is accurate. Consider that one user has an affirmative report, there is only one case that makes the step false: this user's bit string equals the bitwise XOR of all other users' strings. Since this user randomly samples its string from 2^q different choices, the probability for the bitwise XOR result being not accurate is $1 - 1/2^q$. Thus, we have:

$$\begin{aligned} & P(\text{the report and determination is accurate}) \\ = \; & P(\text{all users have negative reports}) \times 1 + \\ & P(\text{any user has an affirmative report}) * (1 - 1/2^q) \\ \geq \; & 1 - 1/2^q \end{aligned}$$

As our Min computation protocol needs to perform "report and determine" for l times, it is not difficult to prove the probability that l times are all correct is no less than $1 - l/2^q$ given q is greater than l. Due to the page limit, we skip this part of the proof here. □

Our Min computation protocol has very good computational efficiency: Once the keys have been established, on average each user mainly needs $2l$ bitwise XOR operations, $2l$ hashing operations, $3l$ comparison operations, and picking no more than ql random bits during each time period; the aggregator mainly needs $(n-1)l$ bitwise XOR operations and l comparison operations during each time period. In terms of communication efficiency, our protocol requires transmissions of $(nq+1)l$ bits on average during each time period.

In terms of security, we have the following theorem.

Theorem 3. *Our Min computation protocol is perfectly privacy-preserving against the aggregator and all users.*

Proof: In our protocol, every user only receives l bits from the aggregator, and these l bits are the same as the l bits in the bit representation of our Min protocol's output. Therefore, our protocol reveals no extra knowledge to the users.

We prove that our protocol reveals no extra knowledge to the aggregator by constructing a probabilistic polynomial-time simulator $\mathcal{S}$ that takes the output of our Min computation protocol, and simulates the aggregator's view during the execution of the protocol as follows.

In our protocol, the aggregator's view consists of the ciphertexts it receives from n users. Denote its view by $V = (V_1, V_2, \ldots, V_l)$ where $V_j = (\overline{d_j^1}, \ldots, \overline{d_j^n})$ $(j = 1, \ldots, l)$ is the ciphertexts of all users' bit strings generated to determine the j-th bit of the Min.

On input of $d^\star$ $(d^\star = \Sigma_{j=1}^{l} d_j^\star \times 2^{l-j})$, the output of our Min computation protocol, $\mathcal{S}$ simulates each V_j for determining $d_j^\star$ using $V_j' = (v_j^1, \ldots, v_j^n)$ as follows.

If $d_j^\star = 0$, $\mathcal{S}$ generates $v_j^1, \ldots, v_j^{n-1}$ independently as random q-bit strings uniformly distributed on $\{0,1\}^q$ and sets v_n to $v_j^1 \oplus v_j^2 \oplus \ldots, \oplus v_j^{n-1}$. If $d_j^\star = 1$, $\mathcal{S}$ generates $v_j^1, \ldots, v_j^n$ independently as random q-bit strings uniformly distributed on $\{0,1\}^q$. Below we prove the computational indistinguishability between V and $V' = (V_1', \ldots, V_l')$.

Due to the pseudo-randomness of the pseudo-random functions used in the cipher's key generation, it is easy to verify that

$$\overline{d_j^i} \overset{c}{\equiv} U_j^i \tag{2}$$

for every i, j and $\overline{d_j^1}, \overline{d_j^2}, \ldots, \overline{d_j^{n-1}}$ are independent variables, where U_j^i $(i = 1, \ldots, n; j = 1, \ldots, l)$ are random variables that are uniformly distributed over $\{0,1\}^l$.

Since the cipher is homomorphic on the bitwise XOR computation, we know

$$(\overline{d_j^1}, \ldots, \overline{d_j^{n-1}}, \overline{d_j^n}) \overset{c}{\equiv} (U_j^1, \ldots, U_j^{n-1}, D_j \oplus U_j[1:n-1]) \tag{3}$$

,where $D_j = d_j^1 \oplus \ldots \oplus d_j^n$ and $U_j[1:n-1] = U_j^1 \oplus, \ldots, \oplus U_j^{n-1}$.

When $d_j^\star = 0$, we have $D_j = 0^l$ according to the coding scheme, and $V_j' = (U_j^1, \ldots, U_j^{n-1}, D_j \oplus U_j[1:n-1])$. When $d_j^\star = 1$, we know $D_j \overset{c}{\equiv} U_j^n$, thus $(U_j^1, \ldots, U_j^{n-1}, D_j \oplus U_j[1:n-1]) \overset{c}{\equiv} (U_j^1, \ldots, U_j^{n-1}, U_j^n)$. In both cases,

$$V_j \overset{c}{\equiv} V_j'. \tag{4}$$

Moreover, due to the pseudo-randomness of the keys, we know $V_1, \ldots, V_l$ are independent. Thus, we know

$$V \overset{c}{\equiv} V'. \tag{5}$$

□

Remark: as we have explained in Section 3.1, here we consider all of the protocol's participants including the aggregator and the users, are entitled to know the output of the protocol. In cases where users are not entitled to know the output, we can conclude our Min protocol is weakly privacy-preserving against all users in the sense that it reveals the value of the Min to all users. (Note that this would allow each user to know whether its input is the minimum.) The proof would be almost the same as above. Only this time, we let $\mathcal{K}_i$ (i.e. knowledge that the protocol reveals to user i) equal the Min and use it as the input of the simulator constructed to simulate user i's view.

5 Privacy-Preserving k-th Min Computation

In this section, we work on the secure computation of the k-th Min in the aggregation of all users' data. First, we present a secure "pseudo-counting" protocol and analyze its accuracy. Then we show how to build our privacy-preserving k-th Min protocol based on it.

5.1 A secure pseudo-counting protocol and its accuracy

Below, we propose a secure pseudo-counting protocol, which will be used in constructing our k-th Min protocol. (By "pseudo-counting", we mean the protocol intends to count, but the result can have an error with a certain probability.) Suppose there are n users. Consider a question, for which each user has two possible answers: "affirmative" or "negative". Our protocol enables the aggregator to compute the total number of affirmative answers without knowing any specific user's reply.

