2014

Privacy-Preserving Ad-Hoc Equi-Join on Outsourced Data

Hwee Hwa PANG
Singapore Management University, hhpang@smu.edu.sg

Xuhua DING
Singapore Management University, xhding@smu.edu.sg

DOI: https://doi.org/10.1145/2629501

Privacy-Preserving Ad-Hoc Equi-Join on Outsourced Data

HWEEHWA PANG and XUHUA DING, Singapore Management University

In IT outsourcing, a user may delegate the data storage and query processing functions to a third-party server that is not completely trusted. This gives rise to the need to safeguard the privacy of the database as well as the user queries over it. In this article, we address the problem of running ad hoc equi-join queries directly on encrypted data in such a setting. Our contribution is the first solution that achieves constant complexity per pair of records that are evaluated for the join. After formalizing the privacy requirements pertaining to the database and user queries, we introduce a cryptographic construct for securely joining records across relations. The construct protects the database with a strong encryption scheme. Moreover, information disclosure after executing an equi-join is kept to the minimum—that two input records combine to form an output record if and only if they share common join attribute values. There is no disclosure on records that are not part of the join result.

Building on this construct, we then present join algorithms that optimize the join execution by eliminating the need to match every record pair from the input relations. We provide a detailed analysis of the cost of the algorithms and confirm the analysis through extensive experiments with both synthetic and benchmark workloads. Through this evaluation, we tease out useful insights on how to configure the join algorithms to deliver acceptable execution time in practice.

Categories and Subject Descriptors: H.2.7 [**Database Management**]: Security, Integrity, and Protection

General Terms: Security, Algorithms, Performance

Additional Key Words and Phrases: Data and query privacy, equi-join, query over encrypted data

1. INTRODUCTION

IT outsourcing including cloud computing and database-as-a-service offers many advantages like enhanced data availability, flexibility in provisioning resources according to usage demand, and economy of scale. Nevertheless, adoption has been slowed by persistent concerns over control and protection of data, as well as conflicting legislative frameworks (e.g., U.S. Patriot Art versus EU Data Protection Art). In a recent high-profile case, *Computer Weekly* [Saran 2011] reported that defense contractor BAE Systems aborted plans to adopt Microsoft Office 365, due to concerns that the Patriot Act would prevent Microsoft from guaranteeing that BAE's data would not leave Europe.

The concerns over data control and protection may be mitigated if data leave customers' premises only in encrypted form. While a small dataset may be encrypted and retrieved in its entirety from an outsourced server each time some data are used, doing so is impractical with large relational databases. Rather, a database system on the

This work is funded in the Singapore Management University through the research grant 13-C220-SMU-004 from the Ministry of Education Academic Research Fund Tier 1.
Authors' addresses: H. Pang (corresponding author) and X. Ding, School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902; email: hhpang@smu.edu.sg.

DOI: http://dx.doi.org/10.1145/2629501

server needs to support ad hoc queries to select and compose data from an encrypted database.

In this article, we focus on the problem of executing ad hoc equi-join queries on an outsourced database while preserving the privacy of the queries and data. An example of equi-join is the following query to retrieve back-to-back appointments.

SELECT * FROM APPT **R**, APPT **S**
WHERE $\mathbf{R}.StartTime = \mathbf{S}.EndTime$

Following the database-as-a-service model (adopted in many studies like Hacigumus et al. [2002] and Mykletun and Tsudik [2006]), our system setting includes a User and an untrusted Server. The User deposits a database, within which are two relations **R** and **S** with confidential attributes $\mathbf{R}.A$ and $\mathbf{S}.B$, respectively, with the Server. At runtime, the User may authorize the Server to perform an equi-join operation $\mathbf{R} \bowtie_{A=B} \mathbf{S}$ in a way that satisfies the following (high-level) privacy and functionality requirements.

—*Initial confidentiality* mandates that, from its copy of the database, the Server derives no information on the confidential values in $\mathbf{R}.A$ and $\mathbf{S}.B$.
—*Query safety* embodies three aspects: (a) after executing equi-join $\mathbf{R} \bowtie_{A=B} \mathbf{S}$, the Server knows only that a pair of records $r \in \mathbf{R}$ and $s \in \mathbf{S}$ combine to form a record in the query answer if and only if $r.A = s.B$. The actual values of $r.A$ and $s.B$ are still not revealed; (b) the Server gains no advantage in deducing the values in records that are not part of the query answer; (c) the Server is unable to abuse the authorization for $\mathbf{R} \bowtie_{A=B} \mathbf{S}$ to join **R** or **S** with some other relations.
—*Noninteractivity* requires the Server to produce the query answer without involving the User in processing the join. An important reason for this requirement is to avoid incurring costly network exchanges in the course of query execution.

The preceding privacy requirements, defined more precisely in Section 3.1, are the strongest possible while allowing the untrusted Server to execute the equi-join. In particular, initial confidentiality requires the use of strong encryption to protect the confidential values, whereas query safety caps the information disclosure stemming from the equi-join to the minimum necessitated by the semantics of the join.

Deterministic encryption schemes, with which records sharing the same attribute value can be observed to have the same ciphertext, violate initial confidentiality as well as point (b) of query safety. Consider the example in Figure 1 that includes a patient relation and a diagnosis relation. The diagnosis codes in the patient relation, as well as the attributes in the diagnosis relation, are encrypted deterministically and stored in the **E(DiagCode)** and **E(Name)** attributes, respectively. Without recovering the original diagnosis codes, an adversary can observe from their ciphertexts that Alice and Bob have the same diagnosis. Furthermore, if Alice is known to have HIV, a join of the two relations on **E(DiagCode)** would reveal that the ciphertext E(HIV) corresponds to HIV. Also, deterministic encryption preserves the frequency of the plaintext values in the ciphertexts; this allows an adversary who possesses prior knowledge of the plaintext frequency distribution to decipher the ciphertext through frequency analysis. Such inference exposure from deterministic encryption poses a real privacy threat and has been highlighted in Damiani et al. [2003].

To prevent inference exposure, we need a probabilistic scheme that incorporates randomness in encrypting data. Thus, the diagnosis codes of Alice and Bob in the patient relation, as well as the diagnosis code for HIV in the diagnosis relation, will encrypt to different ciphertexts, making it impossible for an adversary to link the three records from their encrypted diagnosis codes. This, however, raises the challenge of enabling the untrusted Server to perform an equi-join $\mathbf{R} \bowtie_{A=B} \mathbf{S}$, which involves

Name	E(DiagCode)
Alice	α
Benjamin	γ
Bob	α
...	

(a) **patient**

E(DiagCode)	E(Name)
α	E(HIV)
β	E(flu)
γ	E(glaucoma)
...	

(b) **diagnosis**

Fig. 1. Example of deterministic encryption.

determining whether two records $r \in \mathbf{R}$ and $s \in \mathbf{S}$ satisfy the condition $r.A = s.B$ from their probabilistically generated ciphertexts.

In cryptography, the predicate encryption schemes of Katz et al. [2008] and Shen et al. [2009] allow the Server to determine from the ciphertext of $r.A$ and $s.B$ whether $r.A - s.B = 0$, by treating $r.A$ as message and $s.B$ as predicate. However, the schemes do not meet the second aspect of query safety defined previously and they are very computationally expensive. Another construct called hidden vector encryption [Boneh and Waters 2007] enables the Server to evaluate whether two encrypted vectors for $r.A$ and $s.B$ embed the same value, but with an overhead proportional to the domain size of $r.A$ and $s.B$ that is impractical in our problem setting.

The existing study that is closest to our work is Carbunar and Sion [2012]. Their study is the first we are aware of to formalize the twin requirements of initial confidentiality and query safety for private join. The authors also offered a scheme for an untrusted Server to perform private join. As the data encryption in the scheme contains components that are produced deterministically, it is vulnerable to the inference exposure problem described before and hence falls short of our privacy requirements. We aim to avoid this shortcoming in our method.

Contributions. In this article, we propose a solution for an untrusted Server to perform ad hoc equi-join $\mathbf{R} \bowtie_{A\_B} \mathbf{S}$ while safeguarding the privacy of the database and query. Our solution is the first that achieves constant complexity per pair of records that are evaluated for the join. The query is ad hoc in the sense that, when the database is first deposited with the Server, it is not known that the equi-join will be performed. Consequently, the encryption of attributes $\mathbf{R}.A$ and $\mathbf{S}.B$ cannot build in clues to facilitate the processing of $\mathbf{R} \bowtie_{A\_B} \mathbf{S}$. Our contribution to existing work includes a cryptographic construct for joining encrypted records across relations that satisfies the aforesaid privacy requirements, as well as optimized join algorithms that reduce the number of record pairs from the input relations that need to be matched. The contributions are further elaborated next.

We begin by defining precisely, in the context of equi-join, the notions of initial confidentiality and query safety in our privacy framework. To implement the privacy framework, we then introduce a cryptographic construct for the User to enable the Server to match the records across two input relations on their encrypted join attributes. The cryptographic construct is built on bilinear groups of prime order (see Definitions 2.4 and 2.5). We show that our construct correctly preserves the semantics of the equi-join while enforcing the privacy requirements, predicated on commonly accepted hard problems in bilinear groups.

In view that the equi-join is executed over encrypted data, we should expect to incur additional overheads relative to working directly with cleartext data. As operations in bilinear groups are computationally expensive, for performance reasons it is imperative to avoid matching every record pair from the input relations. We achieve this through two techniques, namely equivalence partitioning (EP) and hash partitioning (HP).

In EP, we exploit the fact that records in **R** that join with the same **S** record must share a common value in attribute *A*. As these **R** records are discovered, they are grouped into an equivalence class so that, if any representative matches a subsequent **S** record, the remaining members in the equivalence class can be determined to match that **S** record without further tests. This technique applies also in grouping the records of **S** into equivalence classes. We provide an EP algorithm that concurrently partitions **R** and **S** into equivalence classes in the course of performing the join.

In HP, the User applies a hash function to map each record on its join attribute to a partition identifier, so that the Server needs to join an **R** record only with those **S** records that have the same partition identifier. The partition identifiers are encrypted to ensure initial confidentiality and disclosed only at the time of the join. Unlike EP, HP discloses extra information, and we show how to quantify this disclosure.

To assess the effectiveness of EP and HP, we provide a detailed analysis of their costs in common query settings. We also evaluate them extensively on synthetic and benchmark workloads. The evaluation reveals useful insights on how to configure the join algorithms to achieve practical query execution time.

The remainder of this article is organized as follows. Section 2 summarizes related work and background on the cryptographic techniques that we employ in this work. In Section 3, we formalize the private join problem and introduce our cryptographic construct for joining records across selected relations. The security of the construct is proved in Section 4. Section 5 then develops the cryptographic construct into join algorithms, incorporating the EP and HP techniques to reduce computation cost. We analyze the cost of EP in Section 6, while Section 7 empirically evaluates the join algorithms. Finally, Section 8 concludes.

2. BACKGROUND

This section begins with a review of existing schemes for private join [Carbunar and Sion 2012] before discussing related literature in cryptography. We also briefly cover the cryptography background for our work.

2.1. Related Work on Private Join

In the OPES scheme [Agrawal et al. 2004], a cleartext attribute value is converted to a ciphertext for query processing (including joins) through two order-preserving mapping functions. The first function maps from the source (i.e., cleartext value) distribution to a uniform distribution, which the second function then maps to a target distribution. Without knowledge of the mapping functions, an adversary cannot reverse them to recover the cleartext value from a ciphertext. As explained in Agrawal et al. [2004], OPES (and any order-preserving scheme) fails when the data distribution or the cleartext data are known, as it is then straightforward to correlate an encrypted record with its cleartext counterpart. Moreover, in cases where pairs of cleartext-ciphertext are leaked, an adversary can easily partition the data collection by the compromised cleartext values.

In Hacigumus et al. [2002], the authors proposed to provide the DBMS with hash digests of the attribute values to facilitate query processing. Two types of hash functions were considered. The first type, order-preserving functions, suffers the same limitations as OPES. The second, randomized hashing, destroys the order between attribute values. However, as the hashing function is deterministic, it has a similar vulnerability as deterministic encryption (as described in the Introduction). Clearly, the finer the hash buckets, the more precise the retrieval, leading to better performance but higher disclosure. Hore et al. [2004] provide a systematic treatment of this privacy-performance trade-off. Instead of one hash function, an alternative is to use a Bloom filter [Bloom 1970] that applies multiple hash functions to derive the digests. Even so,

the vulnerability of hashing remains since the same deterministic hash functions are applied to every attribute value.

CryptDB [Popa et al. 2011] embodies an SQL-aware encryption strategy to enable query execution on encrypted data. This means that the type of encryption used on an attribute depends on the query operations that need to be performed on the attribute. For equi-join, the attributes involved are encrypted with a deterministic function. That paper provides a method to encrypt attributes with different keys initially, and then to adjust their encrypted values to the same key (without access to the original values) just prior to the join operation.

In contrast to Agrawal et al. [2004], Hacigumus et al. [2002], Hore et al. [2004], and Popa et al. [2011], our objective in this article is to protect data with a probabilistic encryption approach to achieve stronger privacy, although we also show how to combine our solution with deterministic bucketization to improve runtime performance.

The notion of private join was first formalized by Carbunar and Sion [2012]. The authors also introduced a method to privately perform predicate join. Suppose that $\mathbf{R}$ is a relation of records $\{r_1, r_2, \ldots, r_m\}$ with schema $\langle K_R, A, \ldots\rangle$, where K_R is the primary key and A is a confidential attribute, and $\mathbf{S}$ is a relation of records $\{s_1, s_2, \ldots, s_n\}$ with schema $\langle K_S, B, \ldots\rangle$ where K_S is the primary key and B is a confidential attribute. Let $pred_{FM} : A \times B \rightarrow \{true, false\}$ be a finite match predicate such that: (a) for every value $a \in A$, there is an upper bound on the number of values $b \in B$ for which $pred_{FM}(a, b) = true$; and (b) for every value $b \in B$, there is an upper bound on the number of values $a \in A$ for which $pred_{FM}(a, b) = true$. The private predicate join method comprises three procedures:

—*Setup*. Let p be a large prime number. Let $\mathbb{Z}_p^*$ be a cyclic group of order p, with generator g (see Definition 2.2). Randomly generate encryption key $K \in_R \{0,1\}^*$ and $x_A, y_A, x_B, y_B \in_R \mathbb{Z}_p$. Let $\mathcal{H}(.)$ be a cryptographic hash function and $E_K(.)$ be an encryption of its input with key K. The User keeps $\langle K, x_A, y_A, x_B, y_B\rangle$ secret, whereas $\langle p, g\rangle$ is published to the Server.

—*Encrypt data.* For each value $a_i \in A$ in record $r_i \in \mathbf{R}$, the User generates a tuple $\langle E_K(a_i), O(a_i), BF(a_i)\rangle$, where $O(a_i) = \mathcal{H}(a_i)x_A \bmod p$, and $BF(a_i)$ is a Bloom filter on $\{g^{\mathcal{H}(b)y_A} \bmod p \mid pred_{FM}(a_i, b) = true\}$.

Likewise, for each value $b_i \in B$ in record $s_i \in \mathbf{S}$, the User generates a tuple $\langle E_K(b_i), O(b_i), BF(b_i)\rangle$, where $O(b_i) = \mathcal{H}(b_i)x_B \bmod p$ and $BF(b_i)$ is a Bloom filter on $\{g^{\mathcal{H}(a)y_B} \bmod p \mid pred_{FM}(a, b_i) = true\}$.

The generated tuples are deposited with the Server.

—*Join*. To identify record pairs $r_i \in \mathbf{R}$, $s_j \in \mathbf{S}$ that match predicate $pred_{FM}(a_i, b_j)$ where a_i and b_j are the A and B values in r_i and s_j, respectively, the User supplies a trapdoor $\mathcal{T}_{AB} = g^{y_A/x_B} \bmod p$. The Server decides that r_i matches s_j if $(\mathcal{T}_{AB})^{O(b_j)} \bmod p$ is encoded in $BF(a_i)$.

An advantage of the scheme is that it supports any finite match predicate. At the same time, it has limitations. One limitation is that duplicate values within the A column can be observed by the Server; since $O(.)$ and $BF(.)$ are deterministic functions, they always give the same output for duplicate values within a column. The authors showed that $O(.)$ may be remedied by forcing each duplicate to a unique value, for example, by appending an instance count. Even so, the Server is able to deduce that, if $BF(a_i) = BF(a_j)$ for records $r_i, r_j \in \mathbf{R}$, it is highly likely that $a_i = a_j$. Similarly, duplicates in the B column are apparent from their $BF(.)$ values. Hence the scheme is vulnerable to inference exposure [Damiani et al. 2003].

Another limitation is that $BF(.)$ depends on $pred_{FM}$, so any change to the predicate function requires all the records to be reencoded. Consequently, the method caters only

for predefined predicates but not for ad hoc join queries. Moreover, if there are multiple predicates defined on a column, each predicate would require its own set of Bloom filters.

In this article, we focus on equi-join, one of the most common operations in relational DBMS. The main differentiations of our work from the scheme in Carbunar and Sion [2012] are in: (a) employing probabilistic encryption to protect confidential attributes, and (b) supporting ad hoc equi-join queries.

Instead of processing encrypted data directly, a different approach is to assume the Server contains a trusted hardware that can decrypt data securely for processing. An example of such a study is reported in Arasu and Kaushik [2014], proposing algorithms that ensure the access patterns generated by database operations do not disclose information on the encrypted data. Challenges associated with this approach, including the cost of programming the trusted hardware, the administrative overhead in deployment, and the resource constraint of the trusted hardware, lead us to decide against the approach in this article.

2.2. Relation Work in Cryptography

The predicate encryption schemes of Katz et al. [2008] and Shen et al. [2009] allow the Server to determine whether $r.A - s.B = 0$ for $r \in \mathbf{R}$, $s \in \mathbf{S}$, by treating $r.A$ as an encrypted message and $s.B$ as an encrypted predicate. The schemes rely on bilinear groups (see Definitions 2.4 and 2.5) with composite orders that are products of 3 and 4 large prime numbers, respectively. Operations in such groups incur very high computation costs; for example, on our experiment platform described in Section 7.1, a bilinear mapping operation requires 114 msec. In comparison, the same operation takes only 565 usec in bilinear groups of prime order, the cryptographic setting for our solution. Most importantly, the schemes do not meet the second aspect of query safety, in that they cannot restrict the Server to only specific joins that the User authorizes. For instance, if there is a record t in some other relation that contains encrypted message $t.C$, the Server could also determine whether $t.C - s.B = 0$.

The Hidden Vector Encryption (HVE) scheme proposed by Boneh and Waters [2007] enables a Server to compute the inner product of two encrypted vectors. To apply HVE in an equi-join, we represent each $\mathbf{R}$ record value $r.A \in [1, \ldots, m]$ as an m-bit vector $\overrightarrow{a}$ with value 1 in position $r.A$ and value 0 everywhere else. Likewise, each $\mathbf{S}$ record value $s.B \in [1, \ldots, m]$ is represented as an m-bit vector $\overrightarrow{b}$ with value 1 in position $s.B$ and value 0 everywhere else. The Server then computes and tests whether the inner product $\overrightarrow{a} \cdot \overrightarrow{b} = 1$. With HVE, every attribute value requires a ciphertext that is $O(m)$ in size. Additionally, the inner product computation entails $O(m)$ operations in a bilinear group of composite order that is the product of two large prime numbers. On our experiment platform, a bilinear mapping operation in such a group is $70\times$ slower than in the cryptographic setting for our work, namely bilinear groups of prime order. Such overheads are clearly impractical for general database attributes (e.g., 4-byte integers where m would be 2^{32}). In contrast, the space and computation costs of our construction in this article are constant and specifically independent of the attribute domain size.

There are also symmetric-key encrypted keyword search [Song et al. 2000; Golle et al. 2004; Curtmola et al. 2006] and public-key encrypted keyword search [Boneh et al. 2004b, 2007] schemes for keyword search over encrypted data. While these schemes protect the data with probabilistic encryption, the trapdoor for testing whether the data contain a given keyword is deterministic. To apply such a scheme in an equi-join, the attribute values in one relation $\mathbf{R}$ will be treated as encrypted data while the attribute values in the other relation $\mathbf{S}$ will be trapdoors, meaning that the latter will

be subjected to information exposure. Hence the requirement of initial confidentiality will not be met.

