Defending against Frequency-based Attacks on Distributed Data Storage in Wireless Networks

HONGBO LIU, Stevens Institute of Technology
HUI WANG, Stevens Institute of Technology
YINGYING CHEN, Stevens Institute of Technology
DAYONG JIA, Stevens Institute of Technology

As wireless networks become more pervasive, the amount of the wireless data is rapidly increasing. One of the biggest challenges of wide adoption of distributed data storage is how to store these data securely. In this work, we study the frequency-based attack, a type of attack that is different from previously well-studied ones, that exploits additional adversary knowledge of domain values and/or their exact/approximate frequencies to crack the encrypted data. To cope with frequency-based attacks, the straightforward 1-to-1 substitution encryption functions are not sufficient. We propose a data encryption strategy based on 1-to-n substitution via dividing and emulating techniques to defend against the frequency based attack, while enable efficient query evaluation over encrypted data. We further develop two frameworks, incremental collection and clustered collection, which are used to defend against the global frequency-based attack when the knowledge of the global frequency in the network is not available. Built upon our basic encryption schemes, we derive two mechanisms, direct emulating and dual encryption, to handle updates on the data storage for energy-constrained sensor nodes and wireless devices. Our preliminary experiments with sensor nodes and extensive simulation results show that our data encryption strategy can achieve high security guarantee with low overhead.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (e.g., firwalls)*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Distributed networks, network communication*

General Terms: Algorithms, Security

Additional Key Words and Phrases: Frequency-based attack, Secure distrbuted data storage, Wireless Networks

ACM Reference Format:
Liu, H., Wang, H., Chen, Y., Jia, D. 2011. Ensuring data storage security against frequency-based attacks in wireless networks. ACM Trans. Sensor Netw. XX, XX, Article XX (May 2012), 37 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

As the rapid advancement of wireless technologies has led to a future where wireless networks are becoming a part of our social life, the collected wireless data provides tremendous opportunities to support various applications ranging from environmental sensing, to infrastructure monitoring, to mobile social network analysis. However, as the amount of the wireless data is increasing, one of the biggest challenges in wireless networks is how to store these data. The traditional approach is to store the col-

Author's addresses: H. Liu and Y. Chen and D. Jia, Department of Electrical and Computer Engineering, Stevens Institute of Technology; H. Wang, Department of Computer Science, Stevens Institute of Technology.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

lected wireless data in a centralized manner. For example, in wireless sensor networks (WSNs) the sensing data is collected from each individual sensor and sent back to a central server for data access. However, the centralized approaches may require frequent communication with individual devices collecting data, and may even result in performance bottlenecks of data access and a single point of failure to both server compromise and intentional attacks.

To address these problems, distributed data storage [Pietro et al. 2008; Girao et al. 2007; Shenker et al. 2003; Ghose et al. 2003; Shao et al. 2007] in wireless networks recently have attracted much attention. For instance, the sensed data can be stored by its type at a group of storage nodes in the network to perform data-centric storage or stored at each individual device that collects the data. The distributed data storage has major advantages over centralized approaches: storing the data on the collected wireless devices or in-network storage nodes decreases the need of constant data forwarding back to centralized places, which largely reduces the communication in the network and the energy consumption on individual devices, and consequently complements the centralized storage and enables efficient and resilient data access. Furthermore, as wireless networks become more pervasive, new-generation wireless devices with significant memory enhancement and powerful processing capabilities are available (e.g., smartphones, tablets and laptops), making the deployment of distributed data storage not only feasible but also practical.

However, as attackers may be eavesdropping on communication among nodes and/or compromise nodes and thus gain access to all information stored on them, secure data storage must be achieved before widespread adoption of distributed data storage. Prior work in wireless network security has been focused on network communication security such as key management, secure localization, and intrusion detection [Perrig et al. 2001; Liu and Ning 2003a; Capkun and Hubaux 2005; Chen et al. 2007; Yang et al. 2008]. None of these works have addressed the problem of secure distributed data storage. To fulfill the security requirements raised by the distributed data storage, recent research has started studying distributed access control, data confidentiality, and data integrity. [Joshi et al. 2005] introduced a redundancy-based key distribution scheme that utilizes secret sharing to achieve a decentralized certificate authority. [Nalin et al. 2007] studied to perform secure distributed data storage by developing an adaptive polynomial-based data storage scheme. [Wang et al. 2009] presented a dynamic data integrity checking scheme for verifying the consistency of data shares in a distributed manner, which is constructed based on the principle of algebraic signatures to ensure the integrity of data shares.

Most of these current research aim to provide data confidentiality, dependability, and integrity from the perspective that the adversaries will make efforts to access the data by cracking the data encryption mechanisms with little prior knowledge. None of these studies have investigated the problem of attackers cracking the data encryption by exploiting additional adversary knowledge. In particular, today with rapidly evolving adversarial activities, an attacker may possess the knowledge of domain values and/or their exact/approximate frequencies. For instance, in the scenario that the distributed data storage maintains the locations that people visited, the attacker may know: (1) the most popular locations, and (2) the fact that the frequency of these locations should be higher than that of the other locations. To protect user privacy, their locations will be encrypted in the storage. However, a 1-to-1 encryption scheme on locations is not secure, as the attacker can map the encrypted data values of the highest frequency to the popular locations easily. In reality, the attacker may possess approximate knowledge of the frequencies or may know the exact/approximate supports of a subset of data values in the network. However, in order to make the analysis robust, we adopt the conservative assumption above that the attacker knows the exact

frequency of every unique data value in the network. The problem gets even worse if the attacker knows the *exact* frequency of plaintext data values and utilizes such knowledge to crack the data encryption by matching the encrypted data values with original data values based on their frequency distribution. We call such an attack as the *frequency-based attack*. The frequency-based attack is both feasible and harmful in reality. For instance, an attacker can derive the specific activities of an important officer if the attacker knows the frequency of his visited places revealed by the trajectory of his smartphone, or a hunter can wait at specific locations of an endangered animal by possessing the knowledge of the frequency of the animal's habitation-related movements recorded by monitoring sensors.

To cope with the frequency-based attack, apparently any 1-to-1 substitution encryption scheme is not secure. A robust encryption mechanism is in great need. However, very little work has been done to investigate cryptographic solutions that can defend against the frequency-based attack in the distributed storage. [Wang and Lakshmanan 2006] has developed a secure encryption scheme to mitigate the frequency-based attack in a centralized database. However, the encryption scheme is not suitable for distributed data storage in wireless networks. In this paper, we consider two representative types of frequency attacks: (1) the *global frequency-based attack*, whereby the attacker only has the knowledge of the global occurrence of the data in the network, and (2) the *local frequency-based attack*, whereby the attacker knows the specific occurrence frequency of the data on each individual storage node. We propose a data encryption strategy based on 1-to-n substitution to defend against the frequency-based attack. Our data encryption strategy aims to transform the original frequency distribution of the original data (i.e., plaintext) to a uniform distribution of the encrypted data (i.e., ciphertext) so that the attacker cannot derive the mapping relationship between the encrypted data and the original data based on her knowledge of domain values and their occurrence frequency. In particular, we develop two basic techniques, *dividing* and *emulating*, that can be applied either on an individual storage node (as the defend against the local frequency-based attack) or across the network (as the defend against the global frequency-based attack).

Besides the security issue, another equally important issue is how to evaluate queries over the encrypted data efficiently. A naive method is to transfer all encrypted data in the network to a trusted node for decryption and query evaluation, which will incur tremendous communication overhead. Thus we design an *order-preserving* encryption scheme on top of dividing and emulating, so that both types of point and range queries that are commonly used on distributed storage can be evaluated efficiently over the encrypted data directly.

Furthermore, when defending against global frequency based attack in practice, the global frequency of plaintext may not be a known knowledge.

In order to collect plaintext frequency from the network and then distribute keys across the network to defend against the global frequency based attack, we further develop two frameworks, *incremental collection* and *clustered collection*, to achieve global uniform distribution. The *incremental collection* framework is proposed for the chain-based network topology, but could be applied to any type of network. On the other hand, the *clustered collection* framework is applied to the tree-based network topology, in which the frequency of data values is collected in the bottom-up fashion and the encryption keys are distributed in the top-down style. We conduct the theoretical analysis and empirical study of the performance of both frameworks. In order to evaluate the communication overhead of the proposed two frameworks, we build a testbed with MicaZ motes.

Additionally, it is also important to address how to exploit the basic dividing-and-emulating approach to deal with incremental updates on the data storage, without

degrading the security guarantee. We propose two encryption methods to deal with updates, namely *direct emulating* and *dual encryption*. Direct emulating handles encryption on the updated data values independently from the existing ciphertext in data storage, while dual encryption encrypts the updated information with the existing ciphertext together. We compare the overhead of both encryption schemes with various types of updates for both local and global frequency based attacks.

Our theoretical analysis, preliminary experiments using the wireless sensor testbed, and extensive simulation results show that our 1-to-n substitution data encryption framework can achieve high security guarantee, low computational and communication cost, and efficient query processing. The remainder of the paper is organized as follows. In Section 2, we first set up the network model and describe the attack model used in this work. We then describe our 1-to-n substitution data encryption strategy and present the details of the query evaluation procedure in Section 3. In Section 4 we discuss our simulation methodology, evaluation metrics, and results that validate our approach for coping with both local and global frequency based attacks. The evaluation for global frequency based attacks is under the assumption that global frequency of plaintext is known. In Section 5 we address the problem when the knowledge of global frequency is not available by introducing two frameworks to defend against global frequency based attacks. We then propose two encryption methods to deal with incremental updating in data storage in Section 6, and evaluated the overhead of both encryption methods. Finally, we put our work into the broader context in Section 7 and conclude in Section 8.

2. SYSTEM OVERVIEW

2.1. Network Model

In our system, we consider wireless networks consisting of both static and mobile nodes, where each node represents a wireless device that can take the form of sensor, smartphone, tablet, laptop, or active RFID tag. We assume the collected data will be stored within the network at each node unless it is required to be sent to a centralized storage space for backup. We also consider incremental update in the data storage. By uploading data in a lazy fashion (i.e., on-demand only), distributed data storage enables real-time query evaluation and avoids frequent data transfer from the wireless devices to the centralized storage, and consequently reduces massive battery power consumption and vastly decreases the communication overhead of the network.

To prevent the misuse of data and provide data confidentiality, the data will be encrypted in the network. We refer the original unencrypted data values as *plaintext* and the encrypted values as *ciphertext*.

Besides the device nodes that act as the data storage in the network, there also exist mobile devices that act as *query nodes* that will query the data stored in the network. When the query nodes request specific data in the network, all the storage nodes that contain the data matching the query requests will respond to the query nodes by sending them the corresponding data values in ciphertext. The query nodes are responsible to perform the data decryption.

2.2. Attack Model

To describe the attack model used in this work, we first provide an example to illustrate the frequency-based attack. We then define two types of adversaries, mobile and static. Based on the knowledge and behavior of the adversaries, we categorize the frequency-based attack into the *global* and *local* frequency-based attack.

Adversary knowledge. The attacker in the network may possess some background knowledge using which she can conduct attacks on the encrypted data in the network.

We assume that the attacker knows exactly the set of (plain) data values in the network and their true frequencies. The attacker may have access to such data from a competitor, from published reports, etc. In reality, the attacker may possess approximate knowledge of the frequencies or may know the exact/approximate supports of a subset of data values in the network. However, in order to make the analysis robust, we adopt the conservative assumption above that the attacker knows the exact frequency of every unique data value in the network. The example below gives more details of the adversary knowledge and its harm to data security.

Type	Location
Panda	river A
Panda	river A
deer	wood B
Panda	wood C
deer	wood B
Panda	river A

(a) Original dataset

Type	Location
123	river A
123	river A
128	wood B
123	wood C
128	wood B
123	river A

(b) After 1-to-1 encryption

Type	Location
123	river A
123	river A
128	wood B
125	wood C
128	wood B
125	river A

(c) After 1-to-n encryption

Table I: Data example: (a) the original data table; (b) after 1-to-1 encryption; and (c)after 1-to-n substitution encryption via dividing and emulating.

Example. We assume the wireless devices collect the location information of animals. The original data contains the animal type and the location information (Table I (a)). As the panda is an endangered species, the location information about panda's activities is sensitive and should be protected to avoid the access by poachers. The straightforward way is to use 1-to-1 encryption function to encrypt the animal type in the data set (Table I (b)). However, a poacher may know that more sensors are installed on conservation-reliant species such as panda than other animals, e.g., deer. Therefore, the number of locations that are associated with panda should be more than that of the other animals. Based on such adversary knowledge, the poacher can map the ciphertext values to plaintext ones by their frequency. For example, the attacker can map the ciphertext value 123, the one of the largest number of entries in Table I (b), to the plaintext value panda. Consequently, the poacher can decrypt the encryption successfully and knows all locations where a panda may appear.

We consider two types of attackers, *static* and *mobile*, based on their capabilities of collecting the frequency of ciphertext values in the network.

— **Static attacker:** A static attacker resides at one particular position in the network and has the capability of eavesdropping the regular communications within its sensing range or compromising some particular nodes in the network. By eavesdropping or compromising, the static attacker can collect the frequency distribution of ciphertext values stored on a *subset* of nodes in its neighborhood.
— **Mobile attacker:** A mobile attacker can move around in the network while eavesdropping the communication between any pair of nodes or accessing the data storage of all nodes in the network. Therefore, the mobile attacker can possess the global frequency distribution of all ciphertext values in the network.

Given N nodes in the network and k distinctive plaintext values, we use $f_{j,i}(i \in [1,N], j \in [1,k])$ to denote of the frequency of the plaintext PT_j on node i. Based on the frequency information of ciphertext values that an static or mobile attacker can collect, we categorize the frequency-based attacks into two types: *global* and *local* attack.

