Integrity Verification of Outsourced Frequent Itemset Mining with Deterministic Guarantee

Boxiang Dong
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ, USA
Email: bdong@stevens.edu

Ruilin Liu
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ, USA
Email: rliu3@stevens.edu

Wendy Hui Wang
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ, USA
Email: Hui.Wang@stevens.edu

***Abstract*—In this paper, we focus on the problem of result integrity verification for outsourcing of frequent itemset mining. We design efficient cryptographic approaches that verify whether the returned frequent itemset mining results are correct and complete with deterministic guarantee. The key of our solution is that the service provider constructs cryptographic proofs of the mining results. Both correctness and completeness of the mining results are measured against the proofs. We optimize the verification by minimizing the number of proofs. Our empirical study demonstrates the efficiency and effectiveness of the verification approaches.**

***Keywords*-Data-mining-as-a-service; Cloud computing; integrity verification; frequent itemset mining;**

I. Introduction

The increasing ability to generate vast quantities of data presents technical challenges for efficient data mining. Outsourcing data mining computations to a third-party service provider (server) offers a cost-effective option, especially for data owners (clients) of limited resources. This introduces the data-mining-as-a-service ($DMaS$) paradigm. Cloud computing provides a natural solution for the $DMaS$ paradigm.

Although the $DMaS$ paradigm is advantageous to achieve sophisticated analysis on large volumes of data in a cost effective way, it raises a few security issues. One of the main security concerns is that the server may return incorrect data mining results due to many possible reasons (e.g., software bugs and malicious insiders) [3], [7]. After all, a server has the financial incentive to improve its revenue by returning cheaper (and thus incorrect) result while charging for more [3], [16].

Given the fact that many data mining applications (e.g., fraud detection and business intelligence) are so critical, it is important to provide efficient and practical methods to enable computationally-weak clients to verify the *result integrity* of data mining computations that are outsourced to a potentially untrusted service provider. In this paper, we focus on frequent itemset mining, a popular and important data mining problem. Our goal is to verify whether the server returned correct and complete set of frequent itemsets. By *correctness*, we mean that all itemsets returned by the server are frequent. By *completeness*, we mean that no frequent itemset is missing in the returned result.

Prior Work. [7] defined the notation of *verifiable computation* (VC) that allows a computationally limited client to be able to verify the correctness of the result of expensive computations outsourced to a computationally powerful server. Existing VC protocols can verify high degree polynomial functions [2], boolean functions [16], and set operations [15]. These VC protocols assume the same computation is executed on many different inputs, so that the cost of the expensive one-time setup phase can be amortized over all instances. This is not ideal for the $DMaS$ paradigm. Some protocols (e.g. [4]) consider functions that are encoded as a Boolean circuit and use fully homomorphic encryption [8], which only can handle simple operations such as addition, multiplications and XOR operations. The existing work on result verification of SQL query evaluation over the outsourced databases (e.g., [9], [17]) cannot be applied to the $DMaS$ paradigm, due to the fundamental difference between SQL query evaluation and data mining applications. [19], [5] proposed efficient integrity verification approaches for outsourced frequent itemset mining. However, both of them only can provide probabilistic result integrity guarantee.

Contributions. In this paper, we aim at designing verification techniques that can provide *deterministic* guarantee for both correctness and completeness of outsourced frequent itemset mining. The key idea of our solution is to require the server to construct *cryptographic proofs* of its mining results. Both correctness and completeness of the mining results are measured against the proofs with 100% certainty. Our contributions include the following: (1) we adapt the verification protocol of primitive set intersection operation [15] to our problem, and show how to construct cryptographic proofs that can verify the exact support of itemsets; (2) we optimize the verification algorithm by reducing the number of proofs for both correctness and completeness verification, and show that a small number of proofs is sufficient to verify the correctness and completeness of a large set of frequent itemsets; (3) we extend our verification approaches for the outsourced frequent itemset mining to maximal frequent itemset mining; and (4) we complement our analytical results with extensive experiments evaluating the performance of our verification algorithm. Our experimental results show that our verification approach can achieve both correctness and completeness deterministic guarantees with a small number of short proofs that can be constructed fast.