Another line of related work is private set intersection [Freedman et al. 2004], where two parties engage in a protocol to jointly compute the intersection of their respective lists without leaking any additional information. In that problem, each party has access to its own list of data and uses their values in the protocol. In contrast, in our equi-join problem the Server that carries out query processing cannot know the cleartext values of the input relations.

2.3. Cryptography Foundation

We define next several concepts in cryptography that underlie our proposed solution in the next section.

Definition 2.1 (*Chosen Plaintext Attack (CPA)*). A CPA is a standard adversarial model under which the adversary is able to obtain the encryption for arbitrary cleartexts.

Definition 2.2. A *cyclic group* $(\mathbb{G}, \cdot)$ with generator g is an algebraic structure in which applying binary operation $\cdot$ on any two elements yields a third element, while demonstrating the closure, associativity, identity, and invertibility properties. Moreover, for every $h \in \mathbb{G}$, there is an integer i such that $h = g^i$. The number of elements in $\mathbb{G}$ is known as its order.

Definition 2.3. The *decision linear problem* is to determine, for given cyclic group $\mathbb{G}$ and $u, v, h, u^a, v^b, h^c \in \mathbb{G}$, whether $c = a + b$. The *decision linear Diffie-Hellman assumption* [Boneh et al. 2004a] asserts that the decision linear problem is as hard as the discrete logarithm problem. The latter is widely accepted as intractable in the sense that it cannot be solved within a reasonable amount of time when the problem parameter is large.

Definition 2.4. Let $\mathbb{G}$ be a cyclic group of order p with generator g. A *bilinear map* is a mapping $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, where $\mathbb{G}_T$ is another cyclic group of order p, that exhibits the following properties:

—*bilinearity*: $\forall u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}$, $\mathbf{e}(u^a, v^b) = \mathbf{e}(u, v)^{ab}$;
—*computability:* $\forall u, v \in \mathbb{G}$, $\mathbf{e}(u, v)$ can be computed efficiently in complexity polynomial in $\log p$; and
—*nondegeneracy*: $\mathbf{e}(g, g) \neq 1$.

Definition 2.5. A *bilinear group* is a cyclic group that has a bilinear map on it.

Definition 2.6 (*Linear Encryption [Boneh et al. 2004a]*). Construct a bilinear group $\mathbb{G}$ with generators g_1, g_2, g_3 such that $g_3 = g_1^{\sigma_1} = g_2^{\sigma_2}$ for some $\sigma_1, \sigma_2 \in \mathbb{Z}$. Let the public key be (g_1, g_2, g_3) and the secret key be (σ_1, σ_2). To encrypt a record value $r.A \in \mathbb{G}$, choose random values $\lambda_1, \lambda_2 \in \mathbb{Z}$ and output the ciphertext $(g_1^{\lambda_1}, g_2^{\lambda_2}, r.A \cdot g_3^{\lambda_1+\lambda_2})$. To recover the value $r.A$ in a ciphertext (T_1, T_2, T_3), the user computes $r.A = T_3/(T_1^{\sigma_1} \cdot T_2^{\sigma_2})$. The scheme is secure against CPA, based on the decision linear Diffie-Hellman assumption.

Definition 2.7. In the context of a bilinear group $\mathbb{G}$ of prime order p with generator g, the *Decision 3-party Diffie-Hellman* (D3DH) problem [Boneh et al. 2006] is to decide, for given $g, g^a, g^b, g^c, t \in \mathbb{G}$, whether $t = g^{abc}$. The *Decision 3-party Diffie-Hellman Assumption* asserts that any polynomial algorithm has negligible advantage over a random guess in solving the D3DH problem.

Table I. Notation

Symbol	Meaning	Default
$\|\mathbf{R}\|$	# pages in relation $\mathbf{R}$	5,000
$\|\|\mathbf{R}\|\|$	# records in relation $\mathbf{R}$	20,000
ϕ_A	# distinct values in attribute $\mathbf{R}.A$	1,000
$\|\mathbf{S}\|$	# pages in relation $\mathbf{S}$	5,000
$\|\|\mathbf{S}\|\|$	# records in relation $\mathbf{S}$	20,000
ϕ_B	# distinct values in attribute $\mathbf{S}.B$	1,000
$\mathcal{T}_{AB}$	Token that the User issues to the Server to perform equi-join $\mathbf{R} \bowtie_{A=B} \mathbf{S}$	–
M	# memory pages for join operation	100
$\mathbb{G}, \mathbb{G}_T$	Cyclic groups with bilinear mapping $\mathbf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$	–
p	Large prime number that is the order of $\mathbb{G}$ and $\mathbb{G}_T$	–
c_{exp}	Cost of exponentiation in $\mathbb{G}$ and $\mathbb{G}_T$	–
c_{mul}	Cost of multiplication in $\mathbb{G}$ and $\mathbb{G}_T$	–
c_{map}	Cost of bilinear mapping $\mathbf{e}(\_, \_)$	–
SK	$SK = (\sigma, \sigma_1, \sigma_2)$ is the User's secret key; $\sigma, \sigma_1, \sigma_2 \in_R \mathbb{Z}_p$	–
PK	$PK = (p, g_1, g_2)$ includes the public values of the cryptosystem; $g_1, g_2 \in \mathbb{G}$	–
κ_A, τ_A	Random value generated for $\mathbf{R}$; $\kappa_A, \tau_A \in_R \mathbb{Z}_p$	–
κ_B, τ_B	Random value generated for $\mathbf{S}$; $\kappa_B, \tau_B \in_R \mathbb{Z}_p$	–
$\mathcal{A}_i$	$\mathcal{A}_i = \langle \mathcal{A}_{i,1}, \mathcal{A}_{i,2}, \mathcal{A}_{i,3}, \mathcal{A}_{i,4}, \mathcal{A}_{i,5}, \mathcal{A}_{i,6}, \mathcal{A}_{i,7}, \mathcal{A}_{i,8} \rangle \in \mathbb{G}^8$ is the encryption of attribute A in record $r_i \in \mathbf{R}$	–
$\lambda_{i,1}, \lambda_{i,2}, x_i$	Random values generated for record $r_i \in \mathbf{R}$; $\lambda_{i,1}, \lambda_{i,2} \in_R \mathbb{Z}_p$, $x_i \in_R \mathbb{G}$	–
$\mathcal{B}_j$	$\mathcal{B}_j = \langle \mathcal{B}_{j,1}, \mathcal{B}_{j,2}, \mathcal{B}_{j,3}, \mathcal{B}_{j,4}, \mathcal{B}_{j,5}, \mathcal{B}_{j,6}, \mathcal{B}_{j,7}, \mathcal{B}_{j,8} \rangle \in \mathbb{G}^8$ is the encryption of attribute B in record $s_j \in \mathbf{S}$	–
$\mu_{j,1}, \mu_{j,2}, y_j$	Random values generated for record $s_j \in \mathbf{S}$; $\mu_{j,1}, \mu_{j,2} \in_R \mathbb{Z}_p$, $y_j \in_R \mathbb{G}$	–

3. CONSTRUCT FOR PRIVATE EQUI-JOIN QUERY

This section introduces our privacy-preserving construct for the ad hoc equi-join query. We begin by defining our problem precisely, before presenting our cryptographic construct. Table I summarizes the key notations, which will be explained as they are used.

3.1. Problem Formulation

Our system comprises two parties, namely User and Server. The User's database includes two relations $\mathbf{R}$ and $\mathbf{S}$, wherein $\mathbf{R}$ contains records $\{r_1, r_2, \ldots, r_m\}$; the schema of $\mathbf{R}$ is $\langle K_R, A, \ldots \rangle$, where K_R is the primary key and A is a confidential attribute. $\mathbf{S}$ contains records $\{s_1, s_2, \ldots, s_n\}$ according to schema $\langle K_S, B, \ldots \rangle$, where K_S is the primary key and B is a confidential attribute. For now, we assume that A, B are in the domain $[L, U) \subset [0, p)$ for some large prime number p; we will discuss how to handle other common attribute domains in Section 5.

The User contracts with the Server to host the database and process queries over it. In this work, we focus on equi-join queries of the form $\mathbf{R} \bowtie_{A=B} \mathbf{S}$. Engaging the Server to compose the result of an equi-join from the input relations entails a necessary disclosure that a record in one relation joins with two records in the other relation if and only if the latter share the same join attribute value. Consequently, we can only prevent additional disclosure from the equi-join. Specifically, we aim to achieve the following two objectives: one pertaining to the privacy protection before executing an equi-join on the database, the other the privacy protection thereafter.

P1 *Initial Confidentiality*. When the database is first deposited on the Server, the A values in $\mathbf{R}$ and B values in $\mathbf{S}$ are sensitive and must be protected. In particular, the

adversary's ability to determine the A values and B values is no better than random guesses in their respective domains.

P2 *Safety of Equi-Join*. The Server can perform a join $\mathbf{R} \bowtie_{A=B} \mathbf{S}$ only after receiving an enabling token $\mathcal{T}_{AB}$ from the User. The Server cannot adapt $\mathcal{T}_{AB}$ for a join involving other relations or attributes. After executing the join, the following must hold.

P2A For any two records $r_1, r_2 \in \mathbf{R}$ that join with some record $s \in \mathbf{S}$, the Server knows that $r_1.A = r_2.A$. Nevertheless, the actual value of $r_1.A$ and $r_2.A$ remain hidden from the Server. Likewise, for any two records $s_1, s_2 \in \mathbf{S}$ that join with some record $r \in \mathbf{R}$, the Server knows that $s_1.B = s_2.B$ but not the actual values in $s_1.B$ and $s_2.B$.

P2B The Server gains no other information on any record of $\mathbf{R}$ that does not pair with some $\mathbf{S}$ record, and vice versa, to satisfy the join condition encoded in $\mathcal{T}_{AB}$.

Our objective P1 implies that an adversary must not learn whether two records in $\mathbf{R}$ share the same A value (i.e., $\forall\, r_1, r_2 \in \mathbf{R}$ such that $r_1 \neq r_2$, whether $r_1.A = r_2.A$), two records in $\mathbf{S}$ share the same B value (i.e., $\forall\, s_1, s_2 \in \mathbf{S}$ such that $s_1 \neq s_2$, whether $s_1.B = s_2.B$), or the A value in an $\mathbf{R}$ record is the same as the B value in an $\mathbf{S}$ record (i.e., $\forall\, r \in \mathbf{R}$ and $s \in \mathbf{S}$, whether $r.A = s.B$). This objective is stronger than the corresponding provision in Carbunar and Sion [2012], that leaks information on whether two records share the same A or B values as explained in Section 2.1. As for objective P2, it limits the disclosure to the minimum necessitated by the semantics of the equi-join. P2B in particular cannot be met by fixed bucketization schemes (like Hacigumus et al. [2002] and Hore et al. [2004]).

In general, an equi-join $\mathbf{R} \bowtie_{A=B} \mathbf{S}$ may be performed on any two relations $\mathbf{R}$ and $\mathbf{S}$ in which the join attributes A and B have compatible domains. One interesting instance is where $\mathbf{R}$ and $\mathbf{S}$ are vertical partitions of a larger relation such that A and B correspond to the primary key of the original relation. In this instance, our privacy objectives ensure that the original relation can be reconstructed from $\mathbf{R}$ and $\mathbf{S}$ only after the User issues $\mathcal{T}_{AB}$, and even then the join attributes remain protected. Other common instances involve primary-key-to-foreign-key joins and foreign-key-to-foreign-key joins. Here, we provide the additional assurance that the Server cannot determine whether any particular primary key value and foreign key value, respectively, has matching counterparts in the other relation.

Adversarial model. The adversary may be the Server that is capable of observing the schema, the database, and all the queries, in addition to tampering with the data and query processing procedure. The adversary may also be an external party that somehow manages to inject itself into the protocol of the system. We exclude consideration of any external knowledge that the adversary may exploit to defeat our privacy measures.

3.2. Cryptographic Construction

We now present a protocol for the Server to determine, in any pair of encrypted records $r_i \in \mathbf{R}$ and $s_j \in \mathbf{S}$, whether the condition $r_i.A = s_j.B$ holds. We begin by explaining how our privacy framework leads us to the following design decisions, before introducing details of our protocol.

—To ensure that query token $\mathcal{T}_{AB}$ is necessary for equi-join $\mathbf{R} \bowtie_{A=B} \mathbf{S}$ and cannot be abused, we embed secret values specific to attribute $\mathbf{R}.A$ in the ciphertext of $r_i.A$ in every record $r_i \in \mathbf{R}$, secret values specific to attribute $\mathbf{S}.B$ in the ciphertext of $s_j.B$ in every record $s_j \in \mathbf{S}$, and the secret values of both attributes in $\mathcal{T}_{AB}$. Only by combining $\mathcal{T}_{AB}$ with the ciphertexts of an $r_i.A$ value along with an $s_j.B$ value can the attribute secrets in them be cancelled out to allow for testing whether $r_i.A = s_j.B$.

—The combination of $\mathcal{T}_{AB}$ with the ciphertexts of an $r_i.A$ value and an $s_j.B$ value gives an output that is a function of $(x_i \cdot y_j)^{r_i.A-s_j.B}$, where x_i is a random value specific to each record $r_i \in \mathbf{R}$ and where y_j is a random value specific to each record $s_j \in \mathbf{S}$. Thus, if $r_i.A - s_j.B = 0$, the protocol output is a known constant; otherwise, the output is a random value due to the factor $x_i \cdot y_j$. This property is critical in preventing any inference between $\mathbf{R}$ records (by checking, for $r_1, r_2 \in \mathbf{R}$ and any one $s_j \in \mathbf{S}$, whether the protocol gives the same output on the pair $\{r_1.A, s_j.B\}$ and on $\{r_2.A, s_j.B\}$), as well as between $\mathbf{S}$ records. Since $(r_i.A, x_i)$ and $(s_j.B, y_j)$ belong to independent ciphertexts, our cryptographic construct exploits the bilinearity property of bilinear groups to constitute $(x_i \cdot y_j)^{r_i.A-s_j.B}$ from the ciphertexts.

—In order to compose, from $\mathcal{T}_{AB}$ and the ciphertexts of an $r_i.A$ value and an $s_j.B$ value, an output that is a function of $(x_i \cdot y_j)^{r_i.A-s_j.B}$, the encryption of $r_i.A$ and $s_j.B$ must be additively homomorphic. We choose to extend Boneh et al's linear encryption scheme in Boneh et al. [2004a]; besides being additively homomorphic, it satisfies our requirement of initial confidentiality in providing semantic indistinguishability of encrypted data against CPA. We emphasize, though, that Boneh et al's work provides neither the mechanism for testing whether $r_i.A - s_j.B = 0$ nor the necessary safeguards to limit the information derivable from the test.

—As our data encryption maps each record value to a ciphertext comprising multiple elements of a bilinear group, those elements need to be "chained" by record-specific random values. This is necessary to prevent deviations from our protocol by intermixing elements across the ciphertexts of different records.

We now present details of our cryptographic protocol that consists of four procedures: `Setup`, `EncryptData`, `GenerateQuery`, and `ServerProcessing`.

`Setup`. Construct a cyclic group $\mathbb{G}$ with bilinear mapping $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, where the orders of $\mathbb{G}$ and $\mathbb{G}_T$ are both p, a large prime number such that $[0, p)$ envelops the domain of $\mathbf{R}.A$ and $\mathbf{S}.B$. Let g_1 be a generator of $\mathbb{G}$. Randomly choose σ and σ_1 from $\mathbb{Z}_p$, denoted as $\sigma, \sigma_1 \in_R \mathbb{Z}_p$. Compute $g_2 = g_1^{\sigma}$. The User's secret key is $SK = (\sigma, \sigma_1)$. The public values $PK = (p, g_1, g_2)$ are released to the Server.

`EncryptData`. After choosing $\kappa_A, \tau_A \in_R \mathbb{Z}_p$ for $\mathbf{R}$ and $\kappa_B, \tau_B \in_R \mathbb{Z}_p$ for $\mathbf{S}$, the User encrypts the confidential attribute in the records of the two relations before depositing them with the Server.

—For each $r_i \in \mathbf{R}$, let $\lambda_{i,1}, \lambda_{i,2} \in_R \mathbb{Z}_p$, $x_i \in_R \mathbb{G}$ and represent $r_i.A$ by encrypted tuple $\langle \mathcal{A}_{i,1}, \mathcal{A}_{i,2}, \mathcal{A}_{i,3}, \mathcal{A}_{i,4}, \mathcal{A}_{i,5}, \mathcal{A}_{i,6}, \mathcal{A}_{i,7}, \mathcal{A}_{i,8} \rangle \in \mathbb{G}^8$, where $\mathcal{A}_{i,1} = x_i^{\lambda_{i,1}}$, $\mathcal{A}_{i,2} = g_1^{\lambda_{i,2}}$, $\mathcal{A}_{i,3} = x_i^{\sigma_1 \times r_i.A + \sigma_2} \cdot g_2^{\lambda_{i,1} + \lambda_{i,2}}$, $\mathcal{A}_{i,4} = x_i$, $\mathcal{A}_{i,5} = x_i^{\sigma}$, $\mathcal{A}_{i,6} = x_i^{\sigma/\kappa_A}$, $\mathcal{A}_{i,7} = g_1^{\lambda_{i,1} \times \tau_A}$, $\mathcal{A}_{i,8} = \mathbf{e}(x_i, x_i)^{\sigma_1 \times r_i.A + \sigma_2}$.

—For each $s_j \in \mathbf{S}$, let $\mu_{j,1}, \mu_{j,2} \in_R \mathbb{Z}_p$, $y_j \in_R \mathbb{G}$ and represent $s_j.B$ by encrypted tuple $\langle \mathcal{B}_{j,1}, \mathcal{B}_{j,2}, \mathcal{B}_{j,3}, \mathcal{B}_{j,4}, \mathcal{B}_{j,5}, \mathcal{B}_{j,6}, \mathcal{B}_{j,7}, \mathcal{B}_{j,8} \rangle \in \mathbb{G}^8$, where $\mathcal{B}_{j,1} = y_j^{\mu_{j,1}}$, $\mathcal{B}_{j,2} = g_1^{\mu_{j,2}}$, $\mathcal{B}_{j,3} = y_j^{\sigma_1 \times s_j.B + \sigma_2} \cdot g_2^{\mu_{j,1} + \mu_{j,2}}$, $\mathcal{B}_{j,4} = y_j$, $\mathcal{B}_{j,5} = y_j^{\sigma}$, $\mathcal{B}_{j,6} = y_j^{\sigma/\kappa_B}$, $\mathcal{B}_{j,7} = g_1^{\mu_{j,1} \times \tau_B}$, $\mathcal{B}_{j,8} = \mathbf{e}(y_j, y_j)^{\sigma_1 \times s_j.B + \sigma_2}$.

In the encryption, κ_A, τ_A, κ_B and τ_B are per-attribute secrets that allow the User to control which pair of attributes can be joined by the Server. Moreover, all the group elements in each encrypted tuple are linked to each other, directly or indirectly, by record-specific random variables (i.e., $\lambda_{i,1}, \lambda_{i,2}, x_i$ in the ciphertext of $r_i.A$, and $\mu_{j,1}, \mu_{j,2}, y_j$ in the ciphertext of $s_j.B$). Among them, x_i and y_j will go into the output that is a function of $(x_i \cdot y_j)^{r_i.A-s_j.B}$ when the Server subsequently tests whether $r_i.A - s_j.B = 0$ in `ServerProcessing`.

`GenerateQuery`. For equi-join query $\mathbf{R} \bowtie_{A=B} \mathbf{S}$, the User generates a query token $\mathcal{T}_{AB} = \langle \kappa_B/\tau_A, -\kappa_A/\tau_B \rangle$. $\mathcal{T}_{AB}$ is sent to the Server.

`ServerProcessing`. For any pair of records $r_i \in \mathbf{R}$ and $s_j \in \mathbf{S}$, if the Server detects that the following condition holds

$$\begin{aligned}\mathbf{e}\big(\mathcal{A}_{i,3} \cdot \mathcal{B}_{j,3}^{-1}, \mathcal{A}_{i,4} \cdot \mathcal{B}_{j,4}\big) \;=\; & \mathcal{A}_{i,8} \cdot \mathcal{B}_{j,8}^{-1} \cdot \mathbf{e}\big(\mathcal{A}_{i,1} \cdot \mathcal{B}_{j,1}^{-1}, g_2\big) \cdot \mathbf{e}\big(\mathcal{B}_{j,6}^{\kappa_B/\tau_A}, \mathcal{A}_{i,7}\big) \cdot \mathbf{e}\big(\mathcal{A}_{i,6}, \mathcal{B}_{j,7}^{-\kappa_A/\tau_B}\big) \cdot \\ & \mathbf{e}\big(\mathcal{A}_{i,2} \cdot \mathcal{B}_{j,2}^{-1}, \mathcal{A}_{i,5} \cdot \mathcal{B}_{j,5}\big), \end{aligned} \tag{1}$$

then $\langle r_i, s_j \rangle$ forms an output record of the equi-join.

