UTRA: Universe Token Reusability Attack and Verifiable Delegatable Order-Revealing Encryption

Jaehwan Park*
University of Tennessee, Knoxville
jpark127@utk.edu

Hyeonbum Lee*
Hanyang University, Seoul
leehb3706@hanyang.ac.kr

Junbeom Hur
Korea University, Seoul
jbhur@isslab.korea.ac.kr

Jae Hong Seo**
Hanyang University, Seoul
jaehongseo@hanyang.ac.kr

Doowon Kim**
University of Tennessee, Knoxville
doowon@utk.edu

ABSTRACT

As dataset sizes continue to grow, users face increasing difficulties in performing processing tasks on their local machines. From this, privacy concerns about data leakage have led data owners to upload encrypted data and utilize secure range queries to cloud servers. To address these challenges, order-revealing encryption (ORE) has emerged as a promising solution for large numerical datasets. Building on this, delegatable order-revealing encryption (DORE) was introduced, allowing operations between encrypted datasets with different secret keys in multi-client ORE environments. DORE operates through authorization tokens issued by the data owner. However, security concerns had arisen about unauthorized users exploiting data without permission, leading to the development of a secure order-revealing encryption scheme (SEDORE). These attacks can result in unauthorized data access and significant financial losses in modern cloud service providers (CSPs) utilizing pay-per-query systems. In addition, efficient delegatable order-revealing encryption (EDORE), which improves speed and storage compared to SEDORE with identical security levels, was also introduced.

Although both SEDORE and EDORE were designed to be robust against these attacks, we have identified that they still retain the same vulnerabilities within the same threat model. To address these issues, we propose Verifiable Delegatable Order-Revealing Encryption (VDORE), which protects against attacks by using the Schnorr Signature Scheme to verify the validity of the token that users send. We propose a precise definition and robust proof to improve the unclear definition and insufficient proof regarding token unforgeability in the SEDORE. Furthermore, the token generation algorithm in VDORE provides about a 1.5× speed-up compared to SEDORE.

1 INTRODUCTION

With the expansion of dataset sizes, users find it increasingly challenging to execute all processing tasks on their local computers. Consequently, there is a growing demand for cloud services to address these constraints [48, 49]. However, due to concerns regarding information leakage [5, 24], the cloud service clients are resorting to uploading encrypted data and sending secure range queries, which can be created from users to the cloud server. These approaches not only safeguard the dataset from potential adversaries but also mitigate fundamental operations between the database and queries. To overcome these challenges, order-revealing encryption (ORE) has been proposed for numerical data [7, 9, 10, 13, 18, 27, 29, 36, 40].

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. With the advancement of related techniques, ORE methods tailored for multi-user scenarios have gradually emerged [18, 27, 29, 40]. Furthermore, Li et al. [27] proposed an 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. [18] highlighted vulnerabilities within DORE, demonstrating the potential for unauthorized users to forge authorization tokens under a certain threat model. To address these security concerns, they proposed secure order-revealing encryption (SEDORE). From this attack, an authorized user (traitor) assists an unauthorized user (attacker), in forging tokens to execute queries on the database of the data owner illegally. These attacks cause not only unauthorized data access but also financial problems for the victims, as many modern cloud service providers (CSPs) [11, 31, 50] utilize pay-per-query services. Furthermore, in this attack, the data owner is unable to identify the traitor, known as stealthy. Furthermore, Xu et al. [47] introduced efficient delegatable order-revealing encryption (EDORE), which enhances latency and storage cost compared to SEDORE.

Unfortunately, despite the enhancement of SEDORE and EDORE for practical and reasonable forgery attacks, we discover the same vulnerability of DORE, SEDORE, and EDORE with the identical threat model suggested by [18]. We name this vulnerability as *universe token reusability*. From this, even though the traitor provides the attacker with more information in our attack scenario than in SEDORE, it remains a highly threatening attack technique. Because *it never violates the practical threat model* introduced by [18]. Therefore, our attack scenario ensures that the victim does not know who the traitor is, and the attacker cannot access the traitor's database. We provide a detailed explanation in Section 6 and 7. For these security problems, we suggest a *Verifiable Delegatable Order-Revealing Encryption* (VDORE) which prevents universe token reusability attacks.

We develop VDORE in two main steps. Firstly, similar to SEDORE [18], VDORE maintains the original algorithms of DORE [27] for setup, key generation, encryption, and test algorithms, with

* Both authors contributed equally to this research.
** Both authors are co-corresponding authors.

a modified token generation algorithm to prevent attacks. This approach aims to minimize additional computational and storage costs wherever possible. Secondly, VDORE introduces verification techniques to mitigate attacks while ensuring that the token generation time is shorter than that of SEDORE. To achieve this, we integrate the Schnorr Signature Scheme [42] into the token generation algorithm of DORE. As a result, the token generation process in VDORE operates approximately 1.5 times faster than that of SEDORE. Furthermore, we provide a clear definition and proof to resolve the vague definition and proof of token unforgeability, which was a limitation in Hahn et al. [18]'s work.

In summary, we make the following main contributions:

- We suggest the vulnerability of DORE, SEDORE, and EDORE, *universe token reusability*, under the same threat model in [18]. This attack can result not only in illegal usage from unauthorized users but also in significant economic damage to the victim.
- We propose a verifiable delegatable order-revealing encryption (VDORE) technique that uses the Schnorr Signature Scheme [42] to prevent unauthorized tokens. Additionally, this technique is approximately 1.5× faster in token generation compared to SEDORE, while incurring no latency penalties with other algorithms and providing enhanced security.
- Our VDORE not only achieves indistinguishability against order chosen plaintext attack (IND-OCPA) but also guarantees token unforgeability as suggested by [18]. Specifically, we formalize token unforgeability for provable security and then we prove that VDORE satisfies token unforgeability in Section 8.
- We not only conduct theoretical comparisons regarding computational cost and storage but also implement our technique and previous methods to experimentally evaluate our approach. We demonstrate that VDORE is sufficiently practical and feasible in Section 9.

2 BACKGROUND

In this section, we provide the background for cross-database systems, order-revealing encryption (ORE), pay-per-query and the related attack, and insider attacks.

2.1 Cross-database systems

In our system, similar to DORE [27] , SEDORE [18], and EDORE [47], we consider a cross-database scenario. Illustrated in Figure 1, a cross-database system allows multiple users to upload their encrypted databases onto the server, based on their raw data. If users want to collaborate and share datasets, they 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. As depicted in Figure 1, for example, User 1 has granted authorization tokens to Users 2 and 3, while User 4 has not received authorization. Consequently, Users 2 and 3 can utilize User 1's dataset, while User 4 cannot.

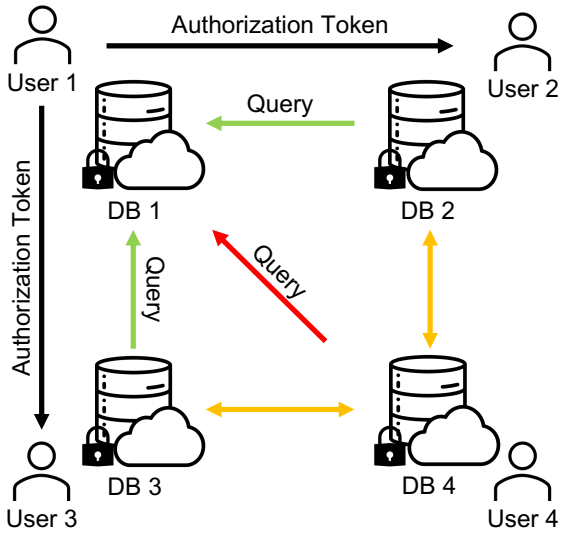


Figure 1: The description of cross-database systems. The green line indicates a valid query, the red line indicates an invalid query, and the yellow line indicates that it may or may not operate based on the possession of an authorization token, respectively.

2.2 Order-Revealing Encryption

With the increasing size of datasets and the growing user base due to advancements in cloud services, the importance of security for data has become prominent. As a result, there is a growing interest in order-revealing encryption (ORE) for enhancing the utility of encrypted numerical data. Furthermore, a delegatable order-revealing encryption scheme has been introduced in a multi-client environment. 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 2, 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.

The *best advantage* of ORE is that users can retrieve the required data from the server through operations on the ciphertext without the need for decryption. Therefore, it applies to vast numerical data requiring encryption. For example, sensitive information such as records of HIV patients or grades, which servers or other entities should not be aware of, can be stored in the cloud. Users can search for the desired data without decryption.

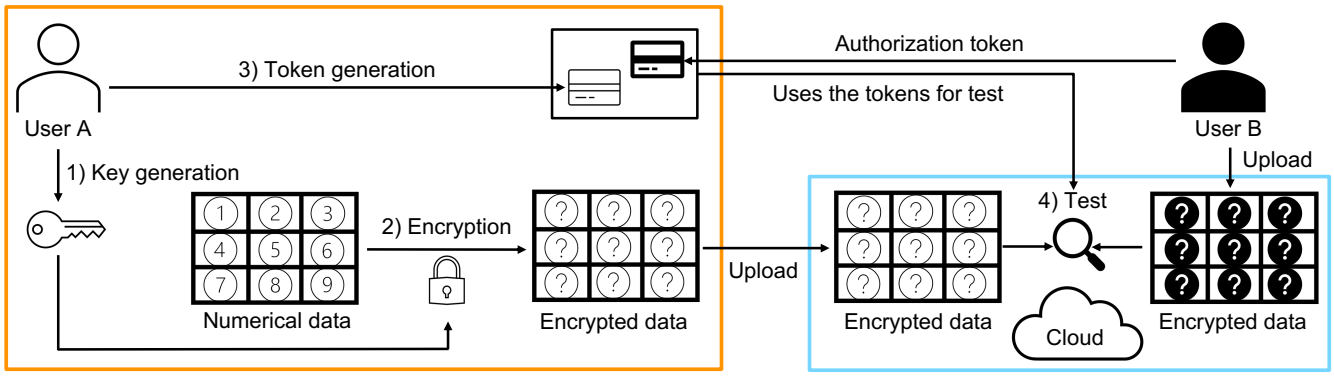


Figure 2: The description of order-revealing encryption. The orange box indicates the operations executed by User A, whereas the blue box represents the processes handled by the cloud.