First, the protocol uses a probabilistic coding scheme that is different from the one used in the secure Min computation protocol to encode users' replies. In particular, each user's reply is encoded by a q-bit string. If its reply is affirmative, the user randomly chooses a bit location x $(1 \leq x \leq q)$ and sets the x-th bit of the string to 1 and all other bits to 0; If its reply is negative, the user sets the bit string to 0^q. Denote by $C'(r)$ the code of a user's reply r in the coding scheme used here. We have

$$C'(r) = \begin{cases} 0^q & \text{if } r = \text{"negative"}, \\ \overbrace{0\ldots0}^{x-1}1\overbrace{0\ldots0}^{q-x} & \text{if } r = \text{"affirmative"}, \end{cases} \quad (6)$$

where $q \in \mathbb{N}$ is the accuracy controlling parameter and x is a random variable that is uniformly distributed over $\{1, \ldots, q\}$.

Then, using the cipher system in Section 3.2, users encrypt their bit strings and send the ciphertexts to the aggregator. Given all users' ciphertexts, the aggregator can compute the bitwise XOR of all users' bit strings. The aggregator counts the number of 1s in the result, and outputs it as the total number of affirmative answers.

Notice that the above process does not always output the correct number of users with affirmative answers. Consider an optimistic case in which all users with affirmative answers have chosen different bit locations. Then it is easy to see that the number of 1s in the bitwise XOR of all users' bit strings is the same as that of the affirmative users. In practice, different users may have collisions, i.e. choosing the same bit location. This would cause the output of our counting protocol to be smaller than the actual number. Therefore, the probability of our counting protocol being accurate equals the probability that the optimistic case happens in a counting process or the "non-collision probability".

Proposition 4. *The probability of our pseudo-counting protocol being accurate in a counting process of n users is greater than or equal to $\frac{q(q-1)\ldots(q-n+1)}{q^n}$.*

Proof: Denote by P_{opt} the probability of our protocol being accurate. P_{opt}'s lower bound can be computed by considering an extreme case where n users' answers are all "affirmative". (The more users with affirmative answers there are, the smaller the non-collision probability is.)

It is easy to see the non-collision probability equals $q(q-1)\ldots(q-n+1)/q^n$ when all n users have affirmative answers. Let us imagine the n users with affirmative answers choose bit location one after another. Each user has q possible choices, and thus there are q^n different outcomes for n users' choices. However, to avoid collisions with the previous users, the second user has only $q-1$ choices, the third user has only $q-2$ choices, ... , the last user has only $q-n+1$ choices, which means there are only $q(q-1)\ldots(q-n+1)$ possible non-collision outcomes.Thus

$$P_{opt} \geq \frac{q(q-1)\ldots(q-n+1)}{q^n} \quad (7)$$

□

Proposition 5. *The probability of our pseudo-counting protocol being accurate in a counting process of n users is around $\frac{q(q-1)\ldots(q-\lceil n/2\rceil+1)}{q^{\lceil n/2\rceil}}$ in the average case.*

Proof: In an average case, users' answers are uniformly random. Thus, the average total number of users with affirmative answers is around $\lceil n/2\rceil$. Denote by P_{opt}^{avg} the probability of our protocol being accurate in the average case. Similar to the analysis in Proposition 4's proof, we know:

$$P_{opt}^{avg} \approx \frac{q(q-1)\ldots(q-\lceil n/2\rceil+1)}{q^{\lceil n/2\rceil}} \quad (8)$$

□

To improve the accuracy, we have two possible approaches: 1) We can use a larger value for q, so that there is a lower probability for two users' chosen bits to collide. 2) We can repeat the counting process for a number of times (denote by p the number of times the counting process is repeated). Then, we can choose the largest output among all repetitions as the final output.

Specifically, the probability for an optimistic case to happen in p times of counting process can be computed as:

$$\begin{aligned} P'_{opt}(p) &= 1-(1-P_{opt})^p \\ &\geq 1-(1-\tfrac{q(q-1)\ldots(q-n+1)}{q^n})^p \end{aligned} \quad (9)$$

$$\begin{aligned} P_{opt}^{avg\prime}(p) &= 1-(1-P_{opt}^{avg})^p \\ &\approx 1-(1-\tfrac{q(q-1)\ldots(q-\lceil n/2\rceil+1)}{q^{\lceil n/2\rceil}})^p \end{aligned} \quad (10)$$

Table 1 lists the theoretical estimations of probability for a non-collision case to happen, computed using Formula 10 with different q and different p in a 100-user scenario in average case.

We formally present our protocol in Algorithm 2.

Table 1
Theoretical estimation probability for a non-collision case to happen in a 100-user scenario in average case.

q	1000	2000	3000	4000	5000
$p=1$	28.77%	53.92%	66.33%	73.53%	78.21%
$p=2$	49.27%	78.77%	88.66%	92.99%	95.25%
$p=3$	63.86%	90.22%	96.18%	98.14%	98.96%
$p=4$	74.26%	95.49%	98.71%	99.51%	99.77%
$p=5$	81.67%	97.92%	99.57%	99.87%	99.95%
$p=6$	86.94%	99.04%	99.85%	99.97%	99.99%

Algorithm 2 The Secure Pseudo-counting Protocol

Require:
A group of n users;
An aggregator;
User i ($i=1,\ldots,n$) has two secret seeds S_a^i and S_b^i;
In each time period, user i ($i=1,\ldots,n$)'s reply is either affirmative or negative;
In each time period, a public known nonce number $t \in [0, 2^f-1]$;
Two precision controlling numbers $p, q \in \mathbb{N}$.

