Fast and Space-Efficient Secure Frequent Pattern Mining by FHE

Hiroki Imabayashi, Yu Ishimaki, Akira Umayabara, Hayato Yamana
Waseda University
Tokyo, Japan
{imabayashi, yuishi, uma, yamana}@yama.info.waseda.ac.jp

Abstract— **In the big data era, security and privacy concerns are growing. One of the big challenges is secure Frequent Pattern Mining (FPM) over Fully Homomorphic Encryption (FHE). There exist some research efforts aimed at speeding-up, however, we have a big room so as to decrease time and space complexity. Apriori over FHE, in particular, generates a large number of ciphertexts during the support calculation, which results in both large time and space complexity. To solve it, we proposed a speed-up technique, around 430 times faster and 18.9 times smaller memory usage than the state-of-the-art method, by adopting both packing and caching mechanism. In this paper, we further propose to decrease the memory space used for caching. Our goal is to discard redundant cached ciphertexts without increasing the execution time. Our experimental results show that our method decreases the memory usage by 6.09% at most in comparison with our previous method without increasing the execution time.**

Keywords—Fully Homomorphic Encryption; Frequent Pattern Mining; Cache Pruning; Ciphertext Caching;

I. Introduction

To execute mining tasks securely on a cloud server, several studies on FHE exist. Liu et al. [1] proposed a state-of-the-art Frequent Patten Mining (FPM) protocol over FHE called P3CC, which however requires long execution time. To reduce it, we adopted both ciphertext-packing and ciphertext-caching [2]. Although this technique accelerates the support calculations with caches, it requires additional memory space in $\mathcal{O}(2^N)$, where N is the number of different items, which leads to a critical problem with large datasets.

In this paper, we propose a novel cache-pruning technique to prevent the server from caching non-reused ciphertexts. Besides, we propose a fast and space-efficient FPM protocol for Apriori over FHE with the cache-pruning technique. The advantage of our protocol is twofold: i) The larger the dataset size is, the more our cache-pruning technique saves the memory space. ii) Our protocol has little effect on the execution time by separating the cache-pruning process on different threads.

II. Related Work

There exist several works to execute mining tasks securely on a cloud server. The works on data mining with cryptosystems [3][4] is classified into two approaches: multi-party computation (MPC) and homomorphic encryption (HE). In an approach by MPC, Kantarcioglu et al. [3] proposed an algorithm for mining frequent patterns from distributed databases while preserving privacy among multi-parties. On the other hand, in an approach by HE, Kaosar et al. [4] proposed a technique to compare numbers to judge "larger than" over encrypted data with FHE in a two-party association rule mining scenario, followed by showing that the approach by MPC is not appropriate for the association rule mining due to large costs on the storage, communications and calculations.

In particular, as to the frequent pattern mining by FHE, Liu et al. [1] proposed Privacy Preserving Protocol for Counting Candidates (P3CC), which encrypts all individual binary-represented components in the item-transaction matrix data, i.e., component-wise encryption, and then applies addition or multiplication to each ciphertext individually. In P3CC, therefore, the total number of ciphertexts increases linearly with the matrix size, which results in the excessive memory usage, communication costs, and the operational costs over encrypted data. To address this issue, we proposed a speed-up technique [2], which adopted ciphertext-packing by Smart and Vercauteren [5][6] with our constructed ciphertext-caching mechanism. With the packing mechanism, multiple components are packed into a ciphertext, which reduces total number of ciphertexts. The caching mechanism reduces the number of redundant multiplications among ciphertexts by reusing previously calculated ciphertexts during the support calculation. Although this technique is able to speed-up around 430 times while reducing memory space 18.9 times in comparison with P3CC, this caching mechanism caches redundant ciphertexts, which uses an additional memory space.

III. Method

Apriori extracts frequent patterns whose frequency called *support* is larger than a given threshold called *minimum support*. However, with FHE, it is hard to judge "larger than" because a comparison over ciphertexts is impossible or takes long time. Thus, in previous implementations [1][2], a client takes over the comparison on behalf of the sever by decrypting them, followed by generating next-pattern-length (here, assumed as k) candidate itemsets, and sending them back to the server. Then, the server calculates supports of the length k candidate itemsets. In our previous proposal [2], during the support calculation on the server, we keep all the previously calculated supports in ciphertext form, hereafter we call them as cached ciphertexts, and reuse them to speed-up the later support calculation for longer-length candidate itemsets.