2.3 Pay-per-query and the related attack

Modern cloud service providers (CSPs) such as Google Cloud [11], Amazon Athena [31], and IBM Cloud [50] charge users based on a pay-per-query model, which depends on the number of tokens sent to the server. While this strategy provides users with cost flexibility, unauthorized usage of the database can lead to significant economic problems for the data owner. A similar real-life case to pay-per-query attacks is the Methbot attack in 2016 [1]. The Methbot attack involved using a bot network to generate a massive inventory of advertisements, which were then displayed or clicked on advertising platforms to generate revenue from advertisers through a pay-per-click model. As a result, it caused significant financial losses, resulting in an average payout of $13.04 per thousand faked views. Therefore, proactive defense against pay-per-query attacks needs to be researched as a necessity.

2.4 Insider attacks

There are many real attack examples caused by authorized insiders [39]. In November 2021, a former employee at the South Georgia Medical Center retrieved confidential data from the medical center's systems onto a USB drive without any apparent justification. From this, the traitor stole Patient test results, names, and birth dates. In April 2022, a former dissatisfied employee illicitly obtained the personal data of users of the mobile payment service Cash App. Following the termination on December 10, 2022, a former disgruntled employee stole sensitive information from Cash App's users, including their full names, brokerage portfolio values, holdings, and stock trading activity. These threats cause secondary harm to users and leave over 66% highly vulnerable, while 63% of organizations lack adequate measures to address them [43]. Therefore, we need to build a secure scheme for potential authorized traitors.

3 PROBLEM STATEMENT

During active research on multi-client environments in ORE, Li et al. [27] introduced DORE, which enables ORE operations without computational costs such as key distribution, by allowing authorized users to exchange tokens without interaction. From this process, Hahn et al. [18] proposed a new attack technique in which authorized users, after receiving authorization tokens from the data owner, collaborate with attackers to create new forged tokens. This technique maintains stealthy characteristics, as the data owner remains unaware of who the traitor is. To address such attacks, SEDORE was introduced. Furthermore, EDORE [47] was introduced to reduce the computational and storage costs compared to SEDORE.

Unfortunately, we discover that SEDORE and EDORE exhibit the same vulnerability as DORE under the identical threat model proposed by Hahn et al. [18]. To tackle the identified vulnerabilities, we propose Verifiable Delegatable Order-Revealing Encryption (VDORE). It achieves enhanced privacy without making major changes to the existing DORE algorithms and also provides faster token generation compared to SEDORE. We will introduce the threat model proposed by [18] in Section 6. Furthermore, we will suggest the vulnerability called *universe token reuability attack* and our VDORE scheme in Section 7 and 8, respectively.

4 REVISIT DORE AND SEDORE

In this section, we introduce the basic notation and revisit DORE and SEDORE.

4.1 Basic Notation

We first define some notations before revisiting the schemes. We denote $\mathbb{Z}_p$ as a prime field which isomorphic to 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 group, 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 computable function satisfies $e(P^a, K^b) = e(P, K)^{ab}$ for all $a, b \in Z_p$.

4.2 DORE, SEDORE, and EDORE Scheme

In this subsection, we scrutinize the DORE [27] and SEDORE [18]. SEDORE reuses key generation, encryption, and test algorithms from DORE, excluding the token generation algorithm. DORE, in

turn, operates based on delegatable equality-revealing encoding (DERE). Due to page limits, we introduce Delegatable Equality-Revealing Encoding (DERE) methods to efficiently describe DORE and SEDORE. It leverages DERE's key generation, encryption, and testing algorithms. Additionally, it encompasses two types of token generation algorithms: type-1 for DORE and type-2 for SEDORE.

Definition 4.1 (DERE Scheme). DERE scheme consists of five algorithms: (Setup, Keygen, Enc, Token, Test). Let $H : \{0,1\}^* \to \mathbb{G}_1$ and $F : \{0,1\}^* \to \mathbb{G}_2$ be cryptographic hash functions. The details of it are as follows:

- $\mathsf{pp} \leftarrow \mathsf{DERE.Setup}(1^\lambda)$: This algorithm takes the security parameter 1^λ as input and returns the public parameter $\mathsf{pp} = (\langle p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e\rangle, H, F)$.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DERE.Keygen}(\mathsf{pp})$: This algorithm receives a public parameter (pp) as input and returns a pair of public key and secret key (pk, sk). From this, it uniformly chooses $a, b \xleftarrow{\$} \mathbb{Z}_p$ and generates sk and the corresponding pk as below:
$$\mathsf{pk} = g_2^a, \quad \mathsf{sk} = (a, b)$$
We denote a key pair of user u as $(\mathsf{pk}_{(u)}, \mathsf{sk}_{(u)}) = (g_2^{a_{(u)}}, (a_{(u)}, b_{(u)}))$.
- $\mathsf{ct} \leftarrow \mathsf{DERE.Enc}(\mathsf{pp}, m, \mathsf{sk})$: This algorithm takes a message $m \in \{0,1\}^*$ and sk as input and returns a ciphertext ct. This algorithm randomly picks $r \xleftarrow{\$} \mathbb{Z}_p$ and computes c^0 and c^1 as below:
$$c^0 = \left(g_1^{rb} H(m)\right)^a, \quad c^1 = g_1^r$$
After that, it returns $\mathsf{ct} = (c^0, c^1)$. For user u, we rewrite ct as $\mathsf{ct}_{(u)} = (c^0_{(u)}, c^1_{(u)})$.
- $\mathsf{tok}_{(v\to u)} \leftarrow \mathsf{DERE.Token}(\mathsf{pp}, \mathsf{pk}_{(v)}, \mathsf{sk}_{(u)})$: This algorithm takes the public key $\mathsf{pk}_{(v)} = g_2^{a_{(v)}}$ of user v and the secret key $\mathsf{sk}_{(u)} = (a_{(u)}, b_{(u)})$ 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 $t^1_{(v\to u)}$. Furthermore, as we discussed earlier, we suggest type-1 and type-2 token generation algorithms for DORE and SEDORE, respectively, as follows:
  – Type-1 (DORE [27]):
  $$t^0_{(v\to u)} = \mathsf{pk}_{(v)}, \quad t^1_{(v\to u)} = \mathsf{pk}_{(v)}^{a_{(u)} b_{(u)}}$$
  – Type-2 (SEDORE [18]):
  $$t^0_{(v\to u)} = F\left(\mathsf{pk}_{(v)}^{a_{(u)}}\right)^{a^{-1}_{(u)}}, \quad t^1_{(v\to u)} = F\left(\mathsf{pk}_{(v)}^{a_{(v)}}\right)^{b_{(v)}}$$
  Finally, it returns $\mathsf{tok}_{(v\to u)} := (t^0_{(v\to u)}, t^1_{(v\to u)})$.
- $0\backslash 1 \leftarrow \mathsf{DERE.Test}(\mathsf{pp}, \mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: This algorithm 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.

Furthermore, we introduce EDORE in Appendix A.

5 SYSTEM MODEL

In this section, we introduce the system model for our VDORE scheme. Same as SEDORE [18], our scheme also provides cross-database environments with encrypted databases. There are three entities in our system model as follows:

- The data owner: It encrypts data using its secret key and uploads it to the server. Then, it provides authorization tokens to users authorized to access their own databases.
- The user: It can request an authorization token from the data owner. If the user receives an authorization token from the data owner, they can use it to perform computations between their and the data owner's databases.
- The server: It serves as a storage for encrypted data uploaded by multiple data owners and performs operations on incoming range queries from users.

Note that while we introduced separate entities for users and data owners above, as shown in Figure 1, 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.

6 THREAT MODEL AND TOKEN FORGERY ATTACK

In this section, we discuss the threat model and *token forgery* attack suggested by Hahn et al. [18]. They proposed two threat models for data privacy violation and token forgeability as follows:

- *Data privacy violation*: The cloud 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 cloud 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, they assumed that $\mathcal{M}$ never shares its secret key with $\mathcal{A}$, as $\mathcal{A}$ can exploit not only $\mathcal{V}$'s database but also $\mathcal{A}$'s database by creating unintentional tokens from $\mathcal{M}$'s secret key. Lastly, this forge attack is stealthy. Because the victim would never divulge $\mathcal{M}$'s credentials (i.e., secret key) unless $\mathcal{M}$ is the only authorized user from $\mathcal{V}$. Furthermore, we consider attacks exploiting the access patterns and/or the information from responses to range queries [15, 17, 25] similar to those described by Hahn et al. [18] and other ORE schemes [13, 26, 27] to be beyond the scope of this study. Thus, mitigating such threats can be achieved through access pattern obfuscation [45] and volume-hiding [22] techniques, which are unrelated to our current research focus.