Ensure:
The aggregator outputs $tot \in \mathbb{N}$, the approximate total number of affirmative users;

the aggregator sets tot to 0;
for $j=1,\ldots,p$ **do**
  user i ($i=1,\ldots,n$) computes $k^i = h_{S_a^i, f+\lceil \log_2 p \rceil, q}(t|j) \oplus h_{S_b^i, f+\lceil \log_2 p \rceil, q}(t|j)$;
  user i ($i=1,\ldots,n$) encodes its answer by a q-bit string d^i according to Formula 6. If user i's status is negative, d^i equals $\{0\}^q$; otherwise, the user randomly choose a bit in d^i, sets the chosen bit to 1 and other bits to 0.
  user i ($i=1,\ldots,n$) computes $\overline{d^i} = d^i \oplus k^i$, and sends $\overline{d^i}$ to the aggregator.
  the aggregator computes the bitwise XOR of all $\overline{d^i}$ and finds out d, the total number of 1s in $\overline{d^i}$.
  the aggregator sets tot to Max$\{tot, d\}$.
end for
return tot as approximate total number of affirmative users;

5.2 Privacy-preserving k-th Min computation protocol

With a pseudo counting protocol, we construct our privacy-preserving k-th Min computation protocol as follows.

Our protocol determines every bit of the k-th minimum value from the MSB to the LSB. The process follows a similar "report-determine-broadcast-compare" pattern as our Min computation protocol.

In particular, the aggregator and all users first run the secure pseudo-counting protocol to find out how many users have a number whose MSB is 0. Denote by tot the total number of affirmative users. If tot is greater than or equal to k, the aggregator knows the k-th minimum value is also the k-th minimum value in all affirmative users' numbers and its MSB is 0. Otherwise, the aggregator knows the k-th minimum value is also the $(k-tot)$-th minimum value in all negative users' numbers and its MSB is 1. To determine the second-MSB, the aggregator only needs to repeat the above counting process among the users who have the same MSB as the k-th minimum value's MSB. To implement this selective counting, the aggregator broadcasts the k-th minimum value's MSB to all users, and the users whose MSB is different would always report negative in the remaining counting process. Similarly, the aggregator can determine the third, the fourth, ..., the LSB of the k-th minimum value.

We summarize our protocol in Algorithm 3.

Algorithm 3 Privacy-preserving k-th Min Computation Protocol

Require:
A group of n users;
An aggregator;
User i ($i=1,\ldots,n$) has two secret seeds S_a^i and S_b^i;
In each time period, user i ($i=1,\ldots,n$) has a number $d^i \in [0, 2^l-1]$;
In each time period, a public known nonce number $t \in [0, 2^l-1]$;
A preset number $K \in \mathbb{N}$.

Ensure:
The aggregator outputs d', the k-th minimum number in $\{d^i\}_{i=1,\ldots,n}$;

the aggregator sets f to 0.
every user sets its status as "effective".
for $j=1,\ldots,l$ **do**
  user i ($i=1,\ldots,n$) sets its reply according to its status and its j-th MSB: If its status is "effective" and the bit equals 0, it sets its reply to affirmative; otherwise, it sets its reply to negative.
  user i ($i=1,\ldots,n$) helps the aggregator to compute tot, the total number of affirmative users, by participating in the secure pseudo-counting protocol;
  If $f+tot<k$, the aggregator updates f's value to $f+tot$, sets the d''s j-th MSB to 1, and broadcasts 1 to every user; otherwise, the aggregator sets the d''s j-th MSB to 0 and broadcasts 0.
  If an "effective" user finds the bit received different from its number's j-th MSB, it sets its status to "ineffective".
end for
return the k-th minimum number d' as $\Sigma_{j=1}^{l} d'_j \times 2^{l-j}$;

5.3 Security Analysis

Here we analyze the security of our k-th Min computation protocol. Define $pcd(k, d^1, \ldots, d^n)$ a function that takes k and n users' numbers as inputs, outputs $\{pc_j\}_{j \in \{1,\ldots,l\}}$ where pc_j equals the count of users whose number has the same j-bit binary prefix as the k-th Min. For example, suppose $l = 3$ and the k-th Min is 100, then pc_1, pc_2, pc_3 equals the count of users whose number is of form $1 * *$, $1\ 0\ *$ and $1\ 0\ 0$ respectively. Since $\{pc_j\}_{j=\{1,\ldots,l\}}$ is actually a distribution of all users' number compared with the k-th Min's prefix, we call $pcd(k, d^1, \ldots, d^n)$'s output the k-th Min's *"prefix-count-distribution"*.

Theorem 6. *Our k-th Min computation protocol is perfectly privacy-preserving against all users and weakly privacy-preserving in the sense that all users learn no more knowledge from it than k-th Min of all users' data, and the aggregator learns no more knowledge than the output of our k-th Min protocol as well as the its prefix-count-distribution.*

Proof: Similar as our Min computation protocol, it is straightforward to see our protocol reveals every bit of the protocol's output to each user only, thus our protocol reveals no extra knowledge to the users.

It is not difficult to see the $\{pc_j\}$ equals the count of effective users in round j of step 3. Using this information, the aggregator could easily simulate all users' ciphertexts in the j-th round by first generating a random q-bit string s that has pc_j bits of 1 and $q - pc_j$ bits of 0, and then generating n random q-bit strings the bitwise XOR of which is s.

Same as our proof of Theorem 3, the computational indistinguishability here follows from the pseudo-randomness of the pseudo-random functions that we use to generate keys and the cipher's homomorphic property on bitwise exclusively-OR computations. □

Remark: Please note that although our k-th Min protocol reveals the k-th Min's prefix-count-distribution, it does not cause significant violation to individual user's privacy since it is a piece of statistic information about all users' numbers. In addition, in cases that users are not entitled to know the k-th Min, we can conclude our k-th Min protocol is weakly privacy-preserving against the users in the sense that it reveals the k-th Min to all users.

5.4 Optimal Choice of (q, p)

Recall that two parameters q (the bit string's length) and p (the number of times of repeating) control our pseudo-counting protocol's accuracy, and thus the k-th Min protocol's accuracy. To guarantee the protocol's accuracy is no less than a particular threshold, we can simply choose a large q and a large p. But since large p or large q would decrease the protocol's efficiency, we are interested in finding the optimal (q, p) pair. In this section, we analyze how to find the optimal (q, p) pair for the pseudo-counting protocol.

