat{\mathsf{tok}_{(\mathcal{C}\to\mathcal{A})}}) = 1$ and $\widehat{\mathsf{tok}_{(\mathcal{A}\to\mathcal{C})}} \notin S_{\mathsf{tok}}$.

Fig. 3: Token forging Game

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, $(\mathsf{pk}_{(u)}, \mathsf{vk}_{(u)}, \mathsf{sk}_{(u)})$, $(\mathsf{pk}_{(v)}, \mathsf{vk}_{(v)}, \mathsf{sk}_{(v)})$ be a pair of keys generated by 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 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 VDORE ensures that the ciphertexts ct generated by Enc algorithm do not leak information except order. VDORE provides data privacy if Enc algorithm satisfies indistinguishability under an ordered chosen plaintext attack (IND-OCPA) [12].

Token Unforgeability. Vfy algorithm of the VDORE scheme should detect forged tokens. To give a 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. Note that token unforgeability can also be considered in the context of a verifiable DERE(VDERE) scheme.

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 VDORE(or VDERE) scheme, respectively.

Definition 1 (Token Unforgeability). *Let* $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Test}, \mathsf{Vfy})$ *be a* VDORE *(or* VDERE*) scheme. We say that* VDORE *(or* 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. We construct an attack game for forging tokens in the above paragraph. The key $(\mathsf{pk}_{(\mathcal{C})}, \mathsf{vk}_{(\mathcal{C})})$ which $\mathcal{C}$ sends represents the key of $\mathcal{V}$. Note that other users can know a public key $\mathsf{pk}_{(\mathcal{V})}$ and verification key $\mathsf{vk}_{(\mathcal{V})}$ of $\mathcal{V}$. After that, we allow $\mathcal{A}$ to send two types of queries: key query and token query. From the queries, $\mathcal{A}$ can get several keys and tokens, which stands for public/verification keys and tokens from other users colluding with $\mathcal{A}$. The purpose of $\mathcal{A}$ is to find a pair of tokens, which goes through the verification algorithm Vfy. The hardness of finding a pair of tokens means those of forging tokens, so our token unforgeability implies security against token forgery attacks.

7 TUDORE: Token Unforgeable DORE

In this section, we first construct our novel VDERE scheme, called TUDERE. And then, following the framework in [12], we complete the TUDORE scheme by using the TUDERE scheme. One of the main concerns is how to construct verify algorithm. To guarantee token unforegability, we apply digital signature schemes.

7.1 Signature Schemes compatible with DERE scheme.

A digital signature is a cryptographic scheme for verifying 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 λ as input and returns the public parameter 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 sk_{sig} and verifying key vk_{sig}.
– $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{pp}_{sig}, \mathsf{sk}_{sig}, m)$: It takes public parameter pp_{sig}, signing key sk_{sig} and message as input and outputs signature σ.
– $0\backslash 1 \leftarrow \mathsf{Sig.Verify}(\mathsf{pp}_{sig}, \mathsf{vk}_{sig}, m, \sigma)$: It takes public parameter pp_{sig}, the verifying key vk_{sig} , message m, and signature σ as input. If the signature is valid, it returns 1; otherwise, it returns 0.

DERE scheme in Figure 2 utilize discrete logarithmic 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 [22] and BLS signature [6]. 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 full version [19].

7.2 Construct TUDORE using Signature Scheme

By combining type-1 DERE scheme in Figure 2 with signature schemes 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 [12], 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 5.

7.3 Security Analysis

In [12], the authors provided security proof for the correctness and IND-OCPA for the DORE scheme in the GGM. With the proof in [12] and 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 5) under the* TUDERE *scheme (Figure 4) satisfies the correctness, data privacy and token unforgeability under the generic group model.*

Proof Sketch. *(Correctness)* The correctness of TUDERE holds in the similar way of [12]. 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 Schnorr signature, Vfy should be output 1 so that TUDERE satisfies the correctness. Since $\mathsf{TUDORE.Test}$ consists of several correct $\mathsf{TUDERE.Test}$ algorithms, it follows that TUDORE satisfies correctness as well.
(Data privacy) Since our underlying encryption algorithm is the same as DERE [12], we can prove IND-OCPA in a similar way in [12]. 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.