We introduce the token forgery attack against DORE [27] suggested by [18] as follows:

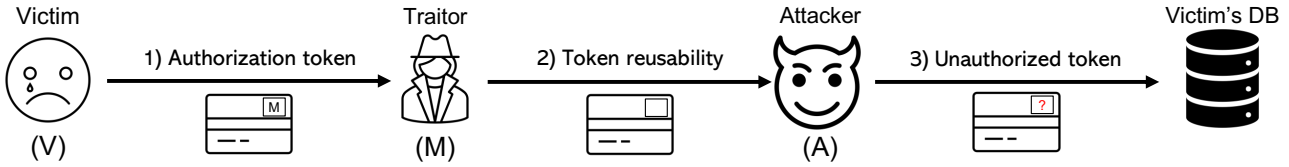


Figure 3: The description of universe token reusability. 1) the victim sends a token to the traitor. 2) the traitor passes a token to the attacker that cannot be traced back to the victim. 3) the attacker creates a token that allows unauthorized access to the victim's database.

1. The victim $\mathcal{V}$ sends the authorization token $\mathsf{tok}_{(\mathcal{M}\to\mathcal{V})} = (g_2^{a_{(\mathcal{M})}}, g_2^{a_{(\mathcal{M})}a_{(\mathcal{V})}b_{(\mathcal{V})}})$ to $\mathcal{M}$.
2. Upon receiving the tokens, $\mathcal{M}$ computes $t^1_{(\square\to\mathcal{V})}$ by using her/his secret key and sends it to $\mathcal{A}$ as follows:
$$\tilde{t^1}_{(\square\to\mathcal{V})} = (g_2^{a_{(\mathcal{M})}a_{(\mathcal{V})}b_{(\mathcal{V})}})^{a^{-1}_{(\mathcal{M})}} = g_2^{a_{(\mathcal{V})}b_{(\mathcal{V})}}$$
3. And then, $\mathcal{A}$ returns $\mathsf{tok}_{(\mathcal{V}\to\mathcal{A})} \leftarrow \mathsf{DERE.Token}(\mathsf{pk}_{(\mathcal{V})}, \mathsf{sk}_{(\mathcal{A})})$. After that, it gets the unauthorized token $\mathsf{tok}_{(\mathcal{A}\to\mathcal{V})}$ which consists of $\tilde{t^0}_{(\mathcal{A}\to\mathcal{V})}$ and $\tilde{t^1}_{(\mathcal{A}\to\mathcal{V})}$ as follows:
$$\tilde{t^0}_{(\mathcal{A}\to\mathcal{V})} = g_2^{a_{(\mathcal{A})}},$$
$$\tilde{t^1}_{(\mathcal{A}\to\mathcal{V})} = (\tilde{t^1}_{(\square\to\mathcal{V})})^{a_{(\mathcal{A})}} = g_2^{a_{(\mathcal{A})}a_{(\mathcal{V})}b_{(\mathcal{V})}}$$
4. Finally, $\mathcal{A}$ sends $\mathsf{tok}_{(\mathcal{A}\to\mathcal{V})}$ and $\mathsf{tok}_{(\mathcal{V}\to\mathcal{A})}$ to server and it runs the DERE.Test.

7 UNIVERSE TOKEN REUSABILITY

In this section, we demonstrate the vulnerability of SEDORE with the same threat model outlined in [18] by providing a concrete attack. Before explaining the attack, we propose the notion of universal forged token in 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 to database of $\mathcal{V}$ without authorized token from $\mathcal{V}$. We define an universal forged token as $\mathsf{uft}_{(\mathcal{V}),h_2} = (h_2^{a^{-1}_{(\mathcal{V})}}, h_2^{b_{(\mathcal{V})}})$ in our attack. We show the overall description of the attack method in Figure 3 and introduce it as follows:

1) The user $\mathcal{V}$ creates authorization token $\mathsf{tok}_{(\mathcal{M}\to\mathcal{V})}$ by using type-2 DERE.Token algorithm and sends it to user $\mathcal{M}$ as below:
$$\mathsf{tok}_{(\mathcal{M}\to\mathcal{V})} = \left(F\left(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}}\right)^{a^{-1}_{(\mathcal{V})}}, F\left(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}}\right)^{b_{(\mathcal{V})}}\right)$$

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:
$$\begin{aligned}\mathsf{uft}_{(\mathcal{V}),h_2} &= \mathsf{tok}^r_{(\mathcal{M}\to\mathcal{V})} \\ &= \left(\left(F(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}})^r\right)^{a^{-1}_{(\mathcal{V})}}, \left(F(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}})^r\right)^{b_{(\mathcal{V})}}\right)\end{aligned}$$
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 $\mathcal{A}$. And it is intractable to find a secret key of $\mathcal{M}$ by the cryptographic hash function F. For this reason, $\mathcal{M}$ may help adversary $\mathcal{A}$ without concern about leaking $\mathcal{M}$'s secret.

3) When $\mathcal{A}$ receives $\mathsf{uft}_{(\mathcal{V}),h_2}$ and h_2, she samples her secret key $\mathsf{sk}_{(\mathcal{A})} = (a_{(\mathcal{A})}, b_{(\mathcal{A})}) \xleftarrow{\$} \mathbb{Z}_p^2$ and then computes the counterpart forged token $\mathsf{uft}_{(\mathcal{A}),h_2}$ as follows:
$$\mathsf{uft}_{(\mathcal{A}),h_2} = (h_2^{a^{-1}_{(\mathcal{A})}}, h_2^{b_{(\mathcal{A})}})$$
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.
$$\begin{aligned}d_0 &= \frac{e(c^0_{(\mathcal{V})}, \mathsf{uft}^0_{(\mathcal{V}),h_2})}{e(c^1_{(\mathcal{V})}, \mathsf{uft}^1_{(\mathcal{V}),h_2})} = \frac{e((g_1^{b_{(\mathcal{V})}r_{(\mathcal{V})}}H(m))^{a_{(\mathcal{V})}}, h_2^{a^{-1}_{(\mathcal{V})}})}{e(g_1^{r_{(\mathcal{V})}}, h_2^{b_{(\mathcal{V})}})} \\ &= \frac{e(g_1^{b_{(\mathcal{V})}r_{(\mathcal{V})}}H(m), h_2)}{e(g_1^{b_{(\mathcal{V})}r_{(\mathcal{V})}}, h_2)} = e(H(m), h_2)\end{aligned}$$
$$\begin{aligned}d_1 &= \frac{e(c^0_{(\mathcal{A})}, \mathsf{uft}^0_{(\mathcal{A}),h_2})}{e(c^1_{(\mathcal{A})}, \mathsf{uft}^1_{(\mathcal{A}),h_2})} = \frac{e((g_1^{b_{(\mathcal{A})}r_{(\mathcal{A})}}H(m))^{a_{(\mathcal{A})}}, h_2^{a^{-1}_{(\mathcal{A})}})}{e(g_1^{r_{(\mathcal{A})}}, h_2^{b_{(\mathcal{A})}})} \\ &= \frac{e(g_1^{b_{(\mathcal{A})}r_{(\mathcal{A})}}H(m), h_2)}{e(g_1^{b_{(\mathcal{A})}r_{(\mathcal{A})}}, h_2)} = e(H(m), h_2)\end{aligned}$$
In other words, $\mathcal{A}$ can be identified equally 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 DORE and efficient DORE in a similar way. We suggest the detailed attack senario for efficient DORE in the Appendix A.

Limitation of SEDORE. Hahn et al. proposed SEDORE [18] to prevent token forging attacks against colluding users $\mathcal{M}$ and $\mathcal{A}$. Concretely, token generation algorithm in [18] uses a cryptographic hash function. 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 requirement for query to $\mathcal{V}$'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 token.

One of the main reasons is that the Test algorithm does not check who generates tokens; $\mathsf{tok}_{(\mathcal{A}\to\mathcal{V})}$ should not be generated except $\mathcal{V}$. For this reason, before running the test algorithm, the server should check tokens whether they are forged or not.

8 VERIFIABLE DORE (VDORE)

In [27], the authors provided security proof for the DORE scheme in the generic group model (GGM). Following the DORE construction, we give an augmented DORE scheme, called VDORE, whose security is proven under GGM.

To provide token unforgeability, we introduce verifiable delegatable equality-revealing encoding (VDERE) and verifiable delegatable order-revealing encryption (VDORE). Both VDERE and VDORE schemes additionally contain a token verification algorithm, which ensures that the token $\mathsf{tok}_{(u\to v)}$ is generated exclusively by the user v.

Therefore, before introducing our new VDORE, we provide security definitions and basic cryptographic tools to construct it.

8.1 Security Definitions

Definition 8.1 (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:

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

Definition 8.2 (EUF-CMA). Let Sig = (Setup, Keygen, Sign, 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 $\mathsf{Adv}^{\mathsf{EUF-CMA}}[\mathcal{A}, \mathrm{Sig}]$ to the game in Figure 4 is $\mathsf{negl}(\lambda)$.

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

(1) **Setting Phase**: 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 C with chosen message m. Then, C generates signature $\sigma_m \leftarrow \mathsf{Sign}(\mathsf{pp}_{sig}, \mathsf{sk}_{sig}, m)$ and return it to $\mathcal{A}$. Additionally, 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 $m^* \notin \mathcal{M}$.