4. ANALYSIS OF EQUI-JOIN CONSTRUCT

We now give a formal analysis of the security properties of the equi-join protocol introduced in Section 3.2, following the convention and terminology in cryptography.

4.1. Correctness of Equi-Join Construct

THEOREM 4.1. *The equi-join protocol in Section 3.2 is correct.*

PROOF. Consider any pair of records $r_i \in \mathbf{R}$ and $s_j \in \mathbf{S}$, their ciphertexts $\langle \mathcal{A}_{i,1}, \mathcal{A}_{i,2}, \mathcal{A}_{i,3}, \mathcal{A}_{i,4}, \mathcal{A}_{i,5}, \mathcal{A}_{i,6}, \mathcal{A}_{i,7}, \mathcal{A}_{i,8} \rangle$ and $\langle \mathcal{B}_{j,1}, \mathcal{B}_{j,2}, \mathcal{B}_{j,3}, \mathcal{B}_{j,4}, \mathcal{B}_{j,5}, \mathcal{B}_{j,6}, \mathcal{B}_{j,7}, \mathcal{B}_{j,8} \rangle$, along with query token $\mathcal{T}_{AB}$. Due to the properties of the bilinear group,

$$\begin{aligned}\mathbf{e}\big(\mathcal{A}_{i,1} \cdot \mathcal{B}_{j,1}^{-1}, g_2\big) \;&=\; \mathbf{e}\big(x_i^{\lambda_{i,1}} \cdot y_j^{-\mu_{j,1}}, g_2\big) = \mathbf{e}(x_i, g_2)^{\lambda_{i,1}} \cdot \mathbf{e}(y_j, g_2)^{-\mu_{j,1}} \\ \mathbf{e}\big(\mathcal{B}_{j,6}^{\kappa_B/\tau_A}, \mathcal{A}_{i,7}\big) \;&=\; \mathbf{e}\big(y_j^{\sigma/\tau_A}, g_1^{\lambda_{i,1} \times \tau_A}\big) = \mathbf{e}(y_j, g_2)^{\lambda_{i,1}} \\ \mathbf{e}\big(\mathcal{A}_{i,6}, \mathcal{B}_{j,7}^{-\kappa_A/\tau_B}\big) \;&=\; \mathbf{e}\big(x_i^{\sigma/\kappa_A}, g_1^{-\mu_{j,1} \times \kappa_A}\big) = \mathbf{e}(x_i, g_2)^{-\mu_{j,1}} \\ \mathbf{e}\big(\mathcal{A}_{i,2} \cdot \mathcal{B}_{j,2}^{-1}, \mathcal{A}_{i,5} \cdot \mathcal{B}_{j,5}\big) \;&=\; \mathbf{e}\big(g_1^{\lambda_{i,2} - \mu_{j,2}}, x_i^{\sigma} y_j^{\sigma}\big) = \mathbf{e}(x_i y_j, g_2)^{\lambda_{i,2} - \mu_{j,2}}. \end{aligned}$$

Therefore,

$$\begin{aligned}\mathbf{e}\big(\mathcal{A}_{i,3} \cdot \mathcal{B}_{j,3}^{-1}, \mathcal{A}_{i,4} \cdot \mathcal{B}_{j,4}\big) \;&=\; \mathbf{e}\big(x_i^{\sigma_1 \times r_i.A + \sigma_2} y_j^{-\sigma_1 \times s_j.B - \sigma_2} g_2^{\lambda_{i,1} + \lambda_{i,2} - \mu_{j,1} - \mu_{j,2}}, x_i y_j\big) \\ &=\; \mathbf{e}\big(x_i^{\sigma_1 \times r_i.A + \sigma_2} y_j^{-\sigma_1 \times s_j.B - \sigma_2}, x_i y_j\big) \cdot \mathbf{e}(g_2, x_i y_j)^{\lambda_{i,1} + \lambda_{i,2} - \mu_{j,1} - \mu_{j,2}} \\ &=\; \mathbf{e}(x_i, x_i)^{\sigma_1 \times r_i.A + \sigma_2} \cdot \mathbf{e}(x_i, y_j)^{\sigma_1 (r_i.A - s_j.B)} \cdot \\ &\quad\; \mathbf{e}(y_j, y_j)^{-\sigma_1 \times s_j.B - \sigma_2} \cdot \mathbf{e}(g_2, x_i)^{\lambda_{i,1} - \mu_{j,1}} \cdot \\ &\quad\; \mathbf{e}(g_2, y_j)^{\lambda_{i,1} - \mu_{j,1}} \cdot \mathbf{e}(g_2, x_i y_j)^{\lambda_{i,2} - \mu_{j,2}} \\ &=\; \mathcal{A}_{i,8} \cdot \mathbf{e}(x_i, y_j)^{\sigma_1 (r_i.A - s_j.B)} \cdot \mathcal{B}_{j,8}^{-1} \cdot \mathbf{e}\big(\mathcal{A}_{i,1} \cdot \mathcal{B}_{j,1}^{-1}, g_2\big) \cdot \\ &\quad\; \mathbf{e}\big(\mathcal{B}_{j,6}^{\kappa_B/\tau_A}, \mathcal{A}_{i,7}\big) \cdot \mathbf{e}\big(\mathcal{A}_{i,6}, \mathcal{B}_{j,7}^{-\kappa_A/\tau_B}\big) \cdot \\ &\quad\; \mathbf{e}\big(\mathcal{A}_{i,2} \cdot \mathcal{B}_{j,2}^{-1}, \mathcal{A}_{i,5} \cdot \mathcal{B}_{j,5}\big). \end{aligned} \tag{2}$$

In formula (2), $\mathbf{e}(x_i, y_j)^{\sigma_1 (r_i.A - s_j.B)} = 1 \Leftrightarrow r_i.A = s_j.B \bmod p$. As we have chosen $[0, p)$ to envelop the domains of $\mathbf{R}.A$ and $\mathbf{S}.B$ and as the order of $\mathbb{G}_T$ is a prime number, we have $r_i.A = s_j.B$ if and only if $\mathbf{e}(x_i, y_j)^{\sigma_1 (r_i.A - s_j.B)} = 1$. Thus, the condition in formula (1) holds. □

In testing the previous join condition, the query token $\mathcal{T}_{AB}$ plays a crucial role in that, without it, the Server can obtain only $\mathbf{e}(\mathcal{B}_{j,6}, \mathcal{A}_{i,7}) = \mathbf{e}(y_j, g_2)^{\lambda_{i,1} \times \tau_A/\kappa_B}$ and $\mathbf{e}(\mathcal{A}_{i,6}, \mathcal{B}_{j,7}^{-1}) = \mathbf{e}(x_i, g_2)^{-\mu_{j,1} \times \tau_B/\kappa_A}$, thus failing to offset the components $\mathbf{e}(y_j, g_2)^{\lambda_{i,1}}$ and $\mathbf{e}(x_i, g_2)^{-\mu_{j,1}}$ in $\mathbf{e}(\mathcal{A}_{i,3} \cdot \mathcal{B}_{j,3}^{-1}, \mathcal{A}_{i,4} \cdot \mathcal{B}_{j,4})$.

4.2. Initial Confidentiality

4.2.1. Definition of Initial Confidentiality. Our privacy objective P1 mandates the use of strong encryption to protect the confidential $\mathbf{R}.A$ and $\mathbf{S}.B$ attributes. To satisfy the objective, we show formally that our data encryption is secure against CPA (see Definition 2.1) through a privacy experiment involving an adversarial algorithm $\mathbb{A}$, relation $\mathbf{R}$, and a security parameter n. Without loss of generality, we suppose that $\mathbb{A}$ targets the privacy of $\mathbf{R}$. The definitions and security proof that follow are also applicable to an adversary targeting relation $\mathbf{S}$.

The privacy experiment, denoted by $\mathsf{Exp}^{REO}_{R,\mathbb{A}}(n)$, follows the standard procedure in cryptography for proving the security of an encryption against CPA. We enable the adversary to carry out CPA by giving it access to the following oracle. Using the oracle, the experiment simulates the ability of the adversary to obtain ciphertext corresponding to any plaintext it chooses, without knowing the encryption key.[1] This ability gives the adversary more advantage than a passive eavesdropper.

> ***R-Enc-Oracle (REO)***. Upon receiving an R-Enc-Oracle "query" representing a record $r \in \mathbf{R}$, REO returns a 8-tuple $\mathcal{A} = \langle \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6, \mathcal{A}_7, \mathcal{A}_8 \rangle \in \mathbb{G}^8$ produced with the `EncryptData` procedure.

The steps in the experiment are next given.

> *Step 1.* Adversary $\mathbb{A}$ issues a polynomial number of R-Enc-Oracle queries to REO to obtain the encryption of arbitrarily chosen records of $\mathbf{R}$.
> *Step 2.* Adversary $\mathbb{A}$ chooses two records $(\bar{r}_0, \bar{r}_1)$ from $\mathbf{R}$. It receives an encrypted tuple $\bar{\mathcal{A}} = \langle \bar{\mathcal{A}}_1, \bar{\mathcal{A}}_2, \bar{\mathcal{A}}_3, \bar{\mathcal{A}}_4, \bar{\mathcal{A}}_5, \bar{\mathcal{A}}_6, \bar{\mathcal{A}}_7, \bar{\mathcal{A}}_8 \rangle \in \mathbb{G}^8$ for $\bar{r}_\beta$, where $\beta \in_R \{0, 1\}$, produced with the `EncryptData` procedure.
> *Step 3.* Adversary $\mathbb{A}$ may issue a polynomial number of R-Enc-Oracle queries to REO to obtain the encryption of arbitrarily chosen records of $\mathbf{R}$, in the same way as in step 1. Finally, $\mathbb{A}$ stops and outputs β'.
> *Output.* The output of the experiment is defined to be 1 if $\beta' = \beta$, and 0 otherwise.

Since attribute $\mathbf{R}.A$ is not necessarily a primary key, $\mathbf{R}$ may have multiple records with the same key attribute value. Step 3 of the experiment thus allows the adversary to issue R-Enc-Oracle queries on either $\bar{r}_0$ or $\bar{r}_1$.

We now formally define privacy objective P1 by stating its mathematical meaning in the context of the attacker's capability.

Definition 4.2. An encryption scheme on a relation $\mathbf{R}$ ensures its *initial confidentiality* if, for all probabilistic polynomial-time algorithms $\mathbb{A}$, there exists a polynomially negligible function $\epsilon(n)$ of the security parameter n such that the adversary's probability of outputting 1 from the $\mathsf{Exp}^{REO}_{R,\mathbb{A}}(n)$ experiment differs from the probability of a successful random guess by $\epsilon(n)$; in other words,

$$\Pr\left[\mathsf{Exp}^{REO}_{R,\mathbb{A}}(n) = 1\right] = 1/2 + \epsilon(n).$$

Informally, the definition means that, in a CPA, an adversary can do no better than a random guess in determining the confidential attribute A in any $\mathbf{R}$ record.

4.2.2. Proof of Initial Confidentiality. We now prove that our protocol given in Section 3.2 achieves initial confidentiality as specified in Definition 4.2. Suppose on the contrary that there exists an algorithm $\mathbb{A}$ offering a nonnegligible advantage in determining

[1] It is a common practice to define oracles in a formal proof to model the adversary's noncomputational attack capability. For instance, an adversary may manipulate the database owner so that a special record (with a value known to the adversary) is encrypted and inserted into the database.

β in the $\mathsf{Exp}^{REO}_{R,\mathbb{A}}(n)$ experiment. This would allow the formulation of an algorithm $\mathbb{B}$ upon algorithm $\mathbb{A}$ that solves the decision linear problem with nonnegligible advantage. As this contradicts the decision linear Diffie-Hellman assumption (in Definition 2.3), algorithm $\mathbb{A}$ cannot exist.

Informally, algorithm $\mathbb{B}$ is built on $\mathbb{A}$ as follows: $\mathbb{B}$ receives an instance of the decision linear problem (x, v, h, x^a, v^b, W) and has to determine whether $W = h^{a+b}$. We set up $\mathbb{B}$ to encrypt records of relation $\mathbf{R}$, so $\mathbb{B}$ can simulate REO in the $\mathsf{Exp}^{REO}_{R,\mathbb{A}}(n)$ experiment performed by $\mathbb{A}$. In step 2 of the experiment, $\mathbb{B}$ receives records $(\bar{r}_0, \bar{r}_1)$ from $\mathbb{A}$ and encrypts $\bar{r}_\beta$, $\beta \in_R \{0, 1\}$, with the input parameters of the decision linear problem. If $\mathbb{A}$ successfully determines β, then $\mathbb{B}$ concludes that W is related to the other input parameters, specifically $W = h^{a+b}$.

THEOREM 4.3. *The encryption scheme presented in Section 3.2 ensures initial confidentiality, assuming that both the decision linear problem and the Decision 3-party Diffie-Hellman (D3DH) problem are intractable.*

PROOF. We give a sketch of the proof here and defer the complete proof to the Appendix.

We show that, if there is a Probabilistic Polynomial-Time (PPT) algorithm $\mathbb{A}$ that can defeat the encryption scheme with nonnegligible advantage (denoted by ϵ) over a random guess, then there exists a PPT algorithm $\mathbb{B}$ that is capable of successfully solving the decision linear problem with nonnegligible advantage.

Algorithm $\mathbb{B}$ takes as input an instance of the decision linear problem, (x, v, h, x^a, v^b, W) where $x, v, h \in \mathbb{G}$ and $a, b, c \in \mathbb{Z}_p$, such that W is random with 0.5 probability and $W = h^{a+b}$ with 0.5 probability. $\mathbb{B}$ successfully solves the problem if it outputs 1 when $W = h^{a+b}$, and 0 otherwise. Let ρ denote $\mathbb{B}$'s success probability.

$\mathbb{B}$ simulates the privacy experiment $\mathsf{Exp}^{REO}_{R,\mathbb{A}}(n)$ for $\mathbb{A}$. Choosing $\kappa_A, \tau_A, \sigma, \sigma_1 \in_R \mathbb{Z}_p$ and setting $g_1 = v, g_2 = g_1^\sigma$, $\mathbb{B}$ has all the secrets used in the `EncryptData` procedure described in Section 3.2 to simulate REO for the R-Enc-Oracle queries in steps 1 and 3 of the experiment. Upon receiving $(\bar{r}_0, \bar{r}_1)$ from $\mathbb{A}$ in step 2 of the experiment, $\mathbb{B}$ chooses $\beta \in_R \{0, 1\}$ and returns $\bar{\mathcal{A}} = \langle \bar{A}_1, \bar{A}_2, \bar{A}_3, \bar{A}_4, \bar{A}_5, \bar{A}_6, \bar{A}_7, \bar{A}_8 \rangle$ where

$$\bar{A}_1 = x^a, \bar{A}_2 = v^b, \bar{A}_3 = x^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2} \cdot W, \bar{A}_4 = x,$$
$$\bar{A}_5 = x^\sigma, \bar{A}_6 = x^{\sigma/\kappa_A}, \bar{A}_7 = g_1^z, z \in_R \mathbb{Z}_p, \bar{A}_8 = \mathbf{e}(x, x)^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2}.$$

If $\mathbb{A}$ determines β correctly from $\bar{\mathcal{A}}$ and the output of $\mathsf{Exp}^{REO}_{R,\mathbb{A}}(n)$ is 1, then $\mathbb{B}$ decides that $W = h^{a+b}$ and outputs 1; otherwise, $\mathbb{B}$ outputs 0.

We consider the two equally likely cases for W in the decision linear problem that $\mathbb{B}$ is trying to solve: (I) W is random; (II) $W = h^{a+b}$.

Case I. As $\bar{A}_3$ would be random in this case, $\mathbb{A}$ guesses randomly and outputs 0 with 1/2 probability. Therefore, $\mathbb{B}$'s success probability is 1/2 as well.

Case II. Except for $\bar{A}_3$ and $\bar{A}_7$, all the components of $\bar{\mathcal{A}}$ in $\mathbb{B}$'s simulation are identical to those in the original privacy experiment $\mathsf{Exp}^{REO}_{R,\mathbb{A}}(n)$.

Let ρ_0 denote the probability that adversary $\mathbb{A}$ manages to discern that $z \neq a \times \tau_A$ in $\bar{A}_7$, given $\bar{A}_1$ to $\bar{A}_6$ and $\bar{A}_8$. In the Appendix, we show that the D3DH problem is reducible to $\mathbb{A}$'s problem of discerning $\bar{A}_7$. Assuming that the D3DH problem is hard, ρ_0 can be only negligibly greater than 1/2.

Let ρ_1 denote the probability that $\mathbb{A}$ successfully distinguishes $\bar{A}_3 = x^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2} h^{a+b}$ in the simulation from $x^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2} g_2^{a+b}$ in the original privacy experiment. In the Appendix, we show that $\mathbb{A}$ has negligible advantage in distinguishing $x^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2} h^{a+b}$ from $x^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2} g_2^{a+b}$ over random guessing. This implies that ρ_1 is also negligibly greater than 1/2.

Since the differences in $\bar{\mathcal{A}}_3$ and $\bar{\mathcal{A}}_7$ are not discernible, $\mathbb{A}$ will complete the experiment with advantage ϵ and lead $\mathbb{B}$ to the right output; else $\mathbb{A}$'s output is of no help to $\mathbb{B}$. Therefore, $\mathbb{B}$'s success probability in Case II is $(1-\rho_0)(1-\rho_1)\epsilon + 1/2$.

Combining Cases I and II, $\mathbb{B}$'s probability of solving the decision linear problem is

$$\begin{aligned}\rho &= 1/2 \times 1/2 + 1/2((1-\rho_0)(1-\rho_1)\epsilon + 1/2) \\ &= 1/2 + \epsilon(1-\rho_0-\rho_1+\rho_1\rho_1)/2.\end{aligned}$$

Assuming that $\mathbb{B}$'s advantage in solving the decision linear problem is insignificant, ϵ must be negligible. This leads to the conclusion that no PPT adversary can defeat our encryption scheme. □

4.3. Safety of Equi-Join

4.3.1. Definition of Equi-Join Safety. We formalize privacy objective P2 through a privacy experiment in a similar fashion as in Section 4.2. Again, we suppose that $\mathbb{A}$ targets the privacy of relation **R**. We prove that any record of **R** that does not join with some **S** record remains secure against CPA. The definitions and security proof that follow are applicable to an adversary targeting relation **S** as well.

The privacy experiment, denoted by $\mathsf{Exp}^{REO,SEO}_{\bowtie(R,S),\mathbb{A}}(n)$, is an extension of the standard procedure in cryptography for proving the security of an encryption against CPA. We enable the adversary to carry out CPA by giving it access to the following oracles to obtain the encryption of arbitrarily chosen records of **R** and **S**.

R-Enc-Oracle (REO). : The same as in $\mathsf{Exp}^{REO}_{R,\mathbb{A}}(n)$.
S-Enc-Oracle (SEO). : Upon receiving an S-Enc-Oracle "query" representing a record $s \in \mathbf{S}$, SEO returns a 8-tuple $\mathcal{B} = \langle \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4, \mathcal{B}_5, \mathcal{B}_6, \mathcal{B}_7, \mathcal{B}_8 \rangle \in \mathbb{G}^8$ produced with the `EncryptData` procedure.

The steps in the experiment are next given.