To prune wasteful cached ciphertexts, our cache-pruning technique generates *pseudo candidates* of length k+1 from the candidates of length k by using item-ids that are not encrypted [1][2]. Here, item-ids are hashed not to be known by the server [1]. Our idea behind is to generate *pseudo candidates* of length k+1 to prune the cache on the server instead of waiting length k+1 real candidates sent from the client. We prune the cached ciphertexts if it is not any of the subsets of length k+1 *pseudo candidates*. Fig. 1 shows our protocol overview, which executes cache-pruning asynchronously to have little effect on the execution time.

IV. Experimental Results and Evaluation

We evaluated the effectiveness of our proposed protocol with cache-pruning technique in terms of the execution time, memory usage and the number of caches.

Our method is implemented with FHE library called *HElib*. The platform consists of two machines: a client with an Intel Xeon E5-2643 v3@3.4 GHz CPU with 512 GB RAM on 12 threads, and a server with an Intel Xeon E7-8880 v3@2.3 GHz CPU with 1 TB RAM on 24 threads, both of which are

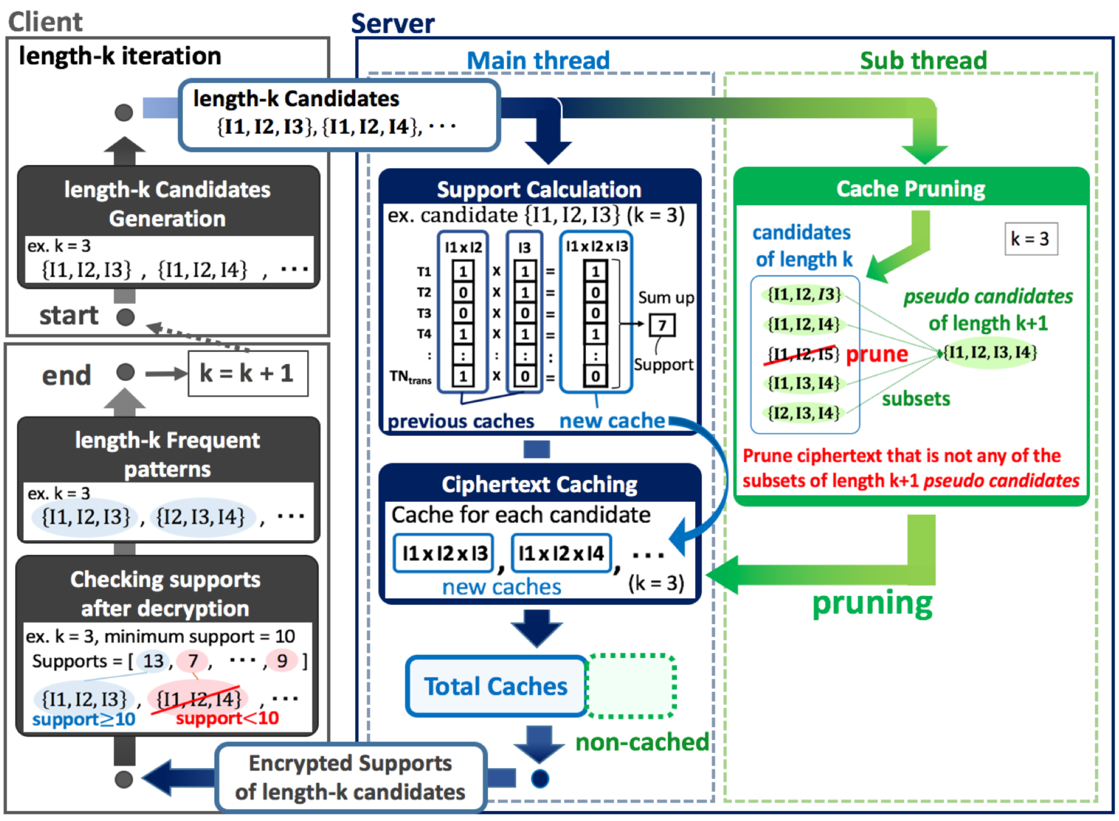
Client
length-k iteration
length-k Candidates
{I1, I2, I3}, {I1, I2, I4}, ···
length-k Candidates Generation
ex. k = 3
{I1, I2, I3} , {I1, I2, I4} , ···
start
end
k = k + 1
length-k Frequent patterns
ex. k = 3
{I1, I2, I3} , {I2, I3, I4} , ···
Checking supports after decryption
ex. k = 3, minimum support = 10
Supports = [13 , 7 , ··· , 9]
{I1, I2, I3} , ~~{I1, I2, I4}~~ , ···
support≥10
support<10
Server
Main thread
Sub thread
Support Calculation
ex. candidate {I1, I2, I3} (k = 3)
I1 x I2
I3
I1 x I2 x I3
T1
T2
T3
T4
TNtrans
Sum up
7
Support
previous caches
new cache
Ciphertext Caching
Cache for each candidate
I1 x I2 x I3 , I1 x I2 x I4 , ···
new caches
(k = 3)
Total Caches
non-cached
Encrypted Supports of length-k candidates
Cache Pruning
candidates of length k
k = 3
{I1, I2, I3}
{I1, I2, I4}
~~{I1, I2, I5}~~
prune
{I1, I3, I4}
{I2, I3, I4}
pseudo candidates of length k+1
{I1, I2, I3, I4}
subsets
Prune ciphertext that is not any of the subsets of length k+1 pseudo candidates
pruning