In the pseudo-counting protocol, the aggregator needs to receive all users' bit strings and performs bitwise XOR computations on them. Compared with the workload of a user, the aggregator's workload is generally much heavier especially when the total number of users is large. Therefore, here we study how to optimize the workload of the aggregator to improve the protocol's efficiency. In particular, we study how to find the optimal (q, p) pair that minimizes the total bits sent to the aggregator, and guarantees the accuracy is no less than the threshold.

Let $b^\star$ be the threshold and let a and b be the non-collision probability lower bounds for running the pseudo-counting protocol once and p times respectively in a n-user counting process. According to formula (7) and (9), we have:

$$a = \frac{q(q-1)\ldots(q-n+1)}{q^n} \tag{11}$$

$$b = 1 - (1-a)^p. \tag{12}$$

Let $Z(q, p)$ be the total bits sent to the aggregator when every user's answer is a q-bit string and the pseudo-counting protocol is repeated for p times. In every time, the aggregator receives n q-bit strings from all users. Therefore,

$$Z(q, p) = nqp, \tag{13}$$

and our goal is to compute $(q^\star, p^\star)$, the solution of the following problem:

$$\begin{aligned} &\textbf{argmin: } Z(q,p) \\ &\textbf{subject to: } b \geq b^\star \end{aligned} \tag{14}$$

According to equation (12), Z can be represented as a function of q as:

$$\begin{aligned} Z(q,p) &= n \ln \tfrac{1}{1-b} \times \tfrac{q}{\ln \frac{1}{1-a}} \\ &= n \ln \tfrac{1}{1-b} \times \tfrac{q}{n \ln q - \ln[q^n - q(q-1)\ldots(q-n+1)]} \end{aligned} \tag{15}$$

The optimal $q^\star$ that minimizes Z could only be an integer near the solution of equation: $\frac{dZ}{dq} = 0$, which is equivalent to

$$\frac{d\left(\frac{q}{n \ln q - \ln[q^n - q(q-1)\ldots(q-n+1)]}\right)}{dq} = 0. \tag{16}$$

Consequently, we have proved the following proposition.

Proposition 7. *Suppose that $\bar{q}$ is the solution to equation (16). Then either $q^\star = \lfloor \bar{q} \rfloor$ or $q^\star = \lceil \bar{q} \rceil$.*

Although equation (16) specifies the optimal value of q, the equation is so complicated that getting an analytic solution would be hard. In practice, we could use some numerical methods to find the optimal (q, p) pair. For example, we can enumerate the possible candidates of p and q, and then use the pair that has the smallest product of p and q in all enumerated pairs as the final $(q^\star, p^\star)$. (Enumeration of possible values is feasible because p and q must be integers located in a certain range.)

Specifically, given any (q, p) pair, we can compute b, the corresponding lower bound of the non-collision probability, using formula (11) and (12). It is easy to see b is an increasing function of p, and also an increasing function of q. Therefore, for any q, there is a minimal p that makes b no less than the threshold $b^\star$. Denote by p_{min} this minimal p and it can be computed as:

$$p_{min} = \max\{\lceil \ln(1-b^\star)/\ln(1-a) \rceil, 1\} \qquad (17)$$

Apparently, for any q, p_{min} computed as above is the optimal value of p that minimizes the product of p and q and satisfies the probability threshold requirement. By enumerating every possible q and its corresponding p_{min}, we traverse every possible candidate of the optimal (q, p) pair.

Note that the enumeration does not need to traverse all possible values of q. Since a is an increasing function of q, we know p_{min} is a non-increasing function of q. Also, we can prove there is q_{max} that is finite, and its corresponding p_{min} is 1. Due to the non-increasing property of p_{min}, the corresponding p_{min} of any q that is greater than q_{max} is 1. (Recall $q \geq 1$.) Therefore, q_{max} and its corresponding p_{min} dominate any q and this q's corresponding p_{min}, when q is greater than q_{max}. The enumeration can be terminated when q reaches q_{max}.

Below, we provide a possible value of q_{max} to prove its existence:

$$q_{max} = \lceil \frac{n-1}{1-(b^\star)^{\frac{1}{n}}} \rceil. \qquad (18)$$

It is easy to verify the corresponding b for $(q_{max}, 1)$ is no less than $b^\star$.

6 Performance Evaluation

In this section, we conduct theoretical analysis and perform experiments to evaluate the accuracy and efficiency of our protocols. In the aspect of accuracy, we consider two metrics—the probabilities of our protocols being accurate, and the relative errors.

In all our experiments, our protocols are implemented using Microsoft Visual Studio 2012 and the Crypto++ library [25]. The experiments are performed on a laptop running the 64-bit Windows 7 Professional operating system with Intel Core i7 3520M CPU and 8 GB memory. Results shown in Figure 2 and Figure 5 are the average of 10000 runs. Other experimental results shown are the average of 1000 runs.

The hash function we use in our Min protocol is HMAC<SHA-1> which can produce a 160-bit hash value as 160-bit is enough in our Min protocol. And HMAC<SHA-512> is used in our secure counting protocol and k-th Min protocol. In particular, HMAC<SHA-512> generates a 512-bit output. In case we need a shorter output of length q, we truncate the output to short bit strings of length q, and then use the bitwise XOR of all these strings as the final output. In case q is greater than 512, we first break the input message into several 512 bit strings and one string of length q' ($q' < 512$), then generate several 512-bit output strings and one q'-bit output string, and finally use the concatenation of these output strings as the final output.

We use the privacy-preserving Min computation protocol from [8] as the baseline and compare our protocols with it. Note that the protocol in [8] is designed for computing the Min. However, it can also be used to find the k-th Min, since the aggregator knows the entire distribution of all users' data from it.

6.1 Communication Cost of Our Protocols

Table 2 shows the total bits sent and received by the aggregator and a user in three protocols. Also, the round complexity of three procotols is shown in it.