Local Frequency-based Attack. By this attack, an attacker has the knowledge of the distribution of plaintext values on each individual node in the network. Particularly, the attacker knows the frequency $f_{j,i}$ for each i and j. Both static and mobile attackers can launch the local frequency-based attack. In particular, when a query node broadcasts a query that may contain a set of keyword values (in ciphertext format) to the network, the attacker can collect the frequency of the ciphertext in the returned answers, and compare it with the its adversary knowledge of frequency of plaintext values, leading to the success of compromising the encrypted data on each node independently.

Global Frequency-based Attack. By this attack, an attacker only has the knowledge of the overall distribution of the data in the network. In particular, the adversary knows the occurrence frequency $f_j (1 \leq j \leq k)$ of the plaintext PT_j, where $f_j = \sum_{i=1}^{N} f_{j,i}$. However, the attacker does not have the knowledge of the frequency of data values on each individual node.

Only mobile attacker can launch the global frequency based attack. The attack is similar to the local frequency based attack.

3. DIVIDING&EMULATING: 1-TO-N SUBSTITUTION ENCRYPTION

In this section, we first describe our *dividing* and *emulating* techniques that are the basis of the 1-to-n substitution encryption. We then present our efficient query processing procedure over encrypted data by using dividing and emulating techniques on static storage. We will present our scheme for data updates in section 6.

3.1. Overview

The example in Table I shows that simply encrypting the sensitive data values by using 1-to-1 encryption functions will make it easy to launch frequency-based attacks and disclose the sensitive data to adversaries. To cope with the frequency-based attacks, we propose to *divide* each plaintext value into one or more ciphertext values in such a way that regardless of the original data distribution, the target distribution remains close to uniform. Furthermore, we propose *emulating* on the divided data to fit the target distribution to a uniform distribution, so that the attacker cannot uniquely crack the identity of ciphertext values, i.e., deriving the corresponding plaintext values, based on his knowledge of data frequency. As an example, Table I (c) shows that after applying dividing and emulating techniques, the distribution of the ciphertext values is uniform (i.e., data values 123, 125, and 128 are of frequency 2), which highly decreases the probability for an adversary to derive the plaintext values by launching a frequency-based attack.

To defend against the global frequency-based attacks, our dividing and emulating techniques exploits the global frequency distribution of plaintext values in the network to achieve uniform frequency distribution of the ciphertext values in the whole network. Whereas to protect the data from the local frequency-based attack, the uniform distribution of the target ciphertext will be achieved on each individual wireless device independently. Thus, it is possible that the frequency of the same data value (in encrypted format) on different nodes is different.

3.2. Dividing

Basically, dividing encrypts each plaintext value PT to multiple ciphertext values whose total frequency equals to the frequency of PT. Intuitively, if k unique plaintext values are encrypted to $m > k$ unique ciphertext values that are of the same frequency, none of these ciphertext values can be explicitly mapped to their corresponding plaintext values by the frequency-based attack. Indeed, the *decipher probability* P that these m ciphertext values can be correctly mapped to k plaintext values by the frequency-based attack equals

$$P = \frac{1}{\binom{m-1}{k-1}}. \tag{1}$$

In practice, the attacker may only be interested in hacking the encryption of specific data values. In particular, when we reason the decipher probability of a specific plaintext value v, we follow these three steps:

— Step 1: construct all possible mapping schemes as candidates. There are $\binom{m-1}{k-1}$ possible mappings;
— Step 2: count the number of correct mappings of v among all candidate mappings. There are $\binom{m-2}{k-2}$ such mappings; and
— Step 3: compute the decipher probability P_v of plaintext value v following the result of Step 1 and 2. This turns out to be:

$$P_v = \frac{\binom{m-2}{k-2}}{\binom{m-1}{k-1}} = \frac{k-1}{m-1} \tag{2}$$

As we consider all the data in the storage as sensitive (and thus is encrypted in the storage), we focus on defending against the attack to crack all the ciphertext values, and require the decipher probability of such attack (shown in Equation 1) should be less than a given threshold σ.

Number of Divided Ciphertext Values. To achieve a threshold σ of the decipher probability, for k unique plaintext values that are encrypted to m unique ciphertext values of the same frequency, they must satisfy $P = \frac{1}{\binom{m-1}{k-1}} \leq \sigma$. Intuitively, the smaller σ is, the more robust the dividing scheme is against the frequency-based attack. With given σ and k values, deriving m from the constraint $\frac{1}{\binom{m-1}{k-1}} \leq \sigma$ is computationally hard. Thus we consider Stirling's approximation, i.e., $m! \approx m^m e^{-m}\sqrt{2\pi m}$. We have:

$$\begin{aligned} P = \frac{1}{\binom{m-1}{k-1}} = \frac{1}{\frac{(m-1)!}{(k-1)!(m-k)!}} &\approx \frac{1}{\frac{(m-1)^{(m-1)}\sqrt{2\pi(m-1)}}{2\pi(k-1)^{(k-1)}(m-k)^{(m-k)}\sqrt{(k-1)(m-k)}}} \\ \leq \frac{1}{\frac{(m-1)^{(m-1)}\sqrt{m-1}}{\sqrt{2\pi}((k-1)^{(k-1)}(m-1)^{(m-k)}\sqrt{(k-1)(m-1)})}} &= (\frac{k-1}{m-1})^{(k-1)}\sqrt{2\pi(k-1)}. \end{aligned} \tag{3}$$

As it is required that $P = \frac{1}{\binom{m-1}{k-1}} \leq \sigma$, from Equation 3, we can infer that $m \geq (k-1)(\frac{\sqrt{2\pi(k-1)}}{\sigma})^{\frac{1}{k-1}} + 1$. It is straightforward that larger m value implies the more robustness of the dividing scheme against the frequency-based attack. However, as we will discuss soon, larger m values will also result in more keys needed for encryption and decryption. To balance the trade-off between the robustness of the scheme and the

cost for key management, we use

$$m = (k-1)(\frac{\sqrt{2\pi(k-1)}}{\sigma})^{\frac{1}{k-1}} + 1 \quad (4)$$

as the number of divided ciphertext values needed to achieve the required robustness of the scheme. Based on the valued m, next, we discuss how to split k unique plaintext values into m unique ciphertext values.

3.2.1. Dividing Factor.. We define the *dividing factor* in our dividing scheme as following.

Definition 3.1. Given k unique plaintext values $PT_j(1 \leq j \leq k)$ that are encrypted as m unique ciphertext values, let $f = \Sigma_{j=1}^{k} f_j$, where f_j is the frequency of the plaintext value PT_j. Then each PT_j is encrypted as $\lceil \frac{f_j}{d} \rceil$ unique ciphertext values, where

$$d = \lceil \frac{f}{m} \rceil. \quad (5)$$

We call d the *dividing factor*.

After dividing, k unique plaintext values are encrypted to m unique ciphertext values, such that $m - \lceil \frac{md-f}{d} \rceil$ of them are of the same frequency d. If $md = f$, then all m ciphertext values are of the same frequency. Otherwise, out of these m values, there will be $\lceil \frac{md-f}{d} \rceil$ of them with frequency of $f_j - \lfloor \frac{f_j}{d} \rfloor \times d$, where f_j is the frequency of their corresponding plaintext values.

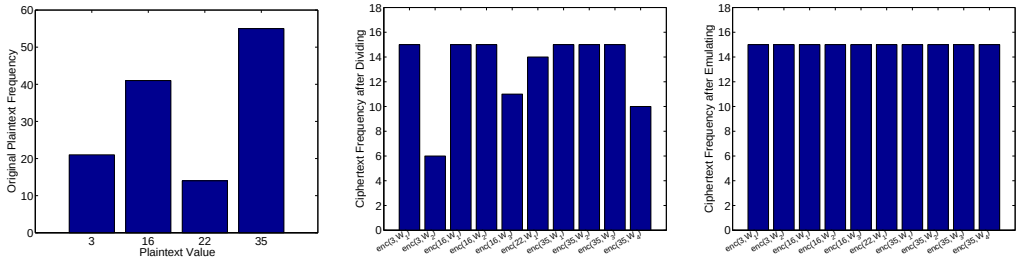
3.2.2. Dividing Procedure.. Next, we describe the details of our dividing procedure that can achieve the goal mentioned above. **Step I. Sorting:** We sort the plaintext values by their frequencies in ascending order. Let $\delta = min(PT_{j+1} - PT_j)(1 \leq j \leq k-1)$ be the minimal interval between any two successive frequency values.

Step II. Dividing: For each plaintext value PT_j, we choose t distinct random numbers $w_1, \ldots, w_t(1 \leq t \leq \lceil \frac{f_j}{d} \rceil)$ as weight values, where f_j is the frequency of PT_j, and d is the dividing factor. We require that w_2 should be unique among all the w_is, as it is needed for value decryption (More details are in Section 3.4). Then we partition f_j number of PT_j values into $\lceil \frac{f_j}{d} \rceil$ partitions, each partition containing d number of PT_j values, except the last one that contains $f_j - \lfloor \frac{f_j}{d} \rfloor \times d$ number of PT_j values. Then the PT_j value in the i-th partition $(1 \leq i \leq \lceil \frac{f_j}{d} \rceil)$ is encrypted to

$$CT_i = enc(PT_j + \sum w_i \delta), 1 \leq i \leq \lceil \frac{f_j}{d} \rceil, \quad (6)$$

where w_i is a distinct random number $\in (0, 1/(\lceil \frac{f_j}{d} \rceil + 1))$ (i.e., $\Sigma w_i < 1$), and $enc()$ is an order-preserving encryption function [Boldyreva et al. 2009]. More specifically, the first partition of occurrence of PT_j is transformed to $enc(PT_j + w_1\delta)$; the l-th partition of occurrences to $enc(PT_j + \sum_{1 \leq i \leq l}(w_i\delta))$. That is to say, the l-th partition is displaced from PT_j by a fraction of the gap δ given by the sum $w_1 + w_2 + \cdots + w_l$. After dividing, there are $\lceil \frac{f_j}{d} \rceil$ number of ciphertext values $CT_1, \ldots, CT_t(1 \leq t \leq \lceil \frac{f_j}{d} \rceil)$, with their total frequencies equal to f_j.

To illustrate the results of ciphertext values by applying our dividing technique, we show a simple example as following: Given two plaintext values PT_1 and PT_2 of frequency 12 and 21, $f = 12 + 21 = 33$. Assume Equation 4 has returned $m = 5$. Then using Definition 3.1, the dividing factor d is calculated as 7. Based on the dividing procedure, PT_1 will be encrypted as 2 unique ciphertext values, one of frequency 7,



(a)Original plaintext freq. (b)Ciphertext freq. after dividing (c)Ciphertext freq. after emulating

Fig. 1: Dividing and Emulating

and one of frequency 5, by using 2 unique keys; PT_2 will be encrypted as 3 unique ciphertext values, each of frequency 7.

Due to the use of order-preserving encryption function $enc()$, a nice property of the dividing scheme is that the ciphertext corresponding to different plaintext values will not straddle each other. More precisely, for any two values $PT_i < PT_j$, and for any ciphertext values CT_i^m, CT_j^n (i.e., the m-th and n-th ciphertext values of PT_i and PT_j respectively), it is necessary that $CT_i^m \leq CT_j^n$. This will enable the efficient query evaluation over the ciphertext values (More details of query evaluation will be discussed in Section 3.4).

Cost of Key Management. For each plaintext value that is divided into r unique ciphertext values, we need r unique keys. To reduce the total number of keys that are needed for dividing k unique plaintext values in the network, we allow these plaintext values to share keys for dividing. Therefore, the number of keys r needed for the dividing scheme equals to $r = max_{1 \leq j \leq k} \lceil \frac{f_j}{d} \rceil$, which largely reduces the total number of unique keys during encryption.

3.3. Emulating

The dividing procedure cannot guarantee that all ciphertext values are of the same frequency. Figure 1 (a) and (b) depict an example of dividing. The 4 plaintext values 3, 16, 22, and 35 of occurrence frequency 21, 41, 14 and 55 (Figure 1 (a)) are divided into 10 unique ciphertext values (Figure 1 (b)), with the dividing factor as 15. Figure 1 (b) shows some ciphertext values, including the second ciphertext value of plaintext value 3, the third ciphertext value of plaintext value 16, the first ciphertext value of plaintext value 22, and the last ciphertext value of plaintext value 35, that are of different frequency from the other ciphertext values. These ciphertext values may face the threat that their encryption can be cracked by the frequency-based attack.

Thus, we apply *emulating* on these values, so that these ciphertext values are indistinguishable from the others by their frequencies. In particular, for these ciphertext values, they are duplicated so that their frequency also equals to d, the frequency of the other ciphertext values. Figure 1 (c) shows the results of the frequencies of these ciphertext values after emulating. Therefore, By performing emulating, these ciphertext values are indistinguishable by the frequency-based attack. However, it incurs additional space overhead for the duplicates, which is called emulating noise. There exists a trade-off between the security guarantee and the space overhead i.e., higher security guarantee may lead to more space overhead. This trade-off will be studied in details in Section 4.

To cope with both global and local frequency-based attacks, we apply the dividing and emulating encryption scheme on the plaintext values. To resist the global frequency-based attack, we apply the scheme on the global distribution information to achieve globally uniform frequency distribution of the ciphertext values (i.e., all unique ciphertext values in the network are of the same frequency). While to defend against the local frequency-based attacks, we exploit the dividing and emulating techniques locally on each individual device, so that the ciphertext values on each device will achieve uniform frequency distribution.