Step 1. Adversary $\mathbb{A}$ issues a polynomial number of R-Enc-Oracle and S-Enc-Oracle queries to REO and SEO to obtain the encryption of arbitrarily chosen records of **R** and **S**, respectively.
Step 2. Adversary $\mathbb{A}$ receives a join query token $\mathcal{T}_{AB}$ and runs the ServerProcessing procedure using $\mathcal{T}_{AB}$ on **R** and **S**.
Step 3. Adversary $\mathbb{A}$ chooses two records $(\bar{r}_0, \bar{r}_1)$ from **R** such that $\forall s \in \mathbf{S}$, $s.B \neq \bar{r}_0.A$ and $s.B \neq \bar{r}_1.A$. It is given an encrypted tuple $\bar{\mathcal{A}} = \langle \bar{\mathcal{A}}_1, \bar{\mathcal{A}}_2, \bar{\mathcal{A}}_3, \bar{\mathcal{A}}_4, \bar{\mathcal{A}}_5, \bar{\mathcal{A}}_6, \bar{\mathcal{A}}_7, \bar{\mathcal{A}}_8 \rangle \in \mathbb{G}^8$ for $\bar{r}_\beta$, where $\beta \in_R \{0, 1\}$, produced with the `EncryptData` procedure.
Step 4. Adversary $\mathbb{A}$ may issue a polynomial number of R-Enc-Oracle and S-Enc-Oracle queries to REO and SEO to obtain the encryption of arbitrarily chosen records of **R** and **S**, respectively, in the same way as in step 1, with the restriction that $\forall s \in \mathbf{S}$, $s.B \neq \bar{r}_0.A$ and $s.B \neq \bar{r}_1.A$. Finally, $\mathbb{A}$ stops and outputs β'.
Output. The output of the experiment is defined to be 1 if $\beta' = \beta$, and 0 otherwise.

As in the previous experiment, the adversary is allowed to issue R-Enc-Oracle queries on either $\bar{r}_0$ or $\bar{r}_1$ in step 4. The main differences between the previous experiment $\mathsf{Exp}^{REO}_{R,\mathbb{A}}(n)$ and $\mathsf{Exp}^{REO,SEO}_{\bowtie(R,S),\mathbb{A}}(n)$ here are that in the latter: (1) $\mathbb{A}$ has access to the S-Enc-Oracle, and (2) $\mathbb{A}$ is given query token $\mathcal{T}_{AB}$.

Definition 4.4. An equi-join scheme $\bowtie$ on two relations **R** and **S** ensures *safety of the equi-join* if, for all probabilistic polynomial-time algorithms $\mathbb{A}$, there exists a polynomially negligible function $\epsilon(n)$ of the security parameter n such that

$$\Pr\left[\mathsf{Exp}^{REO,SEO}_{\bowtie(R,S),\mathbb{A}}(n) = 1\right] = 1/2 + \epsilon(n).$$

Informally, the definition means that, in a CPA, an adversary can do no better than a random guess in determining the confidential attribute A in any $\mathbf{R}$ record that is not part of the join result.

4.3.2. Proof of Equi-Join Safety. We now prove that our protocol given in Section 3.2 achieves query safety as specified in Definition 4.4. The proof is similar to the one in Section 4.2.2 for initial confidentiality: Suppose, on the contrary, that there exists an algorithm $\mathbb{A}$ offering a nonnegligible advantage in determining β in the $\mathsf{Exp}^{REO,SEO}_{\bowtie(R,S),\mathbb{A}}(n)$ experiment. This would allow the formulation of an algorithm $\mathbb{B}$ upon algorithm $\mathbb{A}$ that solves the decision linear problem with nonnegligible advantage. Since this contradicts the decision linear Diffie-Hellman assumption (in Definition 2.3), algorithm $\mathbb{A}$ cannot exist.

THEOREM 4.5. *The equi-join scheme $\bowtie$ presented in Section 3.2 ensures safety of the equi-join, assuming that both the decision linear problem and the Decision 3-party Diffie-Hellman* (D3DH) *problem are intractable.*

PROOF. We show that, if there exists a PPT algorithm $\mathbb{A}$ that breaks the privacy of $\bowtie$ with nonnegligible advantage over a random guess, then there exists a PPT algorithm $\mathbb{B}$ that successfully solves the decision linear problem with nonnegligible advantage as well.

The reduction is similar to the one in Theorem 4.3. To avoid repetition, this proof focuses only on the differences between the two privacy experiments. $\mathbb{B}$ runs in the same way as in the proof of Theorem 4.3, except that, additionally: (i) it simulates SEO by executing the `EncryptData` procedure to handle SEO queries, and (ii) it issues to $\mathbb{A}$ the query token $\mathcal{T}_{AB} = \langle \kappa_B/\tau_A, -\kappa_A/\tau_B \rangle$. Note that $\mathbb{B}$ can perform the simulation because it holds all the necessary secrets.

Next, we show that $\mathbb{A}$'s knowledge of $\mathcal{T}_{AB}$ does not change our security analysis in the proof for Theorem 4.3. Obviously, it does not affect ρ_I in Case I where W is a random element from $\mathbb{G}$. In Case II, $\mathbb{B}$ has already lowered the difficulty for $\mathbb{A}$ by exposing $\sigma, \sigma_1, \sigma_2, \kappa_A, h$ (in the detailed proof in the Appendix), so $\mathcal{T}_{AB}$ provides no additional knowledge to $\mathbb{A}$ in the experiment. Hence, the previous security analysis still holds, leading to the conclusion that $\mathbb{A}$ must have a negligible probability of defeating the safety of the equi-join scheme.

Finally, we observe that the transitivity property of equi-join may permit unintended disclosure. Suppose that the User issues an equi-join on attributes $\mathbf{R}.A$ and $\mathbf{S}.B$ with query token $\mathcal{T}_{AB}$, followed by an equi-join on $\mathbf{R}.A$ and some other attribute $\mathbf{T}.C$ with $\mathcal{T}_{AC}$. If records $r_i \in \mathbf{R}$, $s_j \in \mathbf{S}$ and $t_k \in \mathbf{T}$ meet the conditions $r_i.A = s_j.B$ of the first join and $r_i.A = t_k.C$ of the second join, the Server can deduce that $s_j.B = t_k.C$. This is a necessary consequence of the equi-joins.

5. JOIN PROCESSING ALGORITHMS

Having introduced our cryptographic construct, we now build upon it to create algorithms for equi-join $\mathbf{R} \bowtie_{A\_B} \mathbf{S}$. Our construct applies directly to $\mathbf{R}.A$ and $\mathbf{S}.B$ attributes that are of integer types, namely char, short, int, and long. For bool, float, and double values, we simply treat their internal representations as the binary representation of an integer. A string value can be treated as an array of char's or, more efficiently, an array of char blocks with each block mapped to an integer.

We start with a baseline algorithm that uses the conventional nested loop join strategy and show that it entails a high computation cost. To achieve scalability, we propose two algorithms based on the classical partitioning strategy. The Equivalence Partitioning (EP) algorithm organizes relation $\mathbf{R}$ (and relation $\mathbf{S}$) into equivalence classes, where an equivalence class contains all the $\mathbf{R}$ (respectively, $\mathbf{S}$) records with the same

A (respectively, B) value; this way, the Server only needs to test one representative record of an $\mathbf{R}$ class with a representative record of an $\mathbf{S}$ class to determine whether all the records of the $\mathbf{R}$ class join with all the records of the $\mathbf{S}$ class. To avoid any compromise in privacy, EP forms the equivalence classes gradually, as they are discovered during the join execution. The Hash Partitioning (HP) algorithm is an adaptation of the conventional hash join, with a consequent relaxation in privacy protection. Finally, we explain how EP and HP can be employed in tandem.

5.1. Baseline Algorithm

Algorithm 1 gives the baseline algorithm based on a block nested loop. The cryptographic computations include 6 exponentiations in line 5, along with 10 multiplications and 5 bilinear mappings, in lines 10 to 15. Denoting the cost of exponentiation, bilinear mapping, and multiplication by c_{exp}, c_{map}, and c_{mul}, respectively, and the cardinality of $\mathbf{R}$ and $\mathbf{S}$ by $\|\mathbf{R}\|$ and $\|\mathbf{S}\|$, the computation cost of the baseline algorithm is

$$Cost_{\text{Baseline}} = \|\mathbf{S}\|(6\, c_{exp}) + \|\mathbf{S}\| \cdot \|\mathbf{R}\|(10\, c_{mul} + 5\, c_{map}). \tag{3}$$

In our implementation described in Section 7, c_{exp} and c_{map} are on the order of milliseconds whereas c_{mul} is on the order of microseconds. Hence the computation cost is dominated by the 5 bilinear mappings incurred for every possible pair of an $\mathbf{R}$ record and a $\mathbf{S}$ record in the last component in formula (3).

ALGORITHM 1: Baseline Algorithm for Equi-Join

1: Let M be the number of buffer pages.
2: **while** there are unprocessed records in $\mathbf{S}$ **do**
3: Load the next $M-2$ pages of records from $\mathbf{S}$.
4: **for all** buffered $\mathbf{S}$ records s_j **do**
5: Compute $\mathcal{B}_{j,1}^{-1}$, $\mathcal{B}_{j,2}^{-1}$, $\mathcal{B}_{j,3}^{-1}$, $\mathcal{B}_{j,6}^{\kappa_B/\tau_A}$, $\mathcal{B}_{j,7}^{-\kappa_A/\tau_B}$, $\mathcal{B}_{j,8}^{-1}$.
6: **while** there are unprocessed records in $\mathbf{R}$ **do**
7: Load the next page of records from $\mathbf{R}$.
8: **for all** buffered $\mathbf{S}$ records s_j **do**
9: **for all** buffered $\mathbf{R}$ records r_i **do**
10: Compute $T_0 = \mathbf{e}(\mathcal{A}_{i,3} \cdot \mathcal{B}_{j,3}^{-1}, \mathcal{A}_{i,4} \cdot \mathcal{B}_{j,4})$.
11: Compute $T_1 = \mathbf{e}(\mathcal{A}_{i,1} \cdot \mathcal{B}_{j,1}^{-1}, g_2)$.
12: Compute $T_2 = \mathbf{e}(\mathcal{B}_{i,6}^{\kappa_B/\tau_A}, \mathcal{A}_{j,7})$.
13: Compute $T_3 = \mathbf{e}(\mathcal{A}_{i,6}, \mathcal{B}_{j,7}^{-\kappa_A/\tau_B})$.
14: Compute $T_4 = \mathbf{e}(\mathcal{A}_{i,2} \cdot \mathcal{B}_{j,2}^{-1}, \mathcal{A}_{i,5} \cdot \mathcal{B}_{j,5})$.
15: **if** $(T_0 = \mathcal{A}_{i,8} \cdot \mathcal{B}_{j,8}^{-1} \cdot T_1 \cdot T_2 \cdot T_3 \cdot T_4)$ **then**
16: Add $\langle r_i, s_j \rangle$ to the join result.

5.2. Equivalence Partitioning Algorithm

Our Equivalence Partitioning (EP) algorithm aims to avoid evaluating every possible pair of $\mathbf{R}$ record and $\mathbf{S}$ record. The strategy is to dynamically group those $\mathbf{S}$ records that are found to join with the same $\mathbf{R}$ record, implying that the $\mathbf{S}$ records share the same underlying B value. This way, each subsequent record $r \in \mathbf{R}$ only need be tested once against any group member; if and only if r joins with that member does r join with every $\mathbf{S}$ record in the group. We emphasize that EP adheres to our privacy objective P1 on initial confidentiality, as well as objective P2 on safety of equi-join as detailed in

Section 3.1. This is because the strategy affects only those **S** records that have joined with some **R** record. Following an overview of the essence of EP, we present the data structures and detailed algorithm.

5.2.1. Overview of Equivalence Partitioning. We assume initially that relation **S** fits entirely in memory; the assumption will be removed in the complete EP algorithm. The strategy is as follows.

(1) Load **S** into memory, with each record $s_j \in \mathbf{S}$ forming its own equivalence class $\mathcal{C}_j = \{s_j\}$. Let L denote the resulting set of equivalence classes.
(2) For every record $r \in \mathbf{R}$, initialize $\mathcal{C} = \emptyset$.
   (a) For every equivalence class $\mathcal{C}_j \in L$, let s be any record in $\mathcal{C}_j$. If $\langle r, s\rangle$ satisfy the condition in formula (1), set $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_j$ and $L \leftarrow L \backslash \{\mathcal{C}_j\}$.
   (b) Set $L \leftarrow L \cup \{\mathcal{C}\}$.
   (c) For every record $s \in \mathcal{C}$, add $\langle r, s\rangle$ to the join result.

Provided attribute B is not unique in **S**, the number of equivalence classes in L is expected to decline as we iterate through the records in **R**. Since all the records within an equivalence class $\mathcal{C}_j$ share the same B value, we need only test each **R** record against any one **S** record in $\mathcal{C}_j$ in step 2(a). Consequently, the number of times that formula (1) is evaluated per **R** record should decline over time.

Now, when $|\mathbf{S}|$ is large, the Server may not be able to load the entire relation **S** into memory. In this situation, the Server will process **S** over several passes. In each pass, the Server loads a block of **S** records, groups them into equivalence classes using the records in **R**, then writes the classes to disk. Concurrently, those **R** records that match the same **S** record are grouped into equivalence classes. After processing all the records in **S**, the equivalence classes generated across the passes are merged to derive the final partitioning of **S**. Finally, the Server performs a cross-product of the records in each equivalence class of **S** with the records in the corresponding equivalence class of **R** to produce the join result. This is the quintessence of our EP algorithm.

5.2.2. Data Structures. To reduce the number of passes, EP attempts to pack as many equivalence classes as possible in memory for each pass. In lieu of full records, EP keeps only minimal information for record identification and testing the join condition. The temporary files used in the algorithm include "**R**-classes" for the equivalence classes on **R**, "**S**-classes.i" for the equivalence classes on **S** produced in pass i, and "**R**-unmatched" for the identifiers of the unmatched **R** records.

The data structure for the equivalence classes on **S** is as follows.

```
struct EClass {
   sno : int; // serial number for each class
   Slist : sorted list; // identifier of S records, in ascending order
   B : G^8; // ciphertext for the common B value of the S records in the class
   Rcnt : int; // number of matching R records
};
```

Each equivalence class tracks in *Slist* the (physical) identifier of the **S** records that belong to it, in ascending order. Since all the **S** records in the class share the same B attribute value, it suffices to adopt any one of their ciphertexts for the $\mathcal{B}$ value of the class. *sno* is a serial number that runs across passes, that is, it does not reset on each pass. The *sno* of an equivalence class is assigned automatically upon creation, but may be explicitly overwritten. *Rcnt* is initially set to zero.

To create an equivalence class $\mathcal{C}$ for record $s_j \in \mathbf{S}$, the Server inserts the record identifier $s_j.id$ into $\mathcal{C}.Slist$ and copies into $\mathcal{C}.\mathcal{B}$ the ciphertext for $s_j.B$, $\langle \mathcal{B}_{j,1}, \mathcal{B}_{j,2}, \mathcal{B}_{j,3}, \mathcal{B}_{j,4}, \mathcal{B}_{j,5}, \mathcal{B}_{j,6}, \mathcal{B}_{j,7}, \mathcal{B}_{j,8}\rangle$. To merge equivalence class $\mathcal{C}_1$ *into* class $\mathcal{C}_2$, the Server

ALGORITHM 2: Equivalence Partitioning Algorithm for Equi-Join

```
1: Initialize pass = 1.
2: Invoke EP-Pass1 on the first block of S records.
3: Coalesce Rlist entries for the same sno in R-classes.
4: while there are unprocessed records in S do
5:     Increment pass by 1.
6:     Invoke EP-PassN(pass) on the next block of S records.
7:     Coalesce Rlist entries for the same sno in R-classes.
8: Merge the C_j classes in S-classes.i (1 ≤ i ≤ pass) by C_j.sno into S-classes.
9: for all ⟨sno, Rlist⟩ in R-classes do
10:    Find ⟨sno, Slist⟩ in S-classes.
11:    for all record r_i ∈ R such that r_i.id ∈ Rlist do
12:        for all record s_j ∈ S such that s_j.id ∈ Slist do
13:            Add ⟨r_i, s_j⟩ to the join result.
```

simply sets $\mathcal{C}_2.Slist = \mathcal{C}_2.Slist \cup \mathcal{C}_1.Slist$, $\mathcal{C}_2.Rcnt = \mathcal{C}_2.Rcnt + \mathcal{C}_1.Rcnt$ and discards $\mathcal{C}_1$. With our system configuration (described in Section 7), the merged class is always smaller than the combined space occupied by the two initial classes. This ensures that the memory usage does not grow during a pass, which is important in enabling the Server to load its buffer fully at the start of a pass.

As the Server executes the EP algorithm, it registers in "**R**-classes" the classification of the matched **R** records, in a series of tuples $\langle sno, Rlist\rangle$ denoting that the **R** records corresponding to the identifiers in *Rlist* match the equivalence class of **S** having serial number *sno*. This also implies that these **R** records share the same *A* attribute value. When "**R**-classes" contains two tuples for the same equivalence class of **S**, say $\langle sno, Rlist_1\rangle$ and $\langle sno, Rlist_2\rangle$, they may be coalesced by setting $Rlist_1 = Rlist_1 \cup Rlist_2$ before discarding the second tuple.

5.2.3. Algorithm. Algorithm 2 gives the outline of the EP algorithm. It starts with the **EP-Pass1** procedure that processes the first block of records in **S** by making one pass over **R**. As illustrated in Figure 2(a), the Server creates in L a list of equivalence classes from the first block of **S** records. Subsequently, each **R** record is matched against the classes in L in turn, using formula (1). Suppose that $r_1 \in \mathbf{R}$ matches $\mathcal{C}_2 = \langle 2, [s_2.id], ..\rangle$, a tuple $\langle 2, [r_1.id]\rangle$ is written to **R**-classes. Moving down L, r_1 matches another class $\mathcal{C}_{13} = \langle 13, [s_{13}.id], ..\rangle$, which gets merged into the first matching class (Figure 2(b)). After processing r_1, the merged class $\mathcal{C}_2 = \langle 2, [s_2.id, s_{13}.id]\rangle$ is pushed to the end of L, as in Figure 2(c). The figure also shows that $\mathcal{C}_{13}$ has been removed from between $\mathcal{C}_{12}$ and $\mathcal{C}_{14}$. Now suppose that $r_4 \in \mathbf{R}$ matches $\mathcal{C}_2$, a tuple $\langle 2, [r_4.id]\rangle$ is written to **R**-classes. Any **R** record without a matching class in L is sent to **R**-unmatched. The detailed procedure for **EP-Pass1** will be shortly described. After completing **EP-Pass1**, tuples in **R**-classes that reference the same equivalence class of **S** are combined. For example, $\langle 2, [r_1.id]\rangle$ and $\langle 2, [r_4.id]\rangle$ generated before are combined into $\langle 2, [r_1.id, r_4.id]\rangle$.

Next, EP iterates through the remaining blocks of **S** records. For each **S** block, the Server executes the **EP-PassN** procedure. First, L is repopulated from the current **S** block. As illustrated in Figure 3(a), the *sno* for the classes continues from the previous pass. Next, each equivalence class $\langle sno, Rlist\rangle$ in **R**-classes is matched against the classes in L. This is done by matching *any* record referenced in *Rlist* against L. Suppose that $\langle 2, [r_1.id, r_4.id]\rangle$ matches $\mathcal{C}_{101} = \langle 101, [s_{101}.id], ..\rangle$, the latter is renumbered to $\mathcal{C}_2 = \langle 2, [s_{101}.id], ..\rangle$. Turning to Figure 3(b), $\langle 2, [r_1.id, r_4.id]\rangle$ also matches $\mathcal{C}_{137} = \langle 137, [s_{137}.id], ..\rangle$, so it is merged into $\mathcal{C}_2$. After processing $\langle 2, [r_1.id, r_4.id]\rangle$, the renumbered $\mathcal{C}_2$ is pushed to the back of L as shown in Figure 3(c). Subsequent equivalence classes

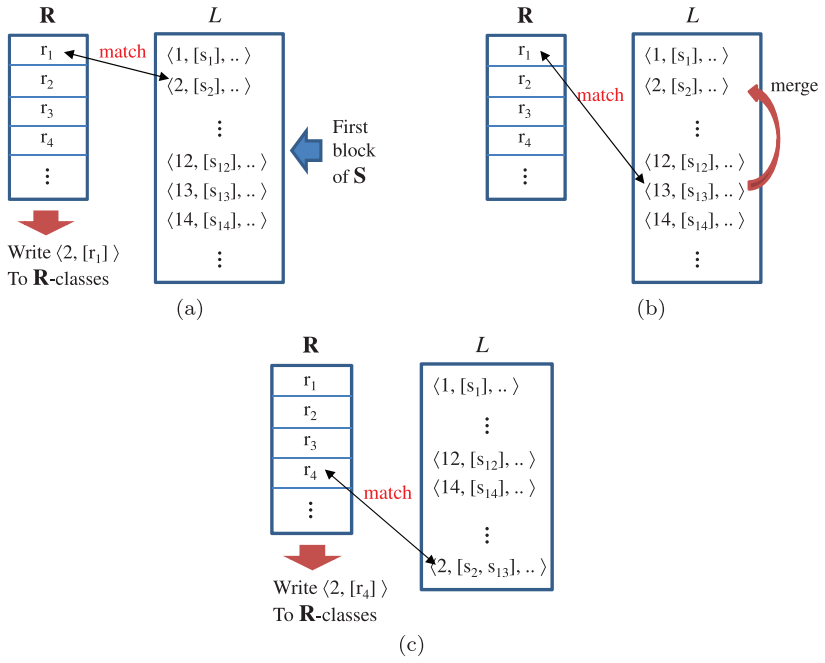


Fig. 2. Equivalence partitioning algorithm: Pass 1.