II. Preliminaries

Frequent Itemset Mining. Given a transaction dataset D that consists of n transactions, let $\mathcal{I}$ be the set of unique items in D. The *support* of the itemset $I \subseteq \mathcal{I}$ is the number of transactions in D that contain I. An itemset I is *frequent* if its support is no less than a minimum support threshold min_{sup} [1].

Outsourcing Setting. The data owner (client) outsources her dataset D, with the minimum support threshold min_{sup}, to the service provider (server). The server performs frequent itemset mining on the received dataset and returns the mining results to the client. We assume the service provider runs exact frequent itemset mining algorithms instead of approximate ones [14]. We allow the client to use privacy-preserving frequent itemset mining algorithms [13], [18] to encrypt the dataset; the server returns the accurate itemsets in encrypted format. Our verification procedure can verify the support of the returned itemsets (in encrypted format).

Verification Goal. We formally define the correctness and completeness of the frequent itemset mining results. Let F be the real frequent itemsets in the outsourced database D, and F^S be the frequent itemsets returned by the server. We define the *precision* P of F^S as $P = \frac{|F \cap F^S|}{|F^S|}$ (i.e., the percentage of returned frequent itemsets that are correct), and the *recall* R of F^S as $R = \frac{|F \cap F^S|}{|F|}$ (i.e., the percentage of correct frequent itemsets that are returned). Our aim is to catch any answer of $P < 1$ and/or $R < 1$ with 100% probability.

III. Set Intersection Verification Protocol

In this paper, we adapt the set intersection verification protocol [15] to our problem. Formally, given a collection sets $\mathcal{S} = \{S_1, \ldots, S_k\}$, $E = S_1 \cap S_2 \cap \cdots \cap S_k$ is the *correct intersection* of $\mathcal{S}$ if and only if:

- $E \subseteq S_1 \wedge \cdots \wedge \subseteq S_k$ (subset condition);
- $(S_1 - E) \cap \cdots \cap (S_k - E) = \emptyset$ (completeness condition).

Here the completeness condition is on the set intersection. To avoid confusion with our completeness verification of frequent itemsets, in the remainder of the paper, we use the term *intersection completeness* for the completeness condition on the set intersection.

Papamanthou et al. proposed a set intersection verification protocol to verify that E is the correct intersection of $\mathcal{S}$ [15]. Next, we explain the protocol details briefly.

Proof construction by the server. Given an intersection result $E = \{e_1, \ldots, e_\delta\}$, to prove that $E = S_1 \cap \cdots \cap S_k$, the set intersection verification protocol requires the server to construct its proof $\Pi(E)$ that consists of four parts: (1) an encoding of the result (as a polynomial) as the coefficients $\mathcal{B}$ $=\{b_\delta, b_{\delta-1}, \cdots, b_0\}$ of the polynomial $(s+e_1)(s+e_2)\cdots(s+e_\delta)$, where s is a randomly chosen value by the client and kept as secret; (2) a set of *accumulation values* $\mathcal{A} =\{acc(S_j)|\forall S_j \in \mathcal{S}\}$ which can be used to verify that the proof is indeed computed from the original dataset D; (3) the *subset witness* $\mathcal{W} = \{W_j|\forall S_j \in \mathcal{S}\}$ as the proof of the subset condition; and (4) the *intersection completeness witness* $\mathcal{C} = \{C_j|\forall S_j \in \mathcal{S}\}$ as the proof of the intersection completeness condition. The total complexity of proof construction is $O(Nlog^3N + m^\epsilon logm)$, where $N = \sum_{j=1}^{k} |S_j|$, m is the number of leaves of the Merkle tree of the dataset D, and $\epsilon \in (0, 1)$ decides the number of levels of the Merkle tree (as $[1/\epsilon]$). More details of the Merkle tree will be present in Section IV.