3.4. Efficient Query Processing over Encrypted Data

We assume that the users issue their original queries that only contain plaintext values. In this paper, we consider two types of queries: *point queries* that return all data values in the network that equal to a given value, and *range queries* that return all data values in the network that fit in a range $[l, u]$. Our goal is to translate the plaintext queries to ciphertext queries that can be applied directly on the encrypted data in the network. This mechanism has two advantages: (1) sending ciphertext queries to the network will protect the queries, especially the plaintext values in the queries, from the malicious attackers, and (2) it supports efficient query evaluation, as data decryption in the network, which in general is costly, is avoided. To achieve the goal, we design the query processing procedure that consists of three phases: query translation at the user side, query evaluation at the nodes in the network, and query post-processing at the user side. Next, we discuss the details of these three phases.

Phase-1: Query translation at user side. We assume a user can access all the auxiliary information including the weight values w_i, the gap values δ, and the order-preserving encryption function $enc()$ that are used in the dividing scheme. He/she will make use of these information to translate the plaintext queries as following.

Point queries: Given a point query $Q : V = v$, the user will translate it to Q' by following the same dividing scheme for encrypting data values in the network. In particular, the plaintext value v will be encrypted to r ciphertext values $CT_1, \ldots, CT_r$. Since these r ciphertext values follow the order that $CT_1 < CT_2 \cdots < CT_r$, the query Q will be translated to $Q' : V \in [CT_1, CT_r]$, where $CT_1 = enc(v + w_1 * \delta)$, and $CT_r = enc(v + \sum_{i=1}^{r} w_i * \delta)$. Here w_i and δ are pre-valued in the dividing scheme (see Section 3.2).

Range queries: Recall that our dividing technique performs order-preserving encryption. Thus the range query $Q : V \in [l, u]$ will be translated to another range query Q'. In particular, let w_i^l and w_i^u be the i-th weight values assigned for dividing l and u, and δ_l, δ_u be the gap values used for dividing l and u values, then the query Q will be translated to $Q' : V \in [CT_1^l, CT_r^u]$, where $CT_1^l = enc(l + w_1^l * \delta_l)$, and $CT_r^u = enc(u + \sum_{i=1}^{r} w_i^u * \delta_u)$. In other words, the plaintext range $[l, u]$ is translated to another range whose lower bound equals to the smallest divided ciphertext value of l, and upper bound equals to the largest divided ciphertext value of u.

Phase-2: Query evaluation at nodes in the network. After translation, the range query $Q' : V \in [CT_l, CT_u]$ (for both point and range plaintext queries), where CT_l and CT_u are the lower bound and upper bound ciphertext values, will be sent to the network. Each node will check whether it has any ciphertext value that satisfies the query Q', and return these ciphertext values if there is any. To ensure successful decryption in Phase-3, we require that there are at least two unique ciphertext values to be returned; if there is only one ciphertext CT value that satisfies Q', the next ciphertext value that is greater than CT will also be sent back, even though it may not satisfy Q'.

Phase-3: Query post-processing at user side. After the user receives the returned ciphertext values $CT_1, CT_2, \cdots, CT_t$ from the network, he/she will decrypt these values and obtain the plaintext values. In particular, with the knowledge of the gap values

δ, he/she calculates $s_v = CT_{v+1} - CT_v$, the distance of every two successive ciphertext values (we assume $CT_1 \leq \cdots \leq CT_t$). If there exists any s_v that equals to $w_2 * \delta$, then the user deciphers CT_v as $(CT_v - w_1 * \delta)$. The reason that only w_2 is used for decryption is that if there exists any answer PT, it must satisfy that for the first and the second divided values CT_1 and CT_2 of PT, $CT_1 = PT + w_1 * \delta$, and $CT_2 = PT + (w_1 + w_2) * \delta$, thus there must exist $s_v = CT_{v+1} - CT_v$ that equals to $w_2 * \delta$. If there is no such s_v that equals to $w_2 * \delta$, then there is no answer to the queries. The success of the Phase-3 decryption is guaranteed by: (1) our design of the dividing scheme that requires that w_2 is unique among all weight values, so that is $s_v = w_2 * \delta$, and (2) our Phase-2 query evaluation procedure that requires that at least two ciphertext values (i.e., at least the first and the second divided values) should be returned.

We illustrate the query post-processing procedure through the following example: Given the plaintext values {10,11,13,14,17} with $\delta = 0.5$, and the weights {0.1, 0,3, 0.2, 0.1}, the divided ciphertext values will be CT = {10.05, 10.15, 11.05, 11.2, 11.3, 13.05, 14.05, 14.2, 14.3, 14.35, 17.05}. Let's consider a plaintext query $Q : V \in [13.45, 14.15]$. It is translated to the ciphertext query $Q' : V \in [13.5, 14.5]$. Applying Q' on CT will return {13.05,14.05,14.2,14.3,14.35}. There exists two ciphertext values 14.05 and 14.2 whose distance equals to $\delta * w_2$ = 0.15. Thus the value 14.05 is deciphered as $14.05 - w_1 * \delta$ = $14.05 - 0.1 * 0.5$ = 14.

It is possible that the decrypted query result contains duplicated (noise) values that is added by the emulating procedure. To remove those duplicated values, we exploit the signature-based strategy introduced in [Xie et al. 2007]. In particular, each node will add a special signature s_h to each (real or noise) tuple $t(c_1, c_2, \cdots, c_n)$ in its dataset. The signature is constructed in a different way for real values and noise values that are added by emulating. In particular, for duplicated (noise) tuple, a flag is added when computing s_h, as shown in Equation 7.

$$s_h = \begin{cases} H\left(c_1 \bigoplus c_2 \bigoplus \cdots \bigoplus c_n\right) & if\ t\ \ is\ \ real \\ H\left(c_1 \bigoplus c_2 \bigoplus \cdots \bigoplus c_n\right) + 1 & if\ t\ \ is\ \ noise(added\ by\ emulating) \end{cases} \tag{7}$$

where $\bigoplus$ indicates string concatenation, and H is a one-way hash function that takes a variable length input string and converts it into a fixed length (e.g., 128 bits) binary sequence. We assume that only authorized users will be assigned with H; the attacker will not be able to access to H and consequently reconstruct s_h. The signature s_h will be associated with its value in the network all the time.

On receiving a tuple $t(c_1, c_2, \cdots, c_n)$ with its signature s_h from querying the network, the authorized user will determine whether the tuple is real for noise. If it is a real tuple, it should be true that $s_h = H\left(c_1 \bigoplus c_2 \bigoplus \cdots \bigoplus c_n\right)$. Otherwise, it must be a noise value added by the emulating procedure. The signatures will only bring negligible computation overhead, as they are of fixed length (128 bits).

4. PERFORMANCE ANALYSIS FOR DIVIDING AND EMULATING

In this section, we evaluate the performance of our proposed dividing and emulating techniques by defining two metrics, *overhead by dividing only* and *total overhead (after dividing and emulating)*. We present both theoretic analysis and simulation results to validate their effectiveness. For defending against global frequency based attack, we perform simulation under the assumption that global frequency of plaintext is known, and in the next section (Section 5) we develop two frameworks to show how to collect global frequency information efficiently in the network for coping with global frequency based attack.

4.1. Metrics

To evaluate the performance of our proposed dividing&emulation approach, we developed the following metrics.

Overhead by dividing only. We measure the number of ciphertext values that are introduced by the dividing process. For the global frequency-based attack, we define the *overhead by dividing* as $\frac{m-k}{k}$, where m and k are the numbers of distinct ciphertext and plaintext values in the network. For the local frequency-based attack, we define the *overhead by dividing* as $\frac{\sum_{i=1}^{N} m_i - \sum_{i=1}^{N} k_i}{\sum_{i=1}^{N} k_i}$, where m_i and k_i are the number of distinct ciphertext and plaintext values on the i-th node $(1 \leq i \leq N)$. Intuitively, the larger the number of distinct ciphertext values after dividing, the more computational overhead is incurred by the dividing approach. We will evaluate the overhead by dividing for various decipher probabilities, including both global and local frequency-based attacks.

we will also quantify the overhead introduced by performing emulating after dividing. We define the *total overhead after dividing and emulating* as $\frac{S_e - S_o}{S_o}$, where S_o and S_e are the sizes of the data memory before and after emulating.

4.1.1. Theoretical Analysis. First, we analyze the overhead of our approach theoretically.

Overhead by dividing only. The overhead by dividing for coping with global frequency based attack should be less than that for coping with local frequency based attack

According to equation 4, for defending against frequency based attack, the number of encryption keys required depends on the number of plaintext involved and prior decipher probability, the only difference resides in that local frequency based attack is based on individual node whereas the whole network for global frequency based attack. With respect to one particular plaintext PT_j of the frequency f_j, given the dividing factor d, it requires $\tau = \left\lceil \frac{f_j}{d} \right\rceil$ encryption keys for coping with global frequency based attack. However, since PT_j is stored on several sensor nodes $n_i, 1 \leq i \leq N$, with frequency f_j^i respectively, each node n_i needs $\tau_i = \left\lceil \frac{f_j^i}{d} \right\rceil$, encryption keys for defending against local frequency based attack if the dividing factor d_i on individual node equals to d. The total number of encryption keys from all the nodes $\sum_{i=1}^{N} \tau_i$ should be no less than τ shown as below.

Assume $f_j^i = P_i d + r_j^i$, so that $f_j = \sum_{i=1}^{N} P_i d + \sum_{i=1}^{N} r_j^i$, where $0 \leq r_j^i \leq d$. Note that r_j^i is the remainder after f_j^i divided by dividing factor d. Therefore for global frequency based attack, we have the overhead by dividing:

$$\tau = \sum_{i=1}^{N} P_i + \left\lceil \frac{\sum_{i=1}^{N} r_j^i}{d} \right\rceil \tag{8}$$

whereas for local frequency based attack, the overhead by dividing is:

$$\tau' = \sum_{i=1}^{N} \tau_i = \sum_{i=1}^{N} P_i + N \tag{9}$$

Next we compare the overhead by dividing τ and τ' for global and local frequency based attack. Since $r_j^i \leq d$, It is obvious that $\left\lceil \frac{\sum_{i=1}^{N} r_j^i}{d} \right\rceil \leq N$. Further, the dataset size on each individual node is only a small portion of the total amount of global dataset, given the same decipher probability, the dividing factor is generally smaller for local

frequency based attack than that in global frequency based attack. $\sum_{i=1}^{N} \tau_i$ would be even larger than τ. Therefore, the overhead by dividing for local frequency based attack is no less than global frequency based attack under different decipher probability or distinct number of plaintext, which can also be observed from figure 2 and figure 3.

Total overhead. With different number of nodes and the distributions of frequency for the plaintext in the network, the overhead for local and global frequency based attacks do not have deterministic relationship.

Given the dividing factor d for coping with global frequency based attack, the total overhead by dividing and emulation the particular plaintext P_j is derived as

$$e_j = d - mod(f_j, d) = d - r_j \tag{10}$$

If the dividing factor on each individual node is $d^i, 1 \leq i \leq N$, for local frequency based attack the overhead is:

$$e_j = \sum_{i=1}^{N} (d^i - mod(f_j^i, d^i)) = \sum_{i=1}^{N} (d^i - r_j^i) \tag{11}$$

If we assume a uniform distribution of frequency of plaintext PT_j, when the frequency of PT_j is much larger than d, the total overhead for PT_j also approximately follows uniform distribution on the range $[0, d]$ when coping with global frequency based attack. Similarly, the overhead by emulation on local node $i, 1 \leq i \leq N$, is also uniformly distributed on the range $[0, d^i]$.

When there are k distinct plaintext, the expected total overhead across all plaintext should be for coping with global frequency based attack.

$$S_e^{global} = \sum_{j=1}^{k} e_j = \frac{dk}{2} \tag{12}$$

However, for coping with local frequency based attack, the expected total overhead for all plaintext in the network is determined as:

$$S_e^{local} = \sum_{j=1}^{k} e_j = \sum_{i=1}^{N} \frac{d^i k}{2} \tag{13}$$

As there is no direct relationship between d and $\sum_{i=1}^{N} d^i$, it is difficult to compare S_e^{global} and S_e^{local}.

4.2. Empirical Analysis

We conducted simulation of a wireless network with multiple nodes using Matlab. Each wireless node collects the data and stores it on itself. We tested on three network sizes with number of nodes set to $N = 20, 60$ and 100 respectively. For each simulation setup, we controlled the total number of distinct plaintext values in the network to be less than or equal to 200. The occurrence frequency of plaintext values on each wireless node is positive integer chosen in the range of $[0, 100]$ that follows a uniform distribution. We evaluated our approach under a broad range of the decipher probability valued from 0.00001, 0.0001, 0.001, 0.01, 0.1 to 1. Our simulation results are the average over 100 runs for each simulation setup.

4.2.1. Coping with Local Frequency-based Attack. We first examine the overhead of the dividing. Figure 2 shows the result with the number of distinct plaintext values fixed as 200 (however, the number of distinct plaintext values on each node is less than 200). As shown in Figure 2 (a), we observed that the overhead by dividing decreases

(from 16% to 4%) as the decipher probability increases. This is obvious as stronger security (i.e., smaller decipher probability) needs more distinct cipher values to deceive the attacker. However, the overhead is always very small even with large decipher probability; only 16% of overhead is incurred to achieve the decipher probability as 0.00001.

Next, we measure the total overhead introduced by dividing and emulating when varying the decipher probability. The result is presented in Figure 2 (b). We found that the total overhead slightly increased (from 22% to 25%) when the decipher probability increases (from 10^{-5} to 1), i.e., the total overhead is not sensitive to the changes of decipher probability. This is because the overhead of the emulating alone, which mainly depends on the frequency of ciphertext values but not the number of distinct cipher values, dominates the total overhead. Therefore, changing decipher probability, which consequently change the number of distinct ciphertext values, will not influence the total overhead by dividing and emulating.

