Secure Top-k Inner Product Retrieval

Zhilin Zhang
Simon Fraser University
zhilinz@sfu.ca

Ke Wang*
Simon Fraser University
wangk@cs.sfu.ca

Chen Lin†
Xiamen University
chenlin@xmu.edu.cn

Weipeng Lin
Simon Fraser University
weipeng_lin@sfu.ca

ABSTRACT

Secure top-k inner product retrieval allows the users to outsource encrypted data vectors to a cloud server and at some later time find the k vectors producing largest inner products giving an encrypted query vector. Existing solutions suffer poor performance raised by the client's filtering out top-k results. To enable the server-side filtering, we introduce an asymmetric inner product encryption AIPE that allows the server to compute inner products from encrypted data and query vectors. To solve AIPE's vulnerability under known plaintext attack, we present a packing approach IP Packing that allows the server to obtain the entire set of inner products between the query and all data vectors but prevents the server from associating any data vector with its inner product. Based on IP Packing, we present our solution SKIP to secure top-k inner product retrieval that further speeds up retrieval process using sequential scan. Experiments on real recommendation datasets demonstrate that our protocols outperform alternatives by several orders of magnitude.

KEYWORDS

Top-k Retrieval; Inner Product Encryption; Known Plaintext Attack

ACM Reference Format:
Zhilin Zhang, Ke Wang, Chen Lin, and Weipeng Lin. 2018. Secure Top-k Inner Product Retrieval. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271791

1 INTRODUCTION

The recent advancement in cloud computing has triggered an emerging trend among individuals and organizations to outsource their potentially sensitive data to external non-fully trustworthy server and leverage the server's power to remotely carry out computations on the outsourced data by querying the server. Among various data outsourcing applications, the top-k inner product retrieval has received extensive attentions, which searches for the top-k outsourced data vectors mostly similar to a given query vector by measuring the similarity with the inner product between them. This task is widely applicable in a large number of domains. In recommender system, the collaborative filtering models the rating of item j by user i as the inner product between the latent user vector of i and the latent item vector of j, and recommends the items with top-k highest ratings to this user. In information retrieval, for the document a data vector of binary values or tf-idfs is built to represent the relevant keywords and for the query a binary query vector is built to indicate the keywords of interest, while the inner products between the data and query vectors specify the similarity and the documents having top ranked inner products are returned. In cloud computing, knn queries search for the k-nearest data vectors to the query vector using Euclidean distance or Cosine similarity. Such knn search in an ℓ-dimensional space is equivient to a top-k inner product search in an ℓ+1-dimensional space [22].

For each application above, a user's data/query vectors specify his sensitive data contents/query interests. To avoid putting the user's privacy at risk, the applications usually demand to encrypt both data and query vectors before outsourcing them and require the server to search for the top-k inner products over encrypted vectors. The encryption, however, impedes the usage of traditional inner product retrieval techniques [13, 17, 19, 21] because these methods commonly depend on the server's possessing plain data and query vectors to compute inner products.

Secure inner product computation [7, 9, 10, 23], the basis of privacy-preserving knn search (see [11, 18] for detailed reviews), is a potential solution to compute inner products from encrypted vectors using homomorphic encryption. However, since the obtain inner products through such homomorphic computation are still encrypted, it relies on the client-side filtering to find the top-k inner products. That is, the client must download the encrypted inner products between the query vector and all data vectors of the database, decrypt them and perform the sorting. The client-side filtering, therefore, incurs unacceptable communication cost and client's computation cost.

1.1 Efficiency/Security Dilemma

To resolve the issue of heavy costs in secure inner product computation, a typical solution is allowing the server to filter out the top-k inner products without compromising the user's privacy. It necessitates revealing *plain* inner products directly from encrypted data and query vectors. This, however, would raise the security vulnerability to a special known plaintext attack (KPA) discussed in [14]. That is, in the ℓ-dimensional vector space, assuming that the server knows ℓ linearly independent plain data vectors and their ciphertext; given any encrypted query vector, the server can obtain plain inner products using encrypted query and data vectors first and then set up ℓ linear equations to solve the unknown query. The server can repeat the above process to obtain ℓ query vectors and use them to solve any encrypted data vectors. Since the attack relies solely on the disclosure of inner products and such disclosure

*Ke Wang was partially supported by a discovery grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).
†Chen Lin was supported by the Natural Science Foundation of China (no. 61472335).

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10.
https://doi.org/10.1145/3269206.3271791

is necessary for server-side filtering, it is a highly challenging task to achieve both the server-side filtering and attack resistance.

1.2 Problem Formalization

1.2.1 System Model. The typical cloud model involves a server and a client (data owner or user), in which the server is "honest-but-curious" and the client is fully trusted, as commonly assumed [3, 5, 14, 15, 20, 22]. The terms server and attacker are interchangeable.

Under this cloud model, we consider the following *secure top-k inner product (IP) retrieval* problem. The client possesses a database $\mathcal{P}$ of n data vectors $\vec{p}_1, \cdots, \vec{p}_n$. Each $\vec{p}_i$ consists of ℓ integers[1] within a finite filed $\mathbb{Z}_m = \{0, 1, \cdots, m-1\}$, i.e., $\vec{p}_i \in \mathbb{Z}_m^\ell$. The client encrypts each $\vec{p}_i$ into $[\vec{p}_i]$ and outsources the encrypted database $[\mathcal{P}] = ([\vec{p}_1], \cdots, [\vec{p}_n])$ to the server. At a later time, the client issues the ciphertext $\langle \vec{q} \rangle$ of a query vector $\vec{q} \in \mathbb{Z}_m^\ell$ to find the top-k data vectors having largest inner products for $\vec{q}$, without disclosing any information about the query $\vec{q}$ and database $\mathcal{P}$, defined as:

Definition 1 (Secure Top-k Inner Product Retrieval). *Given the encryption $\langle \vec{q} \rangle$ of an ℓ-dimensional query vector $\vec{q} \in \mathbb{Z}_m^\ell$ and the encrypted database $[\mathcal{P}]$, find the list of ℓ-dimensional data vectors $\vec{p}_{i_1}, \cdots, \vec{P}_{i_k} \in \mathcal{P}$, for which $\vec{q}^T \vec{p}_{i_j}$ is one of the k largest values in $\vec{q}^T \mathcal{P}$, for all $1 \le j \le k$. Ties are broken arbitrarily.*

To enable server-side filtering, this work does not consider to protect $\vec{q}^T \mathcal{P}$ (the set of inner products between the query $\vec{q}$ and the database $\mathcal{P}$) and the access pattern (which data vectors are the answers of a query). Although the definition assumes the secure top-k IP retrieval over integer message spaces, the solutions provided in this work are applicable to general cases of floating point data by scaling floating point values to integers with finite precision as [8].

1.2.2 Threat Model. We consider known plaintext attack (KPA) [3, 5, 14, 20, 22] as the threat model. That is, the attacker can observe a set of plain data vectors in the database and know their corresponding ciphertext. As discussed earlier, inner products between query and data vectors can facilitate the attack. To better evaluate the privacy strength of a scheme, we classify attackers into different levels based on the knowledge they use.

- **Known plaintext attack without using inner products (KPA-noIP)**: the attack follows traditional KPA model that using only the plaintext and ciphertext of known data vectors to solve the secret keys for decrypting more ciphertext. The attack does not utilize inner products between data and query vectors.
- **Known plaintext attack using inner products (KPA-IP)**: apart from the information used above, the attack further makes use of inner products between data and query vectors to solve unknown queries as shown in Section 1.1. The attack is more harmful as cracking secret keys in KPA-noIP is typically more difficult.

We adopt the common assumption for query vectors as [3, 5, 14, 20, 22]; that is, the attacker cannot observe plain query vectors in all cases. It is reasonable because any encryption schemes would be trivially broken using the attack in Section 1.1 if plain query vectors are known to the attacker. Therefore, we consider query vectors to be adequately protected if the query encryption resists ciphertext only attack (COA) [14, 22].

[1] A negative number $-a$ is equivalent to the positive value $m - a$ over $\mathbb{Z}_m$

1.2.3 Design Goals. This paper is devoted to achieve practical secure top-k IP retrieval by solving the highly challenging dilemma in Section 1.1. To be specific, the following goals are addressed:

- **Performance goals**. *Low query cost*: the protocol should incur low client computation cost, server computation cost, and communication cost (retrieve only exact top-k results). Typically, it should support the server-side filtering of top-k inner products. *Non-interactive*: the client involves only uploading the query to the server and downloading the answers from the server.
- **Security goals**. *Confidentiality*: the server should not learn any plain information of the database $\mathcal{P}$ and query $\vec{q}$. In particular, the protocol should resist KPA-IP attack. *Unlinkability*: the encryption of data and query vectors should be randomized instead of deterministic so that the server can not deduce the relationship of data/query vectors from their corresponding ciphertext, e.g., whether two data/query vectors are the same or contain the same values at some dimensions.