Figure 4: EUF-CMA Game

8.2 Cryptographic Tools

8.2.1 Schnorr Signature Scheme [42]. To achieve token unforgeability, we adopt a digital signature scheme based on a discrete logarithm setting, the Schnorr signature scheme, which is used to construct ECDSA. Schnorr signature SSig consists of four algorithms SSig = (Setup, Keygen, Sign, Verify) as follows:

- $\mathsf{pp}_{sig} \leftarrow \mathsf{SSig.Setup}(\lambda)$: 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 $\mathbb{G}$. $T : \{0,1\}^* \to \mathbb{Z}_p$ is a hash function.
- $(\mathsf{vk}_{sig}, \mathsf{sk}_{sig}) \leftarrow \mathsf{SSig.Keygen}(\mathsf{pp}_{sig})$: 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 $\mathsf{vk}_{sig} = A = g^a$.
- $\sigma \leftarrow \mathsf{SSig.Sign}(\mathsf{pp}_{sig}, \mathsf{sk}_{sig}, m)$: 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 $\sigma = (R, s) \in \mathbb{G} \times \mathbb{Z}_p$
- $0\backslash 1 \leftarrow \mathsf{SSig.Verify}(\mathsf{pp}_{sig}, \mathsf{vk}_{sig}, m, \sigma)$: It takes public parameter pp_{sig}, the verifying key $\mathsf{vk}_{sig} = A$, message m, and signature $\sigma = (R, s)$. If $g^s = R \cdot A^{T(R\|m)}$ it returns 1; otherwise, it returns 0.

Theorem 8.3 (Schnorr Signature [37]). *If T is modeled as a random oracle, the Schnorr signature scheme* SSig *is existentially unforgeable under chosen-message attacks (EUF-CMA) under discrete logarithm (DL) assumption.*

8.2.2 Generic Group Model. A generic group model (GGM) is an idealized model for a group whose operations are carried out by making oracle queries [32, 44]. The GGM is designed to capture the behavior of general algorithms that operate independently of any particular group descriptions. Specifically, we consider the bilinear GGM, which additionally simulates a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ as proposed in [21]. The bilinear GGM is defined by the following:

Definition 8.4 (Bilinear Generic Group Algorithm [21, 44]). A bilinear generic group algorithm $\mathcal{A}$ is an algorithm that can access bilinear generic group oracle O_{BL} to treat group operation. The bilinear generic group oracle runs as follows in Figure 5.

8.3 VDERE

Before introducing the VDORE scheme, we define the VDERE scheme based on the DERE scheme in [27] with a token verification algorithm. VDERE scheme consists of six algorithms (Setup, Keygen, Enc, Token, Test, Vfy). We describe the algorithms as follows:

- $\mathsf{pp} \leftarrow \mathsf{VDERE.Setup}(1^\lambda)$: It takes a security parameter 1^λ as input and returns a public parameter pp.
- $(\mathsf{pk}, \mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{VDERE.Keygen}(\mathsf{pp})$: 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.

Generic Group Oracle O_{BL}

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

Figure 5: Bilinear Generic Group Oracle

Token Forging game

$\mathcal{A}(1^\lambda) \to (\tilde{\text{vk}}_{(\mathcal{A})}, \tilde{\text{tok}}_{(C\to\mathcal{A})}, \tilde{\text{tok}}_{(\mathcal{A}\to C)})$

(1) **Setting Phase**: C runs setup algorithm $\text{pp} \leftarrow \text{Setup}(1^\lambda)$ and key generation algorithm $(\text{sk}_{(C)}, \text{vk}_{(C)}, \text{pk}_{(C)}) \leftarrow \text{Keygen}(\text{pp})$. And then sends $(\text{pp}, \text{vk}_{(C)}, \text{pk}_{(C)})$ to $\mathcal{A}$.
(2) **Query Phase**: $\mathcal{A}$ can query to C:
  (a) **Key Query**: $\mathcal{A}$ sends a query with index i. If $(i, \text{pk}_{(i)}, \text{vk}_{(i)}) \in S_{\text{key}}$, then output $(i, \text{pk}_{(i)}, \text{vk}_{(i)})$. Otherwise, it runs $(\text{sk}, \text{pk}, \text{tk}) \leftarrow \text{Keygen}(\text{pp})$. And it returns (pk, tk) and adds the tuple $(i, \text{pk}, \text{tk})$ to key query set S_{key}
  (b) **Token Query**: If $\mathcal{A}$ sends a query with keys $\text{pk}_{(\mathcal{A})}$, C generates an authorized token $\text{tok}_{(\mathcal{A}\to C)} \leftarrow \text{Token}(\text{pp}, \text{pk}_{(\mathcal{A})}, \text{sk}_{(C)})$ and then sends $\text{tok}_{(\mathcal{A}\to C)}$ to $\mathcal{A}$ and adds $\text{tok}_{(\mathcal{A}\to C)}$ 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 $(\tilde{\text{vk}}_{(\mathcal{A})}, \tilde{\text{tok}}_{(C\to\mathcal{A})}, \tilde{\text{tok}}_{(\mathcal{A}\to C)})$. The $\mathcal{A}$ wins if $\text{Vfy}(\text{pp}, \text{vk}_{(C)}, \tilde{\text{vk}}_{(\mathcal{A})}, \tilde{\text{tok}}_{(\mathcal{A}\to C)}, \tilde{\text{tok}}_{(C\to\mathcal{A})}) = 1$ and $\tilde{\text{tok}}_{(\mathcal{A}\to C)} \notin S_{\text{tok}}$.

Figure 6: Token forging Game

- $\text{ct}_{(u)} \leftarrow \text{VDERE.Enc}(\text{pp}, m_{(u)}, \text{sk}_{(u)})$: It takes numerical data $m_{(u)}$ and secret key $\text{sk}_{(u)}$ for user u and returns ciphertext $\text{ct}_{(u)}$ for u.
- $\text{tok}_{(v\to u)} \leftarrow \text{VDERE.Token}(\text{pp}, \text{pk}_{(v)}, \text{sk}_{(u)})$: This algorithm takes the v's public key $\text{pk}_{(v)}$ and the u's secret key $\text{sk}_{(u)}$, as input and returns token $\text{tok}_{(v\to u)}$ authorized by u.
- $0\backslash 1 \leftarrow \text{VDERE.Test}(\text{pp}, \text{ct}_{(u)}, \text{ct}_{(v)}, \text{tok}_{(v\to u)}, \text{tok}_{(u\to v)})$: It takes the two ciphertext $\text{ct}_{(u)}$ and $\text{ct}_{(v)}$, and two tokens $\text{tok}_{(v\to u)}$ and $\text{tok}_{(u\to v)}$ as input and returns 1(accept) or 0(reject). If the plaintext of $\text{ct}_{(u)}$ is equal to those of $\text{ct}_{(v)}$, it returns 1; otherwise, it returns 0.
- $0\backslash 1 \leftarrow \text{VDERE.Vfy}(\text{pp}, \text{vk}_{(u)}, \text{vk}_{(v)}, \text{tok}_{(v\to u)}, \text{tok}_{(u\to v)})$: It takes the two verification keys $\text{vk}_{(u)}$ and $\text{vk}_{(v)}$, and two tokens $\text{tok}_{(v\to u)}$ and $\text{tok}_{(u\to v)}$ as input. If both tokens $\text{tok}_{(v\to u)}$ and $\text{tok}_{(u\to v)}$ go through verification, it returns 1 (accept); otherwise, it returns 0 (reject).

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

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

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

Data Privacy. The data privacy of VDERE ensures that the ciphertexts ct generated by the Enc algorithm do not leak information except order information. In other words, the Test algorithm only gives order information between queried ciphertexts. We say that VDERE scheme provides data privacy if Enc algorithm satisfies indistinguishability of encodings under authority delegation and distinct chosen-plaintext attack (IND-AD-DCPA) [27].

Token Unforgeability. To prevent the token forging attack, we provide a Vfy algorithm. Vfy algorithm 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 6.

Especially, in the GGM, $\mathcal{A}$ only gets public key $\text{pk}_{(i)} \in \mathbb{G}_2$ from the key query, and cannot generate the public key itself. In this case, C roles GGM oracle for the key generation.

Let define the $\mathcal{A}$'s advantage to the token forging game $\text{Adv}^{\text{TF}}[\mathcal{A}, \text{VDERE}]$, which is a probability that $\mathcal{A}$ wins the token forging game under the VDERE scheme.

Definition 8.5 (Token Unforgeability). Let (Setup, Keygen, Enc, Token ,Test, Vfy) be a VDERE (or VDORE) scheme. We say that VDERE (or VDORE) is token unforgeability if for any PPT adversary $\mathcal{A}$ against token forging game in Figure 6, the $\mathcal{A}$'s advantage to the game $\text{Adv}^{\text{TF}}[\mathcal{A}, \text{VDERE}]$ (or $\text{Adv}^{\text{TF}}[\mathcal{A}, \text{VDORE}]$) is less than $\text{negl}(\lambda)$.

Token forging game and attack scenario. We construct an attack game for forging tokens in the above paragraph. The key $(\text{pk}_{(C)}, \text{vk}_{(C)})$ which C sends represents the key of $\mathcal{V}$. Note that other users can know a public key $\text{pk}_{(\mathcal{V})}$ and verification key $\text{vk}_{(\mathcal{V})}$ 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. We describe the token forging game in Figure 6.