We then measured the overhead of dividing and emulating with various number of distinct plaintext values. Figure 2 (c) shows the overhead by dividing alone when $P = 0.01$ and $N = 60$. The observation is that the overhead decreases (from 45% to 9%) as the number of distinct plaintext values increases from 40 to 200. We also measured the total overhead by dividing and emulating process with various numbers of distinct plaintext. We show the result in Figure 2 (d). We found that the overhead increases (from around 35% to 45%) when the number of distinct plaintext increases (from 40 to 200). This is because the larger number of distinct plaintext values needs the larger number of distinct ciphertext values to achieve the required decipher probability, and consequently more ciphertext values to be emulated.

Additionally, we observed from figure 2 (a) and (b) that the overhead does not increase as the network size increases, suggesting that both the computational cost and memory overhead introduced by our scheme are stable and do not vary with network size. This is advantegous as our scheme is scalable to large networks.

4.2.2. Coping with Global Frequency-based Attack. Next, we turn to study the performance of our scheme to defend against the global frequency-based attack. First, we measure the overhead of our approach under various decipher probability. Figure 3 (a) presents the overhead by dividing in the network under various decipher probability when fixing the number of distinct plaintext values as 200. The key observation from Figure 3 (a) is that the overhead by dividing is always small (under 10%) even when the decipher probability goes to around 10^{-5}. This is encouraging as it indicates that our scheme can achieve a robust security guarantee under the global frequency-based attack with little overhead incurred by the dividing technique. The other observation is that the overhead keeps unchanged for different network sizes. This is because the number of ciphertext values required for dividing does not rely on the number of nodes in the network. This property shows that our approach can be applied to large-scale networks. We also observed that the overhead of our approach to defend against the global frequency-based attack is smaller than (almost half of) the overhead to defend against the local frequency-based attack. This is inline with our theoretical analysis in Section 4.1.1. Since the dataset size on each individual node is only a small portion of the total amount of global dataset, given the same decipher probability, the dividing factor is generally smaller for local frequency based attack than that needed in global frequency based attack. Therefore, the overhead by dividing for defending against local frequency based attack is no less than that in global frequency based attack under different decipher probability or distinct number of plaintext.

We then measured the overhead by dividing and emulating with the same setup of decipher probability, network sizes, and the number of distinct plaintext values.

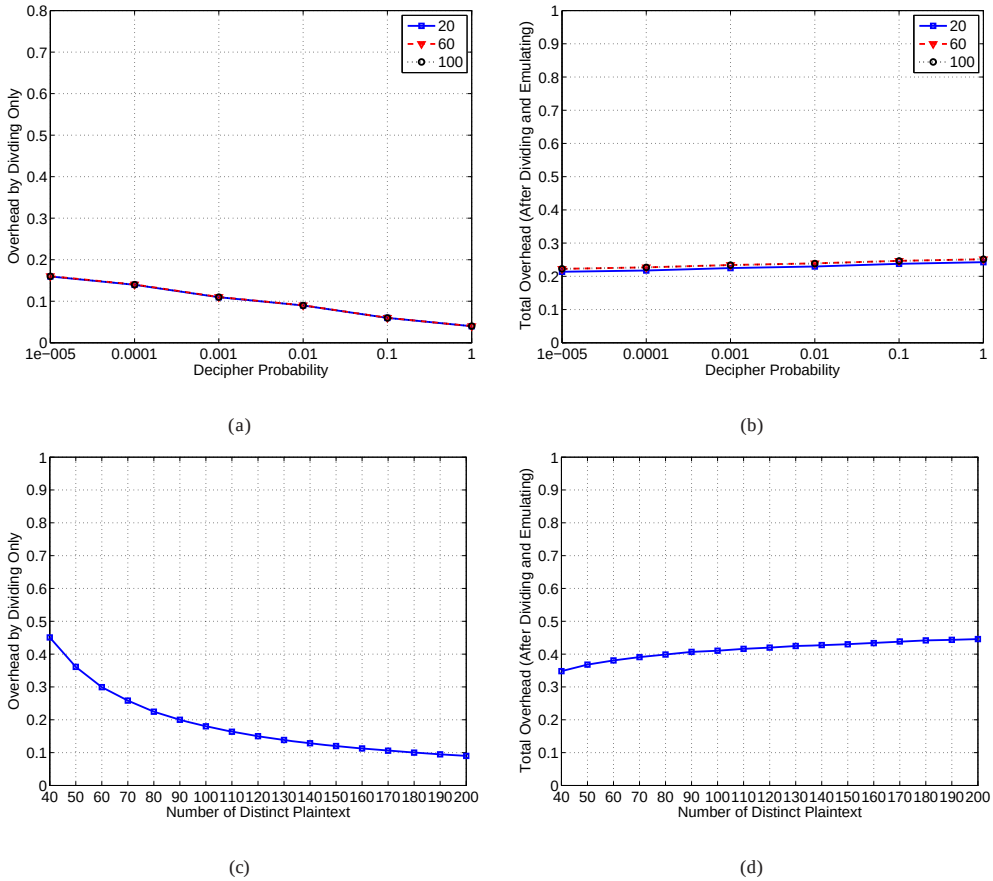


Fig. 2: Overhead of Diving & Emulating when coping with local frequency-based attack with network size $N = 20, 60$ and 100: (a) and (b) measures the overhead by dividing and emulating under various decipher probability, with the number of distinct plaintext values fixed at 200; (c) and (d) shows the overhead by dividing and emulating under various distinct plaintext values, with the decipher probability fixed as 0.01 and network size fixed as 60.

Figure 3 (b) shows the result. We observed that the overhead by emulating is not sensitive to the decipher probability. It goes up slightly from 22% to 25% as the decipher probability increases, which is not significant. We further found that the overhead by emulating does not change with the network size. These discoveries suggest that our scheme is robust in terms of the amount of overhead even for large networks.

We further investigated the overhead of our approach when varying the number of distinct plaintext values in the network. We started from the measurement of overhead by dividing alone. Figure 3 (c) depicts the overhead by dividing with various number of distinct plaintext values, when the decipher probability $P = 0.01$ and the network size $N = 60$. We found that similar to our overhead measurement for the local frequency-based attack case, the overhead by dividing decreases (from 22% to 8%) when the number of distinct plaintext values increases (from 40 to 200). We also examined the total overhead of dividing and emulating with various number of distinct plaintext val-

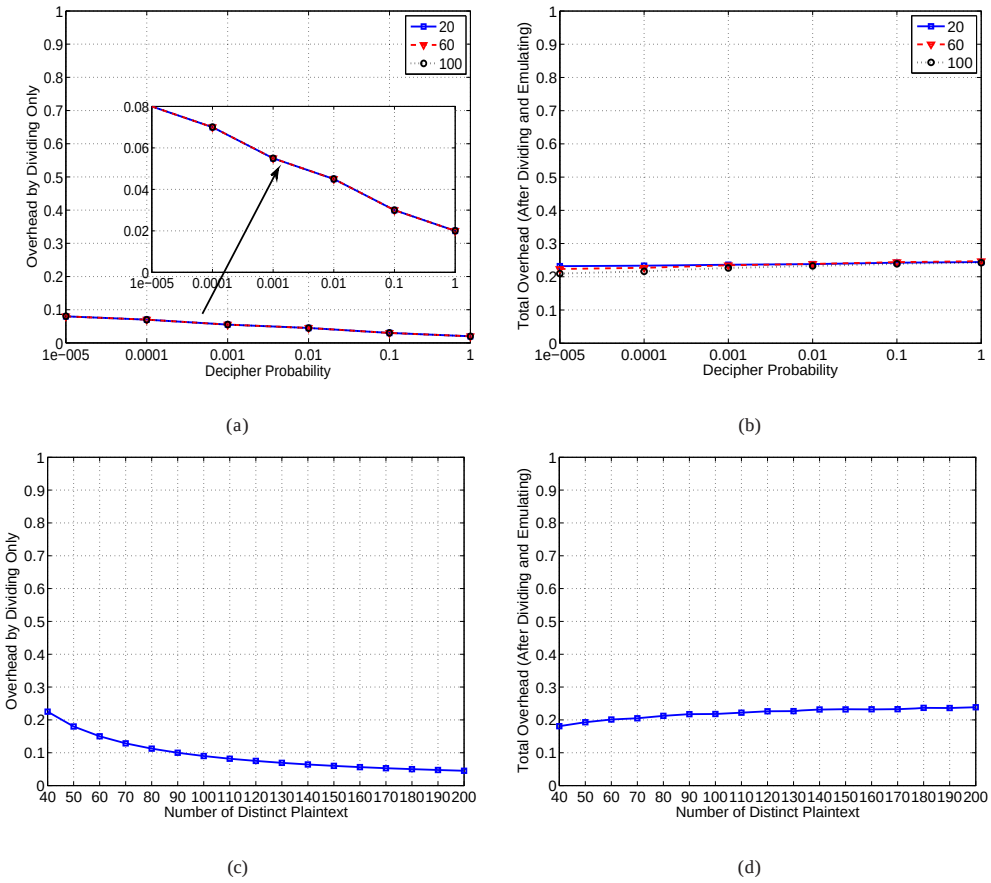


Fig. 3: Overhead of Dividing and Emulating when coping with the global frequency-based attack: (a) and (b) show the overhead by dividing and emulating under various decipher probability and network sizes $N = 20, 60$ and 100, with the number of distinct plaintext values fixed as 200; (c) and (d) present the overhead by dividing and emulating under various distinct plaintext values with the decipher probability fixed as 0.01.

ues. The result is shown in Figure 3 (d). We found that the total overhead only has a slight increase with increasing number of distinct plaintext in the network. In particular, the overhead by emulating generally maintains at around 20% when the number of distinct plaintext values increases from 40 to 200 in the network. This observation implies that our proposed algorithm is robust in terms of the amount of overhead of dividing and emulating with the presence of different number of plaintext values. We also observed that similar to the overhead of dividing alone, the total overhead of our approach to defend against the global frequency-based attack is smaller than (almost half of) the overhead to defend against the local frequency-based attack. These observations indicate that our scheme does not require more overhead when coping with the global frequency-based attack than that under the local frequency-based attack.

5. ENCRYPTION FRAMEWORK FOR COPING WITH GLOBAL FREQUENCY BASED ATTACK

In section 4, when evaluating the effectiveness of the proposed diving and emulating techniques under global frequency based attack, we assume the global frequency information is known. However, this assumption may not be applicable to all the cases in practice. To address the issue of collecting the information of the global frequency and distributing the keys of each plaintext back to the network, we propose two frameworks, *incremental collection* and *clustered collection*. Incremental collection is intended for the simple network topology (e.g., chain-based), whereas clustered collection is designed for more advanced network topology such as tree-based. Both of the two frameworks are used to defend against global frequency based attack when the global frequency information is not available.

5.1. Framework overview

The basic idea of our frameworks is to collect the global frequency and distribute keys through the communication among nodes across the network under different topologies. For each framework, we have a two-round communication process, including collecting global plaintext frequency and distributing encryption keys to each node. The *incremental collection* approach is suitable for the network with a chain-based topology [Hatcher 2004], in which the global frequency of plaintext is collected and the encryption keys are distributed by traversing each node in the chain topology. The *clustered collection* approach is applied to the tree-based network [Llc 2010]. We group the network nodes into several clusters, where each cluster includes one cluster head and a set of in-cluster nodes. We identify one node as the network coordinator who serves as the root node in the clustered collection framework. In this approach, each cluster head plays as the intermediate node between in-cluster nodes and network coordinator. The network coordinator is responsible for collecting the global frequency of plaintext and distributing encryption keys back to each cluster across the network. We assume that the communication channel is secure so that the attacker is unable to compromise the process of frequency collection and key dividing.

5.2. Message Format

During the two-round communication process, we define two types of communication messages, *frequency collection message* and *key dividing message*, which carry out the required information for both the incremental and clustered collection frameworks.

Frequency collection message: The frequency collection message is propagated to each node to collect the frequency information of plaintext. It includes the plaintext values and their corresponding frequency. Its format is defined as following:

Definition 5.1. Given k unique plaintext values $PT_j (1 \leq j \leq k)$, let f_j be the frequency of the plaintext value PT_j. The payload of PT_j in *Frequency Collection Message* is shown as:

$$\|PT_j : f_j\| \tag{14}$$

For the incremental collection approach, in each frequency collection message, f_j represents the *accumulated frequency* of the plaintext value PT_j on nodes from n_1 to n_i, whereas for the collected collection approach, f_j indicates the *local frequency* of PT_j at one particular node. The message format is illustrated in Figure 4.

Key dividing message: The key dividing message carries the information for dividing encryption keys assigned to each node or cluster. The packet format is shown in Figure 5. The definition of the key dividing message is given as following:

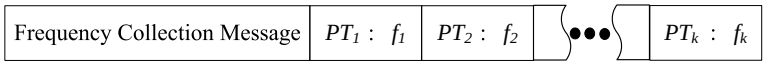


Fig. 4: Packet format of frequency collection message

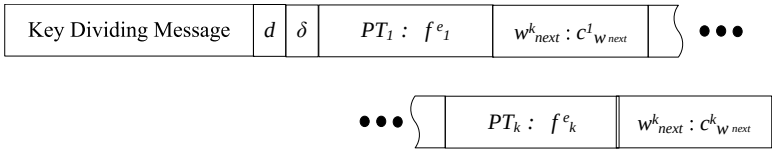


Fig. 5: Packet format of key dividing message

Definition 5.2. Given k unique plaintext values $PT_j(1 \leq j \leq k)$, let f_j^e be the emulation frequency of PT_j, w_{next}^j be the next available encryption key for PT_j, and $c_{w_{next}}^j$ be the available capacity of w_{next}^j, the payload format for PT_j of the **key dividing message** for both the incremental and clustered collection approach is defined as:

$$\|PT_j : f_j^e : w_{next}^j : c_{w_{next}}^j\| \tag{15}$$

For the incremental collection approach, inside the key dividing message sent by node $n_i(1 \leq i \leq N-1)$, f_j^e is the *remaining global emulation frequency* of PT_j for the next neighbor n_{i+1}, and w_{next}^j and $c_{w_{next}}^j$ are the next available encryption key and capacity for n_{i+1}. On the other hand, for the clustered collection approach, f_j^e is the *local emulation frequency* of PT_j assigned to a particular node or cluster, and w_{next}^j and $c_{w_{next}}^j$ are the next available encryption key and capacity for this node or cluster.

