Verifiable Delegated Set Intersection Operations on Outsourced Encrypted Data

Qingji Zheng and Shouhuai Xu

Abstract—We initiate the study of the following problem: Suppose Alice and Bob would like to outsource their encrypted private data sets to the cloud, and they also want to conduct the set intersection operation on their plaintext data sets. The straightforward solution for them is to download their outsourced ciphertexts, decrypt the ciphertexts locally, and then execute a commodity two-party set intersection protocol. Unfortunately, this solution is not practical.

We therefore motivate and introduce the novel notion of *Verifiable Delegated Set Intersection on outsourced encrypted data* (VDSI). The basic idea is to delegate the set intersection operation to the cloud, while (i) not giving the decryption capability to the cloud, and (ii) being able to hold the misbehaving cloud accountable. We formalize security properties of VDSI and present a construction. In our solution, the computational and communication costs on the users are linear to the size of the intersection set, meaning that the efficiency is optimal up to a constant factor.

Index Terms—Verifiable set intersection, delegated set intersection, outsourced encrypted data, verifiable outsourced computing

1 Introduction

Cloud computing allows users to outsource their data to the cloud, but the data privacy issue often makes them reluctant to do so. It is therefore natural to encrypt the outsourced data and delegate the heavy-duty computational tasks on the outsourced encrypted data to the cloud. This leads to a general question: How can the cloud execute the delegated functions on outsourced encrypted data, without being given the decryption capability? Although Fully Homomorphic Encryption (FHE) [1]–[3] is promising to tackle this problem, it is not practical enough for applications that involve a large volume of data [4]. Moreover, FHE in general does not solve another important problem: How can we force the cloud to execute the delegated computational functions honestly? This calls for solutions that can hold the misbehaving cloud accountable.

In this paper, we consider the problem of Verifiable Delegated Set Intersection on outsourced encrypted data (VDSI), which can be seen as the cloud version of the well investigated problem of Private Set Intersection (PSI) [5]–[7]. In the setting of PSI, two parties jointly compute the intersection of their private data sets such that they learn the intersection set but nothing else (the sizes of their private data sets may or may not be deemed as confidential [8]).

In the setting of VDSI, two cloud users, Alice and Bob, outsource their encrypted private data sets to the cloud. They would like to conduct the set intersection operation on their plaintext data sets. The straightforward solution would be for them to download their outsourced ciphertexts, decrypt the ciphertexts locally, and then execute a commodity two-party set intersection protocol. The straightforward solution is not practical, especially when the outsourced data sets are large and when they use wireless systems such as smartphones. Another drawback of this solution is that both Alice and Bob must participate simultaneously. For these reasons, Alice and Bob would prefer delegating the set intersection operation to the cloud, while being able to hold the misbehaving cloud accountable. Note that it is realistic to assume that the cloud is untrusted because it has the incentive not to honestly execute the protocols (e.g., for saving resources or shortening service response time). Moreover, the cloud may have been compromised and the attacker may return Alice and Bob with misleading results.

• *Q. Zheng and S. Xu are with the Department of Computer Science, University of Texas at San Antonio, San Antonio, TX, USA, 78249. E-mail: qingjizheng@gmail.com, shx@cs.utsa.edu*

1.1 Our Contribution

We initiate the investigation of a novel notion called VDSI, a useful primitive for delegating the set intersection operation on outsourced ciphertexts to the untrusted cloud. In contrast to the straightforward solution mentioned above, VDSI solves the problem by enabling the cloud to compute the set intersection, but *without* giving the decryption capability to the cloud. As such, VDSI can be seen as a special-purpose homomorphic cryptographic system for use in cloud computing. Since the cloud is untrusted and possibly malicious, VDSI allows Alice and Bob to verify whether the cloud has faithfully computed the delegated set intersection protocol or not.

Specifically, we formally define security properties of VDSI, and present a concrete VDSI scheme. The scheme is based on two ideas: (i) using proxy re-encryption to enable the cloud to compare equality of plaintexts corresponding to two ciphertexts that are encrypted using different public keys; (ii) using a novel variant of cryptographic accumulator, which can be used to verify the membership of *multiple* elements through a single examination and may be of independent value, to allow the cloud to show the correctness of the resulting intersection set.

Our VDSI scheme has two appealing features. First, it does not require the participation of Alice and Bob, because the cloud conducts the delegated computing. Second, it is much more efficient than the straightforward solution mentioned above. Suppose Alice's (Bob's) data set has n (m) elements, and the intersection set has k elements. Our solution only incurs $O(k)$ computational and communication costs on Alice and Bob, for decrypting and verifying the results received from the cloud. This means that our solution is optimal (up to a constant factor). In contrast, the straightforward solution incurs $O(m+n)$ computational and communication costs on Alice and Bob. Note that it is possible that $m + n >> k$. Experimental evaluation confirms that the VDSI scheme is practical,

We believe that the novel concept of VDSI will inspire many fruitful studies. For example, our solution only achieves a "weak" version of the *function output secrecy* property, which allows the untrusted cloud to launch a *plaintext guessing* attack against Alice's and Bob's private data (i.e., the success probability depends on the size of the plaintext space). It is an outstanding open problem to settle down whether or not this weak guarantee is inherent to the problem that VDSI aims to solve; if not, we need to design a better solution that is immune to this attack. Another outstanding open problem is to enforce fine-grained access control over the delegated set intersection operation, which may or may not need to be traded from the verifiability.

1.2 Related Work

To the best of our knowledge, this is the first work that considers the PSI problem in the cloud computing setting, where cloud users not only outsource their private data but also outsource their set intersection operations, while being able to hold the dishonest cloud vendors accountable for not faithfully executing the delegated operations. Nevertheless, there are prior studies on related problems.

Private Set Intersection. The PSI (private set intersection) problem was initiated in [9] and has become an essential building-block for many applications. Many variants of PSI [6], [8], [10]–[17] have been proposed, with various features (e.g., preventing a malicious party from choosing arbitrary inputs [6], [11], hiding the sizes of the inputs [8]). There have been schemes that aim to reduce the computational and communication complexities (e.g., the RSA-OPRF-based protocol [18], the garbled circuit protocol [17], and the garbled bloom filter protocol [16]). Among the state-of-the-art PSI solutions, the most efficient PSI protocol incurs $O(m + n)$ computational and communication complexities, where m and n are sizes of the respective data sets [16]. We note that [19]–[21] considered the problem of server-aided private set intersection, where cloud users share some secrets with each other to preprocess data sets at the time of outsourcing their data. Such collaborative preprocessing is not needed in the setting of VDSI. Finally, a recent work [22] studies verifiable complex set operations over outsourced plaintext data sets (i.e., the outsourced data is not encrypted). In contrast, we consider outsourced computing on outsourced encrypted data.

Public Key Encryption with Equality Test. The problem of public key encryption with equality test is to decide whether two ciphertexts that are encrypted using two different public keys correspond to the same plaintext or not [23]–[26]. In order to enforce access control over the equality test operation, a variant of the problem is to allow the data owners to authorize who can perform the equality test on the outsourced encrypted data [27]. These protocols do not consider the requirement of verifiability on the equality test results, which is crucial to VDSI in the present paper.

Verifiable Computation. How to securely and efficiently delegate the computation of a function to a remote server has been under active research [28]–[37]. In these solutions, the data owner pre-processes the inputs to the delegated function in question before outsourcing the data to the cloud, and the cloud needs to prove the correctness of the outcome of a function execution. However, these solutions do not solve the problem studied in this paper because (i) the input to their functions is from a single source and known to the delegator in advance, and (ii) some solutions do not consider privacy of the input. In contrast, our model has the following characteristics: the inputs to the delegated functions include other data owners' private data sets, which are not known to the delegators in advance. Finally, [38] considered the notion of verifiable private multi-party computation, but not in the setting of outsourcing data and functions to the cloud.

1.3 Paper Organization

Section 2 reviews some cryptographic preliminaries. Section 3 formulates the problem of VDSI and its security properties. Section 4 introduces the extended accumulator scheme, which is used as a building-block and may be of independent value. Section 5 presents a VDSI scheme and analyzes its security, while Section 6 evaluates its performance. Section 7 concludes the paper.

2 Cryptographic Preliminaries

Let $(e, g, G, G_T, p) \leftarrow \mathsf{MapGen}(1^\ell)$ denote that the bootstrapping algorithm MapGen generates a bilinear map $e : G \times G \rightarrow G_T$, where G and G_T are cyclic groups of order p which is an ℓ-bit prime, g is a generator of G, and the bilinear map e satisfies (i) for $a, b \in \mathbb{Z}_p$, $e(g^a, g^b) = e(g, g)^{ab}$, (ii) $e(g, g)$ is non-degenerate, and (iii) e can be efficiently computed. The bilinear map e is one-way, i.e. the probability of a probabilistic polonomial algorithm inverting e is negligible, which holds when G and G_T are instantiated with Weil or Tate pairing over MNT curves [39]. Table 2 summaries the notions for the algorithms and parameters in the VDSI scheme, multi-accumulator scheme and the signature scheme Sig.

Notation	Description
$\mathsf{D}_a, \mathsf{C}_a$	Alice's data set and its encryption form
$\mathsf{D}_b, \mathsf{C}_b$	Bob's data set and its encryption form
Setup, KeyGen, Enc, Dec, AuGen, SetOp, Verify	algorithms of the VDSI
pm, sk, pk, si, au, rslt, proof	parameters of the VDSI
acKeyGen, acGen, acProve, acVerify	algorithms of the multi-accumulator
acSk, acPk, acDig, acRslt, acWit	parameters of the multi-accumulator
sigKeyGen, sigSign, sigVerify	algorithms of signature scheme Sig
sigSk, sigPk, σ	parameters of the signature scheme

TABLE 1
Notations for algorithms and parameters in the VDSI, multi-accumulator and the signature scheme Sig.

Bilinear q-strong Diffie-Hellman assumption (q-SDH) [39]. For given $(e, g, G, G_T, p) \leftarrow \mathsf{MapGen}(1^\ell)$, and $g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^q}$ where $\alpha \overset{R}{\leftarrow} \mathbb{Z}_p$ and q is bounded by a polynomial in ℓ, there exists no probabilistic polynomial-time algorithm $\mathcal{A}$ that can compute $(s, e(g,g)^{1/(\alpha+s)})$ where $s \in \mathbb{Z}_p$ with a non-negligible probability in ℓ. The probability is defined over the random choices of the parameters and random coins used by $\mathcal{A}$.

Decisional Linear assumption (DL) [39]. For given $(e, g, G, G_T, p) \leftarrow \mathsf{MapGen}(1^\ell)$, and $(f, h, g^{r_1}, f^{r_2}, Q)$ where $f, h, Q \overset{R}{\leftarrow} G$ and $r_1, r_2 \overset{R}{\leftarrow} \mathbb{Z}_p$, there exists no probabilistic polynomial-time algorithm $\mathcal{A}$ that can determine $Q \overset{?}{=} h^{r_1+r_2}$ with a non-negligible advantage, where "advantage" is defined as

$$\Pr[\mathcal{A}(g, f, h, g^{r_1}, f^{r_2}, Q) = 1] \\ - \Pr[\mathcal{A}(g, f, h, g^{r_1}, f^{r_2}, h^{r_1+r_2}) = 1],$$

and the probability is defined over the random choices of the parameters and random coins used by $\mathcal{A}$.

Unforgeable Digital Signature. Let Sig = (sigKeyGen, sigSign, sigVerify) be a secure signature scheme, where sigKeyGen generates a pair of public and private keys, sigSign generates a signature for a message, and sigVerify determines if a message matches a signature. Any signature scheme satisfying the standard definition of unforgeability under adaptive chosen-message attacks [40] is sufficient for the purpose of this paper.