VDERE based on Bilinear Setting. In this paragraph, we introduce our novel VDERE schemes based on bilinear setting. Based on the DERE scheme in [27], we adopt Schnorr's signature scheme and construct our novel token verifier algorithm.

Let $H : \{0,1\}^* \to \mathbb{G}_1$ and $T : \{0,1\}^* \to \mathbb{Z}_p$ be a hash functions. Our VDERE scheme applies the Schnorr signature scheme as a subroutine. We describe our VDERE scheme in Figure 7. We use the frame-box symbol to emphasize additional parts compared with the original DERE [27].

Theorem 8.6. *Assuming H and T are modeled as a random oracle. Then, pairing-based VDERE in Figure 7 satisfies the correctness, data privacy, and token unforgeability under DL assumption with a generic group model.*

Proof. *(Correctness)* The correctness of VDERE holds in the similar way of [18, 27]. Concretely, for a valid token and ciphertext, the intermediate value d_0 and d_1 of Test should be equal by the bilinearility of the pairing operation. Additionally, from the correctness of the Schnorr signature, Vfy should be output 1 so that VDERE satisfies correctness.
(Data privacy) Since our underlying encryption algorithm is the same as DERE [27] and DERE achieves IND-AD-DCPA under a generic group model with random oracle H, our VDERE scheme provides data privacy due to achieving IND-AD-DCPA. Therefore, the VDERE scheme guarantees that the ciphertext does not leak any information without the equality test.
(Token Unforgeability) Because the Vfy algorithm contains sign verification, VDERE satisfies token unforgeability by EUF-CMA of Schnorr signature scheme, which holds under DL assumption with random oracle T.

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 $\mathsf{tok}_{(i\to C)}$ without corresponding key query for $\mathsf{pk}_{(i)}$. Therefore, we claim that VDERE satisfies token unforgeability if the underlying signature scheme satisfies EUF-CMA.

Let $\mathcal{A}$ and $\mathcal{B}$ be adversaries against the token forging game (Figure 6) and EUF-CMA game (Figure 4) 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 O_{BL} in Figure 5.

Simulation C against $\mathcal{A}$. As we mentioned, $\mathcal{B}$ should simulate challenger C for the token forging game in Figure 6. We describe how to simulate C in Figure 8.

- $\mathsf{pp} \leftarrow \mathsf{VDERE.Setup}(1^\lambda)$: This algorithm takes the security parameter 1^λ as input and returns the public parameter $\mathsf{pp} = (\langle p, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e\rangle, H, T)$. Specially, we set $\mathsf{pp}_{sig} := (\langle p, \mathbb{G}_1, g_1\rangle, T)$.
- $(\mathsf{pk}, \boxed{\mathsf{vk}}, \mathsf{sk}) \leftarrow \mathsf{VDERE.Keygen}(\mathsf{pp})$: 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 $\mathsf{vk}_{sig} := \mathsf{vk}$.
- $\mathsf{ct} \leftarrow \mathsf{VDERE.Enc}(\mathsf{pp}, m, \mathsf{sk})$: 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 $ct = (c^0, c^1)$.
- $\mathsf{tok}_{(u\to v)} \leftarrow \mathsf{VDERE.Token}(\mathsf{pp}, \mathsf{pk}_{(u)}, \mathsf{sk}_{(v)})$: 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:
$$t^0_{(u\to v)} = \mathsf{pk}_{(u)}, \quad t^1_{(u\to v)} = \mathsf{pk}_{(u)}^{a_{(v)} b_{(v)}}$$
$$\sigma_v \leftarrow \mathsf{SSig.Sign}(\mathsf{pp}_{sig}, \mathsf{sk}_{sig,(v)}, (t^0_{(u\to v)}, t^1_{(u\to v)}))$$
- $0\backslash 1 \leftarrow \mathsf{VDERE.Test}(\mathsf{pp}, \mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: 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.
- $0\backslash 1 \leftarrow \mathsf{VDERE.Vfy}(\mathsf{pp}, \mathsf{vk}_{(u)}, \mathsf{vk}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: 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. And then, it checks the following:
(1) $1 = \mathsf{SSig.Vfy}(\mathsf{pp}_{sig}, \mathsf{vk}_{(v)}, (t^0_{(u\to v)}, t^1_{(u\to v)}), \sigma_v)$
(2) $1 = \mathsf{SSig.Vfy}(\mathsf{pp}_{sig}, \mathsf{vk}_{(u)}, (t^0_{(v\to u)}, t^1_{(v\to u)}), \sigma_u)$

Figure 7: Pairing based VDERE Scheme

In the setting phase, $\mathcal{B}$ generates $\mathsf{pk}_{(\mathcal{B})}$ using generic group oracle O_{BL} in Figure 5. And then, sends verificatino key $\mathsf{vk}_{(\mathcal{B})} = \mathsf{vk}_{sig}$ received by C_{sig} and public key $\mathsf{pk}_{(\mathcal{B})}$ of $\mathcal{B}$ to $\mathcal{A}$.

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

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

(1) **Setting Phase**: C_{sig} runs setup algorithm $\mathsf{pp}_{sig} = \mathsf{pp}_{sig} := (\langle p, [\mathbb{G}_1, g_1]_{O_{BL}} \rangle, T) \leftarrow \mathsf{Setup}(1^\lambda)$ 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 C against $\mathcal{A}$**: $\mathcal{B}$ roles token forging game challenger C against $\mathcal{A}$.

(a) **Setting Phase**: $\mathcal{B}$ construct $\mathsf{pp} = (\langle p, [\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, \mathbb{G}_T, e]_{O_{BL}} \rangle, H, T)$ using pp_{sig}. And then $\mathcal{B}$ samples $a_{(\mathcal{B})} \xleftarrow{\$} \mathbb{Z}_p$ and access generic group oracle O_{BL} to get a public key $\mathsf{pk}_{(\mathcal{B})} \leftarrow O_{BL}(a_{(\mathcal{B})}, 0, \times_2)$. 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 6. If $(i, \mathsf{pk}_{(i)}, \mathsf{vk}_{(i)}) \in S_{\mathsf{key}}$, then output $(i, \mathsf{pk}_{(i)}, \mathsf{vk}_{(i)})$. Otherwise, it runs $(\mathsf{sk}, \mathsf{pk}, \mathsf{tk}) \leftarrow \mathsf{Keygen}^{O_{BL}}(\mathsf{pp})$. 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 O_{BL} to get random string $t^1_{(\mathcal{A}\to\mathcal{B})} \leftarrow O_{BL}(a_{(\mathcal{A})} a_{(\mathcal{B})}, 0, \times_2)$. 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 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 $(\tilde{\mathsf{vk}}_{(\mathcal{A})}, \tilde{\mathsf{tok}}_{(\mathcal{B}\to\mathcal{A})}, \tilde{\mathsf{tok}}_{(\mathcal{A}\to\mathcal{B})})$ from $\mathcal{A}$.

(3) **Challenge Phase**: $\mathcal{B}$ answers $\tilde{\mathsf{tok}}_{(\mathcal{A}\to\mathcal{B})} = \left((\tilde{t}^0_{(\mathcal{A}\to\mathcal{B})}, \tilde{t}^1_{(\mathcal{A}\to\mathcal{B})}), \tilde{\sigma_{\mathcal{B}}}\right) = (\tilde{m}, \tilde{\sigma})$ to C_{sig}.

Figure 8: Construct $\mathcal{B}$ using $\mathcal{A}$

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 O_{BL}. After then, $\mathcal{B}$ gets signature $\sigma_{\mathcal{B}}$ from signature query to C_{sig}. Therefore $\mathcal{B}$ can response the token query by getting $\mathsf{tok}_{(\mathcal{A}\to C)}$ from O_{BL}.

Finally, $\mathcal{B}$ receives the forged token from $\mathcal{A}$ and then uses it to win the EUF-CMA game (Figure 4).

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 6 is indistinguishable from the simulated game by $\mathcal{B}$ in Figure 8.

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

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

where $\mathsf{Adv}^{\mathsf{TF}}[\mathcal{A}, \mathsf{VDERE}]$ is $\mathcal{A}$'s advantage to the token forging game (Figure 6) and $\mathsf{Adv}^{\mathsf{EUF-CMA}}[\mathcal{B}, \mathsf{SSig}]$ is $\mathcal{B}$'s advantage to the EUF-CMA game Figure 4 under the security parameter λ.