5.3. Incremental Collection Approach

The incremental collection approach is applied to the networks of chain-based topology as depicted in Figure 6(a). The first node, n_1, is the initial node for both frequency collection and key dividing processes. We next discuss the two-round communication process by steps.

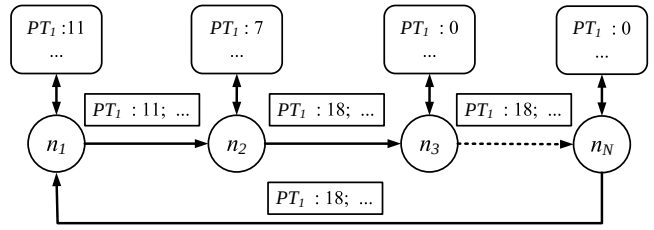
Frequency Collection (1st round): The frequency collection process consists of two steps.

Step 1.1. Frequency Initialization: Node n_1 initializes the frequency collection message. The message contains all local plaintext values on n_1 and their corresponding frequencies. The frequency collection message is then passed to the neighbor n_2 of n_1 on the routing path of the chain topology. For example, consider the plaintext PT_1 in a chain topology as shown in Figure 6(a), assume its frequency f_1^1 on node n_1 is $f_1^1 = 11$. Then in the frequency collection message, the payload of plaintext PT_1 is set as $[PT_1 : 11]$. After that, the frequency collection message is sent to n_2.

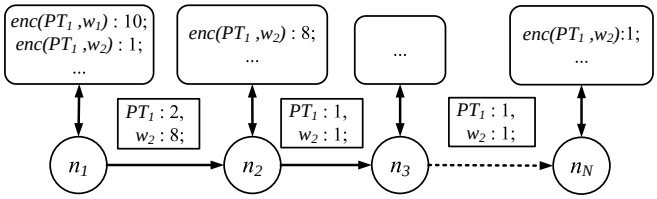
Step 1.2. Frequency Accumulation: Node $n_i(2 \leq i \leq N-1)$ receives the frequency f_j^{i+1} of $PT_j(1 \leq j \leq k)$ from n_{i-1}, and updates the frequency of PT_j by accumulating with its local frequency f_j^i. Then n_i passes the frequency collection message with the updated accumulated frequency $f_j^{i+1} + f_j^i$ of PT_j to its neighbor n_{i+1}. Each node on the routing path repeats the procedure until the last node n_N is reached. Finally n_N sends

the frequency collection message including global frequency f_j of each plaintext PT_j back to n_1.

Continuing the example, assume the local frequency of PT_1 on node n_2 is $f_1^2 = 7$. Then the total frequency of PT_1 is updated to $f_j = 18$ by summing up the frequencies on n_1 and n_2. The updated frequency is then sent to node n_3. This process continues until all the nodes in network are traversed.



(a) Incremental Collection: Frequency collection



(b) Incremental Collection: Key dividing

Fig. 6: Illustration of Incremental Collection framework

Key Dividing (2nd round): The key dividing procedure assigns the encryption keys to each node. The main challenge during this process is to update each node with the available emulation frequency of particular plaintext values:

Step 2.1. Parameter Calculation: Given the global frequency f_j of PT_j, n_1 calculates the dividing factor d based on the required security guarantee in Equation 5, and determines the minimum interval δ between any two successive plaintext values. According to the dividing factor d, n_1 obtains the global emulation frequency $f_j^e = d - (f_j mod\ d)$ for PT_j.

Continuing the example, assuming the dividing factor $d = 10$, and the frequency of PT_1 after emulation is 20, then the global emulation frequency $f_1^e = 2$. In addition, the minimum interval of plaintext is δ.

Step 2.2. Key Dividing Initialization: With the frequency knowledge f_j^1 of plaintext PT_j, its emulation frequency on n_1 is determined as follows: first n_1 picks a random number, i.e., $f_{j,1}^e$, ranging from 0 to f_j^e as PT_j's emulation frequency on n_1, and subtracts $f_{j,1}^e$ from f_j^e. Then the remaining global emulation frequency that is available for the next node is updated as $f_j^e - f_{j,1}^e$; in the meanwhile, PT_j on n_1 is encrypted to $\lceil (f_j^1 + f_{j,1}^e)/d \rceil$ ciphertexts with identical frequency d, but the last ciphertext may not be exactly divided by d. The next available encryption key for PT_j is $w_{next}^j = w_{\lceil \frac{f_{j,1}+f_{j,1}^e}{d} \rceil}$, and its remaining capacity $c_{w_{next}}^j$ is $d - (f_{j,1} + f_{j,1}^e) mod\ d$.

Continuing the example, starting from the initial node n_1, since its local occurrence frequency of PT_1 is 11, the global emulation frequency of PT_1, $f_j^e = 2$, is determined. First n_1 generates a random emulation frequency, $f_{1,1}^e = 1$, of PT_1 at n_1. The frequency after emulating for PT_1 will be up to $f_{1,1}^e + f_1^1 = 12$. n_1 needs two encryption weights w_1 and w_2 for encrypting PT_1, where 10 for w_1 and 2 for w_2, and the capacity of w_2 left for next node is $d - 2 = 8$. In addition, the remaining global emulation frequency decreases to $f_1^e = 1$.

Step 2.3. Message Encapsulation: n_1 initializes the key dividing message, including dividing factor d, minimum interval δ, the plaintext PT_j and its remaining global emulation frequencies f_j^e. In addition, the next available encryption key w_{next}^j and its remaining capacity are also included. All these information will be sent to n_2.

Continuing the example, node n_1 transmits the global remaining emulation frequency $f_1^e = 1$ and the remaining capacity, $c_{w_2}^1 = 8$, of the next encryption weight w_2 to its neighbor n_2, such as $[PT_1 : 1, w_2 : 8]$.

Step 2.4. Key Dividing Iteration: For each node $n_i (1 \leq i \leq N-1)$, it generates a random local emulation frequency $f_{j,i}^e$ for PT_j, and subtracts it from the remaining global emulation frequency f_j^e. Next, the total plaintext frequency of PT_j on n_i is updated $f_{j,i}^e + f_j^i$; then n_i re-uses the encryption key w_{next}^j passed from n_{i-1} to fill the corresponding remaining capacity $c_{w_{next}}^j$. If $c_{w_{next}}^j < (f_{j,i}^e + f_j^i)$, the other $f_{j,i}^e + f_j^i - c_{w_{next}}^j$ plaintext will be encrypted by w_{next}^j's following weights; otherwise, all PT_j appeared on n_i are used to fill the remaining capacity of w_{next}^j. Finally, n_i updates the next available encryption weight and corresponding remaining capacity. If $c_{w_{next}}^j < (f_{j,i}^e + f_j^i)$, $w_{next}^j = w_{l+\lceil \frac{f_{j,i}^e + f_j^i - c_{w_{next}}^j}{d} \rceil}$, where l is the index of next encryption key before updating, and $c_{w_{next}}^j = d - (f_{j,i}^e + f_j^i - c_{w_{next}}^j) mod\ d$; otherwise, w_{next}^j maintains, and $c_{w_{next}}^j = c_{w_{next}}^j - f_{j,i}^e - f_j^i$.

Continuing the example, at n_2, it gives a random local emulation frequency, $f_{1,2}^e = 0$, which means there is no emulation. Then PT_1 of the local occurrence frequency $f_1^2 = 7$ will be encrypted by w_2. Since the remaining capacity of encryption weights for w_2 is 8, which is larger than f_1^2, all PT_1 are encrypted with the same weight w_2. After encryption, the remaining capacity for w_2 becomes 1. This process will repeat on the all nodes at the path, until the last node n_N is reached.

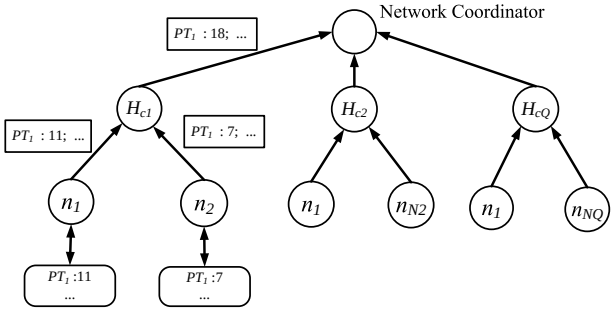
Step 2.5. Key Dividing Ending: When the key dividing message reaches the last node n_N, no matter how many PT_js exist in n_N, the frequency of PT_j will be $f_{j,N-1}^e$ passed from n_{N-1} after emulation. At the end the ciphertext values of the plaintext PT_j achieve global uniform distribution in the network.

Continuing the example, finally, no PT_1 is found on n_N. However, since the remaining global emulation frequency f_j^e passed from n_{N-1} is not zero, it emulates $f_j^r = 1$ ciphertext via w_2. Therefore, the global uniform distribution of the ciphertext values of PT_1 is achieved.

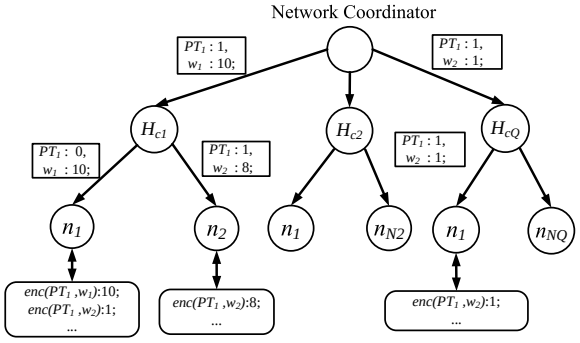
5.4. Clustered Collection Approach

In this section, we present the clustered collection approach which is more suitable for more advanced network topology [Llc 2010]. We use a three-layer tree network topology as an example, where the three layers represent the leaf nodes, the cluster heads, and the network coordinator respectively. The nodes in the network form several clusters. Each cluster elects a cluster head, which is also a node in the network. During the frequency collection procedure, each cluster head collects the plaintext frequency

information from the nodes within its cluster and forwards the total frequency information of its cluster to the network coordinator. During the key dividing procedure, the network coordinator distributes the encryption keys to each cluster. Then the cluster head enforces dividing and emulating by further dividing the encryption keys to each leaf node within the cluster. Figure 7(a) illustrates the framework for the clustered collection. Next, we discuss the details of both the frequency collection and encryption key dividing procedure.



(a) Clustered Collection: Frequency collection



(b) Clustered Collection: Key dividing

Fig. 7: Illustration of Clustered Collection framework

Frequency Collection (1st round): there are two main steps in frequency collection.

Step 1.1. In-cluster frequency collection: The frequency collection starts from the leaf nodes. Assume the leaf node n_i belongs to the cluster c_t. The node n_i transmits the frequency collection message including its local occurrence frequency f_j^i for its plaintext $PT_j (1 \leq j \leq k)$ to its cluster head H_{c_t}. All other nodes in cluster c_t follow the same way to send their local frequency collection messages to H_{c_t}.

For example, given the network topology shown in Figure 7(a), each leaf node first sends its local plaintext frequency to its cluster head. The cluster c_1 has two nodes n_1 and n_2. Assume the frequencies of PT_1 on these two nodes are: $[PT_1 : 11]$ and $[PT_1 : 7]$, and other clusters have no PT_1. The cluster head H_{c_1} will receive two frequency collections messages from PT_1 and PT_2.

Step 1.2. Cluster based frequency collection: After the cluster head H_{c_t} collects all the plaintext frequency within its cluster c_t, it sums up the frequency of PT_j as f_{j,c_t}.

Next, the each cluster head constructs a new frequency collection message including all the plaintext and their frequencies, and forwards the message to the network coordinator for further processing.

Continuing the example, the cluster head H_{c_1} sums up the frequencies of PT_1 and sends the total in-cluster frequency information $[PT_1 : 18]$ to network coordinator.

Key Dividing (2nd round): The key dividing procedure consists of 3 major steps.

Step 2.1. Parameter calculation: The network coordinator calculates the global frequency f_j of each plaintext $PT_j (1 \leq j \leq k)$ from the frequency collection messages collected from all cluster heads. Next, the network coordinator determines the dividing factor d and calculates the global emulation frequency $f_j^e = d - (f_j mod\ d)$. Continuing the example, the network coordinator calculates the global occurrence frequency of PT_1 $f_j = 18$. Then it calculates the dividing factor as 10, and $f_j^e = 10 - (18 mod\ 10) = 2$.