– $\mathsf{pp} \leftarrow \mathsf{TUDERE.Setup}(1^\lambda)$: This algorithm takes the security parameter 1^λ 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 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 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 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

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 to 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 arbitrary token $\mathsf{tok}_{(i\to\mathcal{C})}$ without corresponding key query for $\mathsf{pk}_{(i)}$. Therefore, we claim that TUDERE satisfies token unforgeability

– $\mathsf{pp} \leftarrow \mathsf{TUDORE.Setup}(1^\lambda)$: With the inputs, run $\mathsf{TUDERE.Setup}$ algorithm and output pp.
– $(\mathsf{pk}, \mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{TUDORE.Keygen}(\mathsf{pp})$: With the inputs, run $\mathsf{TUDERE.Keygen}(\mathsf{pp})$ 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 $\mathsf{TUDERE.Token}$ 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 $\mathsf{TUDERE.Vfy}$ 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 $\mathsf{TUDERE.Test}$ iteratively. For $i = 1$ to n, the algorithm follows it:

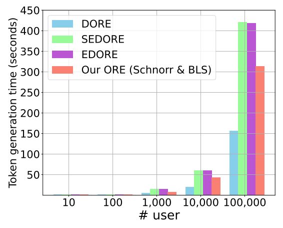
```
1. If $i = n + 1$, then return 0
2. Else Compute $res_u^i$ and $res_v^i$ as follows:
     • $res_u^i \leftarrow \mathsf{TUDERE.Test}(\mathsf{ct}_{(u)}[i,0],\ \mathsf{ct}_{(v)}[i,1])$
     • $res_v^i \leftarrow \mathsf{TUDERE.Test}(\mathsf{ct}_{(u)}[i,1],\ \mathsf{ct}_{(v)}[i,0])$
   (a) If $res_u^i = 1$, then returns 1
   (b) Else if $res_v^i = 1$, then return $-1$
   (c) Else $i \leftarrow i + 1$
```

Fig. 5: TUDORE scheme from TUDERE

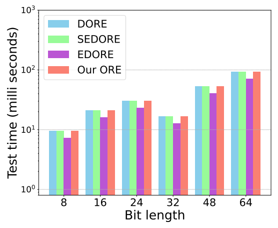
if the underlying signature scheme satisfies EUF-CMA. For the complete proof, please refer to full version [19]. □

8 Experiment

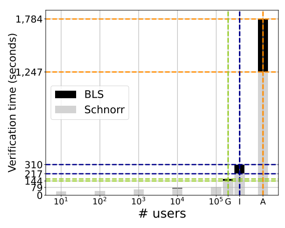
This section evaluates our proposed TUDORE scheme compared to DORE, SE-DORE, and EDORE, which serve as benchmarks.
Experiment environments. To conduct realistic experiments, we implement and evaluate the system in two distinct environments representing the server and the user. The server environment consists of a Linux desktop with a 5.20 GHz Intel i9-12900K CPU and 64GB RAM for the test and verification. In contrast, the user environment is simulated using a Linux laptop with a 1.4 GHz AMD Ryzen 7 4700U CPU and 16GB RAM for token generations. For comparison, we implement our proposed TUDORE scheme alongside the previous schemes.



(a) Token generation time



(b) Test time



(c) Verification time

Fig. 6: The performance graphs for token generation time, test time, and verification time for DORE, SEDORE, and our TUDORE. In Figure 6c's x-axis, "G" indicates Google, "I" indicates IBM, and "A" indicates Amazon, respectively.