1.3 Roadmap

To achieve the above goals, this work includes the following parts for dealing with

- *security and performance goals (Section 3).* We propose a novel *asymmetric inner product encryption* (AIPE) that allows the server to efficiently compute plain IP within arbitrarily large domains using encrypted data and query vectors. Compared with alternative solutions (see Section 2), AIPE achieves better security by resisting KPA-noIP and better performance by eliminating small IP domain restriction and pairing operations. AIPE is the first scheme having all these properties and serves as the major building block of our solutions below to secure top-k IP retrieval.
- *security and performance goals* (Section 4). We present a novel *Inner Product Packing* (IP Packing) solution to secure top-k IP retrieval. IP Packing, benefiting from AIPE, allows the server to obtain the set of all inner products $\vec{q}^T \mathcal{P}$ between the query $\vec{q}$ and the database $\mathcal{P}$ but prevents the server to associate any data vector $\vec{p}_i \in \mathcal{P}$ with its inner product $\vec{q}^T \vec{p}_i \in \vec{q}^T \mathcal{P}$. Due to hiding the link between data vectors and corresponding IP values, IP Packing resists KPA-IP attack and thus solves the challenging dilemma in Section 1.1. Interestingly, IP Packing also greatly improves the performance due to reduced IP computation.
- *performance goals* (Section 5). We further speeds up IP Packing using sequential scan, which gives rise to our complete solution *SKIP* to secure top-k IP retrieval. Compared to IP Packing, SKIP achieves significant performance gains by pruning huge number of IP computations.
- *evaluation goals* (Section 6): We evaluate the performance of proposed schemes with extensive experiments over real recommendation datasets MoiveLens and Yelp. The results validate the several orders of magnitude performance improvements for AIPE and SKIP compared to alternative techniques.

2 RELATED WORKS

The latest advances in function-hiding inner product encryption (IPE) [2, 6, 12, 22] allow to reveal plain inner product with inputting encrypted data and query vectors. In principle, it encrypts a data

Table 1: Performance and security comparison of IPE schemes w.r.t. the vector length ℓ and inner product domain T (According to [16], *Mul* (multiplication) is 10x faster than *Exp* (exponentiation), and *Exp* is 100x faster than *Pairing*)

IPE scheme	Performance								Security (resist the attack)	
	Encrypt		*KeyGen*		*Decrypt*					
	# of Mul	# of Exp	# of Mul	# of Exp	# of Mul	# of Exp	# of Pairing	Discrete logarithm	KPA-noIP	KPA-IP
ASPE [22]	$O(\ell^2)$	–	$O(\ell^2)$	–	$O(\ell)$	–	–	–	✗	✗
Pairing IPE [2, 6, 12]	$O(\ell^2)$	$O(\ell)$	$O(\ell^2)$	$O(\ell)$	$O(\ell)$	–	$O(\ell)$	$O(\sqrt{T})$	✓	✗
AIPE	$O(\ell^2)$	$O(\ell)$	$O(\ell^2)$	–	$O(\ell)$	$O(\ell)$	–	$O(1)$	✓	✗
IP Packing ($d > 1$)	$O(\ell^2/d)$	$O(\ell/d)$	$O(\ell^2)$	–	$O(\ell/d)$	$O(\ell/d)$	–	$O(1)$	✓	✓

vector $\vec{p}$ to $[\vec{p}]$ as usual, but encrypts a query vector $\vec{q}$ in such a way that the ciphertext $\langle\vec{q}\rangle$, with the help of $[\vec{p}]$, can be used to compute the exact inner product $\vec{q}^T\vec{p}$. Let the message space be a finite field $\mathbb{Z}_m$, a secret key IPE scheme can be formalized as follows:

Definition 2 (IPE [12]). *Given a data vector $\vec{p} \in \mathbb{Z}_m^\ell$ and a query vector $\vec{q} \in \mathbb{Z}_m^\ell$ of length ℓ over $\mathbb{Z}_m$, a IPE scheme encrypts $\vec{p}$ into a ciphertext $[\vec{p}]$ and encrypts $\vec{q}$ into a secret key $\langle\vec{q}\rangle$. The combination of $[\vec{p}]$ and $\langle\vec{q}\rangle$ reveals $\vec{q}^T\vec{p}$ but nothing more about $\vec{p}$ and $\vec{q}$. Specifically, it is a tuple of algorithms with the following properties:*

- ***Setup***$(1^\psi, 1^\ell) \rightarrow (pp, msk)$*: On input the security parameter ψ and the vector length ℓ, the setup algorithm outputs the public parameter pp and the master security key msk.*
- ***Encrypt***$(msk, \vec{p}) \rightarrow [\vec{p}]$*: On input the master secret key msk and the data vector $\vec{p} \in \mathbb{Z}_m^\ell$, the encryption algorithm outputs $[\vec{p}]$ as the ciphertext of $\vec{p}$.*
- ***KeyGen***$(msk, \vec{q}) \rightarrow \langle\vec{q}\rangle$*: On input the master secret key msk and the query vector $\vec{q} \in \mathbb{Z}_m^\ell$, the key generation algorithm outputs $\langle\vec{q}\rangle$ as the ciphertext of $\vec{q}$.*
- ***Decrypt***$(pp, [\vec{p}], \langle\vec{q}\rangle) \rightarrow \vec{q}^T\vec{p}$*: On input the public parameter pp, a ciphertext $[\vec{p}]$, and a secret key $\langle\vec{q}\rangle$, the decryption algorithm outputs the plain inner product $\vec{q}^T\vec{p} \in \mathbb{Z}_m$.*

Next, we review existing IPE and summarize them in Table 1.

ASPE: ASPE [22] (and its variants) are lightweight IPE schemes that have been widely used in a variety of applications as reviewed in [14, 18]. ASPE encrypts a data vector $\vec{p}$ by $[\vec{p}] = M^T \cdot \vec{p}$ and a query vector $\vec{q}$ by $\langle\vec{q}\rangle = M^{-1} \cdot \vec{q}$, using a random invertible matrix M as the secret key. The inner product is revealed through computing $\langle\vec{q}\rangle^T[\vec{p}] = \vec{q}^T \cdot M^{-T} \cdot M^T \cdot \vec{p} = \vec{q}^T\vec{p}$. The scheme can be enhanced by random asymmetric splitting and adding artificial dimensions. ASPE does not depend on crypto primitives and thus lacks rigorous security proof. Moreover, [5] recently shows that ASPE is vulnerable to KPA-noIP attack.

Pairing IPE. The cryptographic IPE schemes [2, 6, 12] assure the strong confidentiality of data and query vectors by making use of heavy crypto primitives. Typically, all these schemes are built on bilinear pairing and thus suffer from prohibitive *Decrypt* costs due to the following reasons. First, they require $O(\ell)$ pairings for *Decrypt* over vectors of size ℓ, while pairing is a very expensive operation in terms of CPU cost (10^{-1} *sec*) compared to multiplication (10^{-4} *sec*) or exponentiation (10^{-3} *sec*) as shown in [16]. Second, these schemes require computing a discrete logarithm for obtaining plain inner product; that is, use a group generator g and the value $g^{\vec{q}^T\vec{p}}$ to obtain $\vec{q}^T\vec{p}$. Let T be *inner product domain*, i.e., the range of all possible values for the computed IP. The discrete logarithm is typically very expensive in the bilinear group [1], which takes $O(T)$ time in a naive way (test every possible value in the domain) and $O(\sqrt{T})$ time using Pollard's kangaroo method. Therefore, Pairing IPE usually restricts the inner product domain T to be sufficient small (e.g., by letting data and query vectors be binary).

From Tables 1, we can see that no existing IPE scheme satisfies both performance and security goals of our secure top-k IP retrieval, as discussed in Section 1.2.3. To be specific, ASPE [22] encrypts both data and query vectors with lightweight matrix multiplication to achieve good efficiency but sacrifices the security (not resist KPA-noIP). This contradicts the requirement that the encryption for outsourced data vectors should provide strong protection as the data is permanently stored on the server and out of user's control. Pairing IPE [2, 6, 12] encrypts both of them with heavy crypto primitives to enforce better security but incurs prohibitive costs due to $O(\ell)$ expensive pairings and $O(\sqrt{T})$ discrete logarithm computation (e.g., *Decrypt* takes 1000 sec to compute inner products of 50-dimensional vectors over a domain $T = 10^7$). Such costly Pair IPE makes its usage unrealistic in real applications. Moreover, both ASPE and Pairing IPE fails to resist KPA-IP attack.

3 ASYMMETRIC INNER PRODUCT ENCRYPTION

To eliminate the drawbacks of existing IPE discussed above, we first build an *asymmetric inner product encryption* (AIPE) that well balances the security and performance requirements under the KPA-noIP model (i.e., resist KPA-noIP). That is, AIPE encrypts the data and query vectors in an asymmetric manner by exploiting the widely accepted assumption that the attacker cannot obtain plain query vectors (otherwise, the attacker easily infers the contents of data vectors by simply executing the queries): it chooses a crypto encryption for data vectors to provide strong data confidentiality and a lightweight encryption for query vectors to support efficient query processing. We will enhance AIPE to resist KPA-IP attack while using it to build our solutions for secure top-k IP retrieval in Section 4.

3.1 Construction

The construction involves the newly built functions of Definition 2.

- **Setup**$(1^\psi, 1^\ell) \rightarrow (pp, msk)$: On input the security parameter ψ and vector length ℓ, sample large primes p, q to compute $N = p * q$ and $\lambda = lcm(p-1, q-1)$ by following a Paillier's variant scheme in [4]. Then, sample a vector $\vec{s} = (s_1, \cdots, s_{2\ell})$ with each $s_i \in [1, \lambda N/2)]$, two $\ell \times \ell$ invertible matrices $\{M_1, M_2\}$ over $\mathbb{Z}_N$,

and an integer $h_0 \in \mathbb{Z}^*_{N^2}$. Finally, let

$$h = h_0^{2N} \mod N^2$$

and output the public parameter pp and master secret key msk

$$pp = (N) \qquad msk = \left(\lambda, h, M_1, M_2, \{s_i\}_{i=1}^{2\ell}\right)$$

- **Encrypt**$(msk, \vec{p}) \rightarrow [\vec{p}]$: On input master secret key msk and plain data vector $\vec{p} = (p_1, \cdots, p_\ell)^T \in \mathbb{Z}_N^\ell$, compute

$$\begin{aligned}(\hat{p}_1, \cdots, \hat{p}_\ell) &= \vec{p}^T \cdot M_1 \mod N \\ (\hat{p}_{\ell+1}, \cdots, \hat{p}_{2\ell}) &= \vec{p}^T \cdot M_2 \mod N\end{aligned} \tag{1}$$

Then, sample an integer $a \in \mathbb{Z}_N$, define

$$\begin{cases} C_0 = h^a & \mod N^2 \\ C_i = (1 + \hat{p}_i N) \cdot h^{a \cdot s_i} & \mod N^2, 1 \le i \le 2\ell \end{cases} \tag{2}$$

and output $[\vec{p}] = (C_0, C_1, \cdots, C_{2\ell})$.