Correctness verification by the client. After receiving $\Pi(E) = \{\mathcal{B}, \mathcal{A}, \mathcal{W}, \mathcal{C}\}$ together with E, the client verifies the following: (1) whether the coefficients $\mathcal{B}$ are computed correctly by the server; (2) whether any given accumulation value in $\mathcal{A}$ is indeed calculated from the original dataset; (3) whether E satisfies the subset condition by using $\mathcal{W}$; and (4) whether E satisfies the intersection completeness condition by using $\mathcal{C}$. We omit the details of proof construction and verification due to limited space. Readers can refer to [15] for more details.

IV. Basic Solution

In this section, we present our basic verification approach.

A. Authenticated Data Structure

Before sending the dataset D to the server, the client constructs an authenticated data structure. The authenticated data structure is constructed from the *item-based inverted index*. In particular, given a dataset D that contains p unique items, its *item-based inverted index* E^I consists of p inverted lists $\{L_1, \ldots, L_p\}$, each maintaining the index of transactions that contains the item I_i. We construct the authenticated data structure as the Merkle hash tree $\mathcal{T}$ [12] of the inverted index. In particular, for each leaf l_j of $\mathcal{T}$ that corresponds to the j^{th} inverted list L_j in E^I, the client executes the set intersection verification protocol to construct $acc(l_j)$. Then the client applies a collision-resistant hash function $hash(\cdot)$ recursively over the nodes of $\mathcal{T}$. Each leaf l_j of $\mathcal{T}$ is assigned the value $h_j = hash(acc(l_j)^{(s+j)})$, where s is a randomly chosen value by the client and kept as secret. Each internal node that has children $c_1, \ldots, c_k$ is assigned the value $h_v = hash(h_{c_1}|| \ldots ||h_{c_k})$. The root of the tree is signed to produce signature $sig(E^I)$. The client sends $\mathcal{T}$ to the server with D, and keeps $sig(E^I)$ locally. At this point, besides constructing the Merkle tree, the client finds all frequent and infrequent 1-itemsets from the inverted index. She maintains such information for later verification.

B. Verification Procedure

The frequent itemset verification problem can be mapped to the set intersection verification problem by treating each inverted list L_i as a set. Then verifying whether any itemset I is included in a set of transactions T^I is equivalent to verifying whether T^I is the correct intersection of the inverted lists of all items in I. Following this idea, we design the verification procedure based on the set intersection verification protocol. The details of the procedure are discussed below.

Verification Preparation by the client. Before outsourcing the dataset D to the server, the client constructs the item-based inverted index E^I of D and the Merkle hash tree $\mathcal{T}$ of E^I. The client keeps the hash value of the root element of $\mathcal{T}$ locally, and sends D and $\mathcal{T}$ to the server.

Proof construction by the server. After the server receives D and $\mathcal{T}$ from the client, the server runs frequent itemset mining on D and discovers a set of frequent itemsets. Before the server sends its result F^S to the client, it constructs *proofs* for result integrity verification. In particular, for each itemset $I \in F^S$ and each itemset $I \notin F^S$, the server constructs a proof that can be used to verify the exact support of I. The proof of the itemset $I \in F^S$ ($I \notin F^S$, resp) is used for the correctness (completeness, resp.) verification.

Result verification by the client. For each itemset $I \in F^S$, let $\Pi(I) = \{\mathcal{B}, \mathcal{A}, \mathcal{W}, \mathcal{C}\}$ be its proof, the client launches the following steps for verification: (1) it runs the set intersection verification protocol on $\Pi(I)$ to verify I is contained in a set of transactions T^I; (2) it computes the support of I as $I_{sup} = |\mathcal{B}| - 1$. Then the correctness is to verify whether the support of any $I \in F^S$ is no less than min_{sup}, while the completeness verification is to verify whether the support of any itemset $I \notin F^S$ is less than min_{sup}.

The complexity of the basic verification approach can be prohibitively expensive, due to the fact that the total number of (correctness and completeness) proofs is the same as the size of the search space of frequent itemsets. This will introduce considerable amounts of overhead at both the client and the server sides.