Step 2.2. Cluster-based key dividing: The network coordinator assigns each cluster a random emulation frequency, f_{j,c_t}^e, from the global emulation frequency f_j^e, where $\sum_{t=1}^{Q} f_{j,c_t}^e = f_j^e$, where Q is total number of clusters in the network. It also distributes the next available encryption weights w_{next}^j and its corresponding capacity $c_{w_{next}}^j$ to each cluster. For cluster c_1, combining its local frequency $f_j^{c_1}$ of PT_j and emulation frequency f_{j,c_1}^e, the total frequency of PT_j becomes $f_{j,c_1}^e + f_j^{c_1}$. The next plaintext PT_j in cluster c_1 is assigned with the encryption weights from w_1 to $w_{\lceil (f_{j,c_1} + f_{j,c_1}^e)/d \rceil}$. But the frequency of the last ciphertext may not be as high as d, thus the remaining capacity for $w\lceil (f_{j,c_1}^e + f_{j,c_1})/d \rceil$ is $d - (f_{j,c_1}^e + f_{j,c_1})\ mod\ d$. As shown in Figure 5, the key dividing message sent to cluster c_1 includes the dividing factor d, the emulation frequency f_{j,c_1}^e on cluster c_1, the next available encryption weight w_1, and its corresponding remaining capacity 10. Generally speaking, the updated message sent to cluster c_t includes the dividing factor d, its next available encryption weight $w_{next}^j = w_{\lceil \sum_{i=1}^{t-1} f_{j,c_i}^e + f_{j,c_i}/d \rceil}$, and the remaining capacity for this encryption weight $c_{w_{next}}^j = d - \sum_{i=1}^{t-1} f_{j,c_i}^e + f_{j,c_i}\ mod\ d$. The last cluster will apply emulation to make the capacity of last encryption weight of PT_j 0.

Continuing the example, network coordinator begins to distribute the encryption weights to each cluster. First, the network coordinator generates a random emulation frequency $f_{j,c_1}^e = 1$ for c_1, so the total frequency within c_1 is 19. Consequently PT_1 requires two encryption weights w_1 and w_2, and the remaining capacity for w_2 is 1. The key dividing message for cluster c_1 is $[PT_1 : 1, w_1 : 10]$. The network coordinator then distributes the remaining emulation frequency of PT_1 to other clusters, although there is no PT_1 in other clusters. As shown in Figure 7(b), c_2 is assigned with emulation frequency 0, and 1 for c_Q. The key dividing message to cluster c_Q is $[PT_1 : 1, w_2 : 1]$.

Step 2.3. In-cluster key dividing: Next, each cluster head acts as the cluster coordinator to divide the encryption weights among its leaf nodes. In particular, the cluster head assigns a random emulation frequency, f_{j,n_i}^e, to node n_i in the cluster. Following the same principle of key dividing among clusters, the key dividing message from the cluster head to node n_i consists of the dividing factor d, the first available encryption weight $w_{next}^j = w_{\lceil \sum_{i=1}^{N_t - 1} f_{j,n_i}^e + f_{j,n_i}/d \rceil}$, and the remaining capacity for this encryption weight $c_{w_{next}}^j = d - \sum_{i=1}^{N_t - 1} f_{j,n_i}^e + f_{j,n_i}\ mod\ d$. At the end of this round of communication, all clusters will achieve the global uniform distribution.

Continuing the example, once the encryption weights on clusters are determined, the cluster head H_{c_1} distributes the encryption weights among its cluster. In particular, H_{c_1} gives no emulation frequency assignment to n_1, but 1 for n_2. For n_1, PT_1 uses 10 w_1 and 1 w_2 for encryption; for n_2, 8 w_2 for PT_1. The key dividing message within the cluster should be: $n_1 : [PT_1 : 0, w_1 : 10]$ and $n_2 : [PT_1 : 1, w_2 : 8]$. For cluster c_Q, although no PT_j exists there, it also has the emulation frequency $f_1^{c_Q} = 1$. Therefore the corresponding key dividing message sent to the node n_{N_Q} in c_Q is $n_{N_Q} : [PT_1 : 1, w_2 : 1]$. After dividing and emulation executed on each individual node, the global distribution homogenization is completed.

5.5. Communication Overhead Analysis

We now discuss the communication overhead of the two approaches in terms of the number of packet exchanges during frequency collecting and key dividing. For the communication overhead, we consider two main factors: (1) communication cost and (2) time performance. We measure the communication cost as the number of packets transmitted during the frequency collection and key dividing processes. We evaluate time performance as the time cost for completing the two rounds of communication processes.

5.5.1. Incremental Collection Framework. We assume that the frequency collection and key dividing messages can be fit into single packet Assume the number of nodes in the networks is N, the *communication cost* C_{incr} is:

$$C_{incr} = C_{incr}^{FreqCol} + C_{incr}^{KeyDiv} = N + (N-1) = 2N - 1 \tag{16}$$

where $C_{incr}^{FreqCol}$ and C_{incr}^{KeyDiv} are the communication cost of the frequency collection and the key dividing procedure respectively. Next, regarding the time cost, we find that the time cost increases with the growth of the network size, due to the fact that the incremental collection approach needs to traverse all the node in the network. We assume that for each node in the network, its neighbor is always in its communication range; therefore all messages can be transmitted without any relay. We use T_0 to denote the time unit of transmitting the (frequency collection or key dividing) message between two neighbor nodes. The *time performance* T_{incr} of the incremental collection approach is:

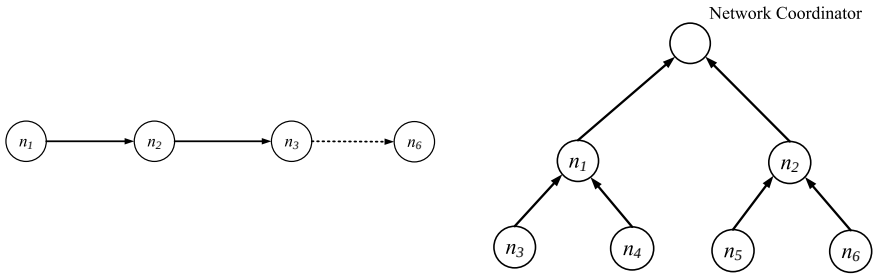
$$T_{incr} = T_{incr}^{FreqCol} + T_{incr}^{KeyDiv} = NT_0 + (N-1)T_0 = (2N-1)T_0 \tag{17}$$

where $T_{incr}^{FreqCol}$ and T_{incr}^{KeyDiv} are the time cost during frequency collection and key dividing respectively.

5.5.2. Clustered Collection Approach. Each cluster head collects the frequency of plaintext PT_j within the cluster, and passes the cluster frequency information to the network coordinator. Therefore, the *communication cost* C_{incr} of the clustered collection approach is:

$$C_{incr} = C_{incr}^{FreqCol} + C_{incr}^{KeyDiv} = N + N = 2N \tag{18}$$

Comparing with the incremental collection approach, the clustered collection approach has one additional network coordinator node that is not used for data storage but for frequency collection and key dividing. Therefore, comparing with the incremental collection approach, the communication cost of the clustered collection approach involves two extra messages, one for frequency collection and one for key dividing, from the network collector.



(a) Incremental collection (b) Clustered collection

Fig. 8: Implementation of both Incremental and Clustered Collection framework

However, since the frequency collection and the key dividing procedures of each cluster can take place simultaneously, whereas the incremental collecting messages have to traverse node by node, the clustered collection approach can be more efficient than the incremental collection. For example, assume there are G clusters of equal sizes, each of $N_G = \lceil \frac{N}{G} \rceil$ nodes. Since all clusters carry out frequency collection and key dividing simultaneously, the total time cost is $O(G + N_G)$. The *time performance* T_{incr} of the clustered collection approach is:

$$T_{incr} = T_{incr}^{FreqCol} + T_{incr}^{KeyDiv} = [(G + N_G)T_0] + [(G + N_G)T_0] = 2(G + \left\lceil \frac{N}{G} \right\rceil)T_0 \quad (19)$$

In particular, the time cost between network coordinator and all cluster heads equals to $G \times T_0$, and the time cost within each cluster equals to $N_G \times T_0$.

5.6. Experimental and Simulation Results

In this section, we perform both experimental and simulation evaluation of the time performance for the two proposed framework approaches. Our testbed-based experimental study focuses on investigating the feasibility of using two frameworks to defend against the global frequency based attack in a small network, while our simulation evaluation provides a more comprehensive performance evaluation of our approaches in networks at larger scales.

Metrics. We measure the *time performance* as the time cost of completing all the required communication process of the encryption framework.

Experimental Evaluation. Our key construction/update mechanism for defending against the global frequency based attack involves two parts: the communication for frequency collection and key distribution, and the specific key construction/update on each involved node. The key construction/update for defending against local frequency based attack only involves the specific dataset on each involved node. Implementation of key construction/updates on the sensor nodes are feasible, since previous work has shown that some popular public key cryptography techniques (e.g., elliptic curve cryptography [Liu and Ning 2003b]) can be implemented and executed on wireless sensors with small overhead in terms of energy consumption and ROM/RAM consumption. We expect that integrating our dividing/emulating method with those techniques will not introduce overwhelming amounts of overhead, especially with the more powerful sensor nodes these days, e.g., smartphone and tablet.

We build a testbed of a wireless sensor network using MicaZ sensor motes [Crossbow Tech. Inc.]. Each MicaZ sensor mote is equipped with a 2.4-2.48 GHz Chipcon CC2420

Preamble(bytes)	Frame Check Seq.(bytes)	Synchronization(bytes)
4	2	1
Frame Payload(bytes)	Frequency Band(MHz)	Address(bytes)
Depends on message type and number of plaintext	2400-2483.5	2
Pkt Trans. Rate(pkt/s)	RF Power(dBm)	Data Seq. Number(bytes)
10	0	1
Frame Control Field(bytes)	Receive Sensitivity(dBm)	Frame Length(bytes)
2	-90	1

Table II: Testbed configuration

Radio, 512KB flash memory, TinyOS 2.x as the operating system, and 802.15.4 protocol. Our testbed configures each sensor mote with B-MAC protocol [Polastre et al. 2004], a carrier sense multiple media access protocol for wireless sensor networks that provides a flexible interface to obtain ultra low power operation, effective collision avoidance, and high channel utilization. When using the B-MAC protocol, we configure the node to use adaptive clear channel assessment (CCA), which is an essential component of carrier sense multiple access (CSMA), the de-facto medium access control (MAC) protocols in many wireless networks. In particular, each network node is only allowed to transmit packets when the channel is idle by using CCA as channel detection. Typically, CCA works as follows: before transmitting, a wireless device samples the ambient noise floor for a short period and it will transmit only if the sampled value is larger than a threshold. Studies [Polastre et al. 2004] have shown that adaptive-CCA, which adjusts the threshold based on the ambient noise floor, can achieve better throughput and latency than using a pre-determined threshold. Besides the CCA scheme deployed in MAC layer, the MAC layer frame components and other related configuration are listed in Table II. We find that, except for the frame payload that carries the real information of frequency collection and key dividing messages, the overhead of the frame is about 13 bytes in total, which has little impact on communication cost. Additionally, the packet transmission rate is set at 10 pkt/s in our testbed, which could ensure lower probability on packet collision during communication, and result in lower latency as well.

We test the proposed two frameworks using this testbed for five scenarios of network sizes of 2, 3, 4, 5 and 6 wireless nodes. In order to evaluate how the time cost varies with the network size, we gradually increase the size of the network starting from 2 nodes in both frameworks. In particular, for the incremental collection framework, we set up a chain topology with all the nodes in the network connecting with one another. In the chain topology, each node forwards the message along the topology until it reaches the destination. The messages exchanged in the framework are frequency collection message and key dividing message. Each node maintains a two-way communication link with its previous and next neighboring nodes. For the clustered collection framework, we setup a tree topology, with each node in the tree connected to at most two leaf nodes at the next lower level in the cluster. The tree topology starts from two tiers involving two nodes, and then grows up to three tiers for the five different network sizes. Different nodes could exchange messages with their leaf nodes simultaneously without interfering with each other. The experimental setup is shown in Figure 8.

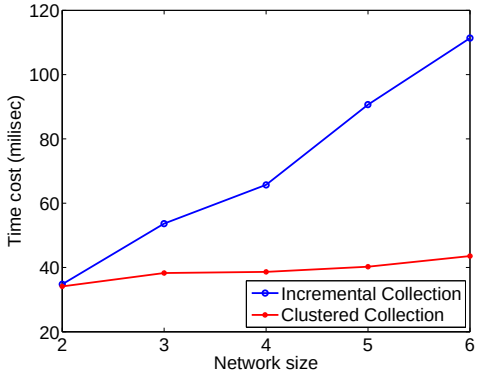


Fig. 9: Time Performance in real implementation of both frameworks

The time performance (in the level of milliseconds) on the frequency collection and key distribution for both topologies is shown in Figure 9. We observe that the time cost for incremental collection is lower than that in clustered collection at different network sizes. Since the incremental collection framework consists of the chain topology and needs to traverse the chain topology twice for frequency collection and key distribution. Furthermore, we find that the time cost for incremental collection increases faster than that for clustered collection, and the growing trend follows a linear relationship to the number of nodes in the network. Since all the clusters in the proposed clustered collection framework performs frequency collection and key distribution in parallel, the time cost for clustered collection approach is less sensitive to the size of network, which is inline with our theoretic analysis. In particular, the time cost increases from 35 ms to over 100 ms when the size of network increases from 2 to 6 nodes for incremental collection, while for clustered collection, the time cost only has a slight increasing, from 35 to 43 ms, when the tree topology increases from 2 tiers with 2 nodes to 3 tiers with 6 nodes. This observation indicates that using the clustered collection framework is more efficient when performing key updating in the network with the time cost grows slowly as the network size increases. In summary, since we find that the growing trend of the time performance under the incremental collection framework follows a linear relationship to the number of nodes in the network, the total communication overhead is still manageable under a large scale network with hundreds of sensor nodes. And with more clusters or each cluster involving more leaf nodes under the clustered collection framework in a large scale network, communication overhead will be much smaller than that in the incremental collection framework, making the clustered collection framework more compelling.

The results from running the testbed show the feasibility of applying our frameworks to real wireless sensor networks. Next, we perform a more comprehensive evaluation of the frameworks using simulation in a large-scale wireless network.