- **KeyGen**$(msk, \vec{q}) \rightarrow \langle \vec{q} \rangle$: On input master secret key msk and plain query vector $\vec{q} = (q_1, \cdots, q_\ell)^T \in \mathbb{Z}_N^\ell$, randomly split $\vec{q}$ into two vectors $\vec{q}_a, \vec{q}_b$ such that $\vec{q} = \vec{q}_a + \vec{q}_b \mod N$, compute

$$\begin{aligned}(\hat{q}_1, \cdots, \hat{q}_\ell) &= \vec{q}_a^T \cdot (M_1^{-1})^T \mod N \\ (\hat{q}_{\ell+1}, \cdots, \hat{q}_{2\ell}) &= \vec{q}_b^T \cdot (M_2^{-1})^T \mod N\end{aligned} \tag{3}$$

Let $b = \sum_{i=1}^{2\ell} s_i \cdot \hat{q}_i \in \mathbb{Z}$. Then, define

$$\begin{cases} K_0 = b & \mod \lambda \\ K_i = \hat{q}_i & \text{for } 1 \le i \le 2\ell \end{cases} \tag{4}$$

and output $\langle \vec{q} \rangle = (K_0, K_1, \cdots, K_{2\ell})$.

- **Decrypt**$(pp, [\vec{p}], \langle \vec{q} \rangle) \rightarrow \vec{q}^T \vec{p}$: On input public parameter pp, ciphertext $[\vec{p}]$, and secret key $\langle \vec{q} \rangle$, compute

$$C_{\vec{q}^T \vec{p}} = C_0^{-K_0} \cdot \prod_{i=1}^{2\ell} C_i^{K_i} \mod N^2 \tag{5}$$

Then, compute the following discrete logarithm to obtain $\vec{q}^T \vec{p}$

$$\frac{C_{\vec{q}^T \vec{p}} - 1 \mod N^2}{N} \tag{6}$$

Theorem 1. *The Decrypt algorithm correctly produces the exact inner product $\vec{q}^T \vec{p} \in \mathbb{Z}_N$.*

Proof. Since M_1, M_2 are invertible matrices over $\mathbb{Z}_N$, Eqn (1) and (3) implies

$$\begin{aligned}\sum_{i=1}^{2\ell} \hat{p}_i \hat{q}_i &= \sum_{i=1}^{\ell} \hat{p}_i \hat{q}_i + \sum_{i=\ell+1}^{2\ell} \hat{p}_i \hat{q}_i \\ &= \left(\vec{q}_a^T (M_1^{-1})^T\right) \cdot \left(\vec{p}^T M_1\right)^T + \left(\vec{q}_b^T (M_2^{-1})^T\right) \cdot \left(\vec{p}^T M_2\right)^T \\ &= (\vec{q}_a + \vec{q}_b)^T \vec{p} = \vec{q}^T \vec{p} \mod N\end{aligned}$$

Thus, the theorem is proved if Eqn (6) outputs $\sum_{i=1}^{2\ell} \hat{p}_i \hat{q}_i$, which is shown as follows:

(1) From [4], $h^\lambda = 1 \mod N^2$ holds. Since $C_0 = h^a \mod N^2$ and $K_0 = b \mod \lambda$, we have

$$C_0^{-K_0} = \left(h^a\right)^{-(b \bmod \lambda)} = h^{-ab} \mod N^2$$

(2) Recall that, for $1 \le i \le 2\ell$, $C_i = (1 + \hat{p}_i N) \cdot h^{a \cdot s_i} \mod N^2$, $K_i = \hat{q}_i$. Also, $b = \sum_{i=1}^{2\ell} s_i \cdot \hat{q}_i$. Thus, we have

$$\begin{aligned}\prod_{i=1}^{2\ell} C_i^{K_i} &= \prod_{i=1}^{2\ell} \left((1 + \hat{p}_i N) \cdot h^{a \cdot s_i}\right)^{\hat{q}_i} = \left(\prod_{i=1}^{2\ell} (1 + \hat{p}_i \hat{q}_i N)\right) \cdot \prod_{i=1}^{2\ell} h^{a \cdot s_i \cdot \hat{q}_i} \\ &= \left(1 + \left(\sum\nolimits_{i=1}^{2\ell} \hat{p}_i \hat{q}_i\right) \cdot N\right) \cdot h^{ab} \mod N^2\end{aligned}$$

(3) Finally, (1) and (2) imply that Eqn (5) producing

$$C_{\vec{q}^T \vec{p}} = 1 + \left(\sum\nolimits_{i=1}^{2\ell} \hat{p}_i \hat{q}_i\right) \cdot N \mod N^2$$

and thus Eqn (6) produces

$$\begin{aligned}\frac{C_{\vec{q}^T \vec{p}} - 1 \mod N^2}{N} &= \frac{1 + \left(\sum_{i=1}^{2\ell} \hat{p}_i \hat{q}_i\right) \cdot N - 1 \mod N^2}{N} \\ &= \sum\nolimits_{i=1}^{2\ell} \hat{p}_i \hat{q}_i\end{aligned}$$

□

3.2 Security and Cost Analysis

We analyze the security and costs of AIPE, and review its advantages over the existing IPE discussed in Section 2. Comparison results are summarized in Table 1.

3.2.1 *Security Analysis.*

Theorem 2. *AIPE achieves the randomized encryption providing semantic security for data vectors and resists KPA-noIP attack.*

Proof. AIPE encrypts a data vector $\vec{p}$ in two steps. First, Eqn (1) encrypts $\vec{p}$ into $(\hat{q}_1, \cdots, \hat{q}_\ell)$ and $(\hat{q}_{\ell+1}, \cdots, \hat{q}_{2\ell})$ in a deterministic manner. Then, Eqn (2) further encrypts each $\hat{p}_i$ into (C_0, C_i) using the randomness a, $1 \le i \le 2\ell$. Specifically, (C_0, C_i) is the randomized encryption of $\hat{p}_i$ by the Paillier's variant in [4], which enforces semantic security for meeting strong data security requirement. Since semantic security implies the security against chosen plaintext attack and thus stronger than the security against known plaintext (data vector) attack [14], AIPE is resilient to KPA-noIP. □

Theorem 3. *AIPE achieves the randomized encryption for query vectors that resists ciphertext only attack (COA).*

Proof. AIPE encrypts a query vector $\vec{q}$ in two steps. It randomly splits $\vec{q}$ into two vectors $\vec{q}_a$ and $\vec{q}_b$ first, and then encrypts $\vec{q}_a, \vec{q}_b$ in Eqn (3) by multiplying secret matrices M_1, M_2, respectively. Since the splitting of $\vec{q}$ is randomly performed at every dimension, it achieves the randomized encryption (K_0 discloses no information due to the secret key λ). Under the COA attack, cracking secret keys M_1, M_2 requires to find unique solutions for two independent linear systems in Eqn (3) as the attacker observes no plain query vector (i.e., $\vec{q}$ is unknown). The task is impossible because each system has ℓ equations but $\ell^2 + \ell$ unknowns, i.e., $\vec{q}_a, M_1$ and $\vec{q}_b, M_2$. □

Compared to lightweight scheme ASPE [22], AIPE significantly improves the security. It achieves semantically secure data confidentiality by encrypting data vectors with a Paillier variant croptosystem [4], and preserves stronger query privacy by randomly splitting query vector at every dimension. In contrast, ASPE encrypts both data and query vectors without involving any cryptosystem and solely performs the random splitting of data and query vectors at certain fixed dimensions, which fails to resist KPA-noIP attack [5].

3.2.2 Cost Analysis.

- *Encrypt* has two steps: (1) multiplication of $O(\ell)$-dimensional vectors and $O(\ell) \times O(\ell)$ matrices in Eqn (1), i.e., $O(\ell^2)$ multiplications; and (2) $O(\ell)$ multiplications and exponentiations in Eqn (2). Thus, the total costs include $O(\ell^2)$ multiplications and $O(\ell)$ exponentiations.
- *KeyGen* computes the multiplication of $O(\ell)$-dimensional vectors and $O(\ell) \times O(\ell)$ matrices in Eqn (3), thus $O(\ell^2)$ multiplications.
- *Decrypt* requires $O(\ell)$ multiplications and exponentiations in Eqn (5), and computes the discrete logarithm at $O(1)$ cost in Eqn (6).

Compared to Pairing IPE [2, 6, 12], AIPE greatly improves the performance in the following aspects. First, the *Decrypt* of Pairing IPE involves computing a discrete logarithm at a polynomial cost of the IP domain T, which restricts T to be sufficiently small. By contrast, AIPE enables the efficient IP computation of arbitrarily large value by computing the discrete logarithm in Eqn (6) at $O(1)$ cost. This property is essential to our solutions for resisting KPA-IP in the next section. Second, AIPE depends only on multiplication and exponentiation in *Decrypt*, which are orders of magnitude faster than pairing needed by Pairing IPE. Such improvement is also essential to practical secure top-k IP retrieval because *Decrypt* time immediately determines the searching time. Finally, AIPE improves the performance of *KeyGen* using only multiplication operations (much faster than exponentiation), which also facilitate practical secure top-k IP retrieval as *KeyGen* time determines the query encryption time. We will validate these advantages empirically in Section 6.1.