Schemes	# Elements		# Operations			
	Enc	Token	Enc	Test	Token	Vfy
DORE [12]	$4n\|\mathbb{G}_1\|$	$2\|\mathbb{G}_2\|$	$6nE_{\mathbb{G}_1}$	$8nP_{\mathbb{G}_T}$	$E_{\mathbb{G}_2}$	$\times$
SEDORE [10]	$4n\|\mathbb{G}_1\|$	$2\|\mathbb{G}_2\|$	$6nE_{\mathbb{G}_1}$	$8nP_{\mathbb{G}_T}$	$3E_{\mathbb{G}_2}$	$\times$
EDORE [25]	$4n\|\mathbb{Z}_p\| + 2n\|\mathbb{G}_2\|$	$2\|\mathbb{G}_1\|$	$2nE_{\mathbb{G}_2}$	$8nE_{\mathbb{G}_1} + 4nP_{\mathbb{G}_T}$	$3E_{\mathbb{G}_1}$	$\times$
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

Table 1: A comparative analysis for element and n-bit comparison, with condensed Group/Pairings section. $\times$ indicates the Verification algorithm is not available.

Furthermore, we use the OpenSSL library for the hash function, the pairing-based cryptography library written in C for bilinear maps with MNT224 curve [14], and the GMP library for large integer arithmetic. Furthermore, we use Schnorr and BLS signatures [22,6] for the verification algorithms. Therefore, we present experiments on two signature schemes, allowing users to choose the one that best suits their environment.

Dataset. We utilize the dataset [24] published by the United Nations, which provides an estimate of the total population (both sexes combined) in five-year age groups for our experiment. However, the range of data for the distribution of the population is limited, we also incorporate the volume data of real stock from FAANG companies [17]. We conduct experiments by extracting 5,000 data from each dataset (i.e., a total of 10,000 data).

When 5,000 samples are randomly drawn from the stock volume dataset, the largest number obtained is 3,372,969,600, which requires 32 bits to represent in binary. Therefore, experiments using 8, 16, and 24 bits are conducted using the age distribution dataset, while experiments using 32, 48, and 64 bits are conducted using the stock dataset.

8.1 Performance

We evaluate our TUDORE with DORE, SEDORE, and EDORE across three categories: token generation time, test algorithm time, and verification time.

Comparative analysis. Before experimentally comparing each scheme, we show a theoretical cost analysis for each algorithm based on one bit. In Table 1, we suggest an analysis of storage cost and computational cost, where the left side (group elements) represents storage cost, and the right side (group/pairings) represents computational cost. Furthermore, $|\mathbb{G}_1|$ and $|\mathbb{G}_2|$ denote the sizes of $\mathbb{G}_1$ and $\mathbb{G}_1$, respectively, while $E_{\mathbb{G}_1}$, $E_{\mathbb{G}_2}$, and $E_{\mathbb{G}_T}$ represent the computational overhead for exponential operation for $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, respectively.

As shown in Table 1, the Enc and Test algorithms of SEDORE and TUDORE are identical to those of DORE, resulting in the same storage and computational costs. EDORE improves speed by reducing group elements and pairing operations. TUDORE requires more storage for tokens due to the inclusion of Schnorr or BLS signatures, but since tokens are distributed to clients, this is acceptable. DORE generates tokens with minimal overhead, needing only one $\mathbb{G}_2$-exponentiation, while SEDORE requires three $\mathbb{G}_2$-exponentiations, EDORE performs three $\mathbb{G}_1$-exponentiations, and TUDORE uses one $\mathbb{G}_2$- and one $\mathbb{G}_1$-exponentiation. Therefore, TUDORE is about 2x slower than DORE but 1.5x faster than SEDORE and EDORE.

Most importantly, only TUDORE provides token unforgeability. For the Schnorr signature, it needs $4E_{\mathbb{G}_1}$ and BLS requires $4P_{\mathbb{G}_T}$ for verification. Therefore, by comparing the size and computational speed of the tokens, users can select the scheme that best meets their needs. Furthermore, we will demonstrate the practicality of the verification process in the subsequent sections.

Note that because the encryption methods of DORE, SEDORE, and TUDORE are identical and there are page limitations, the experimental results of Enc are not presented in this paper. We present comprehensive experiments in [19].