V. Verification Optimization

In this section, we discuss how to reduce the number of proofs for verification. The optimization is based on the concept of *maximal frequent itemsets* (MFI) and *minimal infrequent itemsets* (MII). In particular, given a set of frequent itemsets F^S by the server, MFI is a subset of F^S such that for each itemset $I \in MFI$, there does not exist any itemset $I' \in F^S$ such that $I \subseteq I'$; while MII is a set of itemsets that do not appear in F^S such that for each itemset $I \in MII$, there does not exist any itemset $I' \notin F^S$ such that $I' \subseteq I$.

A. Proof Construction at Server Side

To prepare for correctness (completeness, resp.) verification, the server constructs a cryptographic proof for each MFI (MII, resp.) itemset. The MII itemsets should include all infrequent 1-itemsets. To further reduce the number of proofs, the server only needs to construct the proofs of those MII itemsets that are of length greater than one and only contain frequent 1-itemsets. After the proof construction is finished, the server sends the proofs together with F^S to the client.

B. Verification at Client Side

Correctness verification. The client uses the server's proofs to verify that each MFI itemset is frequent by running the set intersection verification protocol. Though simple, only sending proofs of MFI itemsets to the client raises two new issues. The first issue is to verify that the server has computed the proof of all MFI itemsets of F^S. To do this, for each itemset $I \in F^S$, the client verifies whether there exists a MFI itemset I' satisfies that $I \subseteq I'$. If it does not, the client concludes that the server misses the proof of at least one MFI itemset. Another issue is that the client needs to ensure that all MFI itemsets are honestly constructed from F^S, not from the (correct) mining results of D. This can be achieved by verifying: (1) for each MFI itemset I, whether there exists an itemset $I' \in F^S$ such that $I' \subseteq I$; and (2) for each itemset $I \in F^S$, whether there exists a MFI itemset I' satisfies that $I \subseteq I'$.

Completeness Verification. First, the client verifies the correctness and completeness of returned MII itemsets by constructing MII from MFI, assuming she has verified that MFI is correct and complete. Next, the client uses the proof of MII itemsets to prove the completeness of returned frequent itemsets. Based on their relationships with the returned frequent itemsets F^S, we categorize the missing frequent itemsets into four types. In particular, given a frequent itemset I that is not returned by the server, it belongs to one of the following four types: (1) *type-1 (non-overlap):* for each itemset $I' \in F^S$, $I \cap I' = \emptyset$; (2) *type-2 (subset):* there exists a frequent itemset $I' \in F^S$ such that $I \subseteq I'$; (3) *type-3 (superset):* there exists a frequent itemset $I' \in F^S$ such that $I' \subseteq I$; and (4) *type-4 (overlap):* there is no frequent itemset $I' \in F^S$ such that $I \subseteq I'$ or $I' \subseteq I$. However, there exists a frequent itemset $I' \in F^S$ such that $I \cap I' \neq \emptyset$. Next, we describe how to verify these four types of missing frequent itemsets respectively.

Verification of type-1 missing itemsets. It is easy to infer that any type-1 missing itemset must only contain frequent 1-itemsets that do not appear in F^S. Therefore, the client simply checks whether there is any frequent 1-itemsets of D that is not included in F^S. If there is, the client concludes that there must exist at least one type-1 missing frequent itemset. The complexity of verification is $O(|F^S|)$. Proof-based verification is not needed. The frequent 1-itemsets are collected when the client constructs the inverted index.

Verification of type-2 missing itemsets. The client verifies whether for each MFI itemset, all subsets of its corresponding itemset are included in F^S. In this case, as long as I is frequent (verified by correctness verification), this verification procedure does not need to use any proof.

Verification of type-3 & 4 missing itemsets. We show that if there is any type-3/4 frequent itemset missing, the MII itemsets of F^S include at least one frequent itemset. Therefore, the client verifies whether there exists any type-3/4 missing frequent itemsets by verifying the infrequentness of each MII itemset via its proof. If there exists any MII itemset whose proof cannot show that it is infrequent, there exists at least one type-3/4 missing itemset. The complexity of verification is $O(N)$, where $N = \sum_{\forall I \in MII} |I|$.

We can prove that the optimized verification approach provides the same security guarantee as our basic approach. The details of proof are omitted due to limited space.