Table 2
Communication Cost of Our Protocols (the range of data is $[0, M-1]$, n is the total number of users)

	bits sent/received by the aggregator	bits sent/received by a user	round complexity
Baseline	$0/nM\lceil\log_2 n\rceil$	$M\lceil\log_2 n\rceil/0$	1
Min Protocol	$n\lceil\log_2 M\rceil/nq\lceil\log_2 M\rceil$	$q\lceil\log_2 M\rceil/\lceil\log_2 M\rceil$	$\lceil\log_2 M\rceil$
k-th Min Protocol	$pn\lceil\log_2 M\rceil/pnq\lceil\log_2 M\rceil$	$pq\lceil\log_2 M\rceil/p\lceil\log_2 M\rceil$	$p\lceil\log_2 M\rceil$

According to Proposition 2, $l/2^q$ is the upper bound of our Min protocol's error rate ($l = \lceil\log_2 M\rceil$). It is easy to see that when l is not too small, for example greater than 20, by simply setting $q = l$ our Min protocol would achieve a very good error rate bound (less than 0.0001%). Our Min protocol requires much less data to send/receive compared with the baseline protocol, as long as n is not too much greater than M (i.e. $n = O(M)$).

Notice that the accuracy bound of the pseudo-counting protocol on which our k-th Min protocol is based is determined by p, q and n. Compared with the baseline protocol, our k-th Min protocol requires less data to send and receive, typically when the system has a large M and a relatively small n. For example, when $M = 2^{20}-1$ and $n = 100$, experiments show our k-th Min protocol achieves an accuracy that is above 99.5% by setting $p = 10$ and $q = 1400$. (Please see Figure 4 in the next Section for more details.) It is easy to verify our k-th Min protocol requires much fewer bits to be sent and received by both the aggregator and the users in this setting.

Although the amount of data to be sent and received in either protocol of ours is quite less than that in baseline, it is worth noting that the baseline

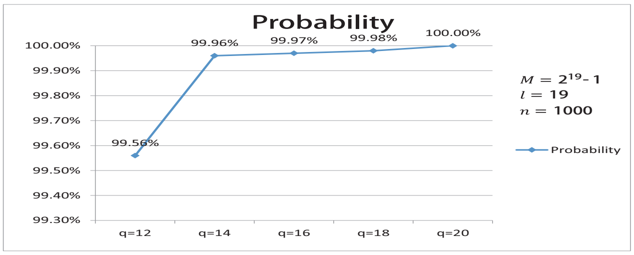

Figure 2. Probability of Min Protocol Being Accurate

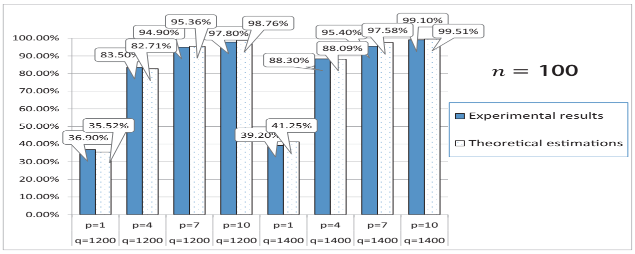

Figure 3. Probability of Pseudo Counting Protocol Being Accurate (Also Called Non-Collision Probability)

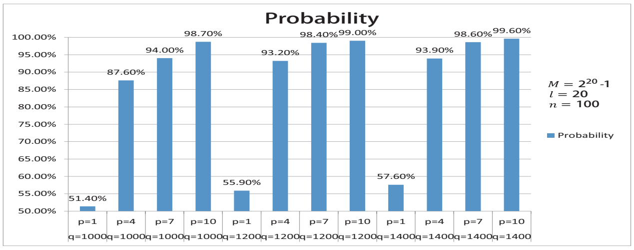

Figure 4. Probability of k-th Min Protocol Being Accurate

protocol has a promising advantage in communication: the baseline protocol only needs one round unidirectional communication (ORUC). Besides the total amount of data that a user needs to send or receive, the round complexity also impacts the amount of time that a user needs to keep being online. Therefore, the baseline protocol suits cases that the network connection is not stable, while our protocol performs well in cases that stable network connections are present.

6.2 Experimental results on probability of being accurate

The accuracy of our privacy-preserving Min computation protocol is closely related to parameters q and l.

In this set of experiments, we set the number of users to 1000, and consider six different lengths of data: $l = 2$, 5, 10, 15, 17, and 19. For all data lengths tested, our protocol is accurate with a probability that is more than 99.9% when $q \geq 14$. Figure 2 shows the probabilities of being accurate for the largest l ($l = 19$). We can see that the probability of being accurate is very high when $q = 20$, which tallies with our theoretical analysis in the previous section.

In addition, we test the probability of our pseudo counting protocol being accurate for a variety of values of p and q. In this set of experiments, we set the number of users to 100 and a user's answer uniformly at random from affirmative and negative. Meanwhile, we present the theoretical estimations of probability for our pseudo counting protocol being accurate in the average case, computed based on Formulas 8 and 10 here. The results are shown in Figure 3.

Also, we fix the number of users as 100 and measure the probability of the k-th Min protocol being accurate. For simplicity, we set l to 20. The results are shown in Figure 4. We can see that, with reasonably large p and q, the probability is pretty high. For example, with $p = 10$ and $q = 1200$, the probability is 99.00%. (We will see in Section 6.4 that the corresponding efficiency is also good.)

6.3 Experimental results on relative error

We are also interested in their relative errors. For our privacy-preserving Min protocol, we measure its relative error for different values of q. Figure 5 shows a curve of the relative error in a set of experiments with 1000 users and $l = 19$. Clearly, the error shrinks quickly when q grows. In this set of experiments, the relative error drops to 0.0003% when $q = 18$.

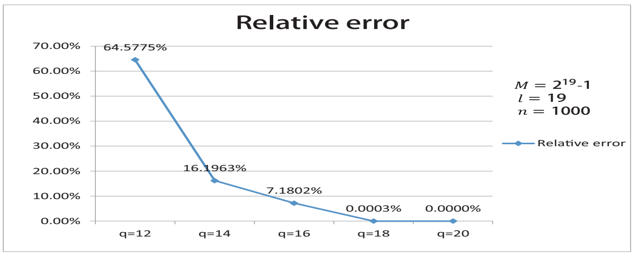

Figure 5. Relative Error of Min Protocol

We also measure the relative error of the k-th Min protocol. In this set of experiments, we set the data length l to 20, and the number of users to 100. The results are shown in Figure 6. We can see that, without reasonably large p and q, the error is very small. For example, with $p = 10$ and $q = 1200$, the relative error is 0.03%. (Again, we will see in Section 6.4 that the corresponding efficiency is also good.)