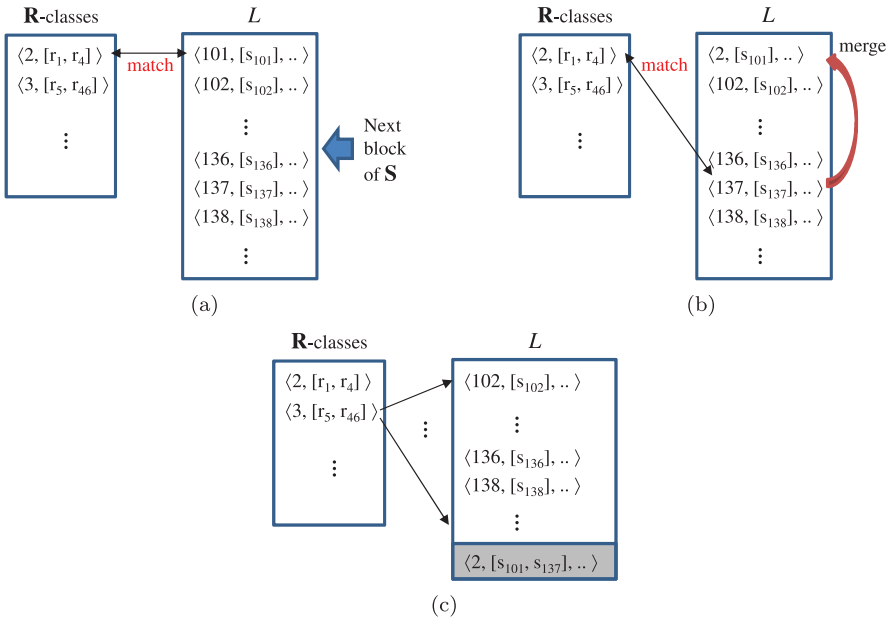


Fig. 3. Equivalence partitioning algorithm: subsequent passes.

in **R**-classes need only be matched with the classes at the front of L, as the former are guaranteed to be disjoint with the renumbered classes (shaded in grey) at the back like $\mathcal{C}_2$. After processing **R**-classes, the **R** records in **R**-unmatched are processed against the classes at the front of L (i.e., skipping the renumbered classes), following the same logic as in **EP-Pass1**. To round out the iteration, **R**-classes is scanned to combine any tuples that reference the same equivalence class of **S**.

After all the **S** records have been processed, line 8 merges the equivalence classes generated in different passes. Consider two classes $\mathcal{C}_i$ and $\mathcal{C}_j$ from different passes. If $\mathcal{C}_i$ and $\mathcal{C}_j$ have the same B value, they must have matched the same **R** record(s) earlier and been set to the same *sno*, for example, $\mathcal{C}_2$ in Figure 2 and the renumbered $\mathcal{C}_2$ in Figure 3. Hence it suffices to merge the equivalence classes by their *sno*'s.

Finally, within each equivalence class, every r_i record in the R list pairs with every s_j record in the corresponding S list to produce a result tuple (lines 9 to 13).

The detailed procedures for **EP-Pass1** and **EP-PassN** are given next.

EP-Pass1: Referring to Algorithm 3, lines 2 to 5 create the equivalence classes from the first block of **S** records and prepare them for evaluation with formula (1). Following this, the Server iterates through the pages within **R**. The **R** records in each page are loaded in line 7. Each **R** record r_i is then matched against the equivalence classes with formula (1) (in lines 13 to 18). Upon finding the first matching class $\mathcal{C}_j$ for r_i (line 19), the Server registers the match to **R**-classes (line 20) and checks whether $\mathcal{C}_j.R$ contains any other record (line 22). If not, r_i is the first **R** record that matches $\mathcal{C}_j.B$; the Server marks $\mathcal{C}_j$ (line 23), then continues with the **for** loop in line 10 to look for more matching classes. If some other **R** record had matched $\mathcal{C}_j.R$ previously, $\mathcal{C}_j$ has already been merged with all the classes sharing the same B value in L, so there is no need to check r_i against the remaining classes (line 25). In case $\mathcal{C}_j$ is not the first matching equivalence class for r_i, the Server merges $\mathcal{C}_j$ into the previous matching class (lines 26 to 28).

One optimization that is built into the algorithm is that the class $\mathcal{C}_{prev}$ that matches the current **R** record is pushed to the back of L (in line 32). By keeping such matched classes that are guaranteed disjoint with every other class, at the back of L, the Server can avoid evaluating them against any **R** record associated with previously unmatched classes (lines 11 to 12).

After processing all the **R** records, the equivalence classes in L that do not have matching **R** records cannot contribute to the join result and are discarded in line 33. The remaining classes are ordered by *sno* and written to **S**-classes.1 (lines 34 to 35).

EP-PassN: Algorithm 4 begins by creating equivalence classes from the next block of **S** records and preparing them for evaluation (lines 2 to 5). For each tuple $\langle sno, Rlist \rangle$ in **R**-classes, *one* of the records $r_i \in \mathbf{R}$ referenced in *Rlist* is retrieved for its encrypted A value (lines 6 to 7), to be used to match against the classes in L (lines 12 to 17). Upon finding the first matching class $\mathcal{C}_j$, its *sno* is reset to the current *sno* (line 19). If this is the first match for $\mathcal{C}_j$, the Server tracks it (in line 22) and continues with the **for** loop in line 9 to look for more matching classes; otherwise, $\mathcal{C}_j$ has already been merged with all the classes sharing the same B value in the current pass, so there is no need to check r_i against the remaining classes (line 24). In case $\mathcal{C}_j$ is not the first matching class for r_i, the former is merged into the previous matching class (lines 25 to 27).

As in the first pass, the matched class $\mathcal{C}_{prev}$ is pushed to the back of L (in line 29). Since the tuples in **R**-classes represent equivalence classes of **R**, a class in L that has matched some tuple of **R**-classes cannot match any more tuples from it. This is why matched classes at the back of L are skipped (lines 10 to 11).

After processing **R**-classes, the matched classes at the back of L are sorted on *sno* and written out (lines 30 to 34). The Server then proceeds to match the records in

ALGORITHM 3: EP-Pass1

1: Allocate 1 input buffer for **R**, 1 output buffer for **R**-classes, 1 output buffer for **R**-unmatched.
2: Load $M - 3$ pages of equivalence classes from the first block of **S** records.
3: Denote the equivalence classes in memory by L.
4: **for all** equivalence class $\mathcal{C}_j \in L$ **do**
5: Compute $\mathcal{B}_{j,1}^{-1}$, $\mathcal{B}_{j,2}^{-1}$, $\mathcal{B}_{j,3}^{-1}$, $\mathcal{B}_{j,6}^{\kappa_B/\tau_A}$, $\mathcal{B}_{j,7}^{-\kappa_A/\tau_B}$, $\mathcal{B}_{j,8}^{-1}$.
6: **while** there are unprocessed records in **R** **do**
7: Load the next page of records from **R**.
8: **for all** buffered **R** records r_i **do**
9: Set $prev = 0$.
10: **for all** equivalence class $\mathcal{C}_j \in L$ **do**
11: **if** (($prev > 0$) and ($\mathcal{C}_j.Rcnt > 0$)) **then**
12: Exit the immediate **for** loop.
13: Compute $T_0 = \mathbf{e}(\mathcal{A}_{i,3} \cdot \mathcal{B}_{j,3}^{-1}, \mathcal{A}_{i,4} \cdot \mathcal{B}_{j,4})$.
14: Compute $T_1 = \mathbf{e}(\mathcal{A}_{i,1} \cdot \mathcal{B}_{j,1}^{-1}, g_2)$.
15: Compute $T_2 = \mathbf{e}(\mathcal{B}_{i,6}^{\kappa_B/\tau_A}, \mathcal{A}_{j,7})$.
16: Compute $T_3 = \mathbf{e}(\mathcal{A}_{i,6}, \mathcal{B}_{j,7}^{-\kappa_A/\tau_B})$.
17: Compute $T_4 = \mathbf{e}(\mathcal{A}_{i,2} \cdot \mathcal{B}_{j,2}^{-1}, \mathcal{A}_{i,5} \cdot \mathcal{B}_{j,5})$.
18: **if** ($T_0 = \mathcal{A}_{i,8} \cdot \mathcal{B}_{j,8}^{-1} \cdot T_1 \cdot T_2 \cdot T_3 \cdot T_4$) **then**
19: **if** ($prev = 0$) **then**
20: Write $\langle \mathcal{C}_j.sno, [r_i.id] \rangle$ to **R**-classes.
21: Increment $\mathcal{C}_j.Rcnt$ by 1.
22: **if** ($\mathcal{C}_j.Rcnt = 1$) **then**
23: Set $prev = j$.
24: **else**
25: Exit the immediate **for** loop.
26: **else**
27: Merge $\mathcal{C}_j$ into $\mathcal{C}_{prev}$.
28: Remove $\mathcal{C}_j$ from L.
29: **if** ($prev = 0$) **then**
30: Write $r_j.id$ to **R**-unmatched.
31: **else**
32: Move $\mathcal{C}_{prev}$ to the back of L.
33: Discard from L all classes $\mathcal{C}_j$ with $\mathcal{C}_j.Rcnt = 0$, and sort the remaining classes in ascending $\mathcal{C}_j.sno$.
34: **for all** class $\mathcal{C}_j \in L$ **do**
35: Write tuple $\langle \mathcal{C}_j.sno, \mathcal{C}_j.Slist \rangle$ to **S**-classes.1.

R-unmatched against the remaining classes in L, following the same logic in the first pass. This is an optimization exploiting the fact that records in **R**-unmatched are outside of the equivalence classes in **R**-classes, so the former cannot match any class of **S** having matched the latter. We defer a detailed analysis of EP's costs to Section 6.

There is an interesting parallel between our EP algorithm and existing work on optimizing queries involving expensive predicates (e.g., Hellerstein and Stonebraker [1993] and Gaede and Günther [1994]). A key technique in the latter is to cache the results of predicate evaluations. Before invoking the predicate function on a value,

ALGORITHM 4: EP-PassN(pass)

Allocate 1 I/O buffer for **R**-classes, 1 input buffer for existing **R**-unmatched, 1 output buffer for new **R**-unmatched.
Load $M-3$ pages of equivalence classes from the next block of **S** records.
... Lines 3 to 5 from Algorithm 3 ...
for all $\langle sno, Rlist\rangle$ in **R**-classes **do**
    Let r_i be *any* **R** record such that $r_i.id \in Rlist$.
    Set $prev = 0$.
    for all equivalence class $\mathcal{C}_j \in L$ **do**
        if $(\mathcal{C}_j.Rcnt > 0)$ **then**
            Exit the immediate **for** loop.
... Lines 13 to 17 from Algorithm 3 ...
        if $(T_0 = \mathcal{A}_{i,8} \cdot \mathcal{B}_{j,8}^{-1} \cdot T_1 \cdot T_2 \cdot T_3 \cdot T_4)$ **then**
            if $(prev = 0)$ **then**
                Set $\mathcal{C}_j.sno = sno$.
                Increment $\mathcal{C}_j.Rcnt$ by 1.
                if $(\mathcal{C}_j.Rcnt = 1)$ **then**
                    Set $prev = j$.
                else
                    Exit the immediate **for** loop.
            else
                Merge $\mathcal{C}_j$ into $\mathcal{C}_{prev}$.
                Remove $\mathcal{C}_j$ from L.
    if $(prev > 0)$ **then**
        Move $\mathcal{C}_{prev}$ to the back of L.
Sort the classes $\mathcal{C}_j$ in L in ascending $\mathcal{C}_j.sno$.
for all class $\mathcal{C}_j \in L$ **do**
    if $(\mathcal{C}_j.Rcnt > 0)$ **then**
        Write tuple $\langle \mathcal{C}_j.sno, \mathcal{C}_j.Slist\rangle$ to **S**-classes.$pass.
        Remove $\mathcal{C}_j$ from L.
for all record $r_i \in \mathbf{R}$ such that $r_i.id$ is in the existing **R**-unmatched **do**
... Lines 9 to 32 of Algorithm 3 ...
Discard from L all classes $\mathcal{C}_j$ with $\mathcal{C}_j.Rcnt = 0$, and sort them in ascending $\mathcal{C}_j.sno$.
for all class $\mathcal{C}_j \in L$ **do**
    Write tuple $\langle \mathcal{C}_j.sno, \mathcal{C}_j.Slist\rangle$ to **S**-classes.$pass.

the cache is checked first to see whether the result for that value has already been computed. The technique is analogous to the way that EP avoids evaluating an **R** record against two **S** records having been found to share the same B value. The difference is that EP is unable to determine directly whether two **S** records share the same B value, because they are probabilistically encrypted. Rather, this fact has to be discovered during query execution, after observing the two **S** records to join with some earlier **R** record.

5.3. Hash Partitioning Algorithm

Independently of EP, a common approach (for example, in Hacigumus et al. [2002] and Hore et al. [2004]) to lower the computation cost for equi-join is to hash-partition **R** on A and **S** on B, so only records in corresponding **R** and **S** partitions may pair with each

other to produce output records. Applied in our setting, hash partitioning can reduce the number of times that formula (1) is evaluated, as explained next.

Let ϕ_A denote the number of distinct values in $\mathbf{R}.A$, and ϕ_B the number of distinct values in $\mathbf{S}.B$. The number of partitions b is computed as

$$b = \min(\phi_A, \phi_B)/u \tag{4}$$

for some given privacy parameter u. This sets the expected number of distinct A (B) values per partition to at least u. The User applies a hash function with a secret key (known as a keyed hash message authentication code) to derive from the A value in each record of $\mathbf{R}$ a partition identifier pid_A, and from the B value in each record of $\mathbf{S}$ a partition identifier pid_B. The pid_A and pid_B values are encrypted to ensure initial confidentiality. To perform an equi-join between $\mathbf{R}$ and $\mathbf{S}$, the User includes the decryption keys for pid_A and pid_B in the query token (in addition to $\mathcal{T}_{AB}$) that is given to the Server. The decrypted pid_A and pid_B values then enable the Server to split $\mathbf{R}$ into partitions by pid_A and $\mathbf{S}$ into partitions by pid_B for the equi-join operation.

The hash partition approach continues to meet privacy objective P1 for initial confidentiality as defined in Section 3.1. However, once decrypted, the partition identifiers pid_A and pid_B disclose information even on records that do not satisfy the join condition. This necessitates the following relaxation of objective P2.

P2B′ For any records $r_i, r_j \in \mathbf{R}$ that do not join with any record of $\mathbf{S}$, the Server's probability of guessing correctly that $r_i.A = r_j.A$ is $\frac{1}{u}$ for given privacy parameter u. At the same time, the Server's probability of guessing correctly that $r_i.A \neq r_j.A$ is $\frac{u-1}{u}$ if r_i, r_j have the same pid_A value, or 1 if they do not, and similarly for any records $s_i, s_j \in \mathbf{S}$ that do not join with any record of $\mathbf{R}$.

To elaborate on P2B′, we observe that, without the pid_A values, the Server's probability of guessing correctly that $r_i.A = r_j.A$ is $\frac{1}{\phi_A}$, as well as the probability of guessing correctly that $r_i.A \neq r_j.A$ is $\frac{\phi_A - 1}{\phi_A}$. Hence, the information disclosed by the pid_A values helps to improve the Server's probability of guessing correctly that $r_i.A = r_j.A$ by $(\frac{1}{u} - \frac{1}{\phi_A})$ and that $r_i.A \neq r_j.A$ by $\max(\frac{\phi_A-1}{\phi_A} - \frac{u-1}{u}, 1 - \frac{\phi_A-1}{\phi_A}) = \max(\frac{\phi_A-1}{\phi_A} - \frac{u-1}{u}, \frac{1}{\phi_A})$. Likewise, the pid_B values raise the Server's probability of guessing correctly that $s_i.B = s_j.B$ by $(\frac{1}{u} - \frac{1}{\phi_B})$ and that $s_i.B \neq s_j.B$ by $\max(\frac{\phi_B-1}{\phi_B} - \frac{u-1}{u}, \frac{1}{\phi_B})$. The User may cap the extent of information disclosure, expressed as

$$\begin{aligned}\text{Disclosure}_{\text{HP}} = \max\Bigg(&\frac{1}{u} - \frac{1}{\phi_A}, \frac{\phi_A - 1}{\phi_A} - \frac{u-1}{u}, \frac{1}{\phi_A}, \\ &\frac{1}{u} - \frac{1}{\phi_B}, \frac{\phi_B - 1}{\phi_B} - \frac{u-1}{u}, \frac{1}{\phi_B}\Bigg),\end{aligned} \tag{5}$$

thus yielding a lower bound on u relative to ϕ_A and ϕ_B, as well as an upper bound on b. Formula (5) measures specifically the additional disclosure in an equi-join due to hash partitioning, in contrast to the general entropy measure advocated in Hore et al. [2004].

The disclosure measure in formula (5) reflects the expected situation where the hash partitions are roughly uniform in cardinality. In the worst case where a partition may contain only records with the same attribute value, the disclosure would obviously be much higher. We decided to base our disclosure measure on the expected situation for two reasons: (i) Hashing has been studied extensively and there are good hash functions that can produce roughly uniform partitions even for skewed distributions.

ALGORITHM 5: Hash Partitioning Algorithm for Equi-Join

1: Decrypt the pid_A and pid_B values.
2: Split $\mathbf{R}$ into b partitions by pid_A.
3: Split $\mathbf{S}$ into b partitions by pid_B.
4: **for** $i = 1$ to b **do**
5: Load partition i of $\mathbf{S}$.
6: **for all** buffered $\mathbf{S}$ records s_j **do**
7: Compute $\mathcal{B}_{j,1}^{-1}$, $\mathcal{B}_{j,2}^{-1}$, $\mathcal{B}_{j,3}^{-1}$, $\mathcal{B}_{j,6}^{\kappa_B/\tau_A}$, $\mathcal{B}_{j,7}^{-\kappa_A/\tau_B}$, $\mathcal{B}_{j,8}^{-1}$.
8: **while** there are unprocessed records in partition i of $\mathbf{R}$ **do**
9: Load the next page of records in partition i of $\mathbf{R}$.
10: **for all** buffered $\mathbf{S}$ records s_j **do**
11: **for all** buffered $\mathbf{R}$ records r_i **do**
12: Compute $T_0 = \mathbf{e}(\mathcal{A}_{i,3} \cdot \mathcal{B}_{j,3}^{-1}, \mathcal{A}_{i,4} \cdot \mathcal{B}_{j,4})$.
13: Compute $T_1 = \mathbf{e}(\mathcal{A}_{i,1} \cdot \mathcal{B}_{j,1}^{-1}, g_2)$.
14: Compute $T_2 = \mathbf{e}(\mathcal{B}_{i,6}^{\kappa_B/\tau_A}, \mathcal{A}_{j,7})$.
15: Compute $T_3 = \mathbf{e}(\mathcal{A}_{i,6}, \mathcal{B}_{j,7}^{-\kappa_A/\tau_B})$.
16: Compute $T_4 = \mathbf{e}(\mathcal{A}_{i,2} \cdot \mathcal{B}_{j,2}^{-1}, \mathcal{A}_{i,5} \cdot \mathcal{B}_{j,5})$.
17: **if** $(T_0 = \mathcal{A}_{i,8} \cdot \mathcal{B}_{j,8}^{-1} \cdot T_1 \cdot T_2 \cdot T_3 \cdot T_4)$ **then**
18: Add $\langle r_i, s_j \rangle$ to the join result.