Up until now, we have achieved the balanced security and performance by AIPE under the KPA-noIP model. However, similar to other IPE schemes, AIPE still suffers the vulnerability to KPA-IP attack (i.e., the dilemma in Section 1.1). In the next section, we will discuss how to enhance AIPE for resisting KPA-IP.

4 INNER PRODUCT PACKING

The key reason of AIPE's vulnerability to KPA-IP attack is that the attacker can associate each known data vector with its corresponding inner product, and then enables the attack described in Section 1.1. Specifically, on ℓ-dimensional vector space, assuming the attacker knows ℓ linearly independent data vectors $\vec{p}_i$ and their ciphertext $[\vec{p}_i]$, $1 \le i \le \ell$; given any encrypted query vector $\langle \vec{q} \rangle$, the attacker can always infer $\vec{q}$ by solving the following equations

$$\vec{q}^T \vec{p}_i = Decrypt(pp, [\vec{p}_i], \langle \vec{q} \rangle), \forall i \in [1, \ell]. \tag{7}$$

To prevent this KPA-IP attack, one method is to hide the linking between $\vec{p}$ and its inner product $\vec{q}^T \vec{p}$ computed by *Decrypt*, w.r.t. any data vector $\vec{p}$ and any query vector $\vec{q}$. It is possible to hide such linking information in many applications. Typically, considering the secure top-k IP retrieval upon a query vector $\vec{q}$ and a database $\mathcal{P} = \{\vec{p}_1, \cdots, \vec{p}_n\}$, the server depends on the entire set $\vec{q}^T\mathcal{P}$ instead of the individual $\vec{q}^T\vec{p}_i$ to filter out the top-k answers. This observation motivates our prevention of KPA-IP attack by applying AIPE in such a way that *the server can learn the set* $\vec{q}^T\mathcal{P}$ *but cannot associate the value* $\vec{q}^T\vec{p}_i$ *with each individual data vector* $\vec{p}_i$*, for all* $1 \le i \le n$.

Using the above idea, we build our solution to secure top-k IP retrieval, called *"Inner Product Packing"* or *"IP Packing"*, that solves the dilemma in Section 1.1. To be specific, IP Packing benefits from AIPE to resist KPA-IP as follows.

(1) IP Packing takes the advantage of AIPE's large message space $\mathbb{Z}_N$ (N has size thousands of bits) to *pack* many data vectors (WLOG, d vectors) into one auxiliary vector of the same length.
(2) Given the ciphertext of an auxiliary vector $\vec{a}$ and a query vector $\vec{q}$, the plain product $\vec{q}^T\vec{a}$ of arbitrarily large value within $\mathbb{Z}_N$ can be efficiently computed because the *Decrypt* cost of AIPE is independent of the domain of $\vec{q}^T\vec{a}$.
(3) IP Packing efficiently *unpacks* $\vec{q}^T\vec{a}$ into the set of inner products between the query $\vec{q}$ and the d data vectors packed into $\vec{a}$.

4.1 Construction

We require that all inner products $\vec{q}^T\mathcal{P}$ between the query vector $\vec{q}$ and the database $\mathcal{P}$ fall into the interval[2] $[0, max]$, i.e., all IP are non-negative. If the original $\vec{q}^T\mathcal{P}$ lie in an interval $[-min', max']$ for some $min', max' > 0$, we can shift all IP to the new interval $[0, max' + min']$ that preserve the original orders among them, by adding one dimension to each $\vec{p}_i \in \mathcal{P}$ and each $\vec{q}$ as follows,

$$\vec{p}_i \leftarrow \begin{pmatrix} \vec{p}_i \\ min' \end{pmatrix} \qquad \vec{q} \leftarrow \begin{pmatrix} \vec{q} \\ 1 \end{pmatrix}. \tag{8}$$

We fix a radix $u > max$ and enforce $u^{d+1} < N$ (N is the public parameter pp of AIPE) by letting

$$d = \left\lfloor \frac{\log N}{\log u} \right\rfloor - 1. \tag{9}$$

Next, we build two helper functions *Pack* and *Unpack*.

- **Pack**$(u, (\vec{p}_i^1, \cdots, \vec{p}_i^d)) \to \vec{a}_i$. On input the radix u and a sequence of d data vectors $(\vec{p}_i^1, \cdots, \vec{p}_i^d)$, compute auxiliary vector $\vec{a}_i$ by

$$\vec{a}_i = \vec{p}_i^1 + u \cdot \vec{p}_i^2 + \cdots + u^{d-1} \cdot \vec{p}_i^d \mod N \tag{10}$$

  Here, $\vec{a}_i$ has the same dimensionality as each $\vec{p}_i^j$ ($1 \le j \le d$), and the value at each dimension of $\vec{a}_i$ is within the message space $\mathbb{Z}_N$, i.e., $\vec{a}_i \in \mathbb{Z}_N^\ell$.

- **Unpack**$(u, j, \vec{q}^T\vec{a}_i) \to \vec{q}^T\vec{p}_i^j$. On input the radix u and the value $\vec{q}^T\vec{a}_i$ computed by $Decrypt(pp, [\vec{a}_i], \langle \vec{q} \rangle)$, for any $1 \le j \le d$

$$\vec{q}^T\vec{p}_i^j = \frac{\vec{q}^T\vec{a}_i \mod u^j - \vec{q}^T\vec{a}_i \mod u^{j-1}}{u^{j-1}} \mod u \tag{11}$$

  The correctness of Eqn (11) is proved in Theorem 4. Note that $\vec{q}^T\vec{a}_i$ can be arbitrarily large value over $\mathbb{Z}_N$. It highlights the necessity of computing discrete logarithm at $O(1)$ cost in *Decrypt* achieved by AIPE.

Based on such helper functions, we build IP Packing as follows.

Initialization (Algorithm 1). The client first generates the public parameter and master secret key (Line 1), sets up the global parameters (Line 2), and privately partitions the database into buckets of size d (Line 3). Here, the database size n is assumed to be dividable by d; if not, achieve it by adding $\vec{0}$ into the database $\mathcal{P}$. For each bucket i, the client re-orders its d vectors into a random sequence, packs them into an auxiliary vector $\vec{a}_i$, and encrypts $\vec{a}_i$ into $[\vec{a}_i]$ (Line 4). Finally, the client outsources all $[\vec{a}_i]$ to the server (Line 5).

[2] max is publicly known or easily estimated in many applications, e.g., $max = 5$ in a recommendation system using the 5-point rating scale.

Algorithm 1 *Initialization* ($\mathcal{P}$)

Require: The client has the database $\mathcal{P} = (\vec{p}_1, \cdots, \vec{p}_n)$ and max

1: call $Setup$ to generate (pp, msk)
2: let $u > max$; $d \leftarrow \left\lfloor \frac{\log N}{\log u} \right\rfloor - 1$; $n' \leftarrow \frac{n}{d}$
3: private partition $\mathcal{P}$ into n' buckets of d data vectors each
4: **for** $i = 1, \cdots, n'$ **do**
   (a) random permute d vectors of bucket i into a sequence $\vec{p}_i^1, \cdots, \vec{p}_i^d$
   (b) $\vec{a}_i \leftarrow Pack(u, (\vec{p}_i^1, \cdots, \vec{p}_i^d))$; $[\vec{a}_i] \leftarrow Encrypt(msk, \vec{a}_i)$
5: outsource $[\mathcal{A}] = ([\vec{a}_1], \cdots, [\vec{a}_{n'}])$ and u to the server

IP Packing (Algorithm 2). During phase 1, the client encrypts the query $\vec{q}$ into $\langle \vec{q} \rangle$ and sends it to the server. During phase 2, the server collects the entire set of IP between the query and the database by computing the plain value $\vec{q}^T \vec{a}_i$ for each outsourced vector $[\vec{a}_i]$ and unpacking $\vec{q}^T \vec{a}_i$ into d inner products (Line 3). Using $\vec{q}^T \mathcal{P}$, the server filters out the top-k answers and returns them to the client (Line 5); that is, for each $\vec{p}_i^j$ producing one of top-k IP, the server returns its index (i, j) and corresponding IP value $\vec{q}^T \vec{p}_i^j$.

Toward real applications, the secure top-k IP retrieval normally serves the purpose of searching for *the index* of the k data vectors producing toppest IP and the ultimate goal is to retrieve certain application-specific data associated with these vectors, e.g., top-k most relevant items in recommendation systems or top-k most relevant documents in information retrieval. Assume that such data are also outsourced to the server and the client knows how to retrieve the data associated with each vector $\vec{p}_i^j$. The client can easily retrieve top-k application-specific data using R.

4.1.1 Correctness and Security Analysis.

Theorem 4. *For any $1 \le j \le d$, Eqn (11) correctly produces $\vec{q}^T \vec{p}_i^j$.*

Proof. Recall that the message space of AIPE is $\mathbb{Z}_N$. We first show that $\vec{q}^T \vec{a}_i \in \mathbb{Z}_N$ does not wrap around *mod N*, i.e., $\vec{q}^T \vec{a}_i < N$:

$$\vec{q}^T \vec{a}_i = \vec{q}^T (\vec{p}_i^1 + u \cdot \vec{p}_i^2 + \cdots + u^{d-1} \cdot \vec{p}_i^d) \quad \text{(a)}$$

$$= \vec{q}^T \vec{p}_i^1 + u \cdot \vec{q}^T \vec{p}_i^2 + \cdots u^{d-1} \cdot \vec{q}^T \vec{p}_i^d \quad \text{(b)}$$

$$< u + u \cdot u + \cdots + u^{d-1} \cdot u < u^{d+1} \quad \text{(c)}$$

(a) comes from Eqn (10). (b) is the expansion of (a). (c) holds as $\vec{q}^T \vec{p}_i^j \in [0, max]$ and $u > max$ (thus $u > 1$). Finally, recalling that $u^{d+1} < N$, we have $\vec{q}^T \vec{a}_i < N$.