Token generation time We introduce the results related to the token generation time with Schnorr and BLS signature schemes. We compare the generation time required for data owners to provide authorization tokens to different users. We conduct experiments assuming the data owner provides tokens to 10, 100, 1,000, 10,000, and 100,000 users, and the relevant results are shown in Figure 6a. SEDORE and EDORE take about 3 times longer than DORE's token generation algorithm and our TUDORE method is approximately 1.5 times faster than the existing SEDORE and EDORE method. Therefore, our method not only offers enhanced security compared to SEDORE and EDORE but is also more efficient in token generation time.

Test time. From Figure 6b, as with the preceding comparative analysis, it can be observed that the test computation costs of DORE, SEDORE, and TUDORE are identical, while EDORE proves to be the fastest. The computational time increases as the bit size ranges from 8 to 24 bits and from 32 to 64 bits in the test algorithm. This is because the padding increases with the increase in bit size, leading to higher computational costs. Moreover, it can be observed that the computational time at 32 bits is faster than that at 16 and 24 bits. This is due to the dataset division in our experiments, with the order based on the 32-bit dataset's most significant bit (MSB).

Verification time We present experimental results on the verification algorithm for two signature schemes [22,6]. The experiments assume token verification for 10, 100, 1,000, 10,000, and 100,000 users. To simulate a realistic scenario, we extend the experiments to CSP companies, testing with the employee counts of Google (182,502)[2], Amazon (1,608,000)[1], and IBM (282,200) [3].

In Figure 6c, it is observed that verification times ranged from 0 to 79 seconds for Schnorr and 0.5 to 84 seconds for BLS across 10 to 100,000 samples. For real-world applications, verification times were approximately 144/164 seconds for Google, 217/310 seconds for IBM, and 1,247/1,783 seconds (20.8/29.7 minutes) for Amazon. Despite the relatively high total time for Amazon, the proposed technique is highly practical, as the average time per person is only 0.77/1.1 milliseconds, ensuring enhanced security.

9 Related Works

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. [9] suggested a practical ORE scheme. Nonetheless, this scheme leaks the most significant different bit (msdb) and thus lacks sufficient security guarantees. After that, Cash et al. [7] introduced parameter-hiding ORE (pORE) with probabilistic characteristics for the single-user scenario, revealing only the equality pattern of msdb. Following this, many schemes have been suggested for multi-client environments [13,18,12,21,25,4].

10 Conclusion

In our paper, we demonstrate the vulnerability of DORE, SEDORE, and EDORE within the same threat model suggested by [10]. While providing additional information beyond the previously outlined attack process, our attack technique remains a menacing and practical threat without violating the attack model presented in [10]. Thus, to address these security concerns, we propose the concept of VDORE with enhanced security proof. Additionally, we introduce TU-DORE, which leverages the Schnorr and BLS signature schemes to ensure security against universe token reusability attacks. We provide a formalized definition and proof to address the unclear definition and proof of token unforgeability in previous work. Furthermore, our scheme offers a faster token generation algorithm than SEDORE and EDORE.

References

1. Amazon employee. https://explodingtopics.com/blog/amazon-employees, 12 2023.
2. Google employee. https://seo.ai/blog/how-many-people-work-at-google, 04 2024.
3. IBM employee. https://stockanalysis.com/stocks/ibm/employees/, 04 2024.
4. Robin Berger, Felix Dörre, and Alexander Koch. Two-party decision tree training from updatable order-revealing encryption. In *International Conference on Applied Cryptography and Network Security*, pages 288–317. Springer, 2024.
5. Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 563–594. Springer, 2015.
6. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of cryptology*, 17:297–319, 2004.
7. David Cash, Feng-Hao Liu, Adam O'Neill, Mark Zhandry, and Cong Zhang. 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*, pages 181–210. Springer, 2018.
8. Stéphanie Challita, Faiez Zalila, Christophe Gourdin, and Philippe Merle. A precise model for google cloud platform. In *2018 IEEE international conference on cloud engineering (IC2E)*, pages 177–183. IEEE, 2018.
9. Nathan Chenette, Kevin Lewi, Stephen A Weis, and David J Wu. 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*, pages 474–493. Springer, 2016.
10. Changhee Hahn and Junbeom Hur. Delegatable order-revealing encryption for reliable cross-database query. *IEEE Transactions on Services Computing*, 2022.
11. Tibor Jager and Andy Rupp. 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*, pages 539–556. Springer, 2010.
12. Yuan Li, Hongbing Wang, and Yunlei Zhao. Delegatable order-revealing encryption. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 134–147, 2019.
13. Chunyang Lv, Jianfeng Wang, Shi-Feng Sun, Yunling Wang, Saiyu Qi, and Xiaofeng Chen. 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*, pages 44–63. Springer, 2021.
14. Ben Lynn. Pairing-based cryptography library. https://crypto.stanford.edu/pbc/, 09 2006.
15. Sajee Mathew and J Varia. Overview of amazon web services. *Amazon Whitepapers*, 105(1):22, 2014.
16. Ueli Maurer. Abstract models of computation in cryptography. In *Cryptography and Coding: 10th IMA International Conference, Cirencester, UK, December 19-21, 2005. Proceedings 10*, pages 1–12. Springer, 2005.