3 VDSI Model and Definition

3.1 System Model

Figure 1 illustrates the *system model* of VDSI (verifiable delegated set intersection operations on outsourced encrypted data). The system has four entities: a trusted third party, a cloud, and two cloud users (i.e., data owners) referred to as Alice and Bob. The trusted third party is responsible for initializing system public parameters used by the cloud and cloud users. Alice and Bob can be either individuals or organizations that outsource their private data sets, denoted by D_a and D_b, to the cloud in encrypted form, denoted by C_a and C_b, respectively. Alice and Bob want to compute the intersection set $\mathsf{D}_a \cap \mathsf{D}_b$, by delegating the set intersection operation to the cloud but *without* giving the cloud the capability to decrypt C_a and C_b.

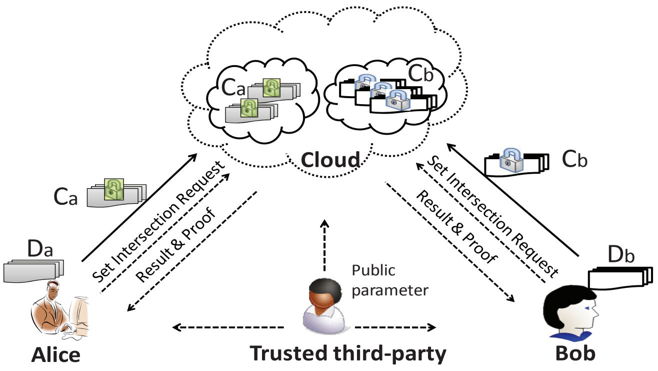


Fig. 1. VDSI system model: data owners Alice and Bob encrypt their data sets (denoted by D_a and D_b), using their respective public keys, outsource to the cloud the resulting ciphertexts (denoted by C_a and C_b), and delegate the computation of $\mathsf{D}_a \cap \mathsf{D}_b$ to the cloud but without giving it the capability to decrypt C_a and C_b.

Remark. The above system model can be easily extended to accommodate the following more general scenarios. First, rather than letting Alice and Bob outsource their encrypted data to the same cloud, they can outsource their encrypted data to two different clouds (dubbed *storage clouds*). Second, rather than letting (one of) the storage cloud(s) conduct the delegated computation on C_a and C_b, another cloud (dubbed *computing* cloud) or any other third party can be used for this purpose. The extension is trivial and omitted.

3.2 Threat Model and Basic Idea of Defense

We assume that the cloud users (i.e., data owners) are honest-but-curious, meaning that they act according to the protocols and use their real data sets as inputs to the protocols, but are curious about each other's private data. However, the cloud is possibly malicious. This means that the cloud can attempt to breach the secrecy of the data outsourced to the cloud, manipulate the integrity of the outsourced data, and deviate from the protocols arbitrarily. The cloud may be controlled by an attacker, who also has control over all the communication channels. This means that denial-of-service attack is inevitable and should be addressed othogonally by another layer of defense. We will use "the attacker" and "the cloud" interchangeably.

The basic idea for defending against the possibly malicious cloud is to ask the cloud to generate a proof, which shows that it has faithfully executed the delegated set intersection operation. By "examining" the result and proof returned by the cloud, Alice and/or Bob can verify whether or not the cloud has faithfully executed the delegated set intersection operations on C_a and C_b or not.

3.3 VDSI Function Definition

In order to simplify the description of both the definition and the concrete scheme that will be presented later, we assume that there is an authenticated user-to-cloud private communication channel, which is used by a cloud user to send some secret information to the cloud (e.g., the secret information that allows the cloud to conduct the delegated set intersection operation). This assumption does not impose any significant restriction because in the case the cloud is controlled by the attacker, the secret information is given to the attacker any way. In practice, the channel can be reaidly realized by encrypting the secret information under the cloud's public key.

With loss of generality, denote by Alice's plaintext data set $\mathsf{D}_a = \{d_{a,0}, \ldots, d_{a,n}\}$ and Bob's plaintext data set $\mathsf{D}_b = \{d_{b,0}, \ldots, d_{b,m}\}$. Alice (Bob) outsources her (his) encrypted version of D_a (D_b), denoted by C_a (C_b), to the cloud. Alice and Bob want to compute $\mathsf{D}_a \cap \mathsf{D}_b$ by delegating the computation to the cloud, but without giving the cloud capability to decrypt C_a and C_b.

Definition 1: A VDSI scheme has seven algorithms:

- $\mathsf{pm} \leftarrow \mathsf{Setup}(1^\ell)$: Given security parameter ℓ, the trusted third party runs this algorithm to bootstrap the public parameters pm.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pm})$: Alice runs this randomized algorithm to generate a pair of public and private keys $(\mathsf{pk}_a, \mathsf{sk}_a)$, where pk_a is made public and sk_a is kept secret by Alice. Similarly, we denote Bob's pair of public and private keys by $(\mathsf{pk}_b, \mathsf{sk}_b)$.
- $(\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a)$: Alice runs this encryption algorithm to encrypt her data set D_a to ciphertext C_a, which is outsourced to the cloud, and some secret information si_a, which is kept secret by Alice. Bob can generate $(\mathsf{C}_b, \mathsf{si}_b)$ similarly.
- $\{\mathsf{D}', \perp\} \leftarrow \mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a)$: Alice runs this decryption algorithm to decrypt ciphertext rslt_a, which is the output of the delegated set intersection operation conducted by the cloud on ciphertexts C_a and C_b, to obtain the intersection set $\mathsf{D}' = \mathsf{D}_a \cap \mathsf{D}_b$. In the case the decryption fails, the algorithm outputs $\perp$ instead. Note this decryption algorithm also can be used to decrypt C_a.
- $\mathsf{au}_a \leftarrow \mathsf{AuGen}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b)$: In order to allow the cloud to conduct the set intersection operation on the outsourced ciphertexts C_a and C_b, Alice runs this algorithm to generate some auxiliary information au_a, which is sent to the cloud through the authenticated user-to-cloud private communication channel (see justification above), where si_a is the Alice's secret information generated by $\mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a)$. Similarly, Bob can generate and send au_b to the cloud, where $\mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a)$.
- $\{(\mathsf{rslt}_a, \mathsf{proof}_a), (\mathsf{rslt}_b, \mathsf{proof}_b)\} \leftarrow \mathsf{SetOp}(\mathsf{C}_a,\ \mathsf{au}_a,\ \mathsf{C}_b,\ \mathsf{au}_b)$: This is the delegated set intersection operation run by the cloud. Depending on the application, the cloud may return $(\mathsf{rslt}_a, \mathsf{proof}_a)$ and $(\mathsf{rslt}_b, \mathsf{proof}_b)$ respectively to Alice and Bob, or return $(\mathsf{rslt}_a, \mathsf{proof}_a)$ to Alice or $(\mathsf{rslt}_b, \mathsf{proof}_b)$ to Bob who requested the delegated set intersection operation, where proof_a and proof_b are proofs that can show that the cloud has faithfully executed the SetOp protocol.
- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a)$: Alice runs this algorithm to verify whether rslt_a is faithfully generated by the cloud according to the SetOp protocol. If so (with output 1), Alice calls the $\mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a)$ algorithm to decrypt rslt_a; otherwise (with output 0), the cloud is cheating.

We say a VDSI scheme is correct if the following holds:

$$\Pr\left[\begin{array}{l} \mathsf{pm} \leftarrow \mathsf{Setup}(1^\ell), \\ (\mathsf{pk}_a, \mathsf{sk}_a) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), \\ (\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), \\ \forall\ \mathsf{D}_a, \mathsf{D}_b, (\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a), \\ \quad (\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b), \\ \quad \mathsf{au}_a \leftarrow \mathsf{AuGen}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b), \\ \quad \mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a), \\ \quad \{(\mathsf{rslt}_a, \mathsf{proof}_a), (\mathsf{rslt}_b, \mathsf{proof}_b)\} \leftarrow \\ \qquad\qquad \mathsf{SetOp}(\mathsf{C}_a, \mathsf{au}_a, \mathsf{C}_b, \mathsf{au}_b) : \\ \quad 1 \leftarrow \mathsf{Verify}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a), \\ \quad 1 \leftarrow \mathsf{Verify}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{rslt}_b, \mathsf{proof}_b), \\ \quad \mathsf{D}_a \cap \mathsf{D}_b = \mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a) = \mathsf{Dec}(\mathsf{sk}_b, \mathsf{rslt}_b) \end{array}\right] = 1.$$

3.4 VDSI Security Definition

Informally, VDSI aims to achieve the following security properties against the afore-discussed threat model. We consider three security properties: *outsourced data secrecy*, *function output secrey* and *verifiability*, which are formally defined below. Let ϵ be a negligible function in security parameter ℓ. We consider a probabilistic polynomial-time (in ℓ) adversary $\mathcal{A}$ controlling the cloud.

Outsourced Data Secrecy: Similar to security against chosen-plaintext attack, this property means that the attacker $\mathcal{A}$ cannot breach secrecy of the outsourced data, unless that $\mathcal{A}$ is given the respect auxiliary information.

Definition 2: (outsourced data secrecy) A VDSI scheme achieves *outsourced data secrecy* if the following holds:

$$\left|\Pr\left[\begin{array}{l} \mathsf{pm} \leftarrow \mathsf{Setup}(1^\ell), \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), \\ (\mathsf{D}_0, \mathsf{D}_1) \leftarrow \mathcal{A}^{\mathsf{Enc}}(\mathsf{pk}), s.t. |\mathsf{D}_0| = |\mathsf{D}_1|, \\ \lambda \xleftarrow{R} \{0, 1\}, (\mathsf{C}_\lambda, \mathsf{si}_\lambda) \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{D}_\lambda), \\ \lambda' \leftarrow \mathcal{A}^{\mathsf{Enc}}(\mathsf{pk}, \mathsf{C}_\lambda, \mathsf{D}_0, \mathsf{D}_1) : \\ \lambda = \lambda' \end{array}\right] - \frac{1}{2}\right| \leq \epsilon$$

This property is necessary but not sufficient because it only assures the secrecy of outsourced data when $\mathcal{A}$ is not given the delegated set intersection operation capability. The following property, function output secrecy, is used to capture the secrecy of oursouced data after $\mathcal{A}$ is granted the capability (i.e, $\mathcal{A}$ is given the auxiliary information au).

Function Output Secrecy: This property means that $\mathcal{A}$ cannot breach secrecy of the resulting intersetction set $\mathsf{D}_a \cap \mathsf{D}_b$. Ideally, given a target ciphertext and the auxiliary information, $\mathcal{A}$ cannot learn the plaintext with a non-negligible probability.

Definition 3: (function output secrecy) A VDSI scheme achieves *function output secrecy* if

$$\Pr\left[\begin{array}{l} \mathsf{pm} \leftarrow \mathsf{Setup}(1^\ell), \\ (\mathsf{pk}_a, \mathsf{sk}_a) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), \\ (\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), \\ \forall \mathsf{D}_a, \mathsf{D}_b, (\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a), \\ (\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b), \\ \forall \mathsf{cph} \in (\mathsf{C}_a \cup \mathsf{C}_b), \\ \mathsf{au}_a \leftarrow \mathsf{AuGen}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b), \\ \mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a), \\ \{d_1, \ldots, d_q\} \leftarrow \mathcal{A}^{\mathsf{Enc},\mathsf{SetOp},\mathsf{Verify}} \\ (\mathsf{pk}_a, \mathsf{au}_a, \mathsf{C}_a, \mathsf{pk}_b, \mathsf{au}_b, \mathsf{C}_b, \mathsf{cph}) : \\ \exists i \in [1, q], d_i = d \end{array}\right] \leq f(\ell, q, |\mathcal{M}|),$$

where q is the maximum number of guessing against cph, d is the plaintext with respect to cph, and $\mathcal{M}$ is the plaintext domain.

Remark. Ideally, we want $f(\ell, q, |\mathcal{M}|)$ to be a negaligible function in ℓ as well. Unfortunately, we are only able to construct a scheme that achieves $f(\ell, q, |\mathcal{M}|) = \frac{q}{|\mathcal{M}|} + \epsilon$, which is a non-negligible function in ℓ because $|\mathcal{M}|$ would not be exponentially in ℓ. The intuition behind $\frac{q}{|\mathcal{M}|}$ is that $\mathcal{A}$ can launch a plaintext-guessing attack (in a way similar to the *online dictionary attack* against passwords), which is specific to our scheme that will be presented later. Designing a VDSI scheme that achieves negligible $f(\ell, q, |\mathcal{M}|)$ in ℓ is left as an open problem for future research. Nevertheless, our definition is general enough to accommodate that scenario.