The above proof shows that $Decrypt(pp, [\vec{a}_i], \langle \vec{q} \rangle)$ outputs the exact inner product $\vec{q}^T \vec{a}_i$ between $\vec{q}$ and $\vec{a}_i$. Therefore, the correctness of Eqn (11) is guaranteed because of

$$\frac{\vec{q}^T \vec{a}_i \mod u^j - \vec{q}^T \vec{a}_i \mod u^{j-1}}{u^{j-1}} =$$

$$\frac{\vec{q}^T (\vec{p}_i^1 + u \cdot \vec{p}_i^2 + \cdots u^{j-1} \cdot \vec{p}_i^j) - \vec{q}^T (\vec{p}_i^1 + u \cdot \vec{p}_i^2 + \cdots u^{j-2} \cdot \vec{p}_i^{j-1})}{u^{j-1}} = \vec{q}^T \vec{p}_i^j \quad \square$$

KPA-IP attack depends on setting up the equations of Eqn (7), which requires the attacker to correctly link each known data vector $\vec{p}_i \in P$ with its inner product $\vec{q}^T \vec{p}_i$. IP Packing hides such linkage to resist the attack by making data vectors indistinguishable, through

Algorithm 2 *IP Packing* ($[\mathcal{A}], u, \vec{q}$)

Require: The client has msk and $\vec{q}$; the server has u, pp, and $[\mathcal{A}]$

Phase 1 (Client):

1: $\langle \vec{q} \rangle \leftarrow KeyGen(msk_\ell, \vec{q})$
2: send $\langle \vec{q} \rangle$ to the server

Phase 2 (Server):

3: **for** $i = 1, \cdots, n'$ **do**
   (a) $\vec{q}^T \vec{a}_i \leftarrow Decrypt(pp, [\vec{a}_i], \langle \vec{q} \rangle)$
   (b) $\vec{q}^T \vec{p}_i^j \leftarrow Unpack(u, j, \vec{q}^T \vec{a}_i)$, for $1 \le j \le d$
4: collect the set of IP between the query $\vec{q}$ and the database $\mathcal{P}$

$$\vec{q}^T \mathcal{P} = \left\{ \vec{q}^T \vec{p}_i^j \mid 1 \le i \le n', 1 \le j \le d \right\} \quad (12)$$

5: send R to the client by letting

$$R = \left\{ (i, j, \vec{q}^T \vec{p}_i^j) \mid \vec{q}^T \vec{p}_i^j \text{ is one of } k \text{ largest values of } \vec{q}^T \mathcal{P} \right\} \quad (13)$$

privately partitioning the database $\mathcal{P}$ into buckets (Line 3, Algorithm 1) and randomly permuting the d vectors of each bucket (Line 4(a)). The next theorem provides the analytical security bound.

Theorem 5. *IP Packing is resilient to KPA-IP attack by limiting the attack's probability of success (Pr) not greater than $\frac{(d \cdot \lceil \frac{\ell}{d} \rceil - \ell)!}{(d!)^{\lceil \frac{\ell}{d} \rceil}}$, w.r.t. the vector length ℓ and the bucket size d.*

Proof. Through IP Packing, the KPA-IP attacker's knowledge has: (1) database $[\mathcal{A}] = ([\vec{a}_1], \cdots, [\vec{a}_{n'}])$ and query $\langle \vec{q} \rangle$; (2) a set of known data vectors $P \subset \mathcal{P}$; and (3) the set of inner products $\vec{q}^T \mathcal{P}$. Since the encryption preserves the security for (1), it suffices to show that the attacker cannot launch the attack using (2) and (3).

To solve the unknown $\vec{q}$ of length ℓ by setting up Eqn (7), the attacker must know ℓ linearly independent plain data vectors (i.e., $|P| = \ell$) and their inner product $\vec{q}^T \vec{p}_i$ for each $\vec{p}_i \in P$ (i.e., $\vec{q}^T P$). Next, we analyze the attacker's probability Pr of finding $\vec{q}^T P$.

Assume that the ℓ known vectors of P are packed into t auxiliary vectors, say $\vec{a}_1, \cdots, \vec{a}_t$, and $\vec{a}_i$ packs d_i vectors of P, $1 \le i \le t$. To simplify the analysis, we also assume that the attacker knows which d_i vectors of P are packed into each $\vec{a}_i$. Let Pr_i be the probability of recognizing the IP of the d_i known vectors from the d inner products *Unpacked* from $\vec{q}^T \vec{a}_i$ (Line 3(b), Algorithm 2). Due to random permutation, the possible choices of selecting d_i ordered IP is $\frac{d!}{(d-d_i)!}$, which gives $Pr_i = \frac{(d-d_i)!}{d!}$. Recall that the attacker's probability of success is the products of all Pr_i. We have

$$Pr = \prod_{i=1}^{t} \frac{(d-d_i)!}{d!} = \left(\prod_{i=1}^{t-1} \frac{(d-d_i)!}{d!} \right) \cdot \frac{(\sum_{i=1}^{t-1} d_i + d - \ell)!}{d!} \quad \text{(a)}$$

$$\le \frac{(\sum_{i=1}^{t-1} (d-d_i) + \sum_{i=1}^{t-1} d_i + d - \ell)!}{(d!)^t} \le \frac{(d \cdot \lceil \frac{\ell}{d} \rceil - \ell)!}{(d!)^{\lceil \frac{\ell}{d} \rceil}} \quad \text{(b)}$$

(a) is obtained by noting $d_t = \ell - \sum_{i=1}^{t-1} d_i$. (b) holds because of $x! \cdot y! \le (x+y)!$ for any $x, y \ge 0$. Finally, since the ℓ vectors of P are packed into *at least* $\lceil \frac{\ell}{d} \rceil$ auxiliary vectors, $t \ge \lceil \frac{\ell}{d} \rceil$ holds and the maximal Pr is obtained in (b) by choosing $t = \lceil \frac{\ell}{d} \rceil$. $\square$

4.2 Discussion

From Theorem 5, the strength of IP Packing for resisting KPA-IP depends on the parameter d (larger d gives better security). We now discuss the strategy to enforce a desired security (i.e., ensure the attack's maximal probability of success Pr satisfies $Pr \le \rho$, for some user-specified probability ρ). If $d \ge \ell$, $\lceil \frac{\ell}{d} \rceil = 1$ holds and the upper bound of Pr is simplified to

$$Pr \le \frac{(d-\ell)!}{d!} \le \frac{1}{(d-\ell+1)^{\ell}}.$$

To achieve a specified security ρ, $\frac{1}{(d-\ell+1)^{\ell}} \le \rho$ must hold, which requires $d \ge \ell - \log_{\ell} \rho - 1$. According to Eqn (9), given an arbitrary u, we can always meet such conditions of $d \ge max(\ell, \ell - \log_{\ell} \rho - 1)$ by choosing a large derived key size of AIPE, i.e., a large N.

Besides enhancing AIPE to resist KPA-IP attack, IP Packing also achieves better performance: each call of *Encrypt* and *Decrypt* is equivalent to encrypt d data vectors and compute d inner products, which has a d-fold cost reduction compared to AIPE. We summarize the security and performance benefits of IP Packing in Table 1.

5 PRUNING WITH SEQUENTIAL SCAN

In this section, we discuss how to speed up IP Packing with pruning and construct our complete secure top-k IP retrieval protocol.

5.1 Sequential Scan

In the plaintext domain, sequential scan (SS) has been widely used to speed up the top-k IP retrieval [13, 21]. Considering an arbitrary query vector $\vec{q}$ and data vector $\vec{p}_i \in \mathcal{P}$, the Cauchy-Schwarz inequality gives the upper bound of its inner product by

$$\vec{q}^T \vec{p}_i \le \|\vec{q}\| \cdot \|\vec{p}_i\| \tag{14}$$

where $\|\cdot\|$ is the L2-norm of the vector. Before the scan, SS sorts the data vectors of the database $\mathcal{P} = (\vec{p}_1, \cdots, \vec{p}_n)$, in decreasing order of their norms. This sorting allows to search for the top-k IP efficiently; that is, SS scans the vectors (columns) of sorted $\mathcal{P}$ from left to right. When SS reaches a vector $\vec{p}_i$, it first checks if $\|\vec{q}\| \cdot \|\vec{p}_i\|$ is less than or equal to the k-th largest inner product found so far, say t. If $\|\vec{q}\| \cdot \|\vec{p}_i\| \le t$, the scan stops and the current top-k inner products are reported as the results, as it is impossible for $\vec{p}_i$ and the remaining columns at the right of $\vec{p}_i$ enter the results.

Now, we exploit how to enable sequential scan over IP Packing. We ignore all involved encryption to highlight the key ideas. Recall that IP Packing reveals the inner products between the query $\vec{q}$ and the database $\mathcal{P}$ in a batch way. That is, for $1 \le i \le n'$, computing $\vec{q}^T \vec{a}_i$ on the server reveals d inner products $\vec{q}^T \vec{p}_i^j$, $1 \le j \le d$. Since it depends on all possible $\vec{q}^T \vec{p}_i^j$ (i.e, the entire set $\vec{q}^T \mathcal{P}$) to find the top-k inner products, IP Packing needs to compute $\vec{q}^T \vec{a}_i$ for all $1 \le i \le n'$. To prune some unnecessary computation of $\vec{q}^T \vec{a}_i$, we give the following definition first.