(ii) The User can check whether the hash partitions are uniform. If they are not, the User can apply a different hash function or decide not to employ HP altogether.

Algorithm 5 gives the hash partition procedure. With the exception of lines 1 to 5 and lines 8 to 9, this algorithm is identical to Algorithm 1. Moreover, the expected number of iterations through lines 12 to 18 is $\frac{1}{b} \cdot \|\mathbf{R}\| \cdot \|\mathbf{S}\|$, therefore, the computation cost of the hash partition algorithm is

$$Cost_{\text{HP}} \;=\; \|\mathbf{S}\|(6\; c_{exp}) + \left\lceil \frac{\|\mathbf{S}\| \cdot \|\mathbf{R}\|}{b} \right\rceil (10\; c_{mul} + 5\; c_{map}). \tag{6}$$

$Cost_{\text{HP}}$ is nearly $\frac{1}{b}$ of $Cost_{\text{Baseline}}$ in formula (3). The difference between the two costs is the performance gain of HP at the expense of the information disclosure quantified by formula (5).

5.4. Combined Hash and Equivalence Partitioning Algorithm

The equivalence partitioning approach in Section 5.2 combines easily with the hash partitioning approach in Section 5.3 as follows: We hash-partition $\mathbf{R}$ and $\mathbf{S}$, then apply EP to each pair of $\mathbf{R}$-$\mathbf{S}$ partitions in turn. We call the combined algorithm Hash-cum-Equivalence Partitioning (HEP). The information disclosure attached to HEP is identical to that of HP in formula (5), while the computation cost of HEP is similar to that of EP as quantified in Section 6, with $\|\mathbf{R}\|$ and $\|\mathbf{S}\|$, respectively, replaced by $\|\mathbf{R}\|/b$ and $\|\mathbf{S}\|/b$.

5.5. Potential for Speedup through Parallel Execution

Even with EP and HP, an equi-join on large $\mathbf{R}$ and $\mathbf{S}$ relations will still invoke the matching operation in Eq. (1) on many pairs of $\mathbf{R}$-$\mathbf{S}$ records, leading to long processing times. An effective way to mitigate this overhead is through parallel execution. We envisage the following means to achieve parallelism.

—*Multithreading for the matching operation*. The five bilinear mappings $\mathbf{e}(\_, \_)$ constitute the computation bottleneck in Eq. (1). On modern multicore CPUs, the bilinear mappings can be carried out by concurrent threads, before combining their results to complete the matching test. This can speed up the join processing by 4× relative to sequential execution, as we show in Section 7.6.
—*Data parallelism*. Parallel execution for HP is identical to that for the classical hash join [Schneider and DeWitt 1989], where every corresponding pair of **R** and **S** partitions is assigned to a different server. With EP, we would iteratively replicate a block of **S** records on multiple processors and send each R record to *one* of the processors for matching. Equivalence classes discovered at a processor could be communicated immediately to the other processors, or the processors could synchronize their equivalence classes at the end of a pass.
—*GPU accelerators*. Several studies (e.g., Harrison and Waldron [2010] and Bose et al. [2013]) have reported how cryptographic protocols can be speeded up effectively by implementing them on GPUs. In particular, Katoh et al. [2011] and Bose et al. [2013] have investigated the implementation of pairing based cryptography on GPUs. These studies show good promises that the execution time of our cryptographic join protocol in Section 3.2 can be improved by porting the underlying cryptographic library to a GPU-based implementation.

6. COST ANALYSIS FOR EQUIVALENCE PARTITIONING ALGORITHM

To understand the behavior of the EP algorithm, this section gives a detailed analysis focusing on three important cases. We denote the number of tuples in **R** and **S** by $\|\mathbf{R}\|$ and $\|\mathbf{S}\|$. For simplicity, $\mathbf{R}.A$ and $\mathbf{S}.B$ have the same number of unique values, namely $\phi_A = \phi_B$. Further, $|EC|$ denotes the fraction of a memory page needed to hold an equivalence class structure. We take into account only computation cost here, which dominates I/O cost as shown in the experiments in Section 7. We also omit the costs of sorting and managing data structures in memory, which are orders of magnitude faster than the cryptographic operations.

Case A. Both $\mathbf{R}.A$ and $\mathbf{S}.B$ contain duplicates, that is, $\|\mathbf{R}\| > \phi_A$ and $\|\mathbf{S}\| > \phi_B$, such as where $\mathbf{R}.A$ and $\mathbf{S}.B$ are foreign keys.

Memory permitting, we want to load just enough tuples (in lines 4 and 5 of Algorithm 3) so that all the ϕ_B values of $\mathbf{S}.B$ are present in the **S** block for the first pass. This enables **R** to be partitioned into ϕ_A equivalence classes, hence minimizing the number of iterations in the **for** statement in line 35 of Algorithm 4. In general, the User will not know exactly how big an **S** block is needed for the first pass without examining the $\mathbf{S}.B$ values. Instead, the block size is estimated probabilistically as explained next. Let ω denote the number of tuples in the **S** block for each pass. For simplicity, we assume that $\frac{\|\mathbf{S}\|}{\omega}$ is a round number.

—In Pass 1, the ω initial classes in L created from the first block of **S** tuples are progressively grouped into ϕ_B equivalence classes by the **R** tuples. Hence, the number of iterations over lines 13 to 18 in Algorithm 3 is roughly $\|\mathbf{R}\| \cdot \frac{\omega+\phi_B}{2}$, giving the cost of Pass 1 as

$$Cost_{\text{EP-1}} = \omega \cdot (6\, c_{exp}) + \|\mathbf{R}\| \cdot \frac{\omega + \phi_B}{2} \cdot (10\, c_{mul} + 5\, c_{map}).$$

As the probability that a given $\mathbf{S}.B$ value does not appear in the first **S** block is

$$q_1 = \left(1 - \frac{1}{\phi_B}\right)^{\omega},$$

we expect to produce $(1-q_1)\phi_A$ **R**-classes and $q_1 \cdot \|\mathbf{R}\|$ tuples in **R**-unmatched for the next pass.

—The second pass starts by using **R**-classes to group $(1-q_1)\omega$ of the ω initial C_j's in L into $(1-q_1)\phi_B$ classes, and then remove them in lines 31 to 34 of Algorithm 4. The process involves $(1-q_1)\phi_A \cdot \frac{\omega+(1-q_1)\phi_B+q_1\cdot\omega}{2}$ iterations over lines 12 to 17 in Algorithm 4. The remaining $q_1 \cdot \omega$ entries in L are grouped into $q_1 \cdot \phi_B$ equivalence classes upon matching against the $q_1 \cdot \|\mathbf{R}\|$ tuples in **R**-unmatched in lines 35 through 59. This entails $q_1 \cdot \|\mathbf{R}\| \frac{q_1(\omega+\phi_B)}{2}$ invocations of formula (1) and results in $(1-q_2)\phi_A$ **R**-classes (including those produced in Pass 1) and $q_2 \cdot \|\mathbf{R}\|$ tuples in **R**-unmatched, where

$$q_2 = \left(1 - \frac{1}{\phi_B}\right)^{2\omega}$$

is the probability that a given $\mathbf{S}.B$ value does not appear within the first two blocks of **S**. The cost of Pass 2 is

$$\begin{aligned} Cost_{\text{EP-2}} \;=\; & \omega \cdot (6\, c_{exp}) \\ & + \left((1-q_1)\phi_A \frac{(1+q_1)\omega + (1-q_1)\phi_B}{2} + q_1^2 \cdot \|\mathbf{R}\| \frac{\omega+\phi_B}{2}\right) \\ & \times (10\, c_{mul} + 5\, c_{map}) \end{aligned}$$

—Similarly, in each subsequent pass i $(2 < i \le \frac{\|\mathbf{S}\|}{\omega})$, the number of iterations over lines 12 to 17 in Algorithm 4 is $(1-q_{i-1})\phi_A \cdot \frac{(1+q_{i-1})\omega+(1-q_{i-1})\phi_B}{2}$, and lines 35–59 involves $q_{i-1} \cdot \|\mathbf{R}\| \frac{q_{i-1}(\omega+\phi_B)}{2}$ invocations of formula (1). The cost of the pass is thus

$$\begin{aligned} Cost_{\text{EP-}i} \;=\; & \omega \cdot (6\, c_{exp}) \\ & + \left((1-q_{i-1})\phi_A \frac{(1+q_{i-1})\omega + (1-q_{i-1})\phi_B}{2} + q_{i-1}^2 \cdot \|\mathbf{R}\| \frac{\omega+\phi_B}{2}\right) \\ & \times (10\, c_{mul} + 5\, c_{map}) \end{aligned}$$

The pass ends with $(1-q_i)\phi_A$ **R**-classes and $q_i \cdot \|\mathbf{R}\|$ tuples in **R**-unmatched, where

$$q_i = \left(1 - \frac{1}{\phi_B}\right)^{i\times\omega}.$$

Summing over all the passes, the total cost of EP is

$$\begin{aligned} Cost_{\text{EP}} \;=\; & \|\mathbf{S}\| \cdot (6\, c_{exp}) \\ & + \left(\phi_A \sum_{i=2}^{\frac{\|\mathbf{S}\|}{\omega}} (1-q_{i-1}) \frac{(1+q_{i-1})\omega + (1-q_{i-1})\phi_B}{2}\right. \\ & \left. + \|\mathbf{R}\| \frac{\omega+\phi_B}{2} \left(1 + \sum_{i=2}^{\frac{\|\mathbf{S}\|}{\omega}} q_{i-1}^2\right)\right) \cdot (10\, c_{mul} + 5\, c_{map}). \end{aligned} \tag{7}$$

If we pick a value for ω such that q_1 is below 10%, q_1^2 as well as $q_2, q_3, \ldots, q_{\frac{\|\mathbf{S}\|}{\omega}}$ are negligible and formula (7) simplifies to

$$\begin{aligned} Cost_{\mathrm{EP}} \;=\; & \|\mathbf{S}\| \cdot (6\; c_{exp}) \\ & + \left(\phi_A (1-q_1) \frac{(1+q_1)\omega + (1-q_1)\phi_B}{2} \right. \\ & + \left. \left(\phi_A \left(\frac{\|\mathbf{S}\|}{\omega} - 2 \right) + \|\mathbf{R}\| \right) \frac{\omega + \phi_B}{2} \right) \\ & \times (10\; c_{mul} + 5\; c_{map}). \end{aligned} \tag{8}$$

Comparing $Cost_{\mathrm{EP}}$ with formula (3) for the baseline algorithm, we see that EP has the potential to reduce computation cost substantially where $\phi_A \ll \|\mathbf{R}\|$. Formulae (7) and (8) are also useful in deciding which of the relations in a foreign-key-to foreign-key join should be designated as $\mathbf{R}$ versus $\mathbf{S}$ in the EP algorithm.

An interesting characteristic of EP is that it may be counterproductive to utilize all the available memory to operate the algorithm with a large block size ω. The reason is that the last component in $Cost_{\text{EP-1}}$ (i.e., $\|\mathbf{R}\| \frac{\omega+\phi_B}{2}(10\; c_{mul} + 5\; c_{map})$) is proportional to ω. Instead, ω should be "just large enough" to lead to a small q_1. Given memory allocation M and a target q_1, w is set as

$$\omega = \min \left(\frac{\log(q_1)}{\log(1 - \frac{1}{\phi_B})}, \frac{M-3}{|EC|} \right). \tag{9}$$

We will investigate the setting of ω empirically in Section 7.

Case B. In performing a join $\mathbf{R} \bowtie_{A=B} \mathbf{S}$ in which $\mathbf{R}.A$ is a primary key and $\mathbf{S}.B$ a foreign key, we have a choice between a foreign-key to primary-key join (FP-join) where $\mathbf{S}$ is designated as outer relation and $\mathbf{R}$ as inner relation $\mathbf{R}$ as in Algorithm 2, or a primary-key to foreign-key join (PF-join) by swapping $\mathbf{R}$ and $\mathbf{S}$ in the algorithm.

We first consider how EP executes in the FP-join setting. Here, $\|\mathbf{R}\| = \phi_A = \phi_B$ and $\|\mathbf{S}\| > \phi_B$. With $M-3$ memory pages for $\mathbf{S}$, EP requires $\frac{\|\mathbf{S}\| \cdot |EC|}{M-3}$ passes as depicted in Figure 4; to simplify the notation, we assume that $\frac{\|\mathbf{S}\| \cdot |EC|}{M-3}$ is a round number. Each pass starts with $\frac{M-3}{|EC|}$ equivalence classes in L. Since $\mathbf{R}.A$ is a primary key, an $\mathbf{R}$ record will not match the merged classes produced by previous $\mathbf{R}$ records. Moreover, each $\mathbf{R}$ record merges roughly $\frac{M-3}{|EC| \cdot \phi_B}$ of the classes with matching B value in L. This is why the number of classes in L that are evaluated against each $\mathbf{R}$ record declines gradually within a pass. The number of iterations over lines 13 to 18 in Algorithm 3 in the first pass, and over lines 12 to 17 in Algorithm 4 in each subsequent pass, is thus $\frac{M-3}{|EC|} \times \frac{\|\mathbf{R}\|+1}{2}$. The total cost is

$$Cost_{\mathrm{EP(F-P)}} \;=\; \|\mathbf{S}\| \cdot \big(6\; c_{exp}\big) + \|\mathbf{S}\| \cdot \frac{\|\mathbf{R}\| + 1}{2} \cdot (10\; c_{mul} + 5\; c_{map}). \tag{10}$$

In this setting, EP roughly halves the computation cost of the baseline algorithm.

Next, we consider the PF-join setting with $\mathbf{R}$ as outer relation and $\mathbf{S}$ as inner relation. As depicted in Figure 5, EP executes in #pass $= \frac{\|\mathbf{R}\| \cdot |EC|}{M-3}$ passes; again, we assume for simplicity that $\frac{\|\mathbf{R}\| \cdot |EC|}{M-3}$ is a round number. Every pass initializes L with the next $\frac{M-3}{|EC|}$ records from $\mathbf{R}$ and matches them with the equivalence classes of $\mathbf{S}$. In the process, each $\mathbf{R}$ record matches roughly $\frac{\|\mathbf{S}\|}{\phi_B}$ of the equivalence classes of $\mathbf{S}$, that at the end of the pass are coalesced in line 3 or 7 of Algorithm 2 to leave $\frac{M-3}{|EC|}(\frac{\|\mathbf{S}\|}{\phi_B} - 1) = \frac{1}{\#\text{pass}}(\|\mathbf{S}\| - \phi_B)$

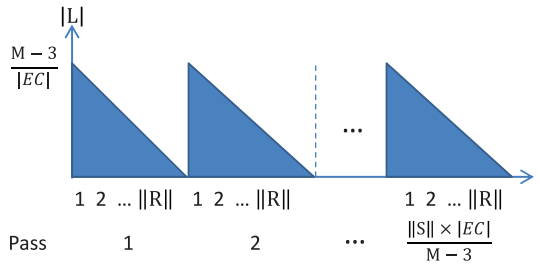
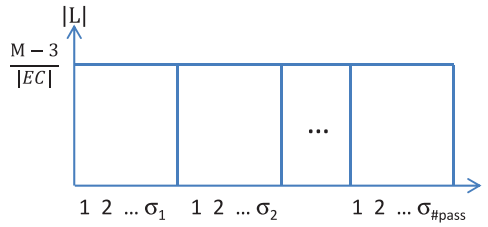


Fig. 4. EP with foreign-key-to-primary-key join.



- #pass = $\frac{\|R\| \times |EC|}{M-3}$
- $\sigma_i = \left(1 - \frac{i-1}{\#pass}\right)\|S\| + \frac{i-1}{\#pass}\phi_B$, $1 \le I \le \#pass$

Fig. 5. EP with primary-key-to-foreign-key join.

fewer $\mathbf{S}$ classes. In Pass i, $1 \le i \le$ #Pass, line 5 of Algorithm 3 or 4 is run $\frac{M-3}{|EC|}$ times, while lines 13 to 18 in Algorithm 3 or lines 12 to 17 in Algorithm 4 are executed $\frac{M-3}{|EC|}((1 - \frac{i-1}{\#pass})\|\mathbf{S}\| + \frac{i-1}{\#pass}\phi_B)$ times. The total cost is

$$\begin{aligned} Cost_{\text{EP(P−F)}} \;=\; & \|\mathbf{R}\| \cdot (6\, c_{exp}) \\ & + \left(\|\mathbf{R}\| \cdot \|\mathbf{S}\| \cdot \frac{\#pass + 1}{2 \cdot \#pass} + \|\mathbf{R}\| \cdot \phi_B \frac{\#pass - 1}{2 \cdot \#pass} \right) \\ & \times (10\, c_{mul} + 5\, c_{map}). \end{aligned} \tag{11}$$

A comparison of the dominant (last) component in formula (10) with the corresponding component in formula (11) indicates that $Cost_{\text{EP(F−P)}} < Cost_{\text{EP(P−F)}}$. We will confirm this finding empirically in Section 7.

Case C. Here $\mathbf{R}.A$ and $\mathbf{S}.B$ are unique attributes, namely $\|\mathbf{R}\| = \phi_A$ and $\|\mathbf{S}\| = \phi_B$, as in a join of the vertical partitions of a relation. In this case, every $\mathbf{R}$ record and $\mathbf{S}$ record belong to their own equivalence class, so EP offers no performance gain. In other words, the computation cost of EP is the same as $Cost_{\text{Baseline}}$ in formula (3).

7. EMPIRICAL EVALUATION

In this section, we empirically evaluate the performance of the privacy-preserving equi-join algorithms proposed in Section 5 and also verify the cost analysis in Section 6. The key questions to be investigated include the following.

—What is a good setting for q_1 in formula (9) for determining the block size ω in the equivalence partitioning algorithm?

—What relationship between M, $|EC|$ and the join operands would enable equivalence partitioning to be effective relative to the baseline algorithm?
—How does hash partitioning trade off between privacy protection and join performance?
—How scalable are the various join algorithms?

7.1. Experiment Setup

We begin by describing the setup of our experiments. The important workload and resource parameters, along with their default values, are included in Table I.

Competing algorithms. In our experiments, we evaluate and compare various join algorithms built on our cryptographic construct in Section 3; these are the `Baseline` algorithm described in Section 5.1 that is based on nested loop join, equivalence partitioning (EP) in Section 5.2, hash partitioning (HP) in Section 5.3, and hybrid Hash-cum-Equivalence partitioning (HEP) in Section 5.4. For HP and HEP, the number of partitions b is 10 by default. For comparison, we also include Carbunar-Sion's (CS) scheme [Carbunar and Sion 2012], which we combine with a block nested loop join strategy. We emphasize that CS offers a weaker privacy protection than our solution, as explained in Section 2.1, and is intended only to provide a performance baseline in the evaluation.

Workload. We use two workloads for our experiments. The first is a synthetic workload that contains two relations, $\mathbf{R}$ and $\mathbf{S}$, according to the schemata in Section 3.1. The attributes $\mathbf{R}.A$ and $\mathbf{S}.B$ are 8-byte integers and encrypted with the scheme in Section 3.2. The workload allows fine-grained control of the parameters $|\mathbf{R}|$, $\|\mathbf{R}\|$, ϕ_A, $|\mathbf{S}|$, $\|\mathbf{S}\|$, ϕ_B that are listed in Table I. The join query that we run is $\mathbf{R} \bowtie_{A=B} \mathbf{S}$.

To corroborate the findings obtained with the synthetic workload, we employ another workload extracted from TPC-E (http://www.tpc.org/tpce), a standard benchmark that models the online transaction processing (OLTP) system of a brokerage firm but is representative of OLTP systems in general. The relations that we use are: (a) the `Security` relation that contains records on 3,425 exchange-traded stocks and has a total size of 507Kbytes; (b) the `Holding_Summary` relation with 50,297 records on the security holdings in customer accounts and a size of 1.1Mbytes; and (c) the `Watch_Item` relation with 500,717 records on the securities in customers' watch lists and a size of 7.9Mbytes. We focus on two equi-join queries.