Verifiability: This property means that any $\mathcal{A}$ not faithfully executing the SetOp protocol is bound to be caught.

Definition 4: (verifiability) A VDSI scheme is *verifiable* if

$$\Pr\left[\begin{array}{l} \mathsf{pm} \leftarrow \mathsf{Setup}(1^\ell), \\ (\mathsf{pk}_a, \mathsf{sk}_a) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), \\ (\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(\mathsf{pm}), \\ (\mathsf{D}_a, \mathsf{D}_b) \leftarrow \mathcal{A}^{\mathsf{Enc},\mathsf{AuGen},\mathsf{SetOp},\mathsf{Verify}}(\mathsf{pk}_a, \mathsf{pk}_b), \\ (\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a), \\ (\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b), \\ \mathsf{au}_a \leftarrow \mathsf{AuGen}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b), \\ \mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a), \\ \{(\mathsf{rslt}_a, \mathsf{proof}_a), (\mathsf{rslt}_b, \mathsf{proof}_b)\} \leftarrow \mathcal{A}^{\mathsf{Enc},\mathsf{SetOp},\mathsf{Verify}} \\ \qquad (\mathsf{pk}_a, \mathsf{au}_a, \mathsf{D}_a, \mathsf{C}_a, \mathsf{si}_a, \mathsf{pk}_b, \mathsf{au}_b, \mathsf{D}_b, \mathsf{C}_b, \mathsf{si}_b) : \\ 1 \leftarrow \mathsf{Verify}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a) \wedge \\ 1 \leftarrow \mathsf{Verify}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{rslt}_b, \mathsf{proof}_b) \wedge \\ (\mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a) \neq \mathsf{Dec}(\mathsf{sk}_b, \mathsf{rslt}_b) \vee \\ \mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a) \neq (\mathsf{D}_a \cap \mathsf{D}_b)) \end{array}\right] \leq \epsilon.$$

4 Building-Block: multi-accumulator

A cryptography accumulator is a primitive for a *verifier* to examine the membership of an element with respect to a (static or dynamic) data set. The examination is based on some public data and membership proof provided by a *prover*. In a single-accumulator scheme, each membership proof allows a verifier to examine the membership of a *single* element with respect to a data set. The idea of single-accumulator has been studied extensively (see, e.g., [41]–[43]). In this paper, we introduce the idea of multi-accumulator by which, each membership proof allows a verifier to examine the membership of *multiple* elements with respect to a data set. in the context of set intersection operations, multi-accumulator allows Alice (Bob) to verify, via a single examination, that $\mathsf{D}_a \cap \mathsf{D}_b \subseteq \mathsf{D}_a$ ($\subseteq \mathsf{D}_b$).

4.1 Function and Security Definitions

Suppose Alice has a data set acD_a and outsources it to the cloud (as the prover). Bob (as the verifier) has a dataset acD_b and queries the cloud for $\mathsf{acD}_a \cap \mathsf{acD}_b$.

Definition 5: A multi-accumulator scheme has the following algorithms:

- $(\mathsf{acSk}, \mathsf{acPk}) \leftarrow \mathsf{acKeyGen}(1^\ell)$: The trusted third party runs this algorithm to generate a pair of public and private key $(\mathsf{acPk}, \mathsf{acSk})$.
- $\mathsf{acDig}_a \leftarrow \mathsf{acGen}(\mathsf{acPk}, \mathsf{acD}_a)$: Alice runs this algorithm to generate a digest acDig for acD_a, which is outsourced to the cloud. Similarly, Bob can generate acDig_b with respect to acD_b.
- $(\mathsf{acRslt}, \mathsf{acWit}) \leftarrow \mathsf{acProve}(\mathsf{acPk}, \mathsf{acD}_b, \mathsf{acD}_a)$: Given the data set acD_b from Bob, the cloud runs this algorithm to generate $\mathsf{acRslt} = (\mathsf{acD}_b \cap \mathsf{acD}_a)$ with an accompanying witness acWit for this fact.
- $\{0, 1\} \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig}_a)$: Bob runs this algorithm to examine if $\mathsf{acRslt} = \mathsf{acD}_b \cap \mathsf{acD}_a$, where acDig_b is the digest with respect to acD_b and acDig is the digest with respect to acD_a. If so, output 1; otherwise, output 0.

A multi-accumulator scheme is correct if

$$\Pr\left[\begin{array}{l} \forall\ \mathsf{acD}_a, \mathsf{acD}_b \\ (\mathsf{acSk}, \mathsf{acPk}) \leftarrow \mathsf{acKeyGen}(1^\ell), \\ \mathsf{acDig}_b \leftarrow \mathsf{acGen}(\mathsf{acPk}, \mathsf{acD}_b), \\ \mathsf{acDig}_a \leftarrow \mathsf{acGen}(\mathsf{acPk}, \mathsf{acD}_a), \\ (\mathsf{acRslt}, \mathsf{acWit}) \leftarrow \mathsf{acProve}(\mathsf{acPk}, \mathsf{acD}_b, \mathsf{acD}_a) : \\ 1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig}_a) \end{array}\right] = 1.$$

A multi-accumulator scheme is secure if a malicious probabilistic polynomial-time prover $\mathcal{A}$ can cheat the honest verifier without being caught. Let ℓ be a security parameter and ϵ be a negligible function in ℓ. Formally, we have:

Definition 6: A multi-accumulator scheme is secure if

$$\Pr\left[\begin{array}{l} (\mathsf{acPk}, \mathsf{acSk}) \leftarrow \mathsf{acKeyGen}(1^\ell), \\ \mathsf{acD}_a \leftarrow \mathcal{A}^{\mathsf{acProve},\mathsf{acVerify}}(\mathsf{acPk}), \\ \mathsf{acDig}_a \leftarrow \mathsf{acGen}(\mathsf{acSk}, \mathsf{acD}_a), \\ (\mathsf{acD}_b, \mathsf{acRslt}, \mathsf{acWit}) \\ \qquad \leftarrow \mathcal{A}^{\mathsf{acProve},\mathsf{acVerify}}(\mathsf{acPk}, \mathsf{acD}_a) : \\ \mathsf{acDig}_b \leftarrow \mathsf{acGen}(\mathsf{acPk}, \mathsf{acD}_b), \\ 1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig}), \\ \mathsf{acRslt} \neq \mathsf{acD}_b \cap \mathsf{acD}_a \end{array}\right] \leq \epsilon.$$

4.2 Construction based on Bilinear Map

A multi-accumulator scheme can be based on a single-accumulator scheme that supports both membership and non-membership proofs, as follows: the cloud generates a witness for each element of acD_b showing the element is a member or non-member of acD_a and simply puts them together as the witness for $\mathsf{acRslt} = \mathsf{acD}_b \cap \mathsf{acD}_a$. However, this straightforward approach is costly because both the computational and communication complexities are linear to $|\mathsf{acD}_b|$.

We present a multi-accumulator scheme, where the size of the witness is constant (i.e., independent of $|\mathsf{acD}_b|$). The proposed multi-accumulator scheme is extended from the single-accumulator scheme due to [41], [43], while adapting the basic idea underlying [44] as follows: Suppose Alice's data set is $\mathsf{acD}_a = \{d_{a,1}, \ldots, d_{a,n}\}$, Bob's data set is $\mathsf{acD}_b = \{d_{b,1}, \ldots, d_{b,m}\}$, and $\mathsf{acRslt} = \mathsf{acD}_a \cap \mathsf{acD}_b$. We can encode acD_a via polynomial $R(x) = \prod_{t \in \mathsf{acD}_a}(x+t)$, encode acD_b via polynomial $W(x) = \prod_{t \in \mathsf{acD}_b}(x+t)$, encode the intersection set acRslt via polynomial $T(x) = \prod_{t \in \mathsf{acRslt}}(x+t)$, and encode the subset $\mathsf{acD}_b - \mathsf{acRslt}$ via polynomial $Q(x) = \prod_{t \in (\mathsf{acD}_b - \mathsf{acRslt})}(x+t)$. These polynomials satisfy the following: (i) $T(x)Q(x) = W(x)$, (ii) $T(x)$ is a divisor of $R(x)$, and (iii) $Q(x)$ is co-prime to $R(x)$. For the special case $\mathsf{acRslt} = \emptyset$, the three conditions also hold since $T(x) = 1$, $Q(x) = W(x) = \prod_{t \in \mathsf{acD}_b}(x+t)$ and $R(x) = \prod_{t \in \mathsf{acD}_a}(x+t)$. Therefore, based on this idea, the multi-accumulator scheme allows the cloud to show the correctness of the intersection set, which can be either empty or non-empty. It can be constructed as follows:

- $\mathsf{acKeyGen}(1^\ell)$: Let $(e, g, G, G_T, p) \leftarrow \mathsf{MapGen}(1^\ell)$, set $\alpha \xleftarrow{R} \mathbb{Z}_p$ and $\mathsf{acPk} = (g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^q}), \mathsf{acSk} = (\alpha)$, where q is bounded by a polynomial in security parameter ℓ.
- $\mathsf{acGen}(\mathsf{acPk}, \mathsf{acD}_a)$: Given Alice's data set $\mathsf{acD}_a = \{d_{a,1}, \ldots, d_{a,n}\} \in \mathbb{Z}_p^n$ where $n \le q$, compute its digest as
$$\mathsf{acDig}_a = g^{\prod_{i=1}^{n}(d_{a,i}+\alpha)}.$$
- $\mathsf{acProve}(\mathsf{acPk}, \mathsf{acD}_b, \mathsf{acD}_a)$: Given Bob's data set $\mathsf{acD}_b = (d_{b,1}, \ldots, d_{b,m}) \in \mathbb{Z}_p^m$ where $m \le q$, compute $\mathsf{acRslt} = \mathsf{acD}_b \cap \mathsf{acD}_a$, and generate a witness as follows:
  - Let $T'(x) = \prod_{t \in (\mathsf{acD}_a - \mathsf{acRslt})}(x+t)$ and compute $g^{T'(\alpha)}$ by substituting x with α.
  - Let $Q(x) = \prod_{t \in (\mathsf{acD}_b - \mathsf{acRslt})}(x+t)$ and $R(x) = \prod_{t \in \mathsf{acD}_a}(x+t)$, and find two polynomials $Q'(x), R'(x)$ such that $Q(x)Q'(x) + R(x)R'(x) = 1 \bmod p$ by taking advantage of $\gcd(Q(x), R(x)) = 1$. Compute $(g^{Q(\alpha)}, g^{Q'(\alpha)}, g^{R'(\alpha)})$ by substituting x with α.

  Set $\mathsf{acRslt} = \mathsf{acD}_b \cap \mathsf{acD}_a$ and $\mathsf{acWit} = (g^{Q(\alpha)}, g^{Q'(\alpha)}, g^{R'(\alpha)}, g^{T'(\alpha)})$.
- $\mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig})$: Given acWit and acRslt from the prover, the verifier proceeds as follows:
  1) If $\mathsf{acRslt} \neq \emptyset$, compute $g^{T(\alpha)}$ according to $T(x) = \prod_{t \in \mathsf{acRslt}}(x+t)$. Otherwise, let $T(x) = 1$ and $g^{T(\alpha)} = g$.
  2) If $e(g^{Q(\alpha)}, g^{T(\alpha)})) \neq e(\mathsf{acDig}_b, g)$, return 0; otherwise, proceed to next step.
  3) If $e(g^{T(\alpha)}, g^{T'(\alpha)}) \neq e(\mathsf{acDig}, g)$, return 0; otherwise, proceed to next step.
  4) If $e(g^{Q(\alpha)}, g^{Q'(\alpha)}) e(\mathsf{acDig}, g^{R'(\alpha)}) \neq e(g, g)$, return 0; otherwise, return 1.