Definition 3 (Index norm). *Let $\vec{a}_i$ pack d data vectors $\vec{p}_i^1, \cdots, \vec{p}_i^d$ as in Eqn* (10)*. The index norm of $\vec{a}_i$, denoted as $\|\vec{a}_i\|$, is the largest norm of these d data vectors, i.e., $\|\vec{a}_i\| = \max_{1 \le j \le d} \|\vec{p}_i^j\|$.*

Using the index norm $\|\vec{a}_i\|$, we can prune $\vec{a}_i$ if we have

$$\|\vec{q}\| \cdot \|\vec{a}_i\| \le t \tag{15}$$

Algorithm 3 *Initialization ($\mathcal{P}$)*

Require: The client has the database $\mathcal{P} = (\vec{p}_1, \cdots, \vec{p}_n)$ and max

1: call $Setup$ to generate (pp_1, msk_1) and (pp_ℓ, msk_ℓ)
2: let $u > max$; $d \leftarrow \left\lfloor \frac{\log N}{\log u} \right\rfloor - 1$; $n' \leftarrow \frac{n}{d}$
3: sort $\mathcal{P}$ into n' buckets
4: **for** $i = 1, \cdots, n'$ **do**
   (a) random permute bucket i to produce a sequence $\vec{p}_i^1, \cdots, \vec{p}_i^d$
   (b) $\vec{a}_i \leftarrow Pack(u, (\vec{p}_i^1, \cdots, \vec{p}_i^d))$; $\|\vec{a}_i\| \leftarrow$ index norm of $\vec{a}_i$
   (c) $[\vec{a}_i] \leftarrow Encrypt(msk_\ell, \vec{a}_i)$; $[\|\vec{a}_i\|] \leftarrow Encrypt(msk_1, \|\vec{a}_i\|)$
5: outsource $[\mathcal{A}] = ([\vec{a}_1], \cdots, [\vec{a}_{n'}])$, $[\|\mathcal{A}\|] = ([\|\vec{a}_1\|], \cdots, [\|\vec{a}_{n'}\|])$, and u to the server

because it implies $\vec{q}^T \vec{p}_i^j \le t$ for all $1 \le j \le d$. This observation inspires the following sequential scan over IP Packing. Assume that all L2-norms of the d vectors in bucket i are not less than the norms in a subsequent bucket i', for all $1 \le i < i' \le n'$. SS sorts the vectors of $\mathcal{A} = (\vec{a}_1, \cdots, \vec{a}_{n'})$, in decreasing order of their index norms. On input a query $\vec{q}$, SS scans the vectors (columns) of sorted $\mathcal{A}$ from left to right. When SS reaches a vector $\vec{a}_i$, it stops the scan and reports the current top-k inner products as the answers if Eqn (15) is satisfied. During this process, the pruning of $\vec{a}_i$ and subsequent vectors $\vec{a}_{i'}$ are effective, because for $1 \le j \le d$, we have $\vec{q}^T \vec{p}_{i'}^j \le \|\vec{q}\| \cdot \|\vec{a}_{i'}\|$ and thus $\vec{q}^T \vec{p}_i^j \le \|\vec{q}\| \cdot \|\vec{a}_i\|$ while

$$\|\vec{q}\| \cdot \|\vec{a}_{i'}\| \le \|\vec{q}\| \cdot \|\vec{a}_i\| \le t.$$

5.2 SKIP: Put All Together

Now, we give out our solution to secure top-k inner product retrieval, called *SKIP*, that achieves the above sequential scan over IP Packing for better satisfying the performance goals of Section 1.2.3.

5.2.1 Initialization. Considering a database $\mathcal{P} = (\vec{p}_1, \cdots, \vec{p}_n)$ of n data vectors in the ℓ-dimensional space. Given an arbitrary query vector $\vec{q}$, we assume that the inner product $\vec{q}^T \vec{p}_i$ falls into the interval $[0, max]$ w.r.t. some $max \ge 1$, for all $1 \le i \le n$.

Initially, the client calls $Setup$ of the AIPE scheme to generate (pp_1, msk_1) and (pp_ℓ, msk_ℓ), i.e., public parameter/master secret key of AIPE for 1- and ℓ- dimensional vectors. Let $N = pp_\ell$, the client chooses some $u > max$ and computes $d = \lfloor \frac{\log N}{\log u} \rfloor - 1$ as Eqn (9).

Then, the client sorts data vectors of the database $\mathcal{P}$ in decreasing order of their L2-norms and splits the sorted $\mathcal{P}$ into $n' = \frac{n}{d}$ equal-sized buckets. In this way, every bucket contains d data vectors and their L2-norms are greater than or equal to the norms of the vectors in all subsequent buckets. Then, for $1 \le i \le n'$, the client randomly permutes the d vectors of bucket i into a sequence $\vec{p}_i^1, \cdots, \vec{p}_i^d$, and packs them into $\vec{a}_i$ using Eqn (10); the client also computes the index norm $\|\vec{a}_i\|$ of $\vec{a}_i$, and encrypts $\vec{a}_i$, $\|\vec{a}_i\|$ to $[\vec{a}_i]$, $[\|\vec{a}_i\|]$ by

$$[\vec{a}_i] \leftarrow Encrypt(msk_\ell, \vec{a}_i)$$
$$[\|\vec{a}_i\|] \leftarrow Encrypt(msk_1, \|\vec{a}_i\|)$$

Here, $\vec{a}_i$ and $\|\vec{a}_i\|$ are vectors of size ℓ and size 1, respectively.

Finally, the client outsources $[\mathcal{A}] = ([\vec{a}_1], \cdots, [\vec{a}_{n'}])$, $[\|\mathcal{A}\|] = ([\|\vec{a}_1\|], \cdots, [\|\vec{a}'_n\|])$, and the value u to the server. The initialization is summarized in Algorithm 3.

Algorithm 4 *SKIP* $([\mathcal{A}], [\|\mathcal{A}\|], u, \vec{q})$

Require: The client has msk_1, msk_ℓ, and $\vec{q}$; the server has pp_1, pp_ℓ, u, $[\mathcal{A}]$, and $[\|\mathcal{A}\|]$

Phase 1 (Client):

1: $\|\vec{q}\| \leftarrow$ L2-norm of $\vec{q}$

2: $\langle \vec{q} \rangle \leftarrow KeyGen(msk_\ell, \vec{q})$; $\langle \|\vec{q}\| \rangle \leftarrow KeyGen(msk_1, \|\vec{q}\|)$

3: send $\langle \vec{q} \rangle$ and $\langle \|\vec{q}\| \rangle$ to the server

Phase 2 (Server):

4: Min-heap $R \leftarrow \emptyset$; $t \leftarrow -\infty$ ▶ R has a maximum size k

5: **for** $i = 1, \cdots, n'$ **do**

(a) $\|\vec{q}\| \cdot \|\vec{a}_i\| \leftarrow Decrypt(pp_1, [\|\vec{a}_i\|], \langle \|\vec{q}\| \rangle)$

(b) **if** $\|\vec{q}\| \cdot \|\vec{a}_i\| \leq t$ **then**

– send R to the client

(c) **else**

– $\vec{q}^T \vec{a}_i \leftarrow Decrypt(pp_\ell, [\vec{a}_i], \langle \vec{q} \rangle)$

– **for** $j = 1, \cdots, d$ **do**

• $\vec{q}^T \vec{p}_i^j \leftarrow Unpack(u, j, \vec{q}^T \vec{a}_i)$

• update R with $(i, j, \vec{q}^T \vec{p}_i^j)$ and t with $\vec{q}^T \vec{p}_i^j$ *if* $\vec{q}^T \vec{p}_i^j > t$

6: send R to the client

5.2.2 Construction. Algorithm 4 gives the detailed SKIP protocol.

Phase 1. The client computes the L2-norm $\|\vec{q}\|$ of inputted query $\vec{q}$ (Line 1), encrypts $\vec{q}$, $\|\vec{q}\|$ into $\langle \vec{q} \rangle$, $\langle \|\vec{q}\| \rangle$ (Line 2), and sends them to the server (Line 3).

Phase 2. The server uses a min-heap R of maximum size k to maintain the top-k results found so far, and an integer t to track the current k-th largest inner product. At first, the server initializes R and t (Line 4). Then, it performs the sequential scan over the outsourced database $[\mathcal{A}]$ (Line 5). Specifically, for each $[\vec{a}_i] \in [\mathcal{A}]$, the server first computes the IP upper bound $\|\vec{q}\| \cdot \|\vec{a}_i\|$ (Line 5(a)), and tries to use it to prune $[\vec{a}_i]$ by Eqn (15) (Line 5(b)). Recall that this test is cheap because it requires only to compute the IP using the ciphertext of two 1-dimensional vectors, i.e., $\|\vec{a}_i\|$ and $\|\vec{q}\|$. If the above pruning fails, the server decrypts the inner product $\vec{q}^T \vec{a}_i$ and unpacks it to $\vec{q}^T \vec{p}_i^j$ for further checking (Line 5(c)); that is, for $1 \leq j \leq d$, if some $\vec{q}^T \vec{p}_i^j$ is greater than the current k-th largest inner product t, the server updates heap R with $(i, j, \vec{q}^T \vec{p}_i^j)$ (if $R.size() < k$, enheap $(i, j, \vec{q}^T \vec{p}_i^j)$ directly; otherwise, deheap the top entry and then enheap $(i, j, \vec{q}^T \vec{p}_i^j)$), and updates t with $\vec{q}^T \vec{p}_i^j$. Finally, the server returns R to the client as the results (Line 6).