C. Complexity Analysis

Proof construction at server side. The total complexity of proof construction is $O(Nlog^3N + p^\epsilon logp)$, where $N = \sum_{I \in MFI \cup MII} |I|$, p is the number of unique items of D, and

$\epsilon \in (0,1)$ is a user-specified constant that is used to specify the number of levels of the Merkle tree (as $[1/\epsilon]$).

Verification at client side. The complexity of verification is $O(N)$, where $N = \sum_{\forall I \in MII \cup MFI} |I|$.

Upper bound of number of MFI and MII itemsets. The complexity of both proof construction and verification is linear in the number of MII and MFI itemsets. We next estimate the theoretical upperbound of total number of MII and MFI itemsets, as well as their total length. The upperbound of the number of MFI itemsets is $\binom{p}{[\frac{p}{2}]-1}$, while the upperbound of the number of MII itemsets is $\binom{p}{[\frac{p}{2}]}$, where p is the number of unique items of D. The upperbound of the total length of MFI and MII itemsets is $\binom{p}{[\frac{p}{2}]-1}([\frac{p}{2}]-1) + \binom{p}{[\frac{p}{2}]}[\frac{p}{2}]$.

VI. Extension

In this section, we explain how to adapt our verification approaches to *maximal frequent itemsets* (i.e., those itemsets that do not have any superset itemset that is frequent). We use MFI and MII that are defined in Section V. In particular, To prove a returned itemset $I \in F^S$ is correct, the proof must show that I is frequent and maximal (i.e., all supersets of I are infrequent). Frequentness can be verified by using the proof of I. To prove that all returned itemsets are maximal, the server constructs the proof for all MFI itemsets of F^S. The server does not need to prepare any other proof for completeness verification; the proofs for correctness verification will be used for completeness verification too.

The correctness verification at the client side is trivial. Regarding the completeness verification, we categorize the possible missing maximal frequent itemsets into two types. In particular, consider a maximal frequent itemset I that is not returned by the server, it should either overlap or not overlap with any returned frequent itemset. The verification of missing non-overlapped maximal frequent itemsets is similar to the verification of type-1 missing itemsets for frequent itemset mining. The verification of missing overlapping maximal frequent itemsets is consumed by the correctness verification via using the proofs of MFI and MII. We can prove that the our verification approach can catch any incorrect/incomplete maximal frequent itemset mining results. The details of proof are omitted due to limited space.

VII. Experiments

We ran a battery of experiments to measure the performance of proof construction at the server side and verification at the client side and explored various factors that impact the verification performance, including various error ratio, frequent itemsets of different lengths, and different database sizes. In this section, we discuss our experimental results.

A. Setup

Hardware. We run our experiment on an Intel machine with 2.4GHz CPU, 4GB memory, running Mac OS X 10.7.5.

Datasets. We use the IBM generator to generate four synthetic datasets S_1, S_2, S_3, and S_4 of various sizes. We also use the real-world *Retail* dataset available at the Frequent Itemset Mining Dataset Repository[1]. The *Retail* dataset R_1 contains 88162 transactions and 16470 items. We also construct a small dataset R_2 from the *Retail* dataset that contains 500 transactions and 100 items. Both Retail datasets are much sparser than the synthetic datasets. Table I shows the details of the datasets and our mining setup.

Dataset	# of trans.	# of items	Avg. trans. length	min_{sup}	# of freq. itemsets
S_1	10^3	49	10	250	36
S_2	10^4	49	10	250	3854
S_3	10^5	49	10	250	149744
S_4	10^6	49	10	250	3074610
R_1	88162	16470	124	50	16778
R_2	500	100	2.4	5	97

TABLE I
Details of Datasets

Simulation of malicious actions. We set the error ratio $r = 1\%, 2\%, 5\%, 10\%$, and 20%. For the simulation of incomplete result, we randomly pick r percent of frequent itemsets from the mining results and remove these picked itemsets. For the simulation of incorrect result, we randomly generate r percent of infrequent itemsets and insert them into the result.