Correctness of the multi-accumulator scheme can be verified easily. We describe its asymptotic efficiency in Table 2. It is worth noting that (i) the witness generated by algorithm $\mathsf{acProve}$ only consists of four group elements, meaning that the complexity is independent of $k = |\mathsf{acD}_b \cap \mathsf{acD}_a|$, and (ii) the computational complexity of algrorithm $\mathsf{acVerify}$ is linear to $k = |\mathsf{acD}_b \cap \mathsf{acD}_a|$.

TABLE 2
Asymptotical efficiency of the multi-accumulator scheme, where Exp denotes the exponentiation operation, Pairing denotes the pairing operation, $n = |\mathsf{acD}_a|$, $m = |\mathsf{acD}_b|$ and $k = |\mathsf{acD}_a \cap \mathsf{acD}_b|$.

	acGen	acProve	acVerify
Computation	nExp	$(n+m)$Exp	kExp + 7Pairing
Output Size	$\lvert G\rvert$	$4\lvert G\rvert$	N/A

Now we prove its security.

Theorem 1: Assume that the q-SDH assumption holds, the multi-accumulator scheme is secure with respect to Definition 6.

Proof: We show that if there is an adversary $\mathcal{A}$ that can break the multi-accumulator scheme with a non-negligible probability, there is an algorithm $\mathcal{B}$ that can break the q-SDH assumption with a non-negligible probability.

Suppose $\mathcal{B}$ is given a challenge instance $(g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^q})$, where $\alpha \xleftarrow{R} \mathbb{Z}_p$ and α is unknown. $\mathcal{B}$ simulates the multi-accumulator scheme for $\mathcal{A}$, according to the game implied by Definition 6. For $\mathsf{acD}_a = \{d_{a,1}, \ldots, d_{a,n}\}$ with digest $\mathsf{acDig}_a = g^{\prod_{t \in \mathsf{acD}_a}(\alpha+t)}$, suppose $\mathcal{A}$ returns acD_b, acRslt and $\mathsf{acWit} = (g^{Q(\alpha)}, g^{Q'(\alpha)}, g^{R'(\alpha)}, g^{T'(\alpha)})$. If $\mathcal{A}$ breaks the security of the multi-accumulator with non-negligible probability, then there exists acD_b, acRslt and acWit such that the followings hold: (i) $1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig}_a)$ where $\mathsf{acDig}_b \leftarrow \mathsf{acGen}(\mathsf{acPk}, \mathsf{acD}_b)$, and (ii) $\mathsf{acRslt} \neq \mathsf{acD}_b \cap \mathsf{acD}_a$, and then $\mathcal{B}$ can break the q-SDH assumption by presenting a tuple $(t', e(g,g)^{1/(\alpha+t')})$ where $t' \in \mathbb{Z}_p$.

First, we claim that $\forall t \in \mathsf{acRslt}$, it holds that $t \in \mathsf{acD}_b$. To prove this, suppose there exists $t' \in \mathsf{acRslt}$ but $t' \notin \mathsf{acD}_b$, meaning that polynomial $\prod_{t \in \mathsf{acD}_b}(x+t)$ cannot be divided by $(x+t')$. Therefore, polynomial $\prod_{t \in \mathsf{acD}_b}(x+t)$ can be represented (in polynomial-time) as $U(x)(x+t') + \lambda$ where $\lambda \neq 0$, and $U(x)$ is a polynomial of degree $|\mathsf{acD}_b| - 1$. On the other hand, $1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig}_a)$ implies the following:
$$e(g^{Q(\alpha)}, g^{\prod_{t \in \mathsf{acRslt}}(\alpha+t)}) = e(\mathsf{acDig}_b, g) = e(g^{\prod_{t \in \mathsf{acD}_b}(\alpha+t)}, g).$$
By substituting $\prod_{t \in \mathsf{acD}_b}(\alpha+t)$ with $U(\alpha)(\alpha+t') + \lambda$ in the right-hand of the above equation, we have
$$e(g^{Q(\alpha)}, g^{\prod_{t \in \mathsf{acRslt}}(\alpha+t)}) = e(g^{U(\alpha)(\alpha+t')+\lambda}, g).$$

This leads to:

$$e(g^{Q(\alpha)}, g^{\prod_{t\in\mathsf{acRslt}, t\neq t'}(\alpha+t)}) = e(g^{U(\alpha)+\frac{\lambda}{\alpha+t'}}, g).$$

Therefore,

$$e(g,g)^{\frac{1}{\alpha+t'}} = (e(g^{Q(\alpha)}, g^{\prod_{t\in\mathsf{acRslt}, t\neq t'}(\alpha+t)})e(g,g)^{-U(\alpha)})^{\frac{1}{\lambda}}.$$

That is, if (i) $\mathcal{A}$ breaks the multi-accumulator scheme with a non-negligible probability and (ii) $\exists t' \in \mathsf{acRslt}$ such that $t' \notin \mathsf{acD}_b$, then $\mathcal{B}$ can break the q-SDH assumption by outputting $(t', e(g,g)^{1/(\alpha+t')})$ with a non-negligible probability.

Second, we claim that $\forall t \in \mathsf{acRslt}$ it holds that $t \in \mathsf{acD}_a$. This can be proved similarly.

Third, we claim that $\forall t \in (\mathsf{acD}_b - \mathsf{acRslt})$, it holds that $t \notin \mathsf{acD}_a$. To prove this, suppose there exists $t' \in (\mathsf{acD}_b - \mathsf{acRslt})$ but $t' \in \mathsf{acD}_a$. This means that there exists polynomials

$$Q'(x)(x+t')\prod_{t\in(\mathsf{acD}_b-\mathsf{acRslt})}^{t\neq t'}(x+t) + R'(x)(x+t')\prod_{t\in\mathsf{acD}_a}^{t\neq t'}(x+t) = 1,$$

which means

$$Q'(x)\prod_{t\in(\mathsf{acD}_b-\mathsf{acRslt})}^{t\neq t'}(x+t) + R'(x)\prod_{t\in\mathsf{acD}_a}^{t\neq t'}(x+t) = \frac{1}{(x+t')}.$$

On the other hand, $1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acD}_b, \mathsf{acRslt}, \mathsf{acWit}, \mathsf{acDig}_a)$ implies:

$$e(g^{Q'(\alpha)}, g^{\prod_{t\in(\mathsf{acD}_b-\mathsf{acRslt})}(\alpha+t)})e(g^{R'(\alpha)}, g^{\prod_{t\in\mathsf{acD}_a}(\alpha+t)}) = e(g,g).$$

Therefore, we have

$$e(g,g)^{\frac{1}{(\alpha+t')}} = e(g^{Q'(\alpha)}, g^{\prod_{t\in(\mathsf{acD}_b-\mathsf{acRslt}),t\neq t'}(\alpha+t)}) \cdot e(g^{R'(x)}, g^{\prod_{t\in\mathsf{acD}_a,t\neq\mathsf{acD}_a}(\alpha+t)}).$$

That is, if (i) $\mathcal{A}$ breaks the multi-accumulator scheme with a non-negligible probability and (ii) there exists $t' \in (\mathsf{acD}_b - \mathsf{acRslt})$ such that $t' \in \mathsf{acD}_a$, then $\mathcal{B}$ can break the q-SDH assumption by outputting $(t', e(g,g)^{1/(\alpha+t')})$ with a non-negligible probability.

To sum up, since $\forall t \in \mathsf{acRslt}$ it holds that $t \in \mathsf{acD}_b$ and $t \in \mathsf{acD}_a$, and $\forall t' \in (\mathsf{acD}_b - \mathsf{acRslt})$ it holds that $t' \notin \mathsf{acD}_a$, we conclude that $\mathsf{acRslt} = \mathsf{acD}_b \cap \mathsf{acD}_a$. Therefore, the multi-accumulator scheme is secure with respect to Definition 6. □

5 The VDSI Scheme

Basic Ideas. In order to attain a VDSI scheme, we need to resolve two issues: (i) How can we enable the cloud to compare the equality of two ciphertexts that are encrypted under two different public keys pk_a and pk_b, respectively? (ii) How can we enable the cloud to generate a proof for showing that it has faithfully executed the SetOp protocol, ideally without using zero-knowledge proof for the sake of better efficiency?

To resolve the above (i), we adopt the idea of proxy re-encryption as follows: Alice can generate a re-key and send it to the cloud, which can use the re-key to transform ciphertext C_a (encrypted under Alice's public key pk_a) into an intermediate form, say T_a. Similarly, the cloud can transform ciphertext C_b (encrypted under Bob's public key pk_b) into the same kind fo intermediate form, denoted by T_b. Then, the cloud can "compare" T_a and T_b to determine whether they correspond to the same plaintext or not. More specifically, a data item $d_{a,i} \in \mathsf{D}_a$ is encrypted using $\mathsf{pk}_a = (g^{\beta_a}, g^{\gamma_a})$ as $(g^{r_2}, g^{\gamma r_1}, d_{a,i}g^{\beta(r_1+r_2)})$, where $r_1, r_2 \xleftarrow{R} \mathbb{Z}_p$. Alice can give the re-key $\mathsf{rk}_a = g^{\beta_a/\gamma_a}$, rather than her private key $\mathsf{sk}_a = (\beta_a, \gamma_a)$ to the cloud, which now can transform the ciphertext into

$$\frac{e(d_{a,i}g^{\beta_a(r_1+r_2)}, g)}{e(g^{\gamma_a r_1}, g^{\beta_a/\gamma_a})e(g^{r_2}, g^{\beta_a})} = e(d_{a,i}, g).$$

Similarly, for data item $d_{b,i} \in \mathsf{D}_b$, the cloud can transform the corresponding ciphertext into $e(d_{b,i}, g)$. If $d_{a,i} = d_{b,i}$, then $e(d_{a,i}, g) = (d_{b,i}, g)$. While this method is sufficient to allow the cloud to determine whether the two ciphertexts correspond to the same plaintext or not, it does not achieve the desired semantic security because the cloud can launch the *plaintext guess* attack against elements $d_{a,i}$ and $d_{b,i}$. We have tried without success to eliminate this attack while preserving the other properties (especially the *verifiability*). We therefore leave it as an open problem.

To resolve the above issue (ii), we observe that the cloud, as illustrated above, can generate $e(d_{a,i}, g)$ for each $d_{a,i} \in \mathsf{D}_a$ and $e(d_{b,i}, g)$ for each $d_{b,i} \in \mathsf{D}_b$. As a result, the cloud can use the multi-accumulator scheme to generate a proof as follows: For Bob, let $e(d_{a,i}, g)$'s as acD_a and $e(d_{b,i}, g)$'s as acD_b, the cloud applies acProve to generate witness acWit_b for showing $\mathsf{acRslt} = \mathsf{acD}_a \cap \mathsf{acD}_b$ is the correct intersection set with respect to $e(d_{a,i}, g)$'s and $e(d_{b,i}, g)$'s. Given witness acWit_b, digest acDig_b of $e(d_{b,i}, g)$'s and digest acDig_a of $e(d_{a,i}, g)$, Bob can verify the correctness of acRslt, which can be computed from the returned intersection set of C_a and C_b. Similarly, the cloud can generate witness acWit_a for Alice, who can then conduct the same kind of verification. That is, by using the multi-accumulator scheme in section 4, the cloud users can verify the correctness of the intersection set, which contains zero or more common elements.