By Theorem 8.3, $\mathsf{Adv}^{\mathsf{EUF-CMA}}[\mathcal{B}, \mathsf{SSig}]$ 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. □

8.4 VDORE

In this subsection, we show the process of VDORE for user u who is the data owner, and user v. VDORE is equivalent to VDERE except test algorithm. The test algorithm outputs an order result between two ciphertexts. VDORE consists of six algorithms as follows:

- $\mathsf{pp} \leftarrow \mathsf{VDORE.Setup}(1^\lambda)$: It takes a security parameter 1^λ as input and returns a public parameter pp.
- $(\mathsf{pk}, \mathsf{tk}, \mathsf{sk}) \leftarrow \mathsf{VDORE.Keygen}(\mathsf{pp})$: It takes a public parameter as input and returns a key tuple of public key, verification key, and secret key, $(\mathsf{pk}, \mathsf{tk}, \mathsf{sk})$. The verification key is used to verify the validity of tokens.
- $\mathsf{ct}_{(u)} \leftarrow \mathsf{VDORE.Enc}(\mathsf{pp}, m_{(u)}, \mathsf{sk}_{(u)})$: It takes numerical data $m_{(u)}$ and secret key $\mathsf{sk}_{(u)}$ for user u and returns ciphertext $\mathsf{ct}_{(u)}$ for u.
- $\mathsf{tok}_{(v\to u)} \leftarrow \mathsf{VDORE.Token}(\mathsf{pp}, \mathsf{pk}_{(v)}, \mathsf{vk}_{(v)}, \mathsf{sk}_{(u)})$: This algorithm takes the v's public key $\mathsf{pk}_{(v)}$ and verification key $\mathsf{vk}_{(v)}$, and the u's secret key $\mathsf{sk}_{(u)}$, as input and returns token $\mathsf{tok}_{(u\to v)}$ authorized by 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 $\mathsf{ct}_{(v)}$, and two tokens $\mathsf{tok}_{(v\to u)}$ and $\mathsf{tok}_{(u\to v)}$ as input. If the u's plaintext is larger than that of v, it returns 1; else if the plaintext of v is larger than that of u, it returns -1; otherwise, it returns 0.
- $0\backslash 1 \leftarrow \mathsf{VDORE.Vfy}(\mathsf{pp}, \mathsf{vk}_{(u)}, \mathsf{vk}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: It takes the two verification key $\mathsf{vk}_{(u)}$ 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 verification, it returns 1; otherwise, it returns 0.

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:

- $\mathsf{Vfy}(\mathsf{pp}, \mathsf{vk}_{(u)}, \mathsf{vk}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)}) = 1$
- $\mathsf{Test}(\mathsf{pp}, \mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)}) = res$
  - 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) [27].

Token Unforgeability. The verifier algorithm of VDORE is identical to VDERE so that the security definition of VDORE is equivalent to those of VDERE.

Construct VDORE using VDERE. Following the construction of DORE from DERE [18, 27], we first construct VDERE scheme and then convert VDORE from it. The Setup, Keygen, Token, and Vfy algorithms of VDORE follows those of VDERE in Figure 7. The Enc is defined by bit-wise VDERE.Enc encryption and Test is defined by iterative running of VDERE.Test.

- $\mathsf{ct} \leftarrow \mathsf{VDORE.Enc}(\mathsf{pp}, m, \mathsf{sk})$: This algorithm takes a message $m \in \{0,1\}^*$ and sk as input and returns a ciphertext ct as follows:

  $$\epsilon(m_i, a) = (i, m_1 m_2 \dots m_i || 0^{n-i}, a),$$

  where $a \in \{0,1,2\}$. The algorithm encrypts $\epsilon(m_i, a)$ using bitwise encoding in the following manner.
  If $m_i = 0$, it computes ciphertexts $\mathsf{ct}[i] = (\mathsf{ct}[i,0], \mathsf{ct}[i,1])$ as follows:

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

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

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

  Finally, this algorithm returns $\mathsf{ct} = (\mathsf{ct}[1], \dots \mathsf{ct}[n])$.
- $res \leftarrow \mathsf{VDORE.Test}(\mathsf{pp}, \mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)})$: This algorithm takes $\mathsf{ct}_{(u)}, \mathsf{ct}_{(v)}, \mathsf{tok}_{(v\to u)}, \mathsf{tok}_{(u\to v)}$ as input. Test algorithm runs VDERE.Test iteratively. For $i = 1$ to n, the algorithm follows it:
  (1) If $i = n+1$, then return 0
  (2) Else if $res_u^i = 1$, then returns 1
  (3) Else if $res_v^i = 1$, then return -1
  (4) Else $i \leftarrow i + 1$
  , where res_u^i and res_v^i are VDERE.Test results of $(\mathsf{ct}_{(u)}[i,0], \mathsf{ct}_{(v)}[i,1])$ and $(\mathsf{ct}_{(u)}[i,1], \mathsf{ct}_{(v)}[i,0])$ respectively.

Theorem 8.7. *Assuming H and T are modeled as a random oracle. Then the above* VDORE *satisfies the correctness, data privacy, and token unforgeability under DL assumption with a generic group model.*

Proof. The security properties: correctness, data privacy, and token unforgeability are inherited from the underlying VDERE scheme. By Theorem 8.6, VDORE also satisfies these properties.
(Correctness) Since VDORE.Test consists of several VDERE.Test algorithms, and our base VDERE scheme in Figure 7 satisfies correctness, it follows that VDORE satisfies correctness as well.
(Data Privacy) Regarding data privacy, as demonstrated in Section 8.3, our VDERE scheme achieves IND-AD-DCPA security. Since the DORE scheme, based on an IND-AD-DCPA secure DERE, satisfies indistinguishability under IND-OCPA [27], our VDORE is also IND-OCPA secure. Therefore, VDORE achieves data privacy.

Table 1: Security feature comparison.

	DORE [27]	SEDORE [18]	VDORE
Data privacy	✓	✓	✓
Token unforgeability	×	×	✓

✓ indicates satisfaction of security features.
× indicates dissatisfaction of security features.

(Token Unforgeability) For token unforgeability, the Vfy algorithm of VDORE is identical to VDERE. That means, to show token unforgeability of VDORE is equivalent with those of VDERE. Therefore, by Theorem 8.6, token unforgeability holds on VDORE. □

Security Comparison. In Table 1, we show the security feature comparison for three ORE methods, DORE [27], SEDORE [18], and our VDORE. As shown in the table, our VDORE only features token unforgeability. We showed this in Theorem 8.7.

9 EXPERIMENT

In this section, we present evaluations of our proposed VDORE scheme compared to DORE and SEDORE, which serve as benchmarks.

9.1 Experiment environments

We implement our VDORE and previous schemes in C language with a Linux desktop with a 5.20 GHz Intel i9-12900K CPU and 64GB RAM. Furthermore, We use the OpenSSL library for the hash function, the pairing-based cryptography library written in C for bilinear maps with MNT224 curve [30], and the GMP library for large integer arithmetic.

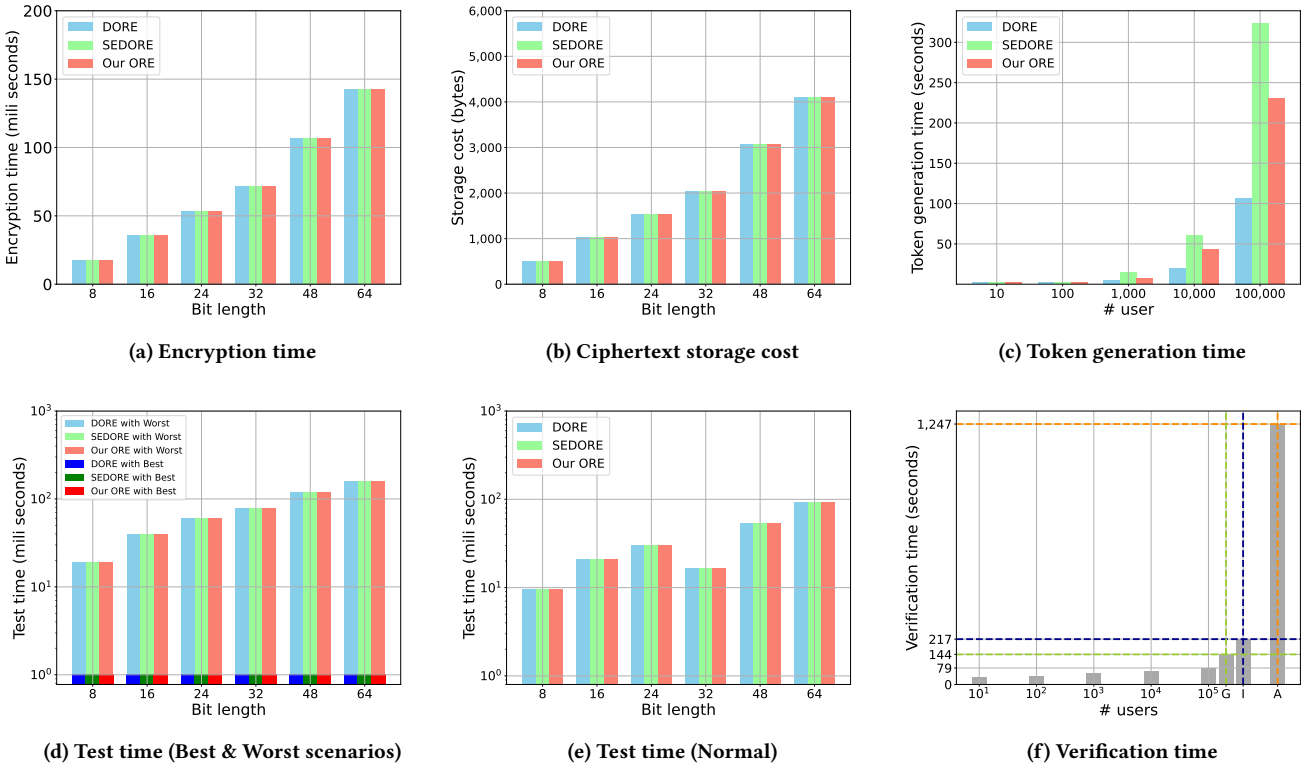
9.2 Dataset

We utilize the dataset [46] 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 [33]. We conduct experiments by extracting 5,000 data from each dataset.

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. Additionally, after randomly shuffling the extracted 5000 samples, we created two datasets, which were then utilized to experiment with the test algorithm.