—Query 1:
SELECT * FROM Security $\mathbf{R}$, Holding_Summary $\mathbf{S}$
WHERE $\mathbf{R}$.S_SYMB = $\mathbf{S}$.HS_S_SYMB
—Query 2:
SELECT * FROM Holding_Summary $\mathbf{R}$, Watch_Item $\mathbf{S}$
WHERE $\mathbf{R}$.HS_S_SYMB = $\mathbf{S}$.WI_S_SYMB

The join attribute, a 15-character stock symbol, is the primary key in `Security` and a foreign key in both `Holding_Summary` and `Watch_Item`.

System configuration. Our experiments are carried out on Centos Linux servers that are equipped with Intel Xeon X5460 3.16 GHz quad-core CPUs and 16GB RAM. The databases are replicated on IBM-ESXS MBA3147RC hard disks that are formatted with 1Kbyte pages and mounted on each server. We implemented our cryptographic construct in C++ on the Stanford PBC library (http://crypto.stanford.edu/pbc). In instantiating the cryptographic construct, we set the group order p to a 160-bit prime number, thus providing 80 bits of security and equivalent in strength to 1024-bit RSA keys [Barker et al. 2012]. At this group order, each ciphertext is 140 bytes in size. This

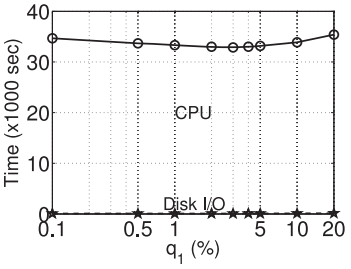


Fig. 6. Block size in EP.

translates to 17.5× space overhead for each integer attribute in the synthetic workload and 9.3× space overhead for each stock symbol in the TPC-E workload.

Performance metrics. Our primary concerns are the execution time of our equi-join algorithms, as well as any information disclosure as measured through formula (5). While we also report the space overhead for storing the ciphertext of the encrypted attributes, this is a secondary consideration to the first two metrics as modern disk storage offers abundant space at low cost.

7.2. Block Size in EP Algorithm

We begin by investigating the setting for q_1, that determines the block size ω in EP according to formula (9). Using the synthetic workload, we vary q_1 from 0.1% to 20%, while keeping the workload parameters at their default values in Table I that model a foreign-key-to-foreign-key join. The observed query execution times are shown in Figure 6. The figure breaks down the execution time between I/O time and computation time, respectively below and above the dotted line marked by star labels. Clearly, CPU cost dominates I/O cost here.

More importantly, the figure confirms that the number of equivalence classes populated from each block of $\mathbf{S}$ tuples should be neither too big nor too small, exactly as predicted in the cost analysis in Section 6. The lowest execution time is observed at $q_1 = 3\%$, so we adopt this setting by default.

7.3. Impact of Memory Allocation

Keeping to the synthetic workload, we next investigate the impact of memory allocation M to the performance of the various join algorithms. At $q_1 = 3\%$, the target number of equivalence classes $\omega = 3505$ from the block of $\mathbf{S}$ records occupy only 521 memory pages, implying EP should utilize at most 524 memory pages for this workload, including the input buffer for $\mathbf{R}$ and output buffers for $\mathbf{R}$-classes and $\mathbf{R}$-unmatched. Thus, we increase M progressively to 500 in the experiment. Figure 7 summarizes the CPU, I/O, and overall execution times.

Referring to Algorithm 1 for the baseline, a low memory allocation necessitates several iterations over $\mathbf{R}$, leading to high I/O cost. The I/O cost declines steadily with more memory. However, the dominant factor is CPU cost, which is determined by the product of the cardinality of $\mathbf{R}$ and $\mathbf{S}$ and independent of M. This is why the overall execution time of the baseline declines only marginally with a higher M.

For EP, the I/O cost falls steadily, as a more liberal memory allocation allows fewer passes. Concurrently, the CPU cost also drops, as a rising M allows the algorithm to operate closer to the target block size ω. Overall, EP's strategy of partitioning the input relations into equivalence classes is effective in giving it a 10 to 40× speedup over Baseline.

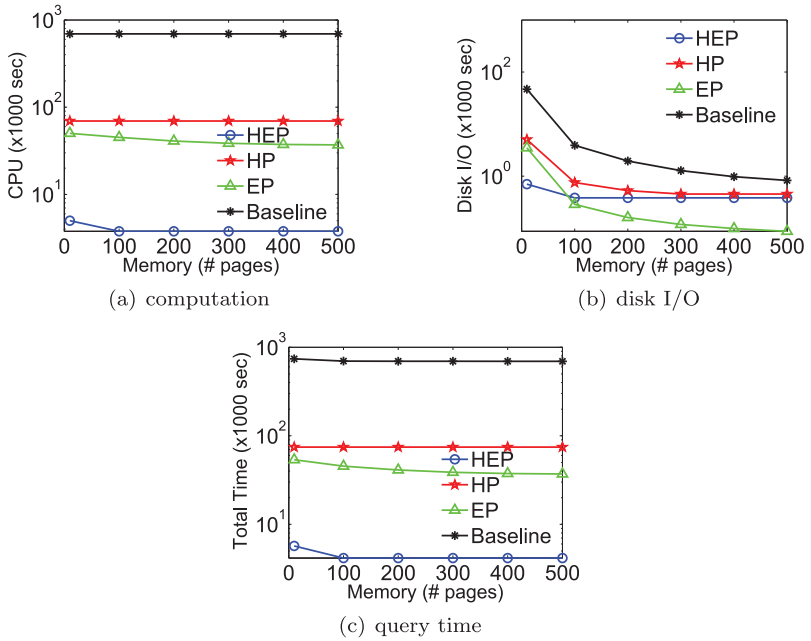


Fig. 7. Impact of memory allocation.

Turning to HP, we observe that at $b = 10$ the computation time is just over 10% that of Baseline, as predicted by formula (6). Like the two previous algorithms, the I/O cost incurred by HP reduces as the memory allocation permits fewer iterations over each $\mathbf{R}$ partition. Unlike EP that offers the same privacy protection as Baseline, HP achieves its performance improvement at the expense of information disclosure. At $b = 10$ partitions, the disclosure is 0.009 as determined with formula (5).

As for HEP, it incurs a lower I/O cost in iterating over the $\mathbf{R}$ partitions as compared to EP, which iterates over $\mathbf{R}$ itself. As more memory is allocated, the I/O cost for HEP stabilizes because of the fixed cost of carving $\mathbf{R}$ and $\mathbf{S}$ into partitions. However, the I/O consideration is overshadowed by CPU cost. Here, HEP performs substantially less computation in joining the smaller $\mathbf{R}$ and $\mathbf{S}$ partitions. With $\frac{\phi_A}{b} = \frac{\phi_B}{b} = 100$ unique key values per partition, HEP requires only up to 55 memory pages according to formula 9. This is why the CPU cost holds steady from $M = 100$ onwards in Figure 7(a). Overall, HEP achieves speedups of 12 to 244×, 1.3 to 6×, and 1.2 to 25× over Baseline, EP, and HP, respectively. Finally, we remark that HEP has an information disclosure of 0.009 here, the same as HP.

7.4. Choice of Outer versus Inner Relations

A question that pertains to all of our join algorithms is: where the input relations are asymmetrical, which should be the outer relation versus inner relation in executing the join? A particularly important scenario is where the join involves a primary key in one input relation and a foreign key in the other. In this experiment, we aim to find out whether the primary-key-to-foreign-key join (PF-join) with the primary key in the outer relation and foreign key in the inner relation is preferable over or the foreign-key-to-primary-key join (FP-join) that reverses the role of the two relations, in terms of the join algorithm.

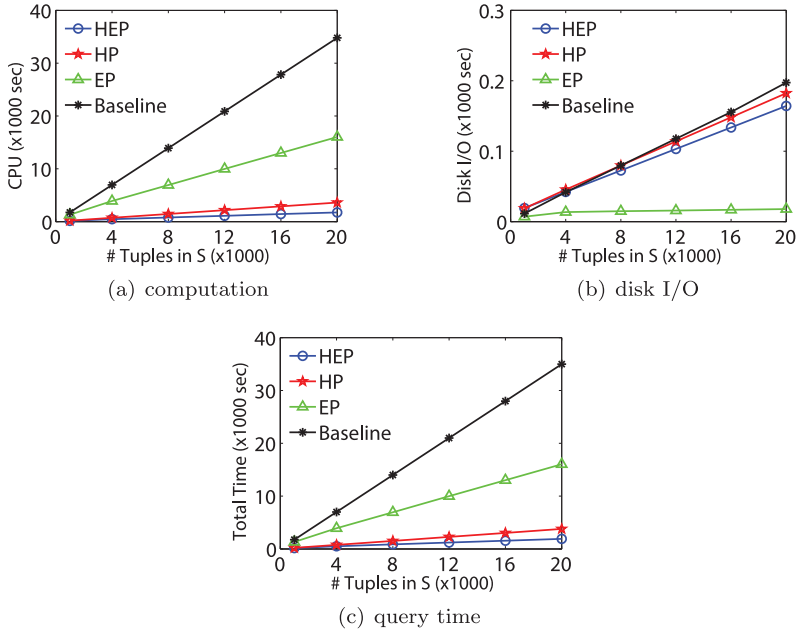


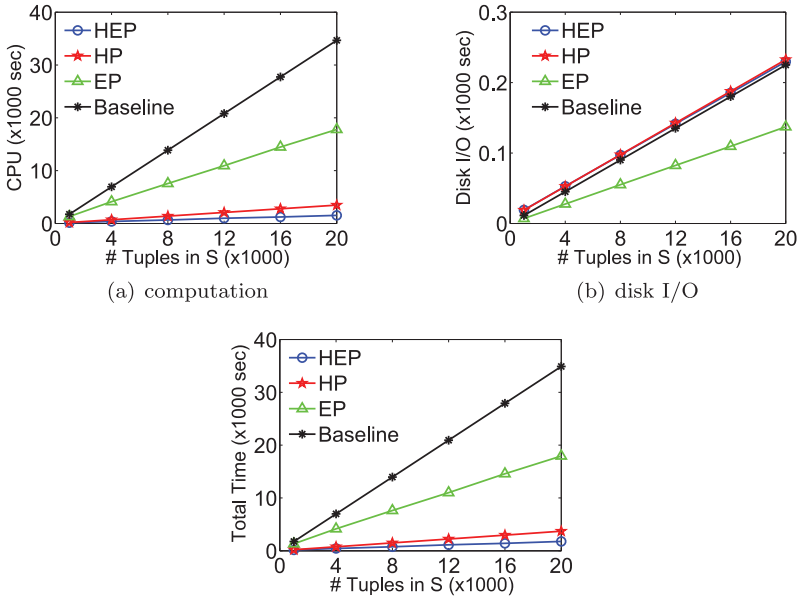
Fig. 8. Foreign-key-to-primary-key join.

In setting up the synthetic workload for FP-join, we fix $\phi_A = \phi_B = \|\mathbf{R}\| = 1{,}000$, $|\mathbf{R}| = 250$ pages, and $M = 100$ pages, while varying $\|\mathbf{S}\|$ and $|\mathbf{S}|$ together such that each page contains four records. For EP and HEP, we set $q_1 = 3\%$ following the guidance in Section 7.2. The results are plotted in Figure 8. Keeping the experiment parameters unchanged, we then swap the two relations such that $\mathbf{R}$ and $\mathbf{S}$ become the outer and inner relations, respectively, for PF-join; the results are summarized in Figure 9.

With the baseline algorithm, the number of I/Os is $|\mathbf{S}| + \frac{|\mathbf{S}|}{M-2} \cdot |\mathbf{R}|$ in FP-join and $|\mathbf{R}| + \frac{|\mathbf{R}|}{M-2} \cdot |\mathbf{S}|$ in PF-join. Since $|\mathbf{S}| > |\mathbf{R}|$, we might have expected FP-join to have a higher I/O than PF-join. However, the results in Figures 8(b) and 9(b) indicate the opposite. A closer examination reveals that this happens because the outer relation is fetched sequentially in blocks of $M-2$ pages, whereas the inner relation is fetched page-by-page, thus incurring a random I/O for each page. However, PF-join incurs a smaller CPU cost (though the difference is not obvious in the figures), which is consistent with formula (3). Overall PF-join is marginally more favorable to Baseline and thus by extension HP, that invokes the baseline algorithm on the partitions.

With EP, PF-join incurs higher CPU cost than FP-join as predicted by formulae (10) and (11). Even the I/O cost is higher in PF-join because, for each $\mathbf{S}$ record, an entry $\langle \mathcal{C}_j.sno, [s_i.id] \rangle$ is written to $\mathbf{S}$-classes (in line 20 of Algorithm 3 and line 47 of Algorithm 4) and these entries have to be loaded back into memory, merged, and written out after each pass (in line 8 of Algorithm 2). Overall, FP-join gives EP a clear advantage over PF-join. Similarly, while the difference between PF-join and FP-join is not apparent due to the scale of the figures, the latter is more favorable for HEP.

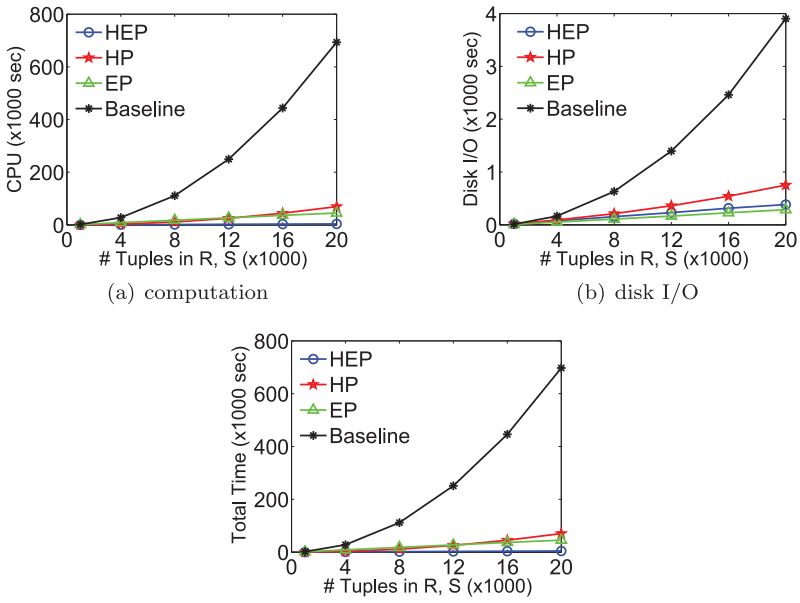
Moving to the next part of the experiment, we fix $\phi_A = \phi_B = 1{,}000$ and $M = 100$ pages while scaling $\|\mathbf{R}\|$ and $\|\mathbf{S}\|$ in tandem; $|\mathbf{R}|$ and $|\mathbf{S}|$ are also increased accordingly to maintain four records per page. As formula (3) predicts, the cost of Baseline is proportional to the product of $\|\mathbf{R}\|$ and $\|\mathbf{S}\|$. The prediction is confirmed in Figure 10 where Baseline's cost quadratically grows. This is so also for HP. Interestingly, the costs of EP and HEP



(a) computation (b) disk I/O

(c) query time

Fig. 9. Primary-key-to-foreign-key join.



(a) computation (b) disk I/O

(c) query Time

Fig. 10. Foreign-key-to-foreign-key join

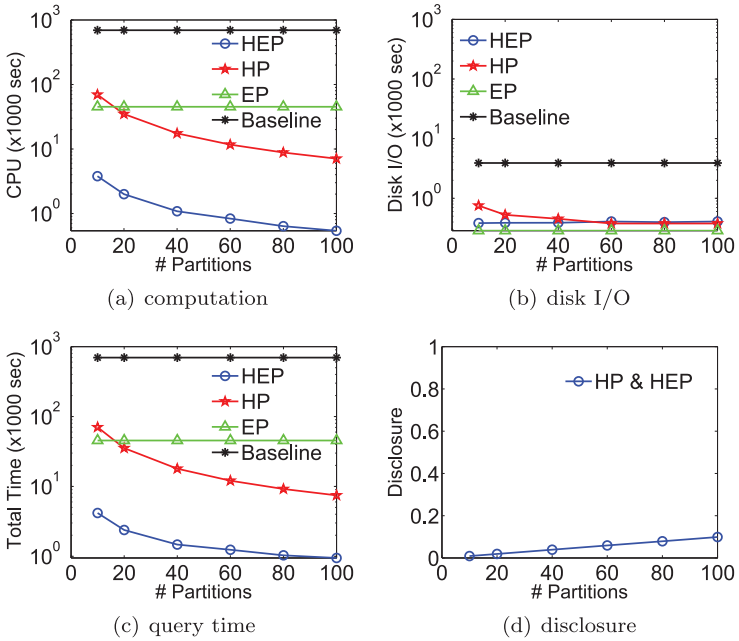


Fig. 11. Number of partitions.

rise only linearly even though both $\|\mathbf{R}\|$ and $\|\mathbf{S}\|$ are growing, owing to the ability of the two algorithms to quickly cluster duplicate key values into equivalence classes. This ability is responsible for the substantial performance gains enjoyed by EP and HEP over Baseline. For EP, the total time starts on par with Baseline at $\|\mathbf{R}\| = \|\mathbf{S}\| = 1{,}000$ records and drops to 6.5% of Baseline's at $\|\mathbf{R}\| = \|\mathbf{S}\| = 20{,}000$. In the case of HEP, the total time declines to merely 0.6% that of Baseline.

7.5. Impact of Hash Partitioning

The previous experiments show that hash partitioning leads to significant cost reduction in HP over Baseline and in HEP over EP. We now investigate how varying the number of hash partitions facilitates different trade-offs between performance and privacy protection.

In Figure 11, we vary the number of partitions b while keeping the synthetic workload and resource parameters at their default settings as given in Table I. Among the join algorithms, Baseline and EP are independent of partitioning. For HP, the computation cost is roughly inversely proportional to b, according to formula (6) and as reflected in the trend in Figure 11(a). Furthermore, an increasing b lowers the number of passes over the inner partitions, until $b = 52$ where every outer partition can be loaded entirely into memory. The reduction in CPU and I/O costs explains HP's improvement in performance with more partitions. Of course, the performance gain is derived at the expense of disclosing more information as quantified in Figure 11(d). As for HEP, we were initially surprised that its I/O cost rises with b. Upon closer examination, we discovered that this is because a larger b reduces ϕ_B per partition, leading in turn to a smaller outer block size ω according to formula (9). This mandates more passes over the inner partitions, hence incurring higher I/O costs.

Through this experiment, we confirm that hash partitioning provides an effective mechanism for HP and HEP to trade off execution time against information disclosure.

Table II. TPC-E Workload (in hours)

	CS	Baseline	EP	HP	HEP
Query 1	13.1	83.0	40.4	2.9	1.4
Query 2	1917 (est.)	12,120 (est.)	458.6	405.1	16.3

Table III. TPC-E Workload (in hours) with Multithreaded Implementation

	Baseline	EP	HP	HEP
Query 1	19.1	9.3	0.7	0.3
Query 2	2800 (est.)	105.6	93.4	3.8

For instance, by lifting b from 10 to 100, the speedup of HP over Baseline increases from $10\times$ to $93\times$, whereas the speedup of HEP over Baseline improves from $180\times$ to $1076\times$.

7.6. TPC-E Workload

In this experiment, we switch to running the join algorithms on the TPC-E workload described in Section 7.1. To accommodate the larger relations, we use $M = 2{,}000$ pages. We also set $q_1 = 3\%$ for both EP and HEP as before. Moreover, for HP and HEP, we set $b = 30$, entailing a disclosure of 0.0085 according to formula (5). In the case of CS (from Carbunar and Sion [2012]), we initialize its Bloom filters with a 0.1% false positive rate. The total execution times (made up of CPU and I/O costs) for the two queries are tabulated in Table II.

For Query 1 involving a primary-key-to-foreign-key join, the relative performances among the join algorithms that we observe here support our cost analysis in Section 6 and the results obtained in the previous experiments. In particular, EP's execution time is half that of Baseline (in line with formula (10)); HEP doubles the speed of HP (formula (10)), which in turn is about $b = 30\times$ faster than Baseline (formula (6)). The execution time of CS is around 16% that of Baseline and a third of EP's for this query.

As for the foreign-key-to-foreign-key join in Query 2, Baseline is too slow for us to run it to completion. Instead, we executed Baseline on samples of the relations and extrapolated the total query time. Again, the time taken by HP is roughly $\frac{1}{b} = \frac{1}{30}$ that of Baseline. Here, EP is $26\times$ faster than Baseline, whereas HEP is $25\times$ faster than HP. These trends are consistent with those obtained earlier with the synthetic workload. Here, the execution time of CS is also estimated like Baseline. The time taken by CS is again around 16% that of Baseline, but more than $4\times$ slower than EP (due to the latter's ability to avoid evaluating every record in an equivalence class).