5.1 The Scheme

The scheme is a modular construction based on (i) a secure multi-accumulator scheme Ac = (acKeyGen, acGen, acProve, acVerify) such as the one described in Section 4, and (ii) a secure digital signature scheme Sig = (sigKeyGen, sigSign, sigVerify). The digital signature scheme is used to authenticate the encryption form of the accumulator digest, which assures that the cloud cannot manipulate it without being detected. Specifically, the scheme is described as follows:

Setup(1^ℓ): Given security parameter ℓ, the trusted third party runs $(e, g, G, G_T, p) \leftarrow \mathsf{MapGen}(1^\ell)$. Let $H : G_T \to \mathbb{Z}_p$ be a collision-resistant hash function. The trusted third party also runs $(\mathsf{acPk}, \mathsf{acSk}) \leftarrow \mathsf{KeyGen}(1^\ell)$ and sets the public parameter as

$$\mathsf{pm} = (\mathsf{acPk}, e, p, g, G, G_T).$$

KeyGen(pm): Alice runs $(\mathsf{sigPk}_a, \ \mathsf{sigSk}_a) \leftarrow \mathsf{sigKeyGen}(1^\ell)$, selects $\beta_a, \gamma_a \overset{R}{\leftarrow} \mathbb{Z}_p$, and sets

$$\mathsf{sk}_a = (\beta_a, \gamma_a, \mathsf{sigSk}_a), \quad \mathsf{pk}_a = (g^{\beta_a}, g^{\gamma_a}, \mathsf{sigPk}_a).$$

Similarly, Bob generates $\mathsf{sk}_b = (\beta_b, \gamma_b, \mathsf{sigSk}_b)$ and $\mathsf{pk}_b = (g^{\beta_b}, g^{\gamma_b}, \mathsf{sigPk}_b)$.

Enc$(\mathsf{pk}_a, \mathsf{D}_a)$: Alice, with $\mathsf{D}_a = (d_{a,1}, \ldots, d_{a,n})$ where $d_{a,i} \in G$ for $1 \le i \le n$, executes as follows:

- Select $d_{a,0} \overset{R}{\leftarrow} G$ (for a security purpose that will be elaborated later).
- For $0 \le i \le n$, select $r_{i1}, r_{i2} \overset{R}{\leftarrow} \mathbb{Z}_p$ and compute
$$\mathsf{cph}_{a,i} = (g^{r_{i2}}, g^{\gamma r_{i1}}, s_i g^{\beta(r_{i1}+r_{i2})}).$$
- For $0 \le i \le n$, let $T_i = H(e(d_{a,i}, g))$ and compute $\mathsf{acDig}_a \leftarrow \mathsf{acGen}(\mathsf{acPk}, \{T_0, \ldots, T_n\})$.
- Set $\mathsf{C}_a = \{\mathsf{cph}_{a,0}, \ldots, \mathsf{cph}_{a,n}\}$ and $\mathsf{si}_a = \mathsf{acDig}_a$.

Similarly, Bob, with $\mathsf{D}_b = (d_{b,1}, \ldots, d_{b,m})$ where $d_{b,i} \in G$ for $1 \le i \le m$, can obtain $\mathsf{C}_b = \{\mathsf{cph}_{b,0}, \ldots, \mathsf{cph}_{b,m}\}$ and $\mathsf{si}_b = \mathsf{acDig}_b$.

Dec$(\mathsf{sk}_a, \mathsf{rslt}_a)$: Given the cloud-generated ciphertext intersection set $\mathsf{rslt}_a = \{\mathsf{cph}_{a,j}, \ldots, \mathsf{cph}_{a,k}\}$ where $1 \le j, k \le n$, Alice decrypts ciphertexts $\mathsf{cph}_{a,i}$ for $j \le i \le k$ as follows:

$$d_{a,i} = d_{a,i} g^{\beta_a(r_{i1}+r_{i2})})/(g^{r_{i2}})^{\beta_a}(g^{\gamma_a r_{i1}})^{\beta_a/\gamma_a}.$$

The decryption of rslt_a is $\mathsf{D}_a \cap \mathsf{D}_b = \{d_{a,j}, \ldots, d_{a,k}\}$. Note that this algorithm can also be used to decrypt C_a *without* involving any delegated set operations. In this case, the integrity of C_a can be easily assured by acDig_a since the plaintexts of C_a should be accumulated to acDig_a.

Similarly, Bob can decrypt the cloud-generated ciphertext intersection set $\mathsf{rslt}_b = \{\mathsf{cph}_{b,j}, \ldots, \mathsf{cph}_{b,k}\}$ where $1 \le j, k \le m$ to obtain $\mathsf{D}_a \cap \mathsf{D}_b$.

AuGen$(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b)$: Given private key sk_a, Alice generates re-key $\mathsf{rk}_a = (g^{\beta_a/\gamma_a})$. Alice encrypts the secret information si_a using Bob's public key pk_b to obtain ciphertext $\mathsf{cph}_B = (g^{r_2}, g^{\gamma_b r_1}, \mathsf{acDig}_a g^{\beta_b(r_1+r_2)})$, where $r_1, r_2 \overset{R}{\leftarrow} \mathbb{Z}_p$. Then, Alice runs $\sigma_a \leftarrow \mathsf{sigSign}(\mathsf{sigSk}_a, \mathsf{cph}_B)$ to obtain a signature σ_a on message cph_B. Finally, Alice sets $\mathsf{au}_a = (\mathsf{rk}_a, \mathsf{cph}_B, \sigma_a)$.

Similarly, Bob can generate $\mathsf{au}_b = (\mathsf{rk}_b, \mathsf{cph}_A, \sigma_b)$.

SetOp$(\mathsf{C}_a, \mathsf{au}_a, \mathsf{C}_b, \mathsf{au}_a)$: Given $\mathsf{C}_a = \{\mathsf{cph}_{a,0}, \ldots, \mathsf{cph}_{a,n}\}$, $\mathsf{C}_b = \{\mathsf{cph}_{b,0}, \ldots, \mathsf{cph}_{b,m}\}$, $\mathsf{au}_a = (\mathsf{rk}_a = g^{\beta_a/\gamma_a}, \mathsf{cph}_B, \sigma_a)$, and $\mathsf{au}_b = (\mathsf{rk}_b = g^{\beta_b/\gamma_b}, \mathsf{cph}_A, \sigma_b)$, the cloud executes as follows:

- Transform ciphertexts $\mathsf{cph}_{a,i}$ for $0 \le i \le n$ into
$$T_{a,i} = \frac{e(d_{a,i} g^{\beta_a(r_{i1}+r_{i2})}, g)}{e(g^{\gamma_a r_{i1}}, g^{\beta_a/\gamma_a}) e(g^{r_{i2}}, g^{\beta_a})} = e(d_{a,i}, g),$$
and compute $T_a = \{H(T_{a,0}), \ldots, H(T_{a,n})\}$.
- Transform ciphertexts $\mathsf{cph}_{b,i}$ for $1 \le i \le m$ into
$$T_{b,i} = \frac{e(d_{b,i} g^{\beta_b(r_{i1}+r_{i2})}, g)}{e(g^{\gamma_b r_{i1}}, g^{\beta_b/\gamma_b}) e(g^{r_{i2}}, g^{\beta_b})} = e(d_{b,i}, g)$$
and compute $T_b = \{H(T_{b,0}), \ldots, H(T_{b,m})\}$.
- Generate the intersection set rslt_a and a proof with respect to C_a as follows: Run $(\mathsf{acRslt}, \mathsf{acWit}_a) \leftarrow \mathsf{acProve}(\mathsf{acPk}, T_a, T_b)$ and set
$$\mathsf{rslt}_a = \{\mathsf{cph}_{a,i} | H(A_{a,i}) \in \mathsf{acRslt}\},$$
$$\mathsf{proof}_a = (\mathsf{acWit}_a, \mathsf{cph}_A, \sigma_b).$$
- Generate the intersection set rslt_b and a proof with respect to C_b as follows: Run $(\mathsf{acRslt}, \mathsf{acWit}_b) \leftarrow \mathsf{acProve}(\mathsf{acPk}, T_b, T_a)$ and set
$$\mathsf{rslt}_b = \{\mathsf{cph}_{b,i} | H(A_{b,i}) \in \mathsf{acRslt}\},$$
$$\mathsf{proof}_b = (\mathsf{acWit}_b, \mathsf{cph}_B, \sigma_a).$$

Verify$(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a)$: Given rslt_a and proof_a, Alice verifies that the cloud faithfully executed the SetOp protocol as follows:

- Verify the integrity of cph_A by running $\mathsf{sigVerify}(\mathsf{sigPk}_b, \ \mathsf{cph}_A, \ \sigma_b)$. If it outputs 0, then return 0; otherwise, proceed to next step.
- Decrypt cph_A using private key sk_a according to
$$\mathsf{acDig}_b = \mathsf{acDig}_b g^{\beta_a(r_1+r_2)})/(g^{r_2})^{\beta_a}(g^{\gamma_a r_1})^{\beta_a/\gamma_a}.$$
- If rslt_a is not empty, decrypt rslt_a to obtain the plaintexts and compute $Y_a = \{e(d_{a,i}, g) | \mathsf{cph}_{a,i} \in \mathsf{rslt}_a\}$. Otherwise, let $Y_a = \emptyset$.
- Run $\mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_a, Y_a, \mathsf{acWit}_a, \mathsf{acDig}_b)$. If it outputs 0, then return 0; otherwise, return 1.

If the algorithm returns 1, **Dec**$(\mathsf{sk}_a, \mathsf{rslt}_a)$ is called to obtain $\mathsf{D}_a \cap \mathsf{D}_b$.

Similarly, Bob can run the same algorithm to verify that the cloud does not cheat.

Remark: why using $d_{a,0}$ and $d_{b,0}$? Since Alice needs to know the accumulator digest acDig_b for the sake of verifying the correctness of rslt_a, we need to assure that Alice cannot use acDig_b to infer useful information about D_b. This is achieved by "blending" the accumulator digest with the randomness, namely the hash value of the randomly selected $d_{b,0}$. This eliminates the usage of zero-knowledge proofs [45], [46], while assuring no useful information is leaked.

Remark. Our VDSI scheme only offers coarse-grained access control in the following sense. Suppose Alice and Bob allow the cloud to conduct the delegated set intersection operation on C_a and C_b, and Alice and Carlos allow the cloud to conduct the delegated set intersection operation on C_a and C_c. Then, the cloud is able to conduct the set intersection operation on C_b and C_c without the authorization from Bob and Carlos. It is a future work to enforce fine-grained access control in VDSI.

5.2 Security Analysis

Correctness of the VDSI scheme can be examined easily. In what follows, we focus on its security properties.

Theorem 2: Under the DL assumption, the scheme achieves *outsourced data secrecy* (Definition 2).

Proof: The proof strategy is: we first show that the VDSI scheme achieves *outsourced data secrecy* when the challenge data set $|\mathsf{D}_0| = |\mathsf{D}_1| = 1$, which is then used as a "building-block" to show that the VDSI scheme achieves *outsourced data secrecy* when $|\mathsf{D}_0| = |\mathsf{D}_1| = n$.

First, we show that given $|\mathsf{D}_0| = |\mathsf{D}_1| = 1$, the VDSI scheme achieves *outsourced data secrecy* under the DL assumption. To prove that, we show that if there is a probabilistic polynomial-time adversary $\mathcal{A}$ that can break *outsourced data secrecy* with a non-negligible probability, then there is an algorithm $\mathcal{B}$ that can break the DL assumption with a non-negligible probability.

Suppose $\mathcal{B}$ is given a DL instance $(f, g, h, g^{r_1}, f^{r_2}, Q)$, where $f, g, Q \xleftarrow{R} G$, $r_1, r_2 \xleftarrow{R} \mathbb{Z}_p$ and r_1, r_2 are unknown to $\mathcal{B}$. Now $\mathcal{B}$ can simulate the game as follows:

Setup: $\mathcal{B}$ treats $f = g^\gamma$ and $h = g^\beta$ for some unknown γ and β, runs $(\mathsf{acPk}, \mathsf{acSk}) \leftarrow \mathsf{acKeyGen}(1^\ell)$ and $(\mathsf{sigPk}, \mathsf{sigSk}) \leftarrow \mathsf{sigKeyGen}(1^\ell)$, sets $\mathsf{pm} = (\mathsf{acPk}, e, p, g, G, G_T)$ and $\mathsf{pk} = (\mathsf{sigPk}, f, g, h)$ and $\mathsf{sk} = (\mathsf{sigSk})$, and finally sends public key pk and public parameter pm to adversary $\mathcal{A}$.