Simulation Evaluation. To further study in a large scale network setup, we conducted simulation of a wireless network with multiple nodes using Matlab. The data is collected and stored at each wireless node. We vary the number of nodes from 30 to 100. For the clustered collection approach, we choose four cluster of sizes, 3, 5, 10 and 20. We distribute the nodes into clusters uniformly. For each simulation setup, the total number of distinct plaintext values appeared in the network is not more than 200. The occurrence frequencies of plaintext values on each wireless node is uniformly distributed within the range of $[0, 100]$. We used the decipher probability valued as

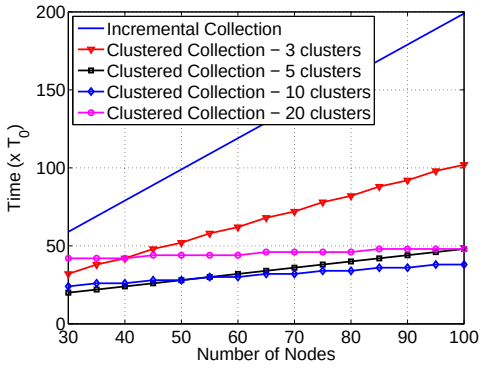


Fig. 10: Time performance of both frameworks

0.00001, 0.0001, 0.001, 0.01, 0.1 and 1. The simulation results are averaged over 100 runs.

Figure 10 shows the time performance of both approaches. First, we observe that the clustered collection approach is always faster than the incremental collection approach, regardless of the cluster size. Second, the time cost increases with the growth of the number of nodes; however, how fast the time increases varies for clusters of different sizes. Furthermore, we observe that the time performance of the clustered collection approach increases slower for larger clusters than the incremental collection approach. Third, there is no consistent trend of the overall time cost when the number of clusters increases. For example, when the cluster number increases from 3 to 5, the time cost decreases with the growth of the nodes. When the number of nodes increases from 40 to 200, for 3 clusters, the time cost increases from $32T_0$ to $102T_0$, and from $20T_0$ to $48T_0$ for 5 clusters.

Fourth, when the number of nodes is small, the time cost of a small number of clusters outperforms that of larger number of clusters. For example, considering the network of 30 nodes, 3 clusters costs 32 time units, whereas 20 clusters costs 42 time units. Since the size of each cluster is small for these small networks, the time cost of the communication between the network coordinator and the cluster heads dominates the total time. Therefore, the parallel operations of clusters indeed benefit the large networks and this is consistent with our findings in the experimental evaluation.

Encouragingly, large-scale simulation results show the similar trend to the observations obtained from our testbed-based experimental study. In both testbed-based experiments and simulation evaluation, we find that using the clustered collection framework is more efficient than using the incremental collection framework.

6. UPDATES ON DATA STORAGE

In previous sections, we only consider static data storage. Due to the fact that devices in wireless networks collect and store data all the time, it is important to deal with updates on (encrypted) data in the network. In this section, we discuss how to update data storage efficiently with provable security guarantee against the frequency-based attacks. First, we discuss two encryption schemes, *direct emulating* and *dual encryption*, that deal with updates on data storage. Second, we conduct a set of experiments to evaluate the performance of our two methods.

6.1. Encryption Schemes to Deal with Updates

Before we start discussing the details of how to update data storage, we must note that the *only* update operation on data storage is *data insertion*, due to the fact that the wireless devices keep collecting and storing new data all the time. A naive approach to encrypt the new data is to first decrypt all existing ciphertext values in the network, then encrypt all data at one time, including the newly inserted one. Obviously this approach is not affordable by the energy-constrained sensor nodes or mobile devices, due to its expensive computational cost. Thus, light-weight methods are needed for the application on sensor nodes and mobile devices. We develop two efficient approaches, the *direct emulation method* and the *dual-encryption method*, that enable secure and efficient data storage updates to cope with both global and local frequency based attack in the network.

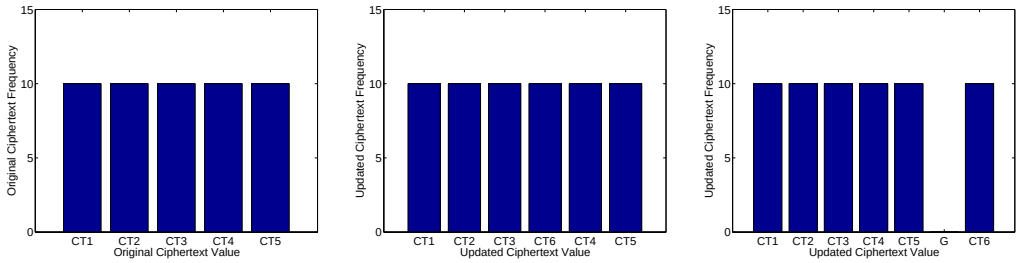
Direct Emulation Method. Assume at time t, a set of plaintext values $PT_j^t (j = 1, \cdots, J)$ with frequency $f_{j,t}$ are collected across the network. Our goal is to encrypt these new values without affecting the order of original ciphertext, while keeping the frequency distribution of the ciphertext values for both old and new ones uniform. There are two possible cases for each new plaintext value $PT_j (j \in [1, J])$.

— **Case 1: PT_j^t already exists in network.** For simplicity, we use PT_j to denote the existing occurrence of PT_j^t in the network, and PT_j^t the newly inserted value. For this case, the keys which have already been used to encrypt the old PT_j cannot be re-used on the newly inserted PT_j. But we use the same dividing factor as the one in original data storage. In particular, assume the dividing factor of the old PT_j is d. Let $w_1, w_2, \cdots, w_{\lceil \frac{f_j}{d} \rceil}$ be the $\lceil \frac{f_j}{d} \rceil$ keys that have been used for encrypting original PT_j. Then we use $\lceil \frac{f_{j,t}}{d} \rceil$ number of keys $w_{\lceil \frac{f_j}{d} \rceil + 1}, w_{\lceil \frac{f_j}{d} \rceil + 2}, \cdots, w_{\lceil \frac{f_{j,t}}{d} \rceil + \lceil \frac{f_j}{d} \rceil}$ for the new PT_j values, without overriding original ciphertext values. The overhead of dividing and emulating (defined in section 3) of plaintext PT_j^t is $d - mod(f_{j,t}, d)$.
If all $PT_j^t (1 \leq j \leq J)$ values exist in the network, the arrival of new data will only increase the number of distinct ciphertext values. Let m_o and m_u be the number of ciphertext before and after updates respectively, and k be the number of distinct plaintext values. It is straightforward that $m_u > m_o$. Next, we prove that the decipher probability will decrease, i.e., the security guarantee will improve, after the updates. The proof is shown in Equation 20. The decipher probability P_u after updates is:

$$P_u = \frac{1}{\binom{m_u - 1}{k-1}} \leq \frac{1}{\binom{m_u - 2}{k-1}} \leq \frac{1}{\binom{m_u - 3}{k-1}} \leq \cdots \leq \frac{1}{\binom{m_o}{k-1}} \leq \frac{1}{\binom{m_o - 1}{k-1}} = P_o. \tag{20}$$

where P_o is the decipher probability of the old data.
For example, let us consider the plaintext value PT_1 and PT_2 that have been mapped to three and two unique ciphertext values $CT_1, \ldots, CT_5$ with frequency 10 (Figure 11 (a)). Assume there are three new PT_1 values collected in the network. Since the first three encryption keys w_1, w_2 and w_3 have been taken by original PT_1, we use a new encryption key w_4 to encrypt the newly added PT_1. After the updates, PT_1 is encrypted to four ciphertext values, CT_1, CT_2, CT_3 and CT_6. For CT_6, the amount of emulating noise is $10 - 3 = 7$. Figure 11 (b) shows the ciphertext values after updates.

— **Case 2. PT_j^t does not exist in the network.** For this case, to avoid the overriding of the order of original ciphertext values in the network, a large value gap, G, between PT_j^t and the largest plaintext value existing in the network, is calculated first. In particular, G should be no less than $R+\delta$, where R is the range of old plaintext values,



(a) (b) (c)

Fig. 11: An example of the direct emulation method. (a) Before updates; (b) PT1 existed in network (3 PT1 updated and encrypted as CT6); (c) PT1 does not existed in network (3 PT1 updated and encrypted as CT6)

and δ is the minimal interval between any two successive plaintext values (defined in section 3). The newly inserted plaintext PT_j^t first is increased to be $PT_j^t + G$. Then we divide the increased PT_j^t by using the same dividing factor for encryption of existing plaintext values. During the emulating step, we duplicate $d - mod(f_j^t, d)$ ciphertext values for the largest split value of PT_j^t to achieve the uniform distribution.

For example, consider the case that there are two plaintext PT_1 and PT_2 in the original data storage that are encrypted to five ciphertext values CT_1 to CT_5 of frequency 10. Now a new plaintext PT_3 of frequency 3 is inserted into the data storage. Following our procedure, first, PT_3 is changed to be $PT_3' = PT_3 + G$. Then a new encryption key, $w_{1'}$ is applied on PT_3 for dividing, which results in a new ciphertext $CT_6 = enc(PT_3, w_1')$. The amount of emulated noise for PT_3 is also 7. The updated data storage is shown in Figure 11 (c).

Next, we prove that the security guarantee will not decrease after the updates for this case. Let k_o and m_o be the number of distinct plaintext and ciphertext values before updates, and k_u and m_u be the number of new plaintext and ciphertext values introduced by the updates. Then the decipher probability P_u after updates is $P_u = \frac{1}{\binom{m_o+m_u-1}{k_o+k_u-1}}$. Note that $m_o \geq k_o$ and $m_u \geq k_u$. Then

$$\begin{aligned} P_u &= \frac{1}{\binom{m_o+m_u-1}{k_o+k_u-1}} = \frac{1}{\binom{m_o+m_u-2}{k_o+k_u-2}\frac{m_o+m_u-1}{k_o+k_u-1}} \leq \frac{1}{\binom{m_o+m_u-2}{k_o+k_u-2}} = \cdots \\ &\leq \frac{1}{\binom{m_o+m_u-k_u-1}{k_o-1}} = \frac{1}{\binom{m_o+m_u-k_u-2}{k_o-1}\frac{m_o+m_u-k_u-1}{m_o+m_u-k_u-k_o}} \leq \frac{1}{\binom{m_o+m_u-k_u-2}{k_o-1}} = \cdots \\ &\leq \frac{1}{\binom{m_o-1}{k_o-1}} = P_o \end{aligned} \tag{21}$$

Dual-encryption Method. Although the direct emulation method can guarantee the required security guarantee after updating, it may incur expensive memory cost. For example, when there is only a single plaintext to be updated, there are $d-1$ ciphertext values that could be updated, which incurs the memory overhead that is $d-1$ times larger than the amount of new data. In the following, we will discuss how to reduce the memory cost without degrading the security guarantee.

Generally speaking, the amount of duplicated noise by emulating is decided by the dividing factor. If we can decrease the dividing factor, it is more likely to save much

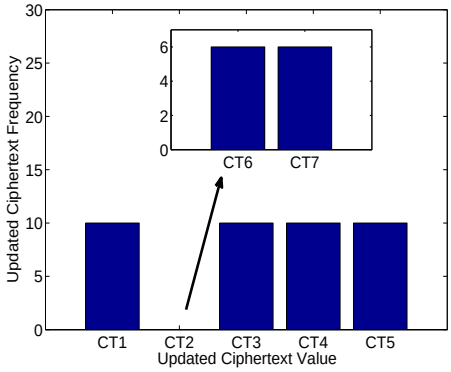


Fig. 12: Distribution of ciphertext values by dual encryption method after updates

memory space. In particular, assume node i receives multiple new plaintext values $PT_j^t (j \in [1, J])$ of occurrence frequency $f_{j,i}$ at time t. For each plaintext $PT_j^t (1 \leq j \leq J)$, we consider the following two cases:

— **Case 1: PT_j^t has appeared in original data storage.** For this case, there exists a set of ciphertext values CT_j of PT_j, each with occurrence frequency d. We apply another round of encryption on the new PT_j^t values together with those existing ciphertext values, by using the same dividing factor d. The encryption keys used for newly inserted plaintext values follow the same encryption key sequence as introduced in previous section. To be more specific, let m_o be the number of ciphertext values encrypted from plaintext PT_j in the original data storage, and w_o^l be the last encryption key of original plaintext PT_j. When there is new PT_j^t inserted into the data storage, the last ciphertext $enc(PT_j, w_o^l)$ is decrypted first, and combined with newly inserted PT_j^t. Let n_l be the number of PT_j encrypted with w_o^l, and n_u be the number of newly inserted PT_j. We generate $mod(n_u + n_l, d) - 1$ new encryption keys to encrypt the new inserted PT_j^t.

— **Case 2: PT_j^t has not appeared in original data storage.** We aim to insert PT_j^t into the place that still maintains the order-preserving relationship with all existing plaintext values. First, we identify two plaintext values PT^l and PT^u in original data storage such that: (1) PT^l and PT^u are the closest to PT_j^t, and (2) $PT^l < PT_j^t$ and $PT^u > PT_j^t$. PT_j^t will locate between PT^l and PT^u. Second, we calculate an interval value, δ_u, between PT_j^t and PT^l. Assume the previous plaintext PT^l has been already been encrypted as $enc(PT_{j_1}, w_u)$, where $u = 1, \cdots, v$. We create a new data interval $\delta_u = PT_j^t - (PT^l + \sum_{u=1}^{v} w_u \delta)$, which is less than δ. Third, we define a particular data interval δ_{new} that is less than δ_u to encrypt the newly inserted plaintext PT_j^t. Figure 12 shows an example of encryption for this case.

Due to the fact that the total number of distinct ciphertext does not decrease after updates, the security guarantee of the dual encryption method is always equal to that before the updates. In other words, the security guarantee does not degrade due to updates.

Query for updated dataset For the query on updated dataset, we still follow the three steps as that for original dataset in Section 3.