B. Implementation

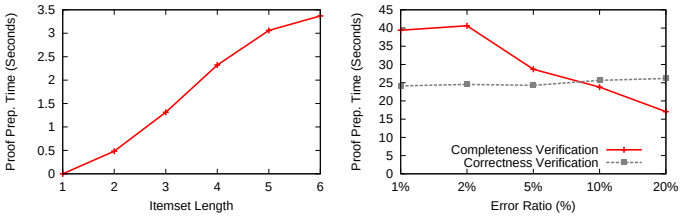
We implement the code in C++. We use the PBC library[2] to implement bilinear pairing, and SHA256 for the implementation of the hash function in Merkle tree construction. The accumulation values in Merkle tree are computed by using the element power function in the PBC library. The coefficients $\mathcal{W}$ in the proof are computed using the NTL library[3]. To improve the performance at the server side, we use multi-threading programming to allow parallel proof construction and client preparation. We implemented the Apriori algorithm [1] for frequent itemset mining.

C. Proof Construction at the Server Side

Time performance. First, we measure the time of preparing one single proof of (in)frequent itemsets of various lengths. As shown in Figure 1 (a), the proof preparation time increases with the length of frequent itemset. This is consistent with our theoretical analysis that the complexity of proof construction is dependent on the size of mining results.

Second, we measure the total time of proof preparation. Figure 1 (b) shows the total preparation time for the mining results of various error ratios. We consider the error ratio $r = 1\%, 2\%, 5\%, 10\%$, and 20%. We observe that the correctness proof preparation time increases slowly with the growth of the error ratio. By further analysis we found that adding infrequent itemsets to the mining results leads to significant change of number of MFI itemsets. For example, when the error ratio changes from 10% to 20%, the number of MFI itemsets changes from 32 to 40 (25% increase). However, the total proof construction time does not increase much since the

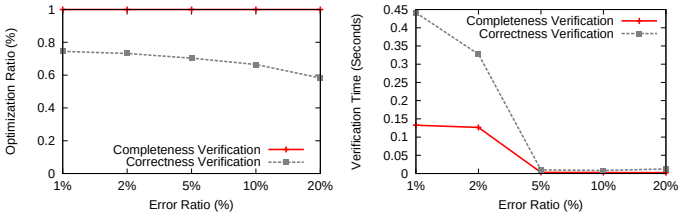
[1]Frequent Itemset Mining Dataset Repository: http://fimi.ua.ac.be/data/.
[2]PBC library: http://crypto.stanford.edu/pbc/manual/.
[3]NTL library: http://www.shoup.net/ntl/doc/tour-intro.html.



(a) Construction time of one single proof

(b) Total proof preparation time

Fig. 1. Proof Construction Time (R_2 dataset)



(a) Proof optimization ratio

(b) Client verification time

Fig. 2. Proof Optimization Ratio & Verification Time (R_2 dataset)

proofs of new MFI itemsets can be constructed very fast. Figure 1 (b) also shows that opposite to the correctness proof preparation, the completeness proof preparation time decreases when the error ratio increases. To find the reason we studied how MII itemsets change when the error ratio grows. We found that the MII itemsets include fewer items when more frequent itemsets are missing. For instance, when the error ratio changes from 2% to 5%, the number of completeness proofs changes from 33 to 27. This explains why the proof construction time decreases indeed.

Proof size. First, we measure the optimization ratio of number of proofs for various error ratios. We define the optimization ratio as $1 - k'/k$, where k and k' are the number of proofs by our basic approach and by our optimization respectively. Intuitively, the closer the optimization to 1, the better. Figure 2 (a) displays the optimization ratio. We observe that the optimization ratio of the number of completeness proofs is very close to 1. In particular, the basic verification approach requires $(2^{|\mathcal{I}|} - |F^S|)$ proofs (for this experiment $|\mathcal{I}| = 100$ and $|F^S| = 97$) for completeness verification, while our optimization approach only requires no more than 34 completeness proofs. This proves that our optimization can dramatically reduce the number of proofs. We also observe that the optimization ratio of the number of correctness proofs deceases when the error ratio increases. The reason is that adding infrequent itemsets into the result leads to more MFI itemsets and thus the number of proofs, which decreases the optimization ratio. Nevertheless, the optimization ratio of the number of correctness proofs is still high; it is at least 0.6 even when the error ratio is 20%.