Phase 1: $\mathcal{A}$ can query the following oracle polynomially-many (in ℓ) times:

- $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pk}, \mathsf{D})$: Given $\mathsf{D} = \{d_1, \ldots, d_n\}$, $\mathcal{B}$ encrypts d_i as $\mathsf{cph}_i = (g^{r_{i2}}, f^{r_{i1}}, d_i h^{(r_{i1}+r_{i2})})$ where $r_{i1}, r_{i2} \xleftarrow{R} \mathbb{Z}_p$, sets $\mathsf{C} = \{\mathsf{cph}_1, \ldots, \mathsf{cph}_n\}$, runs $\mathsf{acDig} \leftarrow \mathsf{acGen}(\mathsf{acPk}, \{H(e(d_i, g)), \ldots, H(e(d_n, g))\})$, and returns C to $\mathcal{A}$.

Challenge: $\mathcal{A}$ outputs two data sets $(\mathsf{D}_0 = \{d_0\}, \mathsf{D}_1 = \{d_1\})$ where $|\mathsf{D}_0| = |\mathsf{D}_1| = 1$ and $\mathsf{D}_0 \neq \mathsf{D}_1$. $\mathcal{B}$ selects $\lambda \xleftarrow{R} \{0, 1\}$, and computes $\mathsf{cph}_\lambda = (g^{r_1}, f^{r_2}, d_\lambda Q)$. Let cph be the ciphertext of a selected random value unkown to $\mathcal{A}$, $\mathsf{C} = \{\mathsf{cph}, \mathsf{cph}_\lambda\}$, $\mathsf{acDig} \leftarrow \mathsf{acGen}(\mathsf{acPk}, \{H(e(d_\lambda, g))\})$. $\mathcal{B}$ returns C to $\mathcal{A}$.

Phase 2: $\mathcal{A}$ can query the oracle the same as Phase 1.

Guess: Lastly, $\mathcal{A}$ outputs λ'. If $\lambda' = \lambda$, then the challenge outputs $h^{(r_1+r_2)} = Q$; otherwise, it outputs $h^{(r_1+r_2)} \neq Q$.

We can see that if $Q = h^{(r_1+r_2)}$, then cph_λ is the valid ciphertext. In this case the probability of $\mathcal{A}$ outputting $\lambda = \lambda'$ is $\frac{1}{2} + \mu$. If Q is a random element from G, then the probability of $\mathcal{A}$ outputting $\lambda = \lambda'$ is $\frac{1}{2}$. Therefore, the probability of $\mathcal{B}$ correctly guessing $Q \overset{?}{=} h^{(r_1+r_2)}$ is $\frac{1}{2}(\frac{1}{2} + \mu + \frac{1}{2}) = \frac{1}{2} + \frac{\mu}{2}$. That is, given the challenge data sets $|\mathsf{D}_0| = |\mathsf{D}_1| = 1$, if $\mathcal{A}$ can break outsourced data secrecy of the proposed scheme with non-negligible advantage μ, then there exists a challenger breaking the DL assumption with non-negligible advantage $\frac{\mu}{2}$.

In what follows we show that given arbitrary size of data sets, e.g. $|\mathsf{D}_0| = |\mathsf{D}_1| = n$ (or m), the proposed scheme achieves outsourced data secrecy. Given the data sets $\mathsf{D}_0 = (d_1, ..., d_n)$ and $\mathsf{D}_1 = (d'_1, ..., d'_n)$ from adversary $\mathcal{A}$, let $\mathsf{C}^{(i)}$ denote the encryption form of $(d_1, \ldots, d_i, d'_{i+1}, \ldots, d'_n)$, so that $\mathsf{C}^{(n)}$ is the encryption form of D_0 and $\mathsf{C}^{(0)}$ is the encryption form of D_1. We simulate the challenge phase with an additional adversary $\mathcal{A}'$ as follows:

- $\mathcal{A}'$ selects an index $i \xleftarrow{R} [1, n]$ and presents (d_i, d'_i) to $\mathcal{B}$. The challenger returns cph_i by encrypting d_i if $\lambda = 0$ and d'_i otherwise.
- $\mathcal{A}'$ encrypts $(d_1, \ldots, d_{i-1})$ and $(d'_{i+1}, \ldots, d'_n)$, and sends $(\mathsf{cph}_1, \ldots, \mathsf{cph}_n)$ to $\mathcal{A}$. $\mathcal{A}'$ outputs λ' that is output by $\mathcal{A}$.

We can see that $\mathcal{A}'$ sends to $\mathcal{A}$ the ciphertexts $\mathsf{C}^{(i)}$ when $\lambda = 0$ and $\mathsf{C}^{(i-1)}$ when $\lambda = 1$. Now let's consider the probability of $\mathcal{A}'$ winning the security game, and have (we denote as $\mathcal{A}(\mathsf{C}^{(i)})$ the guess of $\mathcal{A}$ with ciphertexts $\mathsf{C}^{(i)}$)

$$\begin{aligned}
\Pr[\mathcal{A}'\text{outputs } 0|\lambda = 0] &= \sum_{i=1}^{n} \frac{1}{n} \Pr[\mathcal{A}(\mathsf{C}^{(i)}) = 0] \\
\Pr[\mathcal{A}'\text{outputs } 1|\lambda = 1] &= \sum_{i=1}^{n} \frac{1}{n} \Pr[\mathcal{A}(\mathsf{C}^{(i-1)}) = 1]
\end{aligned}$$

Therefore, the probability of $\mathcal{A}'$ winning the selective security game is

$$\begin{aligned}
&\frac{1}{2}\Pr[\mathcal{A}'\text{outputs } 0|\lambda = 0] + \frac{1}{2}\Pr[\mathcal{A}'\text{outputs } 1|\lambda = 1] \\
&= \sum_{j=1}^{n} \frac{1}{2n} \Pr[\mathcal{A}(\mathsf{C}^{(j)}) = 0] + \sum_{j=1}^{n} \frac{1}{2n} \Pr[\mathcal{A}(\mathsf{C}^{(j-1)}) = 1] \\
&= \frac{n}{2n} + \frac{1}{2n} \Pr[\mathcal{A}(\mathsf{C}^{(n)}) = 0] + \frac{1}{2n} \Pr[\mathcal{A}(\mathsf{C}^{(0)}) = 1] \\
&= \frac{n}{2n} + \frac{1}{n}(\frac{1}{2} \Pr[\mathcal{A}(\mathsf{C}^{(n)}) = 0] + \frac{1}{2} \Pr[\mathcal{A}(\mathsf{C}^{(0)}) = 1]) \\
&\leq \frac{1}{2} + \epsilon
\end{aligned}$$

Here ϵ is negligible probability of the advantage of $\mathcal{A}'$ winning the security game to guess λ. Therefore, the probability of $\mathcal{A}$ distinguishing $\mathsf{C}^{(0)}$ and $\mathsf{C}^{(n)}$ is

$$\frac{1}{2}\Pr[\mathcal{A}(\mathsf{C}^{(n)}) = 0] + \frac{1}{2}\Pr[\mathcal{A}(\mathsf{C}^{(0)}) = 1] \leq \frac{1}{2} + n\epsilon$$

That is, the advantage of $\mathcal{A}$ distinguishing $\mathsf{C}^{(0)}$ and $\mathsf{C}^{(n)}$ is at most $n\epsilon$, which is negligible with respect to ℓ. Therefore, we show that the proposed scheme achieves outsourced data secrecy under the DL assumption. $\square$

Theorem 3: Given that the bilinear map e is one-way, the VDSI scheme achieves the function output secrecy property (Definition 3).

Proof: We show that given a ciphertext and corresponding re-key, any probabilistic polynomial-time adversary $\mathcal{A}$ infers the plaintext with the probability $\frac{q}{|\mathcal{M}|} + \epsilon$ at most if $\mathcal{A}$ has q times to guess the plaintexts, where $\mathcal{M}$ is the plaintext space.

Setup: The challenger $\mathcal{B}$ runs $\mathsf{pm} \leftarrow \mathsf{Setup}(1^\ell)$ and makes pm publicly known.

Challenge: The challenger runs $\mathsf{KeyGen}(\mathsf{pm})$ to obtain $\mathsf{sk}_a = (\mathsf{sigSk}_a, \beta_a, \gamma_a)$ and $\mathsf{pk}_a = (\mathsf{sigPk}_a, g^{\beta_a}, g^{\gamma_a})$ for Alice, and runs $\mathsf{KeyGen}(\mathsf{pm})$ to obtain $\mathsf{sk}_b = (\mathsf{sigSk}_b, \beta_b, \gamma_b)$, $\mathsf{pk}_b = (\mathsf{sigPk}_b, g^{\beta_b}, g^{\gamma_b})$ for Bob. $\mathcal{B}$ selects two data sets $\mathsf{D}_b, \mathsf{D}_b$, elements of which are randomly selected from the plaintext space $\mathcal{M}$, then runs $(\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a)$ and $(\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b)$, $\mathsf{au}_a \leftarrow \mathsf{AuGen}$ $(\mathsf{sk}_a, \mathsf{si}, \mathsf{pk}_b)$ and $\mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a)$, and returns C_a, C_b, pk_a, pk_b, au_a, au_b to $\mathcal{A}$.

Phase 1: $\mathcal{A}$ can query the following oracles polynomially many times.

- $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pk}_a, \mathsf{D}'_a)$: Given the data set D'_a, the challenger runs $(\mathsf{C}'_a, \mathsf{si}'_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}'_a)$ and returns $\mathsf{C}'_a, \mathsf{si}'_a$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pk}_b, \mathsf{D}'_b)$: Given the data set D'_b, the challenger runs $(\mathsf{C}'_b, \mathsf{si}'_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}'_b)$ and returns $\mathsf{C}'_b, \mathsf{si}'_b$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{Verify}}(\mathsf{pk}_a, \mathsf{rslt}_a, \mathsf{proof}_a)$: The challenger runs $\mathsf{Verify}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a)$ and returns the output to $\mathcal{A}$, where si_a is the secret information with respect to the data set D_a .
- $\mathcal{O}_{\mathsf{Verify}}(\mathsf{pk}_b, \mathsf{rslt}_b, \mathsf{proof}_b)$: The challenger runs $\mathsf{Verify}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{rslt}_b, \mathsf{proof}_b)$ and returns the output to $\mathcal{A}$, where si_b is the secret information with respect to the data set D_b .

Guess: The challenger selects a ciphertext cph from $\mathsf{C}_a \cup \mathsf{C}_b$ uniformly at random. $\mathcal{A}$ has q times to guess the keyword with respect to cph.

We can see that given the ciphertexts C_a, C_b and $\mathsf{au}_a, \mathsf{au}_b$, $\mathcal{A}$ can only get the values $e(d_{a,i}, g)$ for $d_{a,i} \in \mathsf{D}_a$ and $e(d_{b,i}, g)$ for $d_{b,i} \in \mathsf{D}_b$. Therefore, as the bilinear map e is an one-way function, $\mathcal{A}$ inverts the plaintext from the pairing value with negligible probability ϵ. The only way of inferring the plaintext with respect to cph is with brute-force method by enumerating possible elements within the plaintext space. Hence, given that A has q times to guess plaintexts, the probability of $\mathcal{A}$ outputting correct plaintext is $\frac{q}{|\mathcal{M}|} + \epsilon$. □

Theorem 4: Assume that Sig is an unforgeable signature scheme, Ac is a secure multi-accumulator scheme and H is a collision resistance hash function, the VDSI scheme achieves the verifiability property (Definition 4).

Proof: We show that if there exists a probabilistic polynomial-time adversary $\mathcal{A}$ breaking the verifiability of the VDSI scheme (i.e. presenting an incorrect intersection result and succeeding in the verification with non-negligible probability), there exists an algorithm $\mathcal{B}$ breaking the assumptions that Sig is a secure signature scheme, Ac is a secure multi-accumulator scheme or H is a collision resistant hash function. $\mathcal{B}$ proceeds as follows.