6.4 Experimental results on efficiency

To see the efficiency of our protocols, we test their computation time. Here the computation time includes the encryption time for each user, and the decryption time for the aggregator. Note that here (and the following) decryption is the decryption of

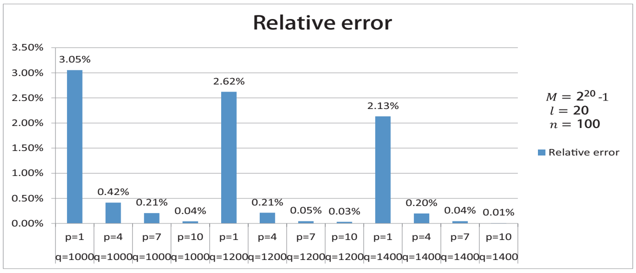

Figure 6. Relative Error of k-th Min Protocol

the bitwise XOR of all users' plaintexts. Also, all operations that encryptions/decryptions need are included in our efficiency experiments, such as the hash operations, comparison operations and bitwise XOR operations.

Table 3
Efficiency of Min Protocol

		$l=20$	$l=25$	$l=30$	$l=32$	$l=34$	$l=36$	$l=38$	$l=40$
Enc.	baseline	$2.5ms$	$33.2ms$	$1.2s$	$4.1s$	$10.4s$	$59.9s$	$282.3s$	$1476.3s$
	ours	$64.4\mu s$	$81.2\mu s$	$96.6\mu s$	$103.0\mu s$	$109.8\mu s$	$117.8\mu s$	$123.3\mu s$	$132.2\mu s$
Dec.	baseline	$33.4ms$	$750.0ms$	$32.2s$	$119.7s$	$328.4s$	$2315.9s$	$> 2500s$	$> 2500s$
	ours	$9.9\mu s$	$12.2\mu s$	$14.0\mu s$	$14.7\mu s$	$15.6\mu s$	$18.8\mu s$	$18.9\mu s$	$19.7\mu s$

The results on the Min protocol are shown in Table 3. In this set of experiments, we fix the number of users to 1000 and let the value of l vary to see how the computation time of the Min protocol changes. We can see that our Min protocol's computation time increases slowly when l grows. When l grows to 40, the encryption time is only 132.2 microseconds and the decryption time is only 19.7 microseconds. In contrast, the baseline protocol's computation time grows very fast. When l reaches 40, the baseline protocol's computation time becomes very long (1476.3 seconds for encryption, and over 2500 seconds for decryption).

Table 4
Efficiency of k-th Min Protocol

		$l=15$	$l=20$	$l=22$	$l=24$	$l=26$	$l=28$
Enc.	baseline	$21.2ms$	$960.5ms$	$1.9s$	$9.8s$	$38.3s$	$187.2s$
	ours	$3.3ms$	$4.4ms$	$4.9ms$	$5.2ms$	$5.7ms$	$6.0ms$
Dec.	baseline	$285.5ms$	$17.2s$	$37.4s$	$216.3s$	$916.7s$	$> 1000s$
	ours	$6.3ms$	$8.3ms$	$9.2ms$	$9.9ms$	$10.8ms$	$11.4ms$

The results on the k-th Min protocol are shown in Table 4. In this set of experiments, we fix the number of users to 100 and $p = 10$, $q = 1400$. When $l = 28$, the baseline protocol's computation time is already very long. It needs 187.2 seconds for encryption and more than 1000 seconds for decryption. However, our k-th Min protocol is still very fast. The encryption/decryption time is only 6.0/11.4 milliseconds.

7 Discussion

In this section, we discuss some potential enhancements of our protocols.

7.1 Dealing with collusion attack

In some cases, the aggregator may be able to collude with a few users in the system. Since the secret that generates a user's key is shared with two other users, this user's data privacy can be easily comprised when the two other users who share the secret happen to be the colluding ones. To deal with such collusion attacks, we present an enhanced keying system which is more secure compared with the basic one.

Specifically, the authority first picks $s_{i,j} \in \{0,1\}^\gamma$ $(i,j = 1,\ldots,n)$ uniformly and independently, where n is the number of users. For each user i $(i = 1,\ldots,n)$, the authority sends $s_{i,1}, s_{i,2}, \ldots, s_{i,n}$ and $s_{1,i}, s_{2,i}, \ldots, s_{n,i}$ to it. And then for each data with the nonce information t, user i computes its secret key using

$$k_i = k_i^r \oplus k_i^c$$

where

$$k_i^r = h_{s_{i,1},f,l}(t) \oplus h_{s_{i,2},f,l}(t) \oplus \ldots \oplus h_{s_{i,n},f,l}(t)$$

$$k_i^c = h_{s_{1,i},f,l}(t) \oplus h_{s_{2,i},f,l}(t) \oplus \ldots \oplus h_{s_{2,i},f,l}(t)$$

The encryption and decryption operations are the same as the cipher presented in Section 3.2. Note that here the decryption operation is the decryption of the bitwise XOR of all users' plaintexts. For a bit-string $x \in \{0,1\}^l$, user i encrypts it by computing

$$\overline{x} = x \oplus k_i$$

As for decryption, the bitwise XOR of all users' ciphertexts equals that of all users' plaintexts, since the bitwise XOR of all users' secret keys equals 0.

In this key generation method, user i's secret key is calculated by $s_{i,1}, s_{i,2}, \ldots, s_{i,n}$ and $s_{1,i}, s_{2,i}, \ldots, s_{n,i}$. It's easy to see that only if the aggregator is in collusion with all remaining $n-1$ users can he recover the user i's key and compute its plaintexts. Therefore the collusion attack can be thwarted by this keying system. Meanwhile, we note that the encryption cost is increased accordingly due to the cost of improved security.