Figure 1. Overview of our protocol with cache-pruning

TABLE 1. Experimental environment

	CPU	RAM	Thread
Client	Intel Xeon E5-2643 v3 @3.4 GHz	512GB	12
Server	Intel Xeon E7-8880 v3 @2.3 GHz	1TB	24

Connected with 10Gbps Ethernet

TABLE 2. Comparison of memory size and #cache

P3CC [1] → Our proposed protocol [2] → Our proposed protocol (N/A: over 1TB)

dataset*	N	execution time (sec)	memory (GB)	#cache
T6I6N30D10kL1k	30	8,481 → 78.7 → 79.2	428 → 15.9 → 15.7	(none) → 2,976 → 2,449
T10I6N50D10kL1k	50	30,783 → 125 → 125	536 → 25.4 → 24.7	(none) → 8,153 → 7,285
T14I6N70D10kL1k	70	54,490 → 187 → 188	738 → 34.9 → 33.0	(none) → 15,438 → 13,299
T20I6N100D10kL1k	100	N/A → 315 → 314	N/A → 55.3 → 53.9	(none) → 30,225 → 27,187
T40I6N200D10kL1k	200	N/A → 1,421 → 1,417	N/A → 230 → 216	(none) → 164,052 → 151,621
T60I6N300D10kL1k	300	N/A → 2,272 → 2,283	N/A → 373 → 352	(none) → 266,352 → 246,789

*Datasets were generated by IBM Quest Dataset Generator.
minimum support = #transactions (D=10,000) * 20% = 2,000

connected with 10Gbps Ethernet as shown in TABLE 1. We experimented with various datasets where both the average length of items per transaction, T, and the number of item-ids, N, are varied. We set the number of transactions, D, to 10,000, minimum support to 20% of D, i.e., 2,000.

Our experimental results compared with P3CC [1] and our previously proposed protocol [2] are shown in TABLE 2. We also show them in figures: i) the execution time in Figure 2, ii) the memory usage in Figure 3, and iii) the number of caches in Figure 4, while varying the number of different items. As shown in Figure 2, the execution time of our proposed protocol [2] and that of our newly proposed protocol are almost same, while P3CC [1] takes much longer time. Figure 3 shows that the saved memory space becomes larger as N increases, while the memory usage of every protocol increases linearly on N. Figure 4 shows that our proposed cache-pruning technique successfully discarded larger number of cached ciphertexts as N increases, while the number of caches increases linearly on