Setup: $\mathcal{B}$ runs $\mathsf{pm} \leftarrow \mathsf{Setup}(1^\ell)$ and makes pm public known. It then runs $\mathsf{KeyGen}(\mathsf{pm})$ to obtain $\mathsf{sk}_a = (\mathsf{sigSk}_a, \beta_a, \gamma_a)$ and $\mathsf{pk}_a = (\mathsf{sigPk}_a, g^{\beta_a}, g^{\gamma_a})$, runs $\mathsf{KeyGen}(\mathsf{pm})$ to obtain $\mathsf{sk}_b = (\mathsf{sigSk}_b, \beta_b, \gamma_b)$, $\mathsf{pk}_b = (\mathsf{sigPk}_b, g^{\beta_b}, g^{\gamma_b})$, and returns $\mathsf{pk}_a, \mathsf{pk}_b$ to $\mathcal{A}$.

Phase 1: $\mathcal{A}$ can query the following oracles polynomially many times.

- $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pk}_a, \mathsf{D}_a)$: Given the data set D_a, $\mathcal{B}$ runs $(\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a)$ and returns $\mathsf{C}_a, \mathsf{si}_a$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pk}_b, \mathsf{D}_b)$: Given the data set D_b, $\mathcal{B}$ runs $(\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b)$ and returns $\mathsf{C}_b, \mathsf{si}_b$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{AuGen}}(\mathsf{pk}_a, \mathsf{D}_a, \mathsf{pk}_b)$: $\mathcal{B}$ runs $\mathsf{au}_a \leftarrow \mathsf{AuGen}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b)$ and returns au_a to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{AuGen}}(\mathsf{pk}_b, \mathsf{D}_b, \mathsf{pk}_a)$: $\mathcal{B}$ runs $\mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a)$ and returns au_a to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{Verify}}(\mathsf{pk}_a, \mathsf{rslt}_a, \mathsf{proof}_a)$: $\mathcal{B}$ runs $\mathsf{Verify}(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{rslt}_a, \mathsf{proof}_a)$ and returns the output to $\mathcal{A}$, where si_a is the secret information with respect to the data set D_a .
- $\mathcal{O}_{\mathsf{Verify}}(\mathsf{pk}_b, \mathsf{rslt}_b, \mathsf{proof}_b)$: $\mathcal{B}$ runs $\mathsf{Verify}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{rslt}_b, \mathsf{proof}_b)$ and returns the output to $\mathcal{A}$, where si_b is the secret information with respect to the data set D_b .

Challenge: $\mathcal{A}$ selects $\mathsf{D}_a, \mathsf{D}_b$ of its choice, and sends them to $\mathcal{B}$. $\mathcal{B}$ runs $(\mathsf{C}_a, \mathsf{si}_a) \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{D}_a)$ and $(\mathsf{C}_b, \mathsf{si}_b) \leftarrow \mathsf{Enc}(\mathsf{pk}_b, \mathsf{D}_b)$, $\mathsf{au}_a \leftarrow \mathsf{AuGen}$ $(\mathsf{sk}_a, \mathsf{si}_a, \mathsf{pk}_b)$ and $\mathsf{au}_b \leftarrow \mathsf{AuGen}(\mathsf{sk}_b, \mathsf{si}_b, \mathsf{pk}_a)$, and returns $\mathsf{C}_a, \mathsf{si}_a, \mathsf{au}_a, \mathsf{C}_b, \mathsf{si}_b, \mathsf{au}_b$ to $\mathcal{A}$, where $\mathsf{si}_a = (\mathsf{acDig}_a)$ and $\mathsf{si}_b = (\mathsf{acDig}_b)$.

Phase 2: $\mathcal{A}$ can query the oracles the same as Phase 1.

Guess: $\mathcal{A}$ outputs $(\mathsf{rslt}_a, \mathsf{proof}_a), (\mathsf{rslt}_b, \mathsf{proof}_b)$ to $\mathcal{B}$.

This completes the simulation. First let us consider the verification for $(\mathsf{rslt}_a, \mathsf{proof}_a)$. Note that cph_A specifed by proof_a cannot be manipulated, otherwise it breaks the unforgeability of Sig. B decrypts cph_A to obtain acDig_b. In addition, $\mathcal{B}$ decrypts $\mathsf{Dec}(\mathsf{sk}_a, \mathsf{rslt}_a)$, and obtains $T_a = \{H(e(d'_{a,i}, g)) | \mathsf{cph}_{a,i} \in \mathsf{rslt}_a\}$ where $d'_{a,i}$ is the plaintext with respect to $\mathsf{cph}_{a,i}$.

Suppose $T = \{H(e(d_{a,i}, g)) | d_{a,i} \in \mathsf{D}_a\} \cap \{H(e(d_{b,i}, g)) |\ d_{b,i} \in \mathsf{D}_b\}$. If $\mathcal{A}$ breaks the verifiability with $(\mathsf{rslt}_a, \mathsf{proof}_a)$, then at least one of the following cases should hold:

Case 1:

$$\begin{aligned}
&1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_a, T_a, \mathsf{acWit}_a, \mathsf{acDig}_b)\\
&1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_a, T, \mathsf{acWit}_a, \mathsf{acDig}_b)\\
&T_a = T\\
&\exists d'_{a,i} \neq d_{a,i}, s.t. H(e(d'_{a,i}, g)) = H(e(d_{a,i}, g))
\end{aligned}$$

Case 2:

$$\begin{aligned}
&1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_a, T_a, \mathsf{acWit}_a, \mathsf{acDig}_b)\\
&1 \leftarrow \mathsf{acVerify}(\mathsf{acPk}, \mathsf{acDig}_a, T, \mathsf{acWit}_a, \mathsf{acDig}_b)\\
&T \neq T_a
\end{aligned}$$

If $\mathcal{A}$ breaks the verifiability with $(\mathsf{rslt}_a, \mathsf{proof}_a)$ with respect to case 1, then it breaks the assumption that H is collision resistant. This is because $d'_{a,i} \neq d_{a,i}$ leads to $e(d'_{a,i}, g)) \neq e(d_{a,i}, g)$ while $H(e(d'_{a,i}, g)) = H(e(d_{a,i}, g))$.

If $\mathcal{A}$ breaks the verifiability with $(\mathsf{rslt}_a, \mathsf{proof}_a)$ with respect to case 2, then it breaks the security of the multi-accumulator scheme by presenting $\mathsf{acRslt} = T_a$, which is different from T.

Therefore, we prove that $\mathcal{A}$ breaks the verifiability of VDSI scheme with respect to $(\mathsf{rslt}_a, \mathsf{proof}_a)$ in a negligible probability under the assumptions that Sig is unforgeable, H is collision resistant and Ac is a secure multi-accumulator scheme. Similarly, we can prove that $\mathcal{A}$ breaks the verifiable of VDSI scheme with respect to $(\mathsf{rslt}_b, \mathsf{proof}_b)$ in a negligible probability. □

TABLE 3
Asymptotic complexity for VDSI scheme, where Exp denotes the exponentiation operation, Pairing denotes the pairing operation, $n = |\mathsf{acD}_a|$, $m = |\mathsf{acD}_b|$ and $k = |\mathsf{D}_a \cap \mathsf{D}_b|$,

Algorithm	Computational Complexity
Enc	$3n\mathsf{Exp} + n\mathsf{Pairing} + \mathsf{acGen}$
Dec	$2n\mathsf{Exp}$
AuGen	$4\mathsf{Exp} + \mathsf{sigSign}$
SetOp	$3(n+m)\mathsf{Pairing} + 2\mathsf{acProve}$
Verify	$2(k+1)\mathsf{Exp} + k\mathsf{Pairing} + \mathsf{acVerify} + \mathsf{sigVerify}$

6 Performance Evaluation

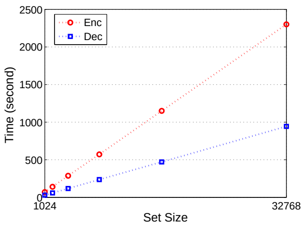
6.1 Asymptotic Complexity

Table 3 describes the asymptotic complexity of algorithms in the VDSI scheme. While algorithm SetOp is more costly when compared with the other algorithms, it is worth noting that algorithm SetOp is executed by the cloud rather than by the users.

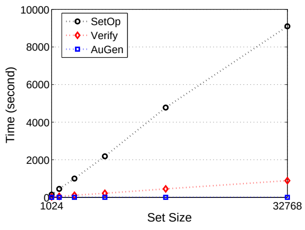
Table 4 summarizes the communication and computational overhead incurred by the VDSI solution, which is grouped into two phases: (i) data outsourcing phase, during which the cloud users outsource their encrypted private data sets to the cloud (i.e. Enc); (ii) set operation phase, during which the cloud users attain the intersection set (i.e. AuGen, SetOp and Verify). We compare the overhead with its counterpart that is incurred by the straightforward solution, namely that Alice and Bob download their outsourced encrypted data from the cloud, decrypt their data, and then run a PSI protocol to jointly compute the intersection set. From the perspective of cloud users, we observe that the VDSI scheme outperforms the straightforward solution in both communication and computational complexities. Assume that the data sets have been stored in the cloud, the VDSI scheme only incurs $O(k)$ computational complexity to obtain the intersection set (including the cost for verification) and $O(k)$ communication complexity for returning the intersection set to the user, where k is the size of intersection set. This means that the VDSI scheme is optimal (up to a constant factor). In contrast, the straightforward solution incurs $O(m+n)$ in computational and communication overhead where m and n are the sizes of the two data sets. The advantage of VDSI scheme is most substantial when $k << m$ or $k << n$.

6.2 Performance Evaluation

Implementation We implemented the VDSI scheme in JAVA based on the Java Pairing Based Cryptography library (jPBC) [47]. In our implementation, we instantiated the bilinear map with Type A pairing ($\ell = 512$), which offers a level of security that is comparable to 1024-bit DLOG [47]. We instantiated the signature scheme with the DSA signature scheme provided by JDK1.6. We varied the set size (m and n) from 2^{10} to 2^{15}. The algorithms run by the cloud users (i.e., Enc, Dec, AuGen and Verify) were executed on a CLIENT MACHINE with Linux OS, 2.93GHz Intel Core Duo CPU (E7500), and 2GB RAM. The algorithm run by the cloud (i.e., SetOp) was executed on a SERVER MACHINE with Linux OS, 4 processors of 2.40GHz Intel Xeon CPU, and 8GB RAM.



(a) Performance of Enc and Dec



(b) Performance of AuGen, SetOp and Verify

Fig. 2. Performance of VDSI, where Alice and Bob outsource their data sets of the same size (i.e., $m = n$), algorithms Enc, Dec, AuGen and Verify run on the CLIENT MACHINE, and algorithm SetOp runs on the SERVER MACHINE.

Evaluation and result In our experiments we set the same size for data sets owned by the two cloud users, i.e., $m = n$, and set the size of intersection set $k = n/2$. For algorithms Enc, Dec, AuGen and Verify, we evaluated each algorithm's execution time for both cloud users and treat their average execution time as the real execution time. Figure 2(a) plots the execution time of Enc and Dec that are run by the cloud users. We observe that the execution times for both algorithms are almost linear to the size of data sets. We also can see that the execution of Enc is more expensive than that of Dec. However, Enc is executed only once when the cloud user outsources data sets. Figure 2(b) shows the execution time of SetOp (run by the cloud) and the execution time of AuGen and Verify (run by the cloud users). We observe that the execution time of AuGen and Verify is much more smaller than that of the algorithm SetOp. This suggests that cloud users should leverage the cloud's computation resources by delegating set intersection operations.

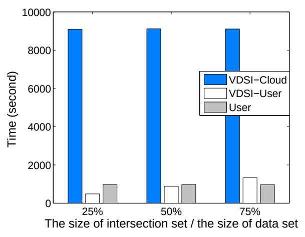
Performance Comparison In order to understand the benefit and limitation of the VDSI solution, we compare it with the straightforward solution, where data sets are encrypted by the algorithm Enc and Dec of the VDSI, and the private set operations between two data users are performed by the protocol (Java version) in [16], which is the most efficient PSI protocol in the literature.