Next, we repeat the experiment with multithreaded implementations of our join algorithms. The results, summarized in Table III, show the same relative performance between the join algorithms that we observed earlier. More importantly, the results confirm that our join algorithms can be speeded up effectively through parallel execution. Similarly, the CS scheme of Carbunar and Sion [2012] can be parallelized, but this is beyond the scope of our work.

7.7. Summary of Experiment Results

We summarize the main findings from our experiments in the following.

—Where the join operands are asymmetrical, the input relation containing a higher ratio of duplicate key values (i.e., the number of records to unique key values) should be

the outer relation in EP and HEP. Moreover, the outer block size ω in the algorithms should be guided by formula (9), with q_1 set to roughly 3%.

—Where privacy requirements prevail over execution-time concerns, EP delivers significant speedup over Baseline across a wide range of workloads without leaking extra information, especially when both input relations contain duplicate key values.

—EP may be slower or faster than Carbunar-Sion's scheme [2012], depending on whether or not the join attributes involve a primary key. While the execution times of EP and CS are roughly of the same order of magnitude, EP achieves a stronger privacy protection between the two schemes.

—Where the application permits, employing HEP is very effective in achieving further cuts in execution time relative to EP. In particular, a disclosure of only 0.01 (as defined by formula (5)) easily brings about a speedup in excess of $10\times$.

—Both EP and HEP are amenable to parallel processing. As discussed in Section 5.5, multithreading (evaluated in Section 7.6), data parallelism, and GPU implementation are viable means to achieving practical query execution time with our join algorithms.

—Join operations on encrypted attributes are expensive, with their CPU costs overshadowing I/O costs. This characteristic contrasts with conventional join operations on cleartext attributes, for which computation overheads are dominated by I/O concerns. In practice, we envisage that most of the attributes in a database will be in cleartext, with only a minority of confidential attributes encrypted. In such a scenario, those join operations involving encrypted attributes should be pushed back as late as possible in a query execution plan, so that the size of the join operands may be trimmed by earlier selection and join operations involving cleartext attributes. This is analogous to joins involving expensive predicate functions [Gaede and Günther 1994].

8. CONCLUSION

In this article, we address the problem of executing ad hoc equi-join queries directly on an outsourced, encrypted database while preserving the privacy of the queries and data. We advocate a privacy framework in which the database itself is protected through strong encryption, whereas the disclosure from executing an equi-join is strictly limited to the minimum necessitated by its semantics, that is, that two input records combine to form an output record if and only if they have common join attribute values. There is no disclosure on records that are not part of the join result. This framework provides stronger privacy protection than existing solutions in the literature. Our solution is the first that achieves constant complexity per pair of records that are evaluated for the join.

To implement our framework, we introduce a cryptographic construct for the data owner to authorize the server to perform equality testing on probabilistically encrypted join attributes. Moreover, we develop join algorithms on the cryptographic construct to reduce the number of equality testing operations. Our join algorithms incorporate two techniques, namely equivalence partitioning and hash partitioning, to lower the computation demand. Equivalence partitioning specifically groups those records that contribute to the query output on their join attribute values during join execution, thus avoiding the need to match them individually again; this technique adheres strictly to the requirements of the privacy framework. In hash partitioning, all the records are assigned into partitions; this discloses some extra information which we quantified. We provide a detailed cost analysis of the join algorithms. We also report on extensive experiments confirming that our partitioning techniques combine to produce practical query execution times.

APPENDIX

In this Appendix, we give a detailed proof for Theorem 4.3 in Section 4.2. We begin by extending the decision linear problem (in Definition 2.3) through the following lemma.

LEMMA A.1. *Given* $(x, v, h_0, h_1, x^a, v^b, Z)$*, where* $x, v, h_0, h_1 \in_R \mathbb{G}$*,* $a, b \in_R \mathbb{Z}_p$ *and* $Z = h_\beta^{a+b}$ *for* $\beta \in_R \{0, 1\}$*, no probabilistic polynomial-time adversary can determine* β *with a probability significantly larger than 1/2.*

PROOF. The lemma can be proven by a reduction from the decision linear problem (in Definition 2.3). Let $\mathbb{A}$ be an algorithm that takes as input $(x, v, h_0, h_1, x^a, v^b, Z)$ and outputs 0 when $\beta = 0$ or 1 otherwise. Suppose that $\mathbb{A}$ successfully guesses β with advantage ϵ_1 over a random guess. We construct on $\mathbb{A}$ an algorithm $\mathbb{B}$ to solve the decision linear problem as follows.

Given an instance of the decision linear problem, specifically a 6-tuple (x, v, h_0, x^a, v^b, Z) such that $Z = h_0^{a+b}$ and $Z \neq h_0^{a+b}$ are equally likely, $\mathbb{B}$ picks $h_1 \in_R \mathbb{G}$ and passes $(x, v, h_0, h_1, x^a, v^b, Z)$ to $\mathbb{A}$. If $Z = h_0^{a+b}$, $\mathbb{A}$ outputs 0 with probability $1/2 + \epsilon_1(n)$. If $Z \neq h_0^{a+b}$, $\mathbb{A}$ outputs 0 with a probability 1/2. Now, if $\mathbb{A}$ outputs 0, $\mathbb{B}$ decides that $Z = h_0^{a+b}$; else $\mathbb{A}$ outputs 1 and $\mathbb{B}$ decides that $Z \neq h_0^{a+b}$. Therefore, $\mathbb{B}$'s success probability is $(1/2 + \epsilon_1(n))/2 + 1/2 \times 1/2 = 1/2 + \epsilon_1(n)/2$. Since the advantage in solving the decision linear problem is negligible, so must $\epsilon_1(n)$. □

With Lemma A.1, we can now proceed to the detailed proof for Theorem 4.3. We show that, if there is a Probabilistic Polynomial-Time (PPT) algorithm $\mathbb{A}$ that can defeat our encryption scheme with nonnegligible advantage over a random guess, then there exists a PPT algorithm $\mathbb{B}$ capable of successfully solving the decision linear problem with nonnegligible advantage. For clarity, we fix the security parameter n and use ϵ instead of $\epsilon(n)$ to denote a negligible polynomial function on n.

Algorithm $\mathbb{B}$ takes as input an instance of the decision linear problem, (x, v, h, x^a, v^b, W), where $x, v, h \in \mathbb{G}$ and $a, b, c \in \mathbb{Z}_p$ such that W is random with 0.5 probability and $W = h^{a+b}$ with 0.5 probability. $\mathbb{B}$ successfully solves the problem if it outputs 1 when $W = h^{a+b}$ and 0 otherwise. Let ρ denote $\mathbb{B}$'s success probability.

In the following, we construct $\mathbb{B}$ that simulates the privacy experiment $\mathsf{Exp}_{R,\mathbb{A}}^{REO}(n)$ for $\mathbb{A}$.

—For the `Setup` procedure of $\mathsf{Exp}_{R,\mathbb{A}}^{REO}(n)$, $\mathbb{B}$ chooses $\kappa_A, \tau_A, \sigma, \sigma_1 \in_R \mathbb{Z}_p$ and sets $g_1 = v, g_2 = g_1^\sigma$.

—$\mathbb{B}$ simulates REO by executing the `EncryptData` procedure described in Section 3.2. It is able to do so because it possesses all the necessary secrets for the procedure.

—Upon receiving $(\bar{r}_0, \bar{r}_1)$ from $\mathbb{A}$ in step 3 of the $\mathsf{Exp}_{R,\mathbb{A}}^{REO}(n)$ experiment, $\mathbb{B}$ chooses $\beta \in_R \{0, 1\}$ and returns $\bar{\mathcal{A}} = \langle \bar{\mathcal{A}}_1, \bar{\mathcal{A}}_2, \bar{\mathcal{A}}_3, \bar{\mathcal{A}}_4, \bar{\mathcal{A}}_5, \bar{\mathcal{A}}_6, \bar{\mathcal{A}}_7, \bar{\mathcal{A}}_8 \rangle$, where

$$\bar{\mathcal{A}}_1 = x^a, \bar{\mathcal{A}}_2 = v^b, \bar{\mathcal{A}}_3 = x^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2} \cdot W, \bar{\mathcal{A}}_4 = x,$$
$$\bar{\mathcal{A}}_5 = x^\sigma, \bar{\mathcal{A}}_6 = x^{\sigma/\kappa_A}, \bar{\mathcal{A}}_7 = g_1^z, z \in_R \mathbb{Z}_p, \bar{\mathcal{A}}_8 = \mathbf{e}(x, x)^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2}.$$

—If $\mathbb{A}$ determines β correctly, namely $\mathsf{Exp}_{R,\mathbb{A}}^{REO}(n) = 1$, $\mathbb{B}$ outputs 1; otherwise, $\mathbb{B}$ outputs 0.

To analyze the success probability of $\mathbb{B}$, we consider the two equally likely cases for W in the decision linear problem that $\mathbb{B}$ is trying to solve: (I) W is a random element from $\mathbb{G}$, or (II) $W = h^{a+b}$.

Case I. Since W is random, $\bar{\mathcal{A}}_3$ is a random element that is independent of the other components in $\mathcal{A}$. Therefore, $\mathbb{A}$ can have no advantage in determining $\bar{r}_\beta$ in $\bar{\mathcal{A}}_3$. Under

the circumstances, there is a probability of 1/2 that $\mathbb{A}$ outputs $\beta = 0$, leading $\mathbb{B}$ to conclude correctly that $W \neq h^{a+b}$. Thus, $\mathbb{B}$'s success probability in this case is $\rho_I = 1/2$.

Case II. Except for $\bar{\mathcal{A}}_3$ and $\bar{\mathcal{A}}_7$, all the components of $\bar{\mathcal{A}}$ that $\mathbb{B}$ produces in our simulation are identical to those in the original privacy experiment $\mathsf{Exp}^{REO}_{R,\mathbb{A}}(n)$. In the following, we first evaluate the probability that $\mathbb{A}$ accepts the simulation, that is, the chance that $\mathbb{A}$ cannot distinguish between the simulation and the original privacy experiment.

Let ρ_0 denote the probability that adversary $\mathbb{A}$ successfully decides that $z \neq a \times \tau_A$ in $\bar{\mathcal{A}}_7$, given $\bar{\mathcal{A}}_1$, $\bar{\mathcal{A}}_3$ to $\bar{\mathcal{A}}_5$; $\bar{\mathcal{A}}_2$, $\bar{\mathcal{A}}_6$ and $\bar{\mathcal{A}}_8$, being independent of $\bar{\mathcal{A}}_7$, are irrelevant here. $\mathbb{B}$ lowers the difficulty for $\mathbb{A}$ by exposing additional information to it, including $\sigma, \sigma_1, \sigma_2, \kappa_A, h$. The rationale is that $\mathbb{A}$'s probability of deciding correctly whether $z = a \times \tau_A$ with the additional information will form an upper bound on the probability of making the same decision correctly using only $\bar{\mathcal{A}}_1$ to $\bar{\mathcal{A}}_6$ and $\bar{\mathcal{A}}_8$. The additional information simplifies $\bar{\mathcal{A}}$ and, after removing redundancy, leaves $\mathbb{A}$ with an easier decision based on $(x, v, h, x^a, x^{\sigma_1 \times \bar{r}_\beta . A} \cdot h^{a+b}, v^z)$. We help $\mathbb{A}$ further by giving it $b, x^{\tau_A}, h^{\tau_A}$ and assuming that it guesses $\bar{r}_\beta . A$ correctly, so the decision is based on $(x, v, h, x^a, h^a, x^{\tau_A}, h^{\tau_A}, v^z)$. Now this is equivalent to solving the D3DH problem in Definition 2.7 on $(x, v, x^a, x^{\tau_A}, v^z)$ in view of the fact that v is a power of x, or solving the D3DH problem on $(h, v, h^a, h^{\tau_A}, v^z)$ considering that v is a power of h. Denoting the probability of solving the D3DH problem by ρ_{d3dh}, we thus have $1/2 \leq \rho_0 \leq \rho_{d3dh}$. As the D3DH assumption asserts that the difference between ρ_{d3dh} and 1/2 is negligible, we conclude that $\rho_0 = 1/2 + \epsilon_0$, where ϵ_0 is a negligible polynomial function.

Next, let ρ_1 denote the probability that $\mathbb{A}$ successfully distinguishes $\bar{\mathcal{A}}_3 = x^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2} h^{a+b}$ in the simulation from $x^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2} g_2^{a+b}$ in the original privacy experiment. Again, $\mathbb{B}$ simplifies the decision for $\mathbb{A}$ by releasing σ_1 and σ_2 to the latter and assuming that it correctly guesses $\bar{r}_\beta . A$. We have shown earlier that $\bar{\mathcal{A}}_7$ is random and independent of $\bar{\mathcal{A}}_3$. Moreover, σ and κ_A are independent of $\bar{\mathcal{A}}_3$, so $\bar{\mathcal{A}}_5 = \bar{\mathcal{A}}_4^{\sigma}$ and $\bar{\mathcal{A}}_6 = \bar{\mathcal{A}}_4^{\sigma/\kappa_A}$ do not provide additional information beyond $\bar{\mathcal{A}}_4$ to $\mathbb{A}$. Also, $\bar{\mathcal{A}}_8 = \mathbf{e}(\bar{\mathcal{A}}_4, \bar{\mathcal{A}}_4)^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2}$ is composed only of values that have been exposed to $\mathbb{A}$ by this time. Therefore, the problem of distinguishing $\bar{\mathcal{A}}_3$ is equivalent to deciding $\bar{\mathcal{A}}_3 / (x^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2}) \stackrel{?}{=} g_2^{a+b}$ given $(x, v, g_2, h, \bar{\mathcal{A}}_1 = x^a, \bar{\mathcal{A}}_2 = v^b, \bar{\mathcal{A}}_3 / (x^{\sigma_1 \times \bar{r}_\beta . A + \sigma_2}))$; $\mathbb{A}$'s advantage in the decision is negligible, according to Lemma A.1. We thus conclude that $\mathbb{A}$ can distinguish the simulation from the original privacy experiment with probability $\rho_1 = 1/2 + \epsilon_1$, where ϵ_1 is a negligible polynomial function.

Let ϵ denote the advantage of $\mathbb{A}$ in correctly guessing β. Consider the two conditions under which $\mathbb{B}$ is able to solve the decision linear problem: (a) $\mathbb{A}$ fails to discern the differences in $\bar{\mathcal{A}}_3$ and $\bar{\mathcal{A}}_7$ and guesses β correctly for $\mathbb{B}$; the associated probability is $(1 - \rho_0)(1 - \rho_1)(1/2 + \epsilon)$. (b) $\mathbb{A}$ detects the differences in $\bar{\mathcal{A}}_3$ or $\bar{\mathcal{A}}_7$ and makes a random guess of β; the probability that $\mathbb{B}$ is then led to conclude that $W = h^{a+b}$ is $(1 - (1 - \rho_0)(1 - \rho_1))/2$. Hence $\mathbb{B}$'s success probability in this case, ρ_{II}, is

$$\begin{aligned} \rho_{II} &= (1 - \rho_0)(1 - \rho_1)(1/2 + \epsilon) + (1 - (1 - \rho_0)(1 - \rho_1))/2 \\ &= \epsilon(1/2 - \epsilon_0)(1/2 - \epsilon_1) + 1/2 \\ &= 1/2 + \epsilon/4 + \epsilon(\epsilon_0\epsilon_1 - \epsilon_0/2 - \epsilon_1/2). \end{aligned}$$

Combining Cases I and II, we have

$$\begin{aligned} \rho &= 1/2 \times \rho_I + 1/2 \times \rho_{II} \\ &= 1/2 + \epsilon \left(\frac{1}{8} + \frac{\epsilon_0\epsilon_1}{2} - \frac{\epsilon_0}{4} - \frac{\epsilon_1}{4} \right). \end{aligned} \tag{12}$$

Under the assumption that the decision linear problem is intractable, $(\rho - 1/2)$ is negligible. This, coupled with the previous observations that ϵ_1 and ϵ_2 are negligible, implies that ϵ must also be negligible. Since $\mathbb{A}$ can have no significant advantage over a random guess in determining β in the privacy experiment, we conclude that the protocol in Section 3.2 satisfies objective P1 on initial confidentiality against all PPT adversaries, hence completing the proof. □

ACKNOWLEDGMENTS

We are grateful to the reviewers for their constructive feedback which helped to improve the article. We also thank Yanjiang Yang for discovering a security weakness in an earlier version.

REFERENCES

Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'04).* 563–574.

Arvind Arasu and Raghav Kaushik. 2014. Oblivious query processing. In *Proceedings of the 17th International Conference on Database Theory (ICDT'14).* 26–37.

Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. 2012. Recommendation for key management part 1: General (revision 3). http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57_Part1_rev3_general.pdf.

Burton Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Comm. ACM* 13, 7, 422–426.

Dan Boneh, Xavier Boyen, and Hovav Shacham. 2004a. Short group signatures. In *Proceedings of the 24th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'04).* 41–55.

Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. 2004b. Public key encryption with keyword search. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'04).* 506–522.

Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. 2007. Public key encryption that allows pir queries. In *Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'07).* 50–67.

Dan Boneh, Amit Sahai, and Brent Waters. 2006. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'06).* 573–592.

Dan Boneh and Brent Waters. 2007. Conjunctive, subset and range queries on encrypted data. In *Proceedings of the Theory of Cryptography Conference (TCC'07).* 535–554.

Utsab Bose, Anup Kumar Bhattacharya, and Abhijit Das. 2013. GPU-based implementation of 128-bit secure eta pairing over a binary field. In *Proceedings of the 6th International Conference on Cryptology in Africa (AFRICACRYPT'13).* 26–42.

Bogdan Carbunar and Radu Sion. 2012. Toward private joins on outsourced data. *IEEE Trans. Knowl. Data Engin.* 24, 1699–1710.

Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS'06).* 79–88.

Ernesto Damiani, S. De Capitani Di Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. 2003. Balancing confidentiality and efficiency in untrusted relational dbmss. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03).* 93–102.

Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient private matching and set intersection. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'04).* 1–19.

Volker Gaede and Oliver Gunther. 1994. Processing joins with user-defined functions. In *Grundlagen von Datenbanken*. 46–50.

Philippe Golle, Jessica Staddon, and Brent Waters. 2004. Secure conjunctive keyword search over encrypted data. In *Proceedings of the 2nd International Conference on Applied Cryptology and Network Security (ACNS'04).* 31–45.

Hakan Hacigumus, Bala Iyer, Chen Li, and Sharad Mehrotra. 2002. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'02).* 216–227.

Owen Harrison and John Waldron. 2010. GPU accelerated cryptography as an os service. In *Transactions on Computational Science XI*. Springer, 4–130.

Joseph M. Hellerstein and Michael Stonebraker. 1993. Predicate migration: Optimizing queries with expensive predicates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'93)*. 267–276.

Bijit Hore, Sharad Mehrotra, and Gene Tsudik. 2004. A privacy-preserving index for range queries. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB'04)*. 720–731.

Yosuke Katoh, Yun-Ju Huang, Chen-Mou Cheng, and Tsuyoshi Takagi. 2011. Efficient implementation of the ETA pairing on GPU. *IACR Cryptol. ePrint Arch. 540*.

Jonathan Katz, Amit Sahai, and Brent Waters. 2008. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of the 27th Annual International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT'08)*. 146–162.

Einar Mykletun and Gene Tsudik. 2006. Aggregation queries in the database-as-a-service model. In *Proceedings of the 20th IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec'06)*. 89–103.

Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*. 85–100.

Cliff Saran. 2011. Office365 fails bae's legal team. http://www.computerweekly.com/news/2240112018/Office365-fails-BAes-legal-team.

Donovan A. Schneider and David J. Dewitt. 1989. A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'89)*. 110–121.

Emily Shen, Elaine Shi, and Brent Waters. 2009. Predicate privacy in encryption systems. In *Proceedings of the Theory of Cryptography Conference (TCC'09)*. 457–473.

Dawn Xiaodong Song, David Wagner, and Adrian Perrig. 2000. Practical techniques for searches on encrypted data. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'00)*. 44–55.

Received July 2013; revised February 2014; accepted April 2014