We also measure the total size of the proofs for various error ratios. The result shows that the trend of the total size of the proofs with regard to various error ratios is consistent with the trend of number of proofs (Figure 2 (a)). In particular, the total size of completeness proofs decreases dramatically when the error ratio grows, while the total number of the correctness proofs increase with the growth of the error ratio. In most of the cases, the total size of proofs does not exceed $16KB$. We omit the results due to limited space.

D. Verification Time at the Client Side

First, we measure the verification time at the client side for various error ratios, and show the result in Figure 2 (b). First, we observe that both completeness and correctness verification are very fast. It never exceeds 0.45 seconds. Second, we observe that the completeness verification time decreases dramatically when the error ratio increases from 2% to 5%, then keeps stable afterward. We analyzed why this happens. It turned out that when the ratio equals to 2% (1% as well), the incomplete itemsets were caught by checking against the proofs, while for the error ratio changes to 5% and more, the incomplete itemsets were caught by missing at least one proper subset or one frequent 1-itemset, which is much faster than using proofs. Third, we observe that the correctness verification time drops sharply when the error ratio changes from 1% to 5%. The reason is that higher error ratio leads to higher chance that the client catches an infrequent itemset at earlier trials. When the error ratio increases from 5% to 20%, the verification time is stable because now the client can catch the infrequent itemset by the first trial.

E. Client VS. Server

We compared the time performance at both the client and the server sides for various support threshold values. We vary the support threshold values so that the number of frequent itemsets is approximately 1%, 5%, and 10% of the number of transactions. Table II shows the comparison result. First, it is not surprising that the verification time at the client side increases with the growth of the number of frequent itemsets. The same pattern also holds for the server side. Second, in all of our testings, the total overhead at the client side is much smaller than that at the server side. Furthermore, the verification procedure at the client side is much faster than the mining at the server side. This proves that the client can outsource the mining task to the server, while verifying the result integrity with the cost that is cheaper than mining locally.

min_{sup}	# of Freq. Itemsets	Client side	Server side	
		Verify	Proof prep.	mining
402	10	0.000164	24.72	0.03707
203	50	0.001358	266.985	0.08984
157	99	0.00332	572.591	0.1355

TABLE II
CLIENT VS. SERVER W.R.T TIME PERFORMANCE (S_1 DATASET)

F. Deterministic VS. Probabilistic Approaches

We implemented a verification approach that returns probabilistic verification guarantee for outsourced frequent itemset mining [5], and compared its performance with our deterministic approach. Table III shows the comparison result on S_3 dataset of various settings. We pick the error ratios of 1%,

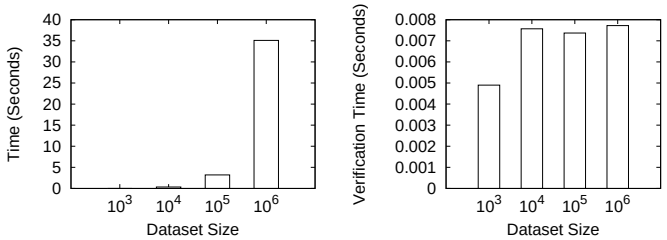
and vary the probabilistic guarantee threshold from 90% to 100% (probability = 100% corresponds to our deterministic approach). Table III shows the details of the comparison result. In general, the deterministic approach brings higher overhead at the server side than the probabilistic approach. However, this is the sacrifice that we have to pay for higher result integrity guarantee. We also observe that in some cases (marked as N/A in Table III), the probabilistic approach fails as it cannot provide required probabilistic correctness guarantee due to the data distribution. The deterministic approach does not have such limit.