9.3 Performance

We evaluate our proposed scheme VDORE along with existing schemes DORE and SEDORE across six categories: ciphertext size, encryption time, token generation time, test algorithm time, and verification time.



(a) Encryption time (b) Ciphertext storage cost (c) Token generation time

(d) Test time (Best & Worst scenarios) (e) Test time (Normal) (f) Verification time

Figure 9: The overall performance graphs for DORE, SEDORE, and our VDORE. In Figure 9f's x-axis, "G" indicates Google, "I" indicates IBM, and "A" indicates Amazon, respectively.

Table 2: A comparative analysis for n-bit comparison

	# elements		# group/pairings			
	(Encryption)	(Token)	(Encryption)	(Token)	(Test)	(Verification)
DORE [27]	$4n\|\mathbb{G}_1\|$	$2\|\mathbb{G}_2\|$	$6nE_{\mathbb{G}_1}$	$E_{\mathbb{G}_2}$	$4nP_{\mathbb{G}_T} + 4nP_{\mathbb{G}_T}$	×
SEDORE [18]	$4n\|\mathbb{G}_1\|$	$2\|\mathbb{G}_2\|$	$6nE_{\mathbb{G}_1}$	$3E_{\mathbb{G}_2}$	$4nP_{\mathbb{G}_T} + 4nP_{\mathbb{G}_T}$	×
VDORE (Ours)	$4n\|\mathbb{G}_1\|$	$1\|\mathbb{G}_1\| + 2\|\mathbb{G}_2\| + 1\|\mathbb{Z}_p\|$	$6nE_{\mathbb{G}_1}$	$E_{\mathbb{G}_1} + E_{\mathbb{G}_2}$	$4nP_{\mathbb{G}_T} + 4nP_{\mathbb{G}_T}$	$4E_{\mathbb{G}_1}$

× indicates the Verification algorithm is not available.

Comparative analysis. Before experimentally comparing each scheme, we show a theoretical cost analysis for each algorithm based on one bit. In Table 2, 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.

When generating tokens, our scheme consumes more storage compared to DORE and SEDORE. Because tokens in VDORE contain the Schnorr signature, which contains one $\mathbb{G}_1$ element and one $\mathbb{Z}_p$ element. However, since the token is not held by the data owner but is distributed to the client, it is considered an acceptable issue. When generating tokens, DORE assumes minimal overhead as it directly utilizes other users' public keys, whereas SEDORE and VDORE compute additional operations. DORE only needs one $\mathbb{G}_2$-exponentiation for generating $t^1_{(v\to u)}$, in contrast SEDORE needs three $\mathbb{G}_2$-exponentiation for generating $t^0_{(v\to u)}$ and $t^1_{(v\to u)}$. In VDORE, the token can be generated only one $\mathbb{G}_2$-exponentiation for generating $t^1_{(v\to u)}$ and one $\mathbb{G}_1$-exponentiation for generating signature σ_u. In conclusion, VDORE is about 2 times slower than DORE but 1.5 times faster than SEDORE. However, only VDORE provides token unforgeability.

Encryption time & Ciphertext size. In Figure 9a and 9b, we show the evaluation of encryption time and ciphertext storage for three ORE schemes. The x-axis of the graph represents the length of plaintexts, while the y-axis represents the time in mili seconds taken for encryption and the storage in bytes for ciphertext storage. As mentioned earlier in the comparative analysis, DORE, SEDORE, and VDORE all use the same encryption algorithm, resulting in identical encryption times and ciphertext sizes. Therefore, our method ensures enhanced privacy for the ciphertext without any degradation in latency.

Token generation time. This paragraph introduces the results related to the token generation time. 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 9c. Regarding token generation, SEDORE takes three times longer than DORE's token generation algorithm due to the additional 2 group operations on $\mathbb{G}_2$. Furthermore, we observe that our VDORE method is approximately 1.5 times faster than the existing SEDORE method. This is because VDORE requires one fewer group operations, and the remaining field operations do not significantly impact latency compared to group operations. Therefore, our method not only offers enhanced security compared to SEDORE but is also more efficient in token generation time.

Test algorithm time. To evaluate the test algorithm, we conduct two separate experiments. The first focuses on the best and worst-case scenarios, while the second utilizes the dataset mentioned earlier for overall computations. In the first experiment, 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.

In Figure 9d, we show the evaluation of the test algorithm for the best and worst scenarios. 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 9e, we show that the computational time falls within the range of Figure 9d. Additionally, 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 attributed to the division of the dataset during our experiments, and the order is determined closer to the most significant bit (MSB) in the dataset used at 32 bits.

Verification time. In this paragraph, we present the results of experiments on the verification algorithm. We conduct experiments based on the assumption that the number of users who need to perform token verification in the test operation is 10, 100, 1,000, 10,000, and 100,000. After that, to apply the experiments to a more realistic scenario, we also perform experiments on the number of employees at CSPs companies introduced in Section 2 with the assumption of issuing authorization tokens to employees within the company. Therefore, we conduct experiments on Google with 182,502 employees [3], Amazon with 1,608,000 employees [2], and IBM with 282,200 employees [4].

In Figure 9f, it is observed that the verification could be conducted within almost 0 to 79 seconds for users ranging from the 10th to the 100,000th. When applying to real companies, Google, IBM, and Amazon, it takes around 144 seconds, 217 seconds, and 1,247 seconds (about 20.8 mins), respectively. Although it took approximately 1,247 seconds for Amazon, our proposed technique proves to be highly practical while ensuring enhanced security, as the average time per person was only about 0.77 milliseconds.

10 RELATED WORKS

The concept of order-preserving encryption (OPE) was initially introduced by Agrawal et al. [6]. Subsequently, Boldyreva et al. [8] 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. To address this, Popa et al. [38] proposed the first IND-OCPA OPE scheme, which employs stateful and interactive techniques utilizing the B-tree structure. Additionally, Kerschbaum et al. [23] introduced Frequency-hiding OPE (FH-OPE), which conceals the frequency information of plaintexts to thwart various inference attacks [12, 28, 34]. Nevertheless, FH-OPE does not offer complete protection against such attacks which results in significant data overhead.

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. The concept of ORE was first introduced by Boneh et al. [9], implemented using multilinear maps [41]. To improve efficiency, Chenette et al. [13] suggested a practical ORE scheme. Nonetheless, this scheme leaks the most significant different bit (msdb) and thus lacks sufficient security guarantees. Subsequently, Lewi et al. [26] proposed an enhanced ORE scheme that only leaks the most significant different block. After that, Cash et al. [10] introduced parameter-hiding ORE (pORE) with probabilistic characteristics for the single-user scenario, revealing only the equality pattern of msdb. Peng et al. [36] suggested the ORE primitive which reduces the computation cost that that of pORE with the same security level.

Following this, Lv et al. [29] suggested m-ORE, which reduces computational overhead from $O(n^2)$ pairings in [10] to $O(n)$ pairings for the comparison stage and supports multi-user environments. After that, Qiao et al. [40] introduced the vulnerability that uses the stealthy property in m-ORE and suggested om-ORE as a countermeasure. Park et al. [35] also identified vulnerabilities in m-ORE when the attacker has a secret key and can eavesdrop on other clients' queries. From this, they proposed msq-ORE as a countermeasure. Furthermore, existing ORE techniques operated

by distributing keys in a multi-user environment, which exhibited weaknesses from a practical standpoint. To address this, Li et al. [27] proposed Delegatable ORE (DORE), enabling functionality even with data encrypted using different keys, provided authorization tokens are utilized.

However, Hahn et al. [18] presented a rational attack model for DORE and threatening forged token attacks, leading to the proposal of SEDORE to counteract such attacks. Unfortunately, we discover the same vulnerability in SEDORE by applying the threat model presented in Hahn et al. [18], and as a countermeasure, we introduce VDORE in the paper. Unfortunately, the various attacks [14, 16, 19, 20] suggested that even in ideal ORE schemes, significant information can be leaked despite only revealing the order. However, these attack techniques rely on the assumption that the adversary has prior knowledge of the distribution of the database. Therefore, in this paper, we do not consider those attacks.

11 CONCLUSION

In our paper, we demonstrate the vulnerability of DORE and SEDORE within the same threat model suggested by [18]. Our attack technique, while providing additional information beyond the previously outlined attack process, remains a menacing and practical threat without violating the attack model presented in [18]. Thus, to address these security concerns, we propose Verifiable Delegatable Order-Revealing Encryption (VDORE), which ensures security against universe token reusability attacks by utilizing the Schnorr signature scheme. Additionally, 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 compared to that of SEDORE.

REFERENCES

[1] 2016. Biggest Ad Fraud. https://www.forbes.com/sites/thomasbrewster/2016/12/20/methbot-biggest-ad-fraud-busted/?sh=1605441b4899.

[2] 2023. Amazon Employee. https://explodingtopics.com/blog/amazon-employees.

[3] 2024. Google Employee. https://seo.ai/blog/how-many-people-work-at-google.

[4] 2024. IBM Employee. https://stockanalysis.com/stocks/ibm/employees/.

[5] Rashmi Agrawal, Leo De Castro, Chiraag Juvekar, Anantha Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. 2023. Mad: Memory-aware design techniques for accelerating fully homomorphic encryption. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 685–697.

[6] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. 563–574.

[7] Robin Berger, Felix Dörre, and Alexander Koch. 2024. Two-Party Decision Tree Training from Updatable Order-Revealing Encryption. In *International Conference on Applied Cryptography and Network Security*. Springer, 288–317.