17. AAYUSH MISHRA. Faang- complete stock data. https://www.kaggle.com/datasets/aayushmishra1512/faang-complete-stock-data, 05 2020. (Accessed on 05/30/2024).
18. Jae Hwan Park, Zeinab Rezaeifar, and Changhee Hahn. Securing multi-client range queries over encrypted data. *Cluster Computing*, pages 1–14, 2024.
19. Jaehwan Park, Hyeonbum Lee, Junbeom Hur, Jae Hong Seo, and Doowon Kim. Utra: Universe token reusability attack and verifiable delegatable order-revealing encryption. *Cryptology ePrint Archive*, 2024.
20. Cong Peng, Rongmao Chen, Yi Wang, Debiao He, and Xinyi Huang. Parameter-hiding order-revealing encryption without pairings. In *IACR International Conference on Public-Key Cryptography*, pages 227–256. Springer, 2024.
21. Hongyi Qiao, Cong Peng, Qi Feng, Min Luo, and Debiao He. Ciphertext range query scheme against agent transfer and permission extension attacks for cloud computing. *IEEE Internet of Things Journal*, 2024.
22. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology—CRYPTO'89 Proceedings 9*, pages 239–252. Springer, 1990.
23. Victor Shoup. 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*, pages 256–266. Springer, 1997.
24. U.Nations. World population prospects - population division - united nations. https://population.un.org/wpp/, 05 2022. (Accessed on 05/30/2024).
25. Jingru Xu, Cong Peng, Rui Li, Jintao Fu, and Min Luo. An efficient delegatable order-revealing encryption scheme for multi-user range queries. *IEEE Transactions on Cloud Computing*, 2024.
26. Jinzy Zhu, Xing Fang, Zhe Guo, Meng Hua Niu, Fan Cao, Shuang Yue, and Qin Yu Liu. Ibm cloud computing powering a smarter planet. In *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1*, pages 621–625. Springer, 2009.

A Universal Token Reusability Attack on efficient DORE

This section shows how to adapt our UTRA method to EDORE [25]. Before describing our attack, we show the EDERE suggested by [25].

A.1 Efficient DERE

EDERE scheme consists of five algorithms: $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Test})$ as follows:

– $\mathsf{pp} \leftarrow \mathsf{EDERE.Setup}(1^\lambda)$: It takes the security parameter 1^λ 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 (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 sk and the corresponding 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 sk as input and returns a ciphertext 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.

A.2 UTRA for EDORE.

Xu et al. also follow the DERE-to-DORE framework to construct EDORE [25]. Due to the equivalence of the Vfy algorithm between EDORE and EDERE, we consider the universal token reusability attack on 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:

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

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.

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

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

B Security Definitions

Definition 2 (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 3 (EUF-CMA). *Let* $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{Vfy})$ *be a signature scheme. We say that* 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 7 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 λ.
3. **Challenge Phase**: $\mathcal{A}$ outputs a message m^* with forging signature σ_{m^*}. The $\mathcal{A}$ wins if $\mathsf{Vfy}(\mathsf{pp}_{sig}, \mathsf{vk}_{sig}, m^*, \sigma_{m^*}) = 1$ and $m^* \notin \mathcal{M}$.