Error ratio	Integrity Prob.	Type	Client	Server	
			Verify	Proof prep.	Mining
1%	90%	R	N/A	0	0.042
		M	1.433	0	1024.53
1%	95%	R	N/A	0	0.042
		M	0.945	0	1204.59
1%	99%	R	N/A	0	0.042
		M	0.689	0	1498.67
1%	100%	R	0.000628	1660.12	0.5707
		M	0.4123	2785.6	0.5707

TABLE III
TIME PERFORMANCE (IN SECONDS) OF DETERMINISTIC VS PROBABILISTIC APPROACHES (S_3 DATASET; R: CORRECTNESS, M: COMPLETENESS)

G. Scalability

To measure the scalability of our verification approaches, we measured the time performance on the datasets of various sizes. Figure 3 (a) shows that the time of constructing one single proof increases with the data size. However, the proof can be constructed fast; it only needs 35 seconds even for a dataset of 10^6 records. Figure 3 (b) shows that the verification time does not change much with the data size, since: (1) the verification complexity is decided by the number of MII and MFI itemsets, and (2) the verification procedure always catches the first incomplete itemset by type-1/2 checking for most of the cases. Similar observations hold on the *Retail* dataset (88162 transactions); it only requires 0.001045 seconds for verification. This convinces us that our approach can be used for efficient verification of outsourced frequent itemset mining.



(a) Construction time of one proof (itemset length = 3)

(b) Client verification time

Fig. 3. Scalability (error ratio=1%)

VIII. CONCLUSION

In this paper, we presented an efficient result integrity verification approach that can provide deterministic guarantee for outsourced frequent itemset mining. The key idea of the approach is to construct cryptographic proofs of all (in)frequent itemsets. We discussed how to optimize the number of proofs to improve the performance. We also extended our study to the verification of maximal frequent itemset mining.

Our solution mainly focuses on correctness and completeness of the mining results. An interesting extension to our method would be the verification of *freshness*, i.e., whether the returned mining results are from the latest version of outsourced data, assuming that the client updates her dataset from time to time.

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.

[2] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, 2011.

[3] Ran Canetti, Ben Riva, and Guy N. Rothblum. Practical delegation of computation using multiple servers. In *CCS*, 2011.

[4] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, 2010.

[5] Boxiang Dong, Ruilin Liu, and Wendy Hui Wang. Result Integrity Verification of Outsourced Frequent Itemset Mining. In *DBSec*, 2013.

[6] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD*, 2002.

[7] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO*, 2010.

[8] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[9] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In *SIGMOD*, 2006.

[10] Ruilin Liu, Hui Wang, Anna Monreale, Dino Pedreschi, Fosca Giannotti, and WengeGuo. Audio: An integrity auditing framework of outlier-mining-as-a-service systems. In *ECML/PKDD*, 2012.

[11] Menezes, Alfred and Vanstone, Scott and Okamoto, Tatsuaki. Reducing elliptic curve logarithms to logarithms in a finite field. In *STOC*, 1991.

[12] R.C.Merkle. Protocols for public key cryptosystems. In *Symposium on Security and Privacy*, 1980.

[13] Ian Molloy, Ninghui Li, and Tiancheng Li. On the (in)security and (im)practicality of outsourcing precise association rule mining. In *ICDM*, 2009.

[14] Jyothsna R. Nayak and Diane J. Cook. Approximate association rule mining. In *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, 2001.

[15] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In *CRYPTO*, 2011.

[16] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: verifiable computation from attribute-based encryption. In *TCC*, 2012.

[17] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. Verifying completeness of relational query results in data publishing. In *SIGMOD*, 2005.

[18] Chih-Hua Tai, Philip S. Yu, and Ming-Syan Chen. k-support anonymity based on pseudo taxonomy for outsourcing of frequent itemset mining. In *SIGKDD*, 2010.

[19] W. K. Wong, David W. Cheung, Ben Kao, Edward Hung, and Nikos Mamoulis. An audit environment for outsourcing of frequent itemset mining. In *PVLDB*, 2:1162–1172, 2009.

[20] Min Xie, Haixun Wang, Jian Yin, and Xiaofeng Meng. Integrity auditing of outsourced data. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2007.

[21] D. Burdick, M. Calimlim, J. Gehrke, MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2001.