7.2 Dealing with offline users

In practice, some users may not be able to provide the sensed value for various reasons (e.g. battery is dead, connection is lost, etc.) during the protocol's process. (We regard such a user to be offline, either intendedly or unintendedly.) In this case, to acquire the correct Min or k-th Min value of the remaining users' data, the protocol has to be restarted by the remaining online users. A straightforward solution would be letting the remaining online users establish a new set of keys, and restart our protocols using the new keys. However, this could be time-consuming, especially when users go offline frequently. Below we provide a method which avoids re-establishing keys every time a user goes offline. It only requires the trusted authority's help once.

The main idea is to set the offline user's sense data as the maximum value $2^l - 1$, so that its data do not impact the result since the Min or k-th Min value is to be computed. If a user goes offline, the aggregator can request the help of the trusted authority. The trusted authority can "simulate" the offline user's behavior (with $2^l - 1$ as its data) in the protocol using its keys and the correct nonce information. Notice that if a user's data is $2^l - 1$, its answers in all l rounds are always "negative" regardless of the meta-result returned by the aggregator. Thus the trusted authority can compute all responses before the protocol is restarted, and send them to the aggregator at one time. Using the responses, the aggregator can complete the protocol with other remaining online users correctly.

7.3 Keeping the minimum value secret from the users

Both protocols presented in this paper allow users to know the output of protocols. There might be some cases in which users are restricted from knowing the minimum/k-th minimum value.

Recall that both our Min protocol and k-th Min protocol consists of l rounds of "report and determine" steps. Starting from the second round, each user needs to determine its private input in the j-th round ($j = 2, \ldots, l$) based on the MSB, the 2nd-MSB, ..., and the $(j-1)$-th-MSB that are received from the aggregator in previous rounds. To keep our protocols functioning and meanwhile hide these bits of the final outputs, we could utilize the well-known oblivious transfer protocols [26], [27]. A 1-out-of-N oblivious transfer protocol (e.g. [28]) allows the sender with N private messages $M_1, M_2, \ldots, M_N$ to transmit M_σ to the receiver as requested by the receiver, and guarantees that the sender does not know the value of σ and the receiver learns no more than M_σ. We can let each user and the aggregator run one 1-out-of-2^{j-1} oblivious transfer protocol in the j-th round to make sure the aggregator receives the correct data (and the correct data only) and the user does not know the bit values. In more details, below we explain how to apply the above technique by taking the interactions between the aggregator and the user i in the j-th round as an example.

Denote by D_j the j-th-MSB of the protocol's output ($j = 1, \ldots, l$), and by $r_j^i(x_1, \ldots, x_{j-1})$ the function that returns user i's private input in the j-th round when the MSB, ..., and the $(j-1)$-th-MSB of the protocol's output equal $x_1, \ldots, x_{j-1}$ respectively ($x_1, \ldots, x_{j-1} \in \{0, 1\}$). Denote by $OT_1^N(M_1, \ldots, M_N; \sigma)$ the 1-out-of-N oblivious transfer protocol with $M_1, \ldots, M_N$ and σ as the sender's private input and the receiver's private input respectively. Specifically, in the beginning of round j, user i computes $M_x = r_j^i(x_1, \ldots, x_{j-1})$ where $\Sigma_{k=1}^{j-1} x_k \times 2^{j-1-k} + 1 = x$ for $x = 1, \ldots, 2^{j-1}$ as its answer in the Algorithm 1 or Algorithm 3, and compute $\overline{M_x}$ by encrypting M_x same as user i does in the Algorithm 1 or by processing M_x as user i does in Algorithm 2 respectively. After this, user i uses $\{\overline{M_x}\}_{x=1,\ldots,2^{j-1}}$ as its private inputs of the oblivious transfer protocol. At the same time, the aggregator computes $\sigma = \Sigma_{k=1}^{j-1} D_k \times 2^{j-1-k} + 1$ as its private input. Next, the aggregator and user i jointly run $OT_1^{2^{j-1}}(\{\overline{M_x}\}_{x=1,\ldots,2^{j-1}}; \sigma)$ to let the aggregator acquire the correct report generated based on $r_j^i(D_1, \ldots, D_{j-1})$ without revealing $D_1, \ldots, D_{j-1}$ to user i. A flow chart is shown in Figure 7.

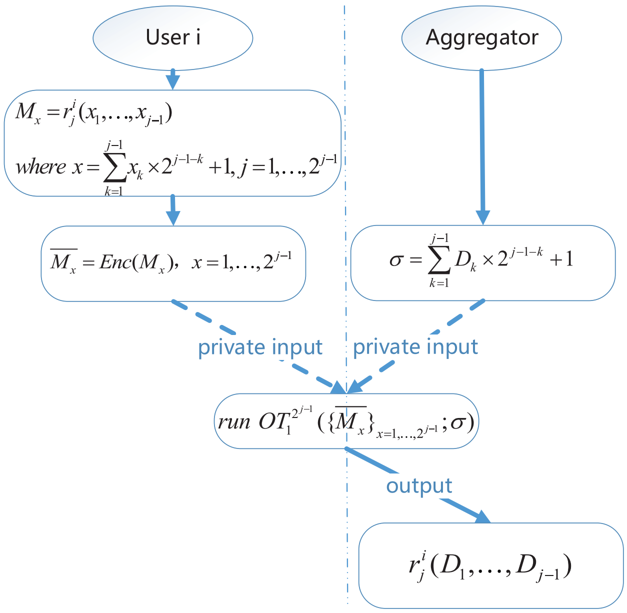

Figure 7. The aggregator and the user i apply OT protocols in the j-th round.

8 Conclusion

In this paper, we study how a data aggregator in a mobile phone sensing scenario can efficiently compute the minimum value or the k-th minimum value in all mobile phone users' private data. Using standard definitions and paradigms in cryptography, we formally prove our protocols are secure and thus are able to protect all users' private data. Compared with existing protocols that are based on arithmetic sum computation, our protocols are based on bitwise XOR computation and thus are more efficient.

References

[1] R. Herring, A. Hofleitner, D. Work, O. Tossavainen, and A. Bayen, *Mobile Millennium-Participatory Traffic Estimation Using Mobile Phones*, 2009.