5.3 Security and Cost Analysis

5.3.1 Security Analysis. Due to to randomized encryption of the data $\vec{a}_i$, $\|\vec{a}_i\|$ and the query $\vec{q}$, $\|\vec{q}\|$ by AIPE, the SKIP protocol preserves the data confidentiality and query privacy as proved in Section 3.2. Typically, as shown in Theorem 2, SKIP resists KPA-noIP attack by enforcing semantically secure data confidentiality due to the Paillier variant cryptosystem [4]. To claim the security of SKIP against much stronger KPA-IP attack, we show that the server cannot utilize the observed inner products $\|\vec{q}\| \cdot \|\vec{a}_i\|$ used by sequential scan (Line 5(a)) and $\vec{q}^T \vec{a}_i$ used by IP Packing (Line 5(c)), accompanied by some known data vectors $P \subset \mathcal{P}$, to infer sensitive information. The proof is as follows:

(1) The server cannot use $\|\vec{q}\| \cdot \|\vec{a}_i\|$ and the known data P to infer more information because the former is independent of the later. Importantly, SKIP conducts sequential scan in a privacy preserving manner as it depends on upper bound $\|\vec{q}\| \cdot \|\vec{a}_i\|$ for pruning while $\|\vec{q}\| \cdot \|\vec{a}_i\|$ is computed from the ciphertext of $\|\vec{q}\|$ and $\|\vec{a}_i\|$.

(2) The server also cannot make use of $\vec{q}^T \vec{a}_i$ and the known data P to infer more information because IP Packing is resilient to the KPA-IP attack as proved in Theorem 5.

(1) and (2) jointly ensure that SKIP is resilient to KPA-IP attack.

5.3.2 Cost Analysis. The cost of SKIP includes: (Client cost) the encryption of ℓ-dimensional vector $\vec{q}$ and 1-dimensional vector $\|\vec{q}\|$ using *KeyGen*; (Server cost) the computation of $\|\vec{q}\| \cdot \|\vec{a}_i\|$ and $\vec{q}^T \vec{a}_i$ using *Decrypt*, for $1 \leq i \leq n'$. The cost of *Unpack* is omitted as it is trivial compared to *Decrypt*; (Communication cost) the uploading of $\langle \vec{q} \rangle$, $\langle \|\vec{q}\| \rangle$ and the downloading of the exact top-k results R.

We highlight the following performance benefits of SKIP and empirically study these advantages in Section 6.2.

- During query processing (Algorithm 4), thanks to IP Packing, the server's computation is proportional to the number $n' = \frac{n}{d}$ of auxiliary data vectors instead of the database size n. That is, IP Packing leads to d times performance improvement.
- During query processing (Algorithm 4), sequential scan further speeds up the above IP Packing by pruning significant IP computation (Line 5).
- During the initialization (Algorithm 3) , thanks to IP Packing, the client encrypts only $n' = \frac{n}{d}$ auxiliary data vectors $\vec{a}_1, \cdots, \vec{a}'_n$ instead of the original database $\mathcal{P}$ of size n. This incurs d-fold reduction of initialization cost. IP Packing also reduces outsourcing communication by uploading $[\mathcal{A}]$ and $[\|\mathcal{A}\|]$ at $O(n'\ell)$ cost instead of the encrypted database $[\mathcal{P}]$ requiring $O(n\ell)$ cost.

6 PERFORMANCE EVALUATION

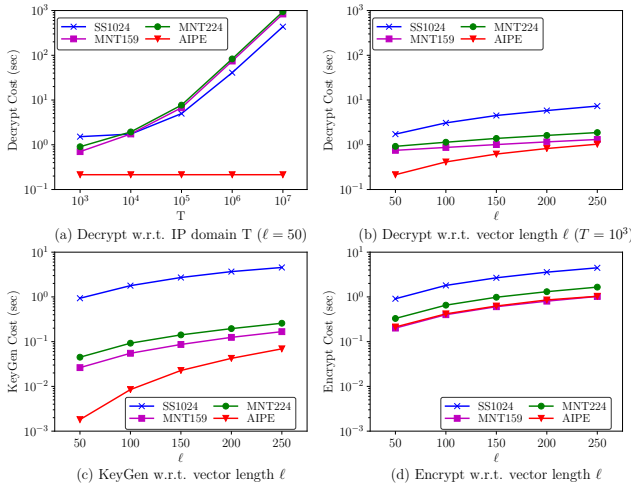
This section evaluated the AIPE scheme and SKIP protocol proposed in this paper. We implemented all methods in C++ and conducted all experiments on a Intel(R) Core(Tm) i7-3770 CPU @ 3.4 GHz machine running Ubuntu 16.04, with 8GB of main memory.

6.1 Performance of AIPE

6.1.1 Experimental Setup. We compare AIPE with currently the most efficient Pairing IPE shceme [12]. We do not consider ASPE [22] as a competitor because it is unable to achieve the same level of security as AIPE and Pairing IPE summarized in Table 1 (i.e., resist KPA-noIP). The Pairing IPE [12] uses different pairing curves to enable varied sizes of message space. In this experiment, we use publicly available source codes[3] for [12] over three different curves: MNT159 (160-bit space), MNT224 (224-bit space), and SS1024 (1024-bit space). MNT159 and MNT224 are also adoptd in [12]. For AIPE, we choose 1024-bit key size to have similar security as these curves, which provides significantly larger message spaces (1024-bit).

6.1.2 Results and Analysis. From Table 1, *Decrypt* is affected by vector length ℓ and IP domain T, and *KeyGen*, *Encrypt* are affected by ℓ. Figure 1 illustrates their costs with respect to varied ℓ, T.

[3] https://github.com/kevinlewi/fhipe



(a) Decrypt w.r.t. IP domain T ($\ell = 50$) (b) Decrypt w.r.t. vector length ℓ ($T = 10^3$)

(c) KeyGen w.r.t. vector length ℓ (d) Encrypt w.r.t. vector length ℓ

Figure 1: Costs of IPE schemes

Decrypt Cost. Figure 1(a) reports the *Decrypt* cost using different T (fix $\ell = 50$). Recall that larger T allows possible IP to lie in a larger interval. Pairing IPE incurs polynomial cost as T increases (up to 1000 sec with $T = 10^7$). It thus imposes that the IP domain must be sufficiently small. In contrast, AIPE completely eliminates such small IP domain restriction because its *Decrypt* cost is not affected by T. This trend is consistent with the theoretical results in Table 1: *Decrypt* involves $O(1)$ cost of computing discrete logarithm for AIPE but $O(\sqrt{T})$ cost for Pairing IPE. AIPE's stable *Decrypt* cost for revealing arbitrarily large IP is highly beneficial to IP Packing because it enables fast computation of large $\vec{q}^T\vec{a}_i$ used in Eqn (11).

Figure 1(b) further compares the *Decrypt* costs of AIPE and Pairing IPE using different vector length ℓ (fix $T = 10^3$). Although the cost of all methods exhibits a linear increase as ℓ increases, AIPE greatly outperforms Paring IPE. It incurs reduced cost than MNT159 and MNT224 for enabling *Decrypt* over a significantly enlarged message space (1024-bit). The cost reduction becomes more impressive over the same size of message space compared to SS1024. The result is explained by the superiority of AIPE shown in Table 1: AIPE requires no pairing operation for *Decrypt* but Pairing IPE requires $O(\ell)$ number of expensive pairings.

KeyGen Cost. Figure 1(c) illustrates another performance gain of AIPE. It involves much less *KeyGen* cost compared to Pairing IPE with varied ℓ. The reason is that AIPE adopts a lightweight encryption for query vectors without needing exponentiations, as shown in Table 1. This efficient query encryption facilitates short retrieval time for outsourcing applications.

Encrypt Cost. Figure 1(d) reports the *Encrypt* cost using varied ℓ. Since both AIPE and Pairing IPE encrypt data vectors with crypto primitives to enforce strong data confidentiality, it has small performance improvements than Pairing IPE. Note worthily, the large message space of AIPE (1024-bit) enables to pack large numbers of data vectors into each encrypted auxiliary vector $[\vec{a}_i]$ (i.e., large d), which benefits IP Packing for resisting KPA-IP.

Since Pairing IPE suffers from small IP domain restriction and expensive costs as shown above, all of our experiments for top-k IP retrieval in the next subsection are based on AIPE.

Table 2: Comparison of secure top-k retrieval protocols

Algorithm	IP Packing	Sequential Scan
Naive	✗	✗
Pack	✓	✗
AIPE-SS	✗	✓
SKIP	✓	✓

Table 3: Dataset statistics ($\ell = 50, max = 5 \times 10^4, u = 10^5$)

Dataset	# of data vectors (n)	# of query vectors
MovieLens	33,670	247,753
Yelp	77,079	552,339

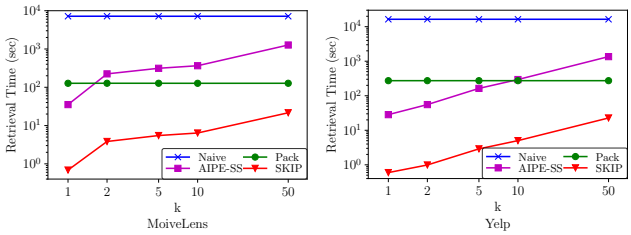


Figure 2: Average retrieval time of 100 random queries

6.2 Performance of SKIP

6.2.1 Experimental Setup. We compare SKIP with three competitors: Naive, Pack, and AIPE-SS. The Naive method adopts only AIPE that initially encrypts each $\vec{p}_i$ of the database $\mathcal{P}$ into $[\vec{p}_i]$ and outsources it to the server, then searches for top-k IP by computing all $\vec{q}^T\vec{p}_i$ with $Decrypt(pp, [\|\vec{p}_i\|], \langle\|\vec{q}\|\rangle)$ on the server. Pack implements IP Packing presented in Section 4. SKIP combines sequential scan with IP Packing, as described in Section 5. AIPE-SS implements sequential scan over encrypted data using AIPE, which corresponds to SKIP without packing (i.e., $d = 1$). The differences among four protocols are summarized in Table 2. To enforce the same security, we choose 1024-bit key size for AIPE in all protocols.