[8] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'neill. 2009. 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*. Springer, 224–241.

[9] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. 2015. 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*. Springer, 563–594.

[10] David Cash, Feng-Hao Liu, Adam O'Neill, Mark Zhandry, and Cong Zhang. 2018. 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*. Springer, 181–210.

[11] Stéphanie Challita, Faiez Zalila, Christophe Gourdin, and Philippe Merle. 2018. A precise model for google cloud platform. In *2018 IEEE international conference on cloud engineering (IC2E)*. IEEE, 177–183.

[12] Xiao Chen, Chaoran Li, Derui Wang, Sheng Wen, Jun Zhang, Surya Nepal, Yang Xiang, and Kui Ren. 2019. Android HIV: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security* 15 (2019), 987–1001.

[13] Nathan Chenette, Kevin Lewi, Stephen A Weis, and David J Wu. 2016. 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*. Springer, 474–493.

[14] F Betül Durak, Thomas M DuBuisson, and David Cash. 2016. What else is revealed by order-revealing encryption?. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1155–1166.

[15] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2018. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 315–331.

[16] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. 2017. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 655–672.

[17] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. 2019. Encrypted databases: New volume attacks against range queries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 361–378.

[18] Changhee Hahn and Junbeom Hur. 2022. Delegatable Order-Revealing Encryption for Reliable Cross-Database Query. *IEEE Transactions on Services Computing* (2022).

[19] Bijit Hore, Sharad Mehrotra, Mustafa Canim, and Murat Kantarcioglu. 2012. Secure multidimensional range queries over outsourced data. *The VLDB Journal* 21 (2012), 333–358.

[20] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access pattern disclosure on searchable encryption: ramification, attack and mitigation.. In *Ndss*, Vol. 20. Citeseer, 12.

[21] Tibor Jager and Andy Rupp. 2010. 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*. Springer, 539–556.

[22] Seny Kamara and Tarik Moataz. 2019. Computationally volume-hiding structured encryption. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part II 38*. Springer, 183–213.

[23] Florian Kerschbaum. 2015. Frequency-hiding order-preserving encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 656–667.

[24] SO Kuyoro, F Ibikunle, and O Awodele. 2011. Cloud computing security issues and challenges. *International Journal of Computer Networks (IJCN)* 3, 5 (2011), 247–255.

[25] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2018. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 297–314.

[26] Kevin Lewi and David J Wu. 2016. Order-revealing encryption: New constructions, applications, and lower bounds. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1167–1178.

[27] Yuan Li, Hongbing Wang, and Yunlei Zhao. 2019. Delegatable order-revealing encryption. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 134–147.

[28] Guanjun Lin, Sheng Wen, Qing-Long Han, Jun Zhang, and Yang Xiang. 2020. Software vulnerability detection using deep neural networks: a survey. *Proc. IEEE* 108, 10 (2020), 1825–1848.

[29] Chunyang Lv, Jianfeng Wang, Shi-Feng Sun, Yunling Wang, Saiyu Qi, and Xiaofeng Chen. 2021. 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*. Springer, 44–63.

[30] Ben Lynn. 2006. Pairing-based cryptography library. https://crypto.stanford.edu/pbc/.

[31] Sajee Mathew and J Varia. 2014. Overview of amazon web services. *Amazon Whitepapers* 105, 1 (2014), 22.

[32] Ueli Maurer. 2005. Abstract models of computation in cryptography. In *Cryptography and Coding: 10th IMA International Conference, Cirencester, UK, December 19-21, 2005. Proceedings 10*. Springer, 1–12.

[33] AAYUSH MISHRA. 2020. FAANG- Complete Stock Data. https://www.kaggle.com/datasets/aayushmishra1512/faang-complete-stock-data. (Accessed on 05/30/2024).

[34] Muhammad Naveed, Seny Kamara, and Charles V Wright. 2015. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 644–655.

[35] Jae Hwan Park, Zeinab Rezaeifar, and Changhee Hahn. 2024. Securing multi-client range queries over encrypted data. *Cluster Computing* (2024), 1–14.

[36] Cong Peng, Rongmao Chen, Yi Wang, Debiao He, and Xinyi Huang. 2024. Parameter-Hiding Order-Revealing Encryption Without Pairings. In *IACR International Conference on Public-Key Cryptography*. Springer, 227–256.
[37] David Pointcheval and Jacques Stern. 2000. Security arguments for digital signatures and blind signatures. *Journal of cryptology* 13 (2000), 361–396.
[38] Raluca Ada Popa, Frank H Li, and Nickolai Zeldovich. 2013. An ideal-security protocol for order-preserving encoding. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 463–477.
[39] Liudmyla Pryimenko. 2024. 7 Examples of Real-Life Data Breaches Caused by Insider Threats. https://www.ekransystem.com/en/blog/real-life-examples-insider-threat-caused-breaches.
[40] Hongyi Qiao, Cong Peng, Qi Feng, Min Luo, and Debiao He. 2024. Ciphertext Range Query Scheme Against Agent Transfer and Permission Extension Attacks for Cloud Computing. *IEEE Internet of Things Journal* (2024).
[41] Daniel S Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. 2016. POPE: Partial order preserving encoding. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1131–1142.
[42] Claus-Peter Schnorr. 1990. Efficient identification and signatures for smart cards. In *Advances in Cryptology—CRYPTO'89 Proceedings 9*. Springer, 239–252.
[43] Holger Schulze. 2021. cybersecurity-insiders.com/wp-content/uploads/2021/06/2021-Insider-Threat-Report-Gurucul-Final-dd8f5a75.pdf. https://www.cybersecurity-insiders.com/wp-content/uploads/2021/06/2021-Insider-Threat-Report-Gurucul-Final-dd8f5a75.pdf. (Accessed on 05/30/2024).
[44] Victor Shoup. 1997. 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*. Springer, 256–266.
[45] Emil Stefanov, Marten van Dijk, Elaine Shi, T-H Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2018. Path ORAM: an extremely simple oblivious RAM protocol. *Journal of the ACM (JACM)* 65, 4 (2018), 1–26.
[46] U.Nations. 2022. World Population Prospects - Population Division - United Nations. https://population.un.org/wpp/. (Accessed on 05/30/2024).
[47] Jingru Xu, Cong Peng, Rui Li, Jintao Fu, and Min Luo. 2024. An Efficient Delegatable Order-Revealing Encryption Scheme for Multi-User Range Queries. *IEEE Transactions on Cloud Computing* (2024).
[48] Yinghui Zhang, Robert H Deng, Shengmin Xu, Jianfei Sun, Qi Li, and Dong Zheng. 2020. Attribute-based encryption for cloud computing access control: A survey. *ACM Computing Surveys (CSUR)* 53, 4 (2020), 1–41.
[49] Yongmin Zhang, Xiaolong Lan, Ju Ren, and Lin Cai. 2020. Efficient computing resource sharing for mobile edge-cloud computing networks. *IEEE/ACM Transactions on Networking* 28, 3 (2020), 1227–1240.
[50] Jinzy Zhu, Xing Fang, Zhe Guo, Meng Hua Niu, Fan Cao, Shuang Yue, and Qin Yu Liu. 2009. IBM cloud computing powering a smarter planet. In *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1*. Springer, 621–625.

A UTRA FOR EFFICIENT DORE

In this section, we show how to adapt our UTRA method to Efficient DORE [47].

Firstly, we scrutinize the Efficient DORE. For this, we show the efficient DERE (EDERE) suggested by [47].

Efficient DERE. . It consists of five algorithms, EDERE.Setup, EDERE.Keygen, EDERE.Enc, EDERE.Tok, and EDERE.Test as below:

- $\mathsf{pp} \leftarrow \mathsf{EDERE.Setup}(1^\lambda)$: 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 receives a public parameter (pp) as input and returns a pair of public key and secret key (pk, 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 takes 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 $\mathsf{tok}_{(v\to u)}$. $\mathsf{tok}_{(v\to u)}$ consists 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 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
$$d_0 = e\left(\prod_{k=1}^{2}(t^k_{(v\to u)})^{c^k_{(u)}}, c^3_{(u)}\right).$$
$$d_1 = e\left(\prod_{k=1}^{2}(t^k_{(u\to v)})^{c^k_{(v)}}, c^3_{(v)}\right).$$
Finally, it compares d_0 and d_1 and returns 1 if $d_0 = d_1$ and 0 otherwise.

UTRA for EDERE. To demonstrate how EDORE is vulnerable to UTRA attacks, we use EDERE to illustrate this. The scenario is as follows:
1) The user $\mathcal{V}$ creates authorization token $\mathsf{tok}_{(\mathcal{M}\to\mathcal{V})}$ by using EDERE.Token algorithm and sends it to user $\mathcal{M}$ as below:
$$\mathsf{tok}_{(\mathcal{M}\to\mathcal{V})} = \left(F(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}})^{b_{(\mathcal{V})}}, \quad F(\mathsf{pk}_{(\mathcal{M})}^{a_{(\mathcal{V})}})^{b_{(\mathcal{V})}\xi_{(\mathcal{V})}}\right)$$
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 $\mathcal{A}$. And it is intractable to find a secret key of $\mathcal{M}$ by the cryptographic hash function F. For this reason, $\mathcal{M}$ may help adversary $\mathcal{A}$ without concern about leaking $\mathcal{M}$'s secret.

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.

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

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