[2] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, “Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones,” in *SenSys*, D. E. Culler, J. Liu, and M. Welsh, Eds. ACM, 2009, pp. 85–98.

[3] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma, “Prism: platform for remote sensing using smartphones,” in *MobiSys*, S. Banerjee, S. Keshav, and A. Wolman, Eds. ACM, 2010, pp. 63–76.

[4] R. Rana, C. Chou, S. Kanhere, N. Bulusu, and W. Hu, “Ear-phone: an end-to-end participatory urban noise mapping system,” in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. ACM, 2010, pp. 105–116.

[5] X. Bao and R. R. Choudhury, “Movi: mobile phone based video highlights via collaborative sensing,” in *MobiSys*, S. Banerjee, S. Keshav, and A. Wolman, Eds. ACM, 2010, pp. 357–370.

[6] R. Johnson and P. Kuby, *Elementary statistics*. Cengage Learning, 2007.

[7] J. Shi, R. Zhang, Y. Liu, and Y. Zhang, “Prisense: Privacy-preserving data aggregation in people-centric urban sensing systems,” in *INFOCOM*. IEEE, 2010, pp. 758–766.

[8] Q. Li and G. Cao, “Efficient and privacy-preserving data aggregation in mobile sensing,” in *ICNP*. IEEE, 2012, pp. 1–10.

[9] Q. Li, G. Cao, and T. F. L. Porta, “Efficient and privacy-aware data aggregation in mobile sensing,” *IEEE Trans. Dependable Sec. Comput.*, vol. 11, no. 2, pp. 115–129, 2014.

[10] C. Castelluccia, A. C.-F. Chan, E. Mykletun, and G. Tsudik, “Efficient and provably secure aggregation of encrypted data in wireless sensor networks,” *ACM Trans. Sen. Netw.*, pp. 1–36, 2009.

[11] V. Rastogi and S. Nath, “Differentially private aggregation of distributed time-series with transformation and encryption,” in *SIGMOD Conference*, A. K. Elmagarmid and D. Agrawal, Eds. ACM, 2010, pp. 735–746.

[12] G. Ács and C. Castelluccia, “I have a dream! (differentially private smart metering),” in *Information Hiding*, ser. Lecture Notes in Computer Science, T. Filler, T. Pevný, S. Craver, and A. D. Ker, Eds., vol. 6958. Springer, 2011, pp. 118–132.

[13] E. G. Rieffel, J. T. Biehl, W. van Melle, and A. J. Lee, “Secured histories: computing group statistics on encrypted data while preserving individual privacy,” *CoRR*, vol. abs/1012.2152, 2010.

[14] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song, “Privacy-preserving aggregation of time-series data,” in *NDSS*. The Internet Society, 2011.

[15] T.-H. H. Chan, E. Shi, and D. Song, “Privacy-preserving stream aggregation with fault tolerance,” in *Financial Cryptography*, ser. Lecture Notes in Computer Science, A. D. Keromytis, Ed., vol. 7397. Springer, 2012, pp. 200–214.

[16] M. M. Groat, W. He, and S. Forrest, “Kipda: k-indistinguishable privacy-preserving data aggregation in wireless sensor networks,” in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 2024–2032.

[17] R. Lu, X. Liang, X. Li, X. Lin, and X. Shen, “Eppa: An efficient and privacy-preserving aggregation scheme for secure smart grid communications,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 9, pp. 1621–1631, 2012.

[18] Y. Yang, X. Wang, S. Zhu, and G. Cao, “Sdap: a secure hop-by-hop data aggregation protocol for sensor networks,” in *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2006, pp. 356–367.

[19] W. He, X. Liu, H. Nguyen, K. Nahrstedt, and T. F. Abdelzaher, “Pda: Privacy-preserving data aggregation in wireless sensor networks,” in *INFOCOM*. IEEE, 2007, pp. 2045–2053.

[20] C. Wang, G. Wang, W. Zhang, and T. Feng, “Reconciling privacy preservation and intrusion detection in sensory data aggregation,” in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 336–340.

[21] B. K. Samanthula, W. Jiang, and S. Madria, “A probabilistic encryption based MIN/MAX computation in wireless sensor networks,” in *2013 IEEE 14th International Conference on Mobile Data Management, Milan, Italy, June 3-6, 2013 - Volume 1*, 2013, pp. 77–86. [Online]. Available: http://dx.doi.org/10.1109/MDM.2013.18

[22] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, “Order-preserving encryption for numeric data,” in *SIGMOD Conference*, G. Weikum, A. C. König, and S. Deßloch, Eds. ACM, 2004, pp. 563–574.

[23] A. Boldyreva, N. Chenette, Y. Lee, and A. O’Neill, “Order-preserving symmetric encryption,” in *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, 2009, pp. 224–241.

[24] O. Goldreich, *Foundations of cryptography: Basic applications*. Cambridge Univ Press, 2004, vol. 2.

[25] W. Dai. (2010) Crypto++ library. [Online]. Available: http://www.cryptopp.com

[26] M. O. Rabin, “How to exchange secrets with oblivious transfer,” 1981.

[27] M. Bellare and S. Micali, “Non-interactive oblivious transfer and applications,” in *Advances in CryptologyCRYPTO89 Proceedings*. Springer, 1990, pp. 547–557.

[28] M. Naor and B. Pinkas, “Oblivious transfer and polynomial evaluation,” in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. ACM, 1999, pp. 245–254.

[29] S. Banerjee, S. Keshav, and A. Wolman, Eds., *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010), San Francisco, California, USA, June 15-18, 2010*. ACM, 2010.

Yuan Zhang received his BS (2005) in Automation from Tianjin University, MS (2009) in Software Engineering from Tsinghua University, and PhD (2013) in Computer Science from State University of New York at Buffalo. He is interested in security, privacy, and economic incentives.

Qingjun Chen received his BS (2013) in computer science from Nanjing University. He is currently working toward the MS degree with the department of computer science and technology, Nanjing University. He is interested in security, privacy, and economic incentives.

Sheng Zhong received his BS (1996), MS (1999) from Nanjing University, and PhD (2004) from Yale University, all in computer science. He is interested in security, privacy, and economic incentives.