Phase-1: Query translation at user side. Similar as query on original dataset, we also assume a user can access all the auxiliary information including the weight values w_i, the gap values δ, and the order-preserving encryption function $enc()$ that are used in the dividing scheme. It follows the same way to translate the plaintext queries to ciphetext queries, i.e., a point query $Q : V = v$ is translated as $Q' : V \in [CT_1, CT_r]$, where $CT_1 = enc(v + w_1 * \delta)$, and $CT_r = enc(v + \sum_{i=1}^{r} w_i * \delta)$, and a range query $Q : V \in [l, u]$ will be translated to $Q' : V \in [CT_1, CT_r]$, where $CT_1 = enc(v + w_1 * \delta)$, and $CT_r = enc(v + \sum_{i=1}^{r} w_i * \delta)$.

Phase-2: Query evaluation at nodes in the network. The translated query $Q' : V \in [CT_l, CT_u]$ (for both point and range plaintext queries), where CT_l and CT_u are the lower bound and upper bound ciphertext values, will be sent to the network. Each node will return these ciphertext values that satisfy the query Q'. It also requires that at least two unique ciphertext values are returned for successful decryption in Phase-3. For dual encryption method, the updated ciphertext values do not affect the query evaluation. For direct emulating method, if the updated plaintext does not exist in original dataset, the updated ciphertext may not be within the query range due to adding the large gap value G during encryption. But these ciphertext values will be also returned to users.

Phase-3: Query post-processing at user side. After the user obtains the returned ciphertext values $CT_1, CT_2, \cdots, CT_t$, he/she will decrypt these values with the same procedures performed in original dataset. During decrypting, if the user encounters the ciphertext values do not match the weight values, it indicates that the ciphertext values have been re-encrypted in dual encryption method. We need to repeat the data decryption process as the way in original dataset by re-using the weight values. Further, for the returned ciphtertext values that are not within the query range due to direct emulating method, the user needs to decrypt them independently. After decrypted, the gap value G will be deducted from the decrypted values, and the original plaintext can be obtained.

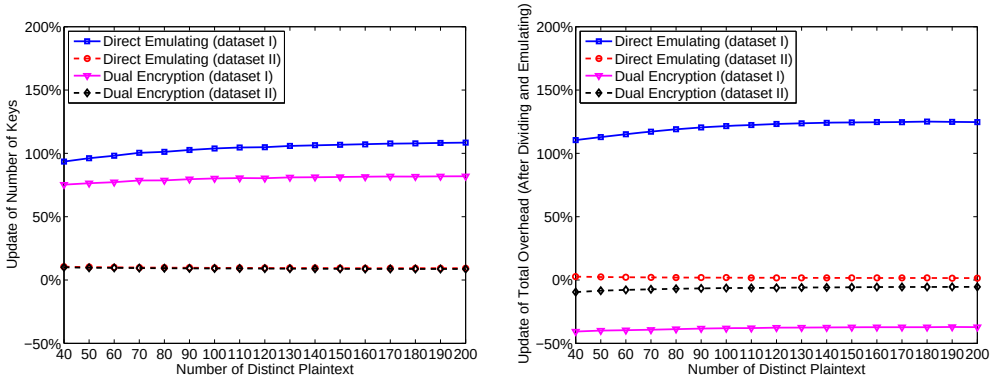
6.2. Simulation Evaluation

We ran a set of experiments to measure the performance of the two encryption approaches of handling updates. The simulation is conducted with Matlab. We tested on one network with size fixed as $N = 60$. For each simulation setup, we varied the total number of distinct plaintext values in the network from 40 to 200. The frequency of original plaintext values on each wireless node is of uniform distribution in the range of $[0, 100]$. The simulation results are averaged over 100 runs.

To evaluate the sensitivity of our approaches to the data distribution of updates, we create two different datasets. For dataset I, the frequency of updated plaintext are uniformly distributed in the range $[0, 30]$; for dataset II, the frequency of updated plaintext is within the range of $[20, 30]$, where the frequency of each updated plaintext is no less than 70% of the dividing factor in the original data storage.

6.2.1. Metrics. To evaluate the performance of our proposed approaches for updating, we developed the following two metrics.

Update of number of keys. We measure the updated number of encryption keys that are introduced by the data updates. For the global frequency-based attack, we define the *update of number of keys* as $\frac{k_u - k_o}{k_o}$, where k_u and k_o are the numbers of distinct ciphertext before and after data updates. For the local frequency-based attack, we define the *update of number of keys* as $\frac{\sum_{i=1}^{N} k_u^i - \sum_{i=1}^{N} k_o^i}{\sum_{i=1}^{N} k_o^i}$, where k_u^i and k_o^i are the number of distinct ciphertext before and after data updates on the i-th node $(1 \leq i \leq N)$. Intuitively, the closer the update of number of keys to 1, the smaller impact of data



(a) (b)

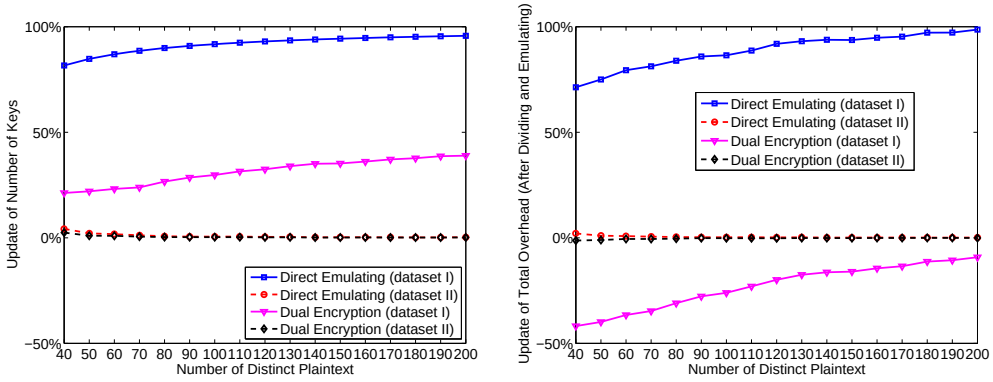
Fig. 13: Data updating: direct emulating and dual encryption methods when coping with local frequency-based attacks.

updates is on the encryption. We will evaluate the update of number of keys for various decipher probabilities, including both global and local frequency-based attacks.

Update of total overhead (after dividing and emulating). In addition to measuring the updated number of keys, we also quantify the update of total overhead after dividing and emulating. We define the *update of total overhead (by dividing and emulating)* as $\frac{S_u - S_o}{S_o}$, where S_u and S_o are the sizes of the overhead in memory cost before and after updating under global frequency based attack. Under local frequency based attack, this metric is defined as $\frac{\sum_{i=1}^{n} S_u^i - \sum_{i=1}^{N} S_o^i}{\sum_{i=1}^{N} S_o^i}$, where S_u^i and S_o^i are the sizes of overhead in memory cost before and after updating on node i. The update of overhead can be either positive or negative; positive value means the overhead increases after updates, while negative value means the overhead is reduced after the update.

6.2.2. Overhead of updating under local frequency based attack. We first examine the overhead of updating the data storage under local frequency based attack for both direct emulating method and dual encryption method. Figure 13 shows the overhead of updating varying with the number of distinct plaintext values, where the decipher probability is fixed at 0.01.

In Figure 13 (a), we show the updated number of encryption keys varying as the number of distinct plaintext for both direct emulating and dual encryption method under local frequency based attack. Both methods are sensitive to the distribution of updated data. For direct emulating method, the updated number of keys increases from 93% to 100.8% on dataset I, while decreases from 10% to 8.7% on dataset II. For dual encryption method, the updated number of encryption keys on dataset I increases from 75% to less than 82%, while decreases from 10.2% to 9.4% on dataset II. Both methods requires much fewer number of updated encryption keys on dataset II. The reason is that, for the direct emulating method, since the frequency of each updated plaintext of dataset II is close to the dividing factor used in the original data storage, it could largely reduce the possibility that each updated plaintext requires multiple encryption keys for encryption, and thus result in fewer number of updated keys. On the other hand, for the dual encryption method, since the frequency distribution of dataset II is restricted in a small range, it reduces the possibility of more encryption keys comparing with dataset I.



(a) (b)

Fig. 14: Data updating: direct emulating and dual encryption methods when coping with global frequency-based attacks.

Figure 13 (b) explores the update of total overhead after dividing and emulating, by varying the number of distinct plaintext values for both direct emulating and dual encryption methods. We observe that the overhead of the direct emulating method increases from 111% to 124% for dataset I, but decreases from 2.7% to 1.4% for dataset II. For the dual encryption method, the overhead slightly increases from -41% to -37% for dataset I, while increases from -9.5% to 5.4% for dataset II. From the observation above, the dual encryption is more suitable than the direct emulating approach regarding memory cost for both types of data updates.

6.2.3. Overhead of updating under global frequency based attack. The updating process under global frequency based attack is based on the framework we introduced in section 5. Figure 14 shows performance of our proposed direct emulating and dual encryption method on data updating with various number of distinct plaintext values.

Figure 14 (a) shows the updated number of encryption keys with various number of distinct plaintext values for both direct emulating and dual encryption methods on two different datasets. We observe that the two methods show different trend. For the direct emulating method, the updated number of keys increases from 81% to 95% for dataset I, while from 4.2% to almost 0 for dataset II. For the dual encryption method, the updated number of keys increases from 21.2% to 38.9% for dataset I, while around -0.01% for dataset II. Based on the observations, the dual encryption method requires fewer new keys to encrypt both datasets. Moreover, for database I, no matter how small the frequency of newly inserted plaintext values, the direct emulating method always requires lager number of encryption keys for encryption on the updated values. However, since the dual encryption method re-uses existing encryption keys in the original data storage, the number of updated keys is still affordable. For example, the number of updated keys by the direct emulating approach is over 40 for dataset I, and fewer than 2 for dataset II. On the other hand, the number of updated keys by the dual encryption approach for dataset I is over 10, and around 1 for dataset II.

Figure 14 (b) explores the update of total overhead after dividing and emulating with regards to various number of distinct plaintext values for both direct emulating and dual encryption method. We observe that the overhead of the direct emulating method increases from 71% to 98% for dataset I, but does not change on dataset II. For the dual encryption method, the overhead increases from -41.8% to -9% on dataset I,

while maintains at almost 0 for dataset II. This shows that both methods are sensitive to the updating dataset type. Since the frequency of each plaintext in dataset II is close to the dividing factor in original data storage, the direct emulating methold could emulate fewer new ciphertext values. However, since the dual encryption method reuses the existing encryption keys, the frequency of emulated ciphertext for dataset II does not influence the overhead of updating. Therefore, the dual encrypiton always produces better update overhead than that of the direct emulating method.

7. RELATED WORK

Distributed storage in wireless networks: In recent years, a few distributed in-network data storage and retrieval schemes [Ye et al. 2002; RatNasamy et al. 2002; Zhang et al. 2003; Newsome and Song 2003; Greenstein et al. 2003; Li et al. 2003; Ghose et al. 2003; Desnoyers et al. 2005; Fang et al. 2006] have been proposed for efficient data management. In the distributed in-network data storage schemes such as TTDD [Ye et al. 2002], DCS [RatNasamy et al. 2002; Newsome and Song 2003], TSRA [Desnoyers et al. 2005], HybridS [Ren et al. 2008], HASS [Xu and Jiang 2009], ZettaDS [Liu et al. 2008], the landmark-based information brokerage system [Fang et al. 2006], and others [Zhang et al. 2003; Greenstein et al. 2003; Li et al. 2003; Ghose et al. 2003], the data collected by a sensor node is stored either locally or at some designated nodes in the network, instead of transferred immediately to a centralized data center that is usually located out of the network. In our proposed work, a fully distributed data storage scheme is deployed, where the data collected are stored locally on individual nodes.

Security in data storage of wireless networks: The past research on sensor network security has focused on securing the information in communication. The research topics include key establishment [Perrig et al. 2001; Eschenauer and Gligor 2002; Liu and Ning 2003a], message authentication [Ye et al. 2004; Yang et al. 2005] and intrusion detection schemes [Wang et al. 2003]. However, securing information in the storage has not received adequate attention from the research community. To protect the stored data from being stolen by the attacker who may be able to compromise the storage nodes, data on each storage node should be encrypted. This raises the issue of how to simultaneously achieve data security in distributed data storage and retrieval systems. Distributed data are vulnerable to random Byzantine failures as well as data pollution attacks, in which the adversary can modify the data and/or inject polluted data into the storage nodes. [Nalin et al. 2007] proposes a series of schemes for securing both distributed data storage and distributed data retrieval. [Wang et al. 2009] presents an efficient and flexible dynamic data integrity checking scheme for verifying the consistency of data shares in a distributed manner. In distributed storage system, data should be encrypted with encryption keys which may compromised by attacker, then data is at the risk of being cracked by attackers. [Seitz et al. 2003] proposes an architecture that allows users to store and share encrypted in the environment. From the view of application, distributed data storage technique has also extended to many promising areas of wireless networks, such as network coding [Wang and Lin 2008], etc.

Defending against the frequency-based attack: Previous works [Wong et al. 2007; Molloy et al. 2009] consider how to defend against the frequency-based attack in the data-mining-as-service paradigm, i.e., the data mining computations are outsourced to a third-party service provider. For example, [Wong et al. 2007] proposes a substitution cipher technique on transactional data for secure outsourcing of association rule mining. It deployed a one-to-n item mapping that transforms transactions non-deterministically. However, the mapping scheme has potential security flaws; [Molloy et al. 2009] introduces an attack that could break the encoding scheme

in [Wong et al. 2007]. Some other works [Wang and Lakshmanan 2006; Agrawal et al. 2004] consider the frequency-based attack in the scenario of the database-as-service paradigm (i.e., outsourcing database and its management to a third-party service provider). The basic idea is to transform the dataset in a way that no matter what the frequency distribution of the plaintext dataset is, the