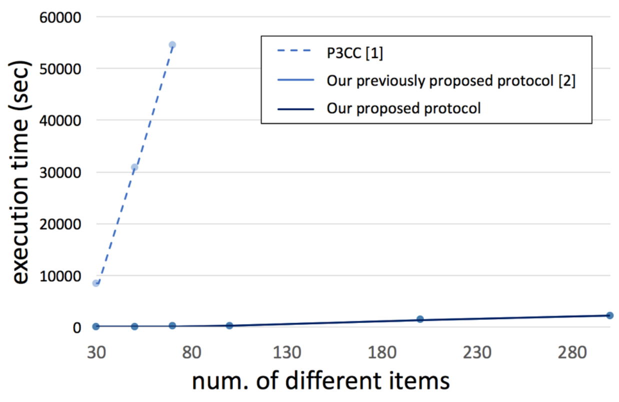

Figure 2. Comparison of execution time

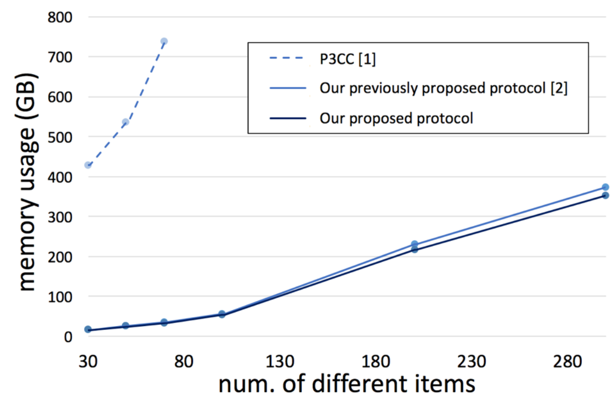

Figure 3. Comparison of memory usage

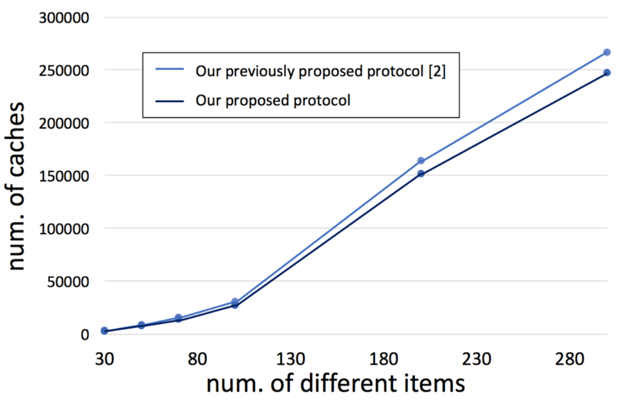

Figure 4. Comparison of #caches

N.

Our method successfully decreased both the number of cached ciphertexts (#cache) and the size of memory without changing the execution time much. At the maximum, we decreased 6.09% of memory when *N* is 200.

V. Conclusion and Discussion

In this paper, we proposed a fast and space-efficient FPM protocol for Apriori over FHE, which decreases a memory space by pruning wasteful cashed ciphertexts. With this cache-pruning technique, the experimental results showed that our proposed protocol decreased memory usage by 6.09% at most.

As for the remaining problem, our protocol still has limitations to the dataset size, since the memory space increases linearly on the number of different items. In real world applications, available computational resources and network bandwidth are limited. We will, therefore, conduct a new construction of FPM protocol where data is processed in stream computation, to further make our FPM protocol fast and space-efficient.

Aknowledgment

This work was supported by CREST, JST.

References

[1] J. Liu, J. Li, S. Xu, et al. “Secure outsourced frequent pattern mining by fully homomorphic encryption.” *Big Data Analytics and Knowledge Discovery*, LNCS, vol. 9264, pp. 70–81, 2015.

[2] H. Imabayashi, Y. Ishimaki, A. Umayabara, et al. “Secure frequent pattern mining by fully homomorphic encryption with ciphertext packing.” *International workshop on Data Privacy Management (DPM 2016)*, vol. 9963, LNCS, 2016.

[3] M. Kantarcioglu and C. Chris. “Privacy-preserving distributed mining of association rules on horizontally partitioned data.” *IEEE Transactions on Knowledge and Data Engineering (TKDE 2004)*, vol. 16, pp. 1026-1037, 2004.

[4] M.G. Kaosar, R. Paulet and X. Yi. “Fully homomorphic encryption based two-party association rule mining.” *Data & Knowledge Engineering*, vol. 76, pp. 1–15, 2012.

[5] N.P. Smart and F. Vercauteren. “Fully homomorphic encryption with relatively small key and ciphertext sizes.” *Public Key Cryptography–PKC 2010,* LNCS, vol. 6056, pp. 420–443, 2010.

[6] N.P. Smart and F. Vercauteren, “Fully homomorphic simd operations.” *Designs, Codes and Cryptography*. vol. 71, no. 1, pp. 57–81, 2014.