We ran the experiments on the same SERVER MACHINE with Linux OS, 4 processors of 2.40GHz Intel Xeon CPU, and 8GB RAM, with the data sets each consisting of 32768

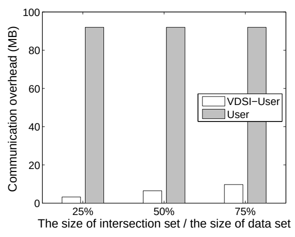
TABLE 4
Asymptotic performance comparison for the VDSI scheme and the straightforward solution. We assume that the straightforward solution adopting the encryption and decryption algorithms of the VDSI scheme. Here n is the size of Alice's data set, m is the size of Bob's data set, k is the size of set intersection, and Comp(PSI) and Comm(PSI) denotes the respective computation and communication complexity of the private set intersection protocol. Note that Comp(PSI) and Comm(PSI) are both linear to the size of data sets $(m+n)$ for the state-of-the-art solution [16].

		VDSI solution				Straightforward solution		
Phase		Alice	Bob	Cloud		Alice	Bob	Cloud
Data outsourcing	Computation	$O(n)$	$O(m)$	N/A		$O(n)$	$O(m)$	N/A
	Communication	$O(n)$	$O(m)$	$O(n+m)$		$O(n)$	$O(m)$	$O(n+m)$
Set operation	Computation	$O(k)$	$O(k)$	$O(m+n)$		$O(n)$+Comp(PSI)	$O(m)$+Comp(PSI)	N/A
	Communication	$O(k)$	$O(k)$	$O(k)$		$O(n)$ +Comm(PSI)	$O(m)$ +Comm(PSI)	$O(n+m)$

elements. We vary the size of the intersection set as 25%, 50% and 75% of the size of the data set respectively, and compare the communication and computation overhead in the set operation phase (we did not compare the cost of the data outsourcing phase because they are the same for both solutions), which is shown in Figure 3. From Figure 3(a) we observe that the computation overhead in the VDSI solution decreases when the size of the intersection set decreases. However, the computation overhead in the straightforward solution remains the same regardless the size of the intersection set. We also can see in Figure 3(b) that the communication overhead for the data users in the VDSI solution is linear to the size of intersection set, and is much less than that of the straightforward solution. This advantage can become more substantial when the size of the intersection size is far less than the size of data set owned by the data users. We note that while the straightforward solution can use other efficient encryption schemes (e.g., symmetric encryption) to encrypt/decrypt data sets to achieve higher efficiency, it cannot reduce the communication cost that is linear to the size of data set. Therefore, our VDSI solution is extremely suitable for computing intersection set whose size is far less than that of the data sets.



(a) Computation overhead.



(b) Communication overhead.

Fig. 3. Performance comparison between our VDSI solution and the straightforward solution, where each data user outsourced the data set of 32768 elements. We vary the size of the intersection set with 25%, 50% and 75% of the size of the data set respectively. VDSI-User and VDSI-Cloud denote the costs spent by the cloud and each data user in the VDSI solution and User represents the cost spent by each data user in the straightforward solution.

6.3 Improvement with Parallelization

In our proposed scheme, the execution of Enc, Dec, SetOp and Verify can be implemented more efficiently with parallelization, because operations related to elements of data sets are independent. In practice, we implemented the algorithms by using multiple threads to compute independent operations (e.g. encrypting elements of the data sets, decrypting ciphertexts, and transforming ciphertexts into a value of G_T). In the parallelization version, we created 4 threads and ran the algorithms Enc, Dec, SetOp and Verify on the SERVER MACHINE with Linux OS, 4 processors of 2.40GHz Intel Xeon CPU, and 8GB RAM. To understand the efficiency gain of parallelization, we also ran the algorithms without parallelization on the same SERVER MACHINE. Figure 4 shows the performance comparison, which indicates that the algorithms using parallelization are about 2 times faster than their counterparts that do not use parallelization. This means that our scheme can leverage the multi-core architecture, and that our scheme is suitable for delegating set intersection over outsourced large data sets.

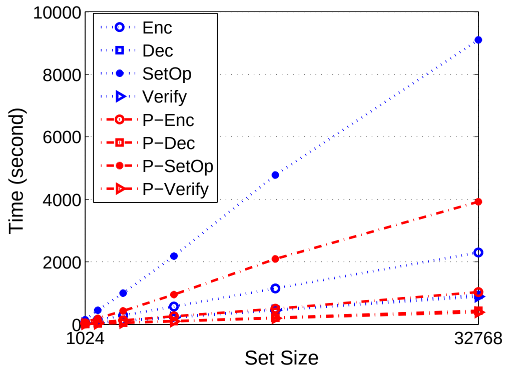


Fig. 4. Performance comparison for algorithms Enc, Dec, AuGen and Verify executed on SERVER MACHINE. The algorithms with prefix "P-" were implemented with parallelization, and the algorithms without prefix were implemented without parallelization.

7 Conclusion

We have introduced the novel notion of VDSI, which allows two users to outsource to the cloud their encrypted data sets as well as the set intersection operation on ciphertexts. This is achieved without giving the cloud the capability to decrypt the encrypted data, while enabling the users to hold the misbehaving cloud accountable.

Our study brings interesting and challenging open problems for future research. In addition to the ones mentioned in the paper (e.g., incorporating fine-grained access control, if possible), we need to design the same kinds of solutions for other set operations.

References

[1] C. Gentry, "Computing arbitrary functions of encrypted data," *Commun. ACM*, vol. 53, no. 3, pp. 97–105, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1666420.1666444

[2] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. New York, NY, USA: ACM, 2012, pp. 309–325. [Online]. Available: http://doi.acm.org/10.1145/2090236.2090262

[3] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 465–482. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29011-4_28

[4] K. Lauter, M. Naehrig, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" Cryptology ePrint Archive, Report 2011/405, 2011, http://eprint.iacr.org/.

[5] P. Baldi, R. Baronio, E. D. Cristofaro, P. Gasti, and G. Tsudik, "Countering gattaca: efficient and secure testing of fully-sequenced human genomes," in *ACM Conference on Computer and Communications Security*, 2011, pp. 691–702.

[6] E. D. Cristofaro and G. Tsudik, "Practical private set intersection protocols with linear computational and bandwidth complexity," *IACR Cryptology ePrint Archive*, vol. 2009, p. 491, 2009.

[7] G. Mezzour, A. Perrig, V. Gligor, and P. Papadimitratos, "Privacy-preserving relationship path discovery in social networks," in *Cryptology and Network Security*, ser. Lecture Notes in Computer Science, vol. 5888. Springer Berlin Heidelberg, 2009, pp. 189–208.

[8] G. Ateniese, E. D. Cristofaro, and G. Tsudik, "(if) size matters: Size-hiding private set intersection," in *Public Key Cryptography*, 2011, pp. 156–173.

[9] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *EUROCRYPT*, 2004, pp. 1–19.

[10] L. Kissner and D. X. Song, "Privacy-preserving set operations," in *CRYPTO*, 2005, pp. 241–257.

[11] J. Camenisch and G. M. Zaverucha, "Private intersection of certified sets," in *Financial Cryptography*, 2009, pp. 108–127.

[12] C. Hazay and Y. Lindell, "Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries," *J. Cryptology*, vol. 23, no. 3, pp. 422–456, 2010.

[13] D. Dachman-Soled, T. Malkin, M. R. 0001, and M. Yung, "Efficient robust private set intersection." in *ACNS*, 2009, pp. 125–142.

[14] C. Hazay and K. Nissim, "Efficient set operations in the presence of malicious adversaries." in *Public Key Cryptography*, 2010, pp. 312–331.

[15] S. Jarecki and X. Liu, "Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection." in *TCC*, 2009, pp. 577–594.

[16] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: An efficient and scalable protocol," Cryptology ePrint Archive, Report 2013/515, 2013, http://eprint.iacr.org/.

[17] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" *19th Network and Distributed Security Symposium*, 2012.

[18] E. D. Cristofaro and G. Tsudik, "Practical private set intersection protocols with linear complexity." in *Financial Cryptography*, 2010, pp. 143–159.

[19] F. Kerschbaum, "Collusion-resistant outsourcing of private set intersection." in *SAC*, 2012, pp. 1451–1456.

[20] ——, "Outsourced private set intersection using homomorphic encryption." in *ASIACCS*, 2012, pp. 85–86.

[21] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian, "Server-aided private set intersection: Scaling to million element sets," 2013, http://research.microsoft.com/pubs/194141/sapsi.pdf.

[22] R. Canetti, O. Paneth, D. Papadopoulos, and N. Triandopoulos, "Verifiable set operations over outsourced databases," Cryptology ePrint Archive, Report 2013/724, 2013, http://eprint.iacr.org/.

[23] G. Yang, C. H. Tan, Q. Huang, and D. S. Wong, "Probabilistic public key encryption with equality test." in *CT-RSA*, 2010, pp. 119–131.

[24] S. Canard, G. Fuchsbauer, A. Gouget, and F. Laguillaumie, "Plaintext-checkable encryption," in *CT-RSA*, 2012, pp. 332–348.

[25] B. Wang, M. Li, S. Chow, and H. Li, "Computing encrypted cloud data efficiently under multiple keys," in *Communications and Network Security (CNS), 2013 IEEE Conference on*, Oct 2013, pp. 504–513.

[26] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev, "Message-locked encryption for lock-dependent messages," in *CRYPTO (1)*, 2013, pp. 374–391.

[27] Q. Tang, "Towards public key encryption scheme supporting equality test with fine-grained authorization." in *ACISP*, 2011, pp. 389–406.

[28] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption," in *TCC*, 2012, pp. 422–439.

[29] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," in *STOC*, 2008, pp. 113–122.

[30] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *CRYPTO*, 2010, pp. 465–482.

[31] K.-M. Chung, Y. T. Kalai, and S. P. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *CRYPTO*, 2010, pp. 483–501.

[32] C. Papamanthou, E. Shi, and R. Tamassia, "Signatures of correct computation," in *TCC*, 2013, pp. 222–242.

[33] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *CRYPTO*, 2011, pp. 111–131.

[34] D. Fiore and R. Gennaro, "Publicly verifiable delegation of large polynomials and matrix computations, with applications," in *Proceedings of the 2012 ACM conference on Computer and communications security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 501–512. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382250

[35] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *EUROCRYPT*, 2013, pp. 626–645.

[36] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," in *ICC*, 2012, pp. 917–922.

[37] B. Thompson, S. Haber, W. G. Horne, T. Sander, and D. Yao, "Privacy-preserving computation and verification of aggregate queries on outsourced databases," in *Privacy Enhancing Technologies*, 2009, pp. 185–201.

[38] L. Zhang, X.-Y. Li, Y. Liu, and T. Jung, "Verifiable private multi-party computation: Ranging and ranking," in *INFOCOM*, 2013, pp. 605–609.

[39] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *In proceedings of CRYPTO 04, LNCS series*. Springer-Verlag, 2004, pp. 41–55.

[40] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, vol. 17, no. 2, pp. 281–308, Apr. 1988. [Online]. Available: http://dx.doi.org/10.1137/0217017

[41] L. Nguyen, "Accumulators from bilinear pairings and applications," in *Proceedings of the 2005 international conference on Topics in Cryptology*, ser. CT-RSA'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 275–292. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30574-3_19

[42] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '02. London, UK, UK: Springer-Verlag, 2002, pp. 61–76. [Online]. Available: http://dl.acm.org/citation.cfm?id=646767.704437

[43] I. Damgård and N. Triandopoulos, “Supporting non-membership proofs with bilinear-map accumulators,” *IACR Cryptology ePrint Archive*, vol. 2008, p. 538, 2008.
[44] C. Papamanthou, R. Tamassia, and N. Triandopoulos, “Optimal verification of operations on dynamic sets,” Cryptology ePrint Archive, Report 2010/455, 2010, http://eprint.iacr.org/.
[45] U. Feige, A. Fiat, and A. Shamir, “Zero-knowledge proofs of identity,” *J. Cryptol.*, vol. 1, no. 2, pp. 77–94, Aug. 1988. [Online]. Available: http://dx.doi.org/10.1007/BF02351717
[46] T. Acar, S. S. M. Chow, and L. Nguyen, “Accumulators and u-prove revocation,” in *Financial Cryptography*, 2013, pp. 189–196.
[47] “The java pairing based cryptography library. http://gas.dia.unisa.it/projects/jpbc/.”