We evaluate the above protocols over two real datasets of recommender systems: Movielens and Yelp. These two datasets, consisting of user (query) and item (data) latent vectors factorized from a user-item rating matrix each, are widely used in collaborative filtering research [13, 19, 21] to recommend top-k items that produce top ranked IP between a user's latent vector and all item vectors. For each dataset, we transform 50-dimensional float data and query vectors generated[4] by [13] to integer vectors by following integer scaling used in [13], which scales the original 5-point IP domain to the interval $[0, 5 \times 10^4]$, i.e., $max = 5 \times 10^4$. For Pack and SKIP, we choose $u = 10^5$ to enforce $u > max$, which gives $d = 60$ in Eqn (9) as 1024-bit key size has $\log_2 N = 1024$, and thus KPA-IP attacker's probability of success is less than 4.3×10^{-76} by Theorem 5. The datasets are summarized in Table 3.

6.2.2 Results and Analysis. **Retrieval Cost**. Figure 2 reports the retrieval cost with varied k. For each dataset, we randomly select 100 queries and report the average retrieval cost (in sec). We only

[4]https://github.com/Hui-Li/MFRetrieval. Secure top-k IP retrieval is orthogonal to how data/query vectors are generated and thus does not affect recommendation quality.

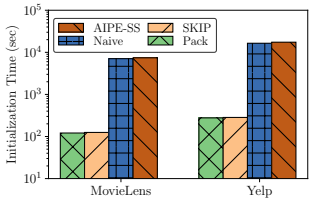


Figure 3: Initialization cost

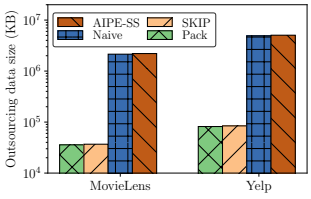


Figure 4: Outsourcing cost

account for the cost of Phase 2 because for all methods the costs of Phase 1 are trivial ($\leq 0.01sec$) and the same. We observe that,

IP Packing reduces retrieval cost. The IP packing based methods SKIP and Pack significantly improve (60x) their non-packing counterparts AIPE-SS and Naive, respectively. The retrieval cost is dominated by computing inner products required for finding top-k results, which is proportional to the number of calls to *Decrypt*. With the help of IP Packing, each calling of *Decrypt* (e.g., Line 5(c), Alg 4 for SKIP) computes d inner products between the query and data vectors, at trivial extra cost by *Unpack*. Therefore, IP Packing incurs d times less calls of *Decrypt*. Since $d = 60$ as discussed above, SKIP and Pack have 60x speed up using IP Packing, besides enhanced security for resisting KPA-IP.

Sequential scan reduces retrieval cost. The sequential scan based methods SKIP and AIPE-SS are much faster (6x-200x for MovieLens, 12x-460x for Yelp) than their non-pruning counterparts Pack and Naive, respectively. Sequential scan reduces the great majority of IP computation by pruning calls of *Decrypt* (e.g., Line 5(c), Alg 4 for SKIP), at the cost of computing IP upper bound (Line 5(a)) for pruning checks (Line 5(b)). Compared to computing actual IP (Line 5(c)), computing IP bound (Line 5(a)) is significantly more efficient as it applies *Decrypt* over 1-dimensional vectors. This explains why sequential scan is highly beneficial. Note that the performance of SKIP and AIPE-SS degrades with larger k. This is because the smallest IP (t) in the current top-k set (heap R) becomes smaller and it makes harder for SKIP and AIPE-SS to prune (Line 5(b)).

Since IP Packing and sequential scan produces independently retrieval time reduction each, the combination of them in SKIP offers the best performance gain (3 to 4 orders of magnitude).

Initialization and Outsourcing Cost. Figure 3 report the time of initialization (in sec). Over both datasets, the IP packing based methods (SKIP and Pack) incur 60 times less cost than their non-packing competitors (AIPE-SS and Naive), due to reduced calls of *Encrypt* (e.g., Line 4(c), Alg 3 for SKIP). This further illustrates the unique advantage of IP Packing on improving the performance and security simultaneously. Compared to non-pruning methods (Pack and Naive), the sequential scan based competitors (SKIP and AIPE-SS) have sightly higher cost because of computing (Line 4(b)) and encrypting (Line 4(c)) index norms $\|\vec{a}_i\|$. This demonstrates the high benefits of sequential scan that uses very limited cost (typically, the initialization is only conducted once) to achieve excessive pruning of IP computation as shown in Figure 2. Finally, Figure 4 reports the size of outsourced data (in KB). It exhibits the same trends as Figure 3; that is, IP Packing reduces the cost besides enhancing the security and sequential scan incurs only limited extra cost.

7 CONCLUSIONS

In this paper, we have studied the secure top-k inner product retrieval problem. We proposed a new IPE scheme AIPE that balances the security and performance. On top of AIPE, we presented an IP Packing method to enable secure top-k IP retrieval that resists KPA-IP attack and reduces retrieval costs. To further improve query performance, we proposed a pruning based approach SKIP that speeds up IP Packing with sequential scan. Thorough security analysis and extensive experimental evolutions illustrate the validity, security and efficiency of our schemes.

REFERENCES

[1] Shweta Agrawal, Benoit Libert, and Damien Stehlé. 2016. Fully secure functional encryption for inner products, from standard assumptions. In *CRYPTO*. Springer, 333–362.

[2] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. 2015. Function-hiding inner product encryption. In *Asiacrypt*. Springer, 470–491.

[3] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. 2011. Order-preserving encryption revisited: improved security analysis and alternative solutions. In *CRYPTO*. Springer, 578–595.

[4] Emmanuel Bresson, Dario Catalano, and David Pointcheval. 2003. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *Asiacrypt*. Springer, 37–54.

[5] Gu Chunsheng and Gu Jixing. 2014. Known-plaintext attack on secure kNN computation on encrypted databases. *Security and Communication Networks* 7, 12 (2014), 2432–2441.

[6] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. 2016. Functional encryption for inner product with full function privacy. In *PKC*. Springer, 164–195.

[7] Liu Fang, Wee Keong Ng, and Wei Zhang. 2014. Encrypted scalar product protocol for outsourced data mining. In *CLOUD*. IEEE, 336–343.

[8] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: applying neural networks to encrypted data with high throughput and accuracy. In *ICML*. 201–210.

[9] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. 2004. On private scalar product computation for privacy-preserving data mining. In *International Conference on Information Security and Cryptology*. Springer, 104–120.

[10] Manish Kesarwani, Akshar Kaul, Prasad Naldurg, Sikhar Patranabis, Gagandeep Singh, Sameep Mehta, and Debdeep Mukhopadhyay. 2018. Efficient secure k-nearest neighbours over encrypted data. In *EDBT*.

[11] Hyeong-Il Kim, Hyeong-Jin Kim, and Jae-Woo Chang. 2017. A secure kNN query processing algorithm using homomorphic encryption on outsourced database. *Data & Knowledge Engineering* (2017).

[12] Sam Kim, Kevin Lewi, Avradip Mandal, Hart William Montgomery, Arnab Roy, and David J Wu. 2016. Function-hiding inner product encryption is practical. *IACR Cryptology ePrint Archive* 2016 (2016), 440.

[13] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. FEXIPRO: fast and exact inner product retrieval in recommender systems. In *SIGMOD*. ACM, 835–850.

[14] Weipeng Lin, Ke Wang, Zhilin Zhang, and Hong Chen. 2017. Revisiting security risks of asymmetric scalar product preserving encryption and its variants. In *ICDCS*. IEEE, 1116–1125.

[15] Xuying Meng, Suhang Wang, Kai Shu, Jundong Li, Bo Chen, Huan Liu, and Yujun Zhang. 2018. Towards privacy preserving social recommendation under personalized privacy settings. *World Wide Web* (2018), 1–29.

[16] Sikhar Patranabis and Debdeep Mukhopadhyay. 2017. Lightweight symmetric-key hidden vector encryption without pairings. Cryptology ePrint Archive, Report 2017/796. (2017). https://eprint.iacr.org/2017/796.

[17] Parikshit Ram and Alexander G Gray. 2012. Maximum inner-product search using cone trees. In *SIGKDD*. ACM, 931–939.

[18] Zihao Shan, Kui Ren, Marina Blanton, and Cong Wang. 2017. Practical secure computation outsourcing: a survey. *Comput. Surveys* 51 (2017).

[19] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Advances in Neural Information Processing Systems*. 2321–2329.

[20] Sen Su, Yiping Teng, Xiang Cheng, Ke Xiao, Guoliang Li, and JunLiang Chen. 2015. Privacy-preserving top-k spatial keyword queries in untrusted cloud environments. *IEEE Transactions on Services Computing* (2015).

[21] Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. 2015. Lemp: Fast retrieval of large entries in a matrix product. In *SIGMOD*. ACM, 107–122.

[22] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. 2009. Secure knn computation on encrypted databases. In *SIGMOD*. ACM, 139–152.

[23] Yunchen Wu, Ke Wang, Zhilin Zhang, Weipeng Lin, Hong Chen, and Cuiping Li. 2018. Privacy preserving group nearest neighbor search. In *EDBT*. 277–288.