Fig. 7: EUF-CMA Game

C Signature Schemes

Schnorr Signature Scheme [22] Schnorr Signature scheme is a EUF-CMA signature scheme under the DL assumption. The Schnorr signature scheme Sch consists of four algorithms (Setup, Keygen, Sign, Verify) as follows:

– $\mathsf{pp}_{sig} \leftarrow \mathsf{Sch.Setup}(\lambda)$: This algorithm takes the security parameter λ 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 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 pp_{sig}, the verifying key $\mathsf{vk}_{sig} = A$, message m, and signature $\sigma = (R, s)$. If $g^s = R{\cdot}A^{T(R\parallel m)}$ it returns 1; otherwise, it returns 0.

BLS Signature Scheme [6] 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 BLS consists of four algorithms (Setup, Keygen, Sign, Verify) as follows:

– $\mathsf{pp}_{sig} \leftarrow \mathsf{BLS.Setup}(\lambda)$: This algorithm takes the security parameter λ 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 $a \overset{\$}{\leftarrow} \mathbb{Z}_p$. 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 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 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 [23,16] 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 [23,16]. In other words, GGM implies DL assumption, which serves as the underlying assumption of both Schnorr's signature [22] and BLS signatures [6].

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 [11]. The bilinear GGM is defined by the following:

Definition 4 (Bilinear Generic Group Algorithm [23,11]). *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 8.*

D 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 7) 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 8.

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 9.

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. 8: Bilinear Generic Group Oracle

In the setting phase, $\mathcal{B}$ generates $\mathsf{pk}_{(\mathcal{B})}$ using generic group oracle $\mathcal{O}_{BL}$ in Figure 8. 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_{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 $\mathcal{O}_{BL}$.

Finally, $\mathcal{B}$ receives the forged token from $\mathcal{A}$ and then uses it to win the EUF-CMA game (Figure 7).

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 9.

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.

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

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

1. **Setting Phase**: $\mathcal{C}_{sig}$ runs setup algorithm $\mathsf{pp}_{sig} = \mathsf{pp}_{sig} := (\langle p, [\mathbb{G}_1, g_1]_{\mathcal{O}_{BL}} \rangle, T) \leftarrow \mathsf{Setup}(1^\lambda)$ 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 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_{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_{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 $\mathcal{C}_{sig}$.

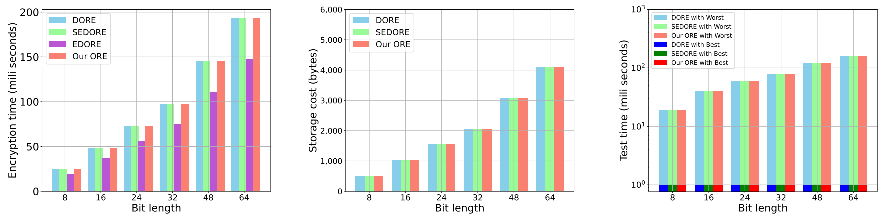
Fig. 9: Construct $\mathcal{B}$ using $\mathcal{A}$

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 7 under the security parameter λ.

By our premise, $\mathsf{Adv}^{\mathsf{EUF-CMA}}[\mathcal{B}, \mathsf{SSig}]$ is at most negligible to λ. Then, we can claim that $\mathsf{Adv}^{\mathsf{TF}}[\mathcal{A}, \mathsf{VDERE}] < \mathsf{negl}(\lambda)$. Thus, we can conclude that VDERE satisfies token unforgeability. $\square$

E Additioanl Experiments

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 10, 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 10c, 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



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

Fig. 10: The performance graphs for encryption time, ciphertext storage cost, and test time for DORE, SEDORE, and our VDORE. In Figure 10c, B indicates Best and W indicates Worst.

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 6b, we show that the computational time falls within the range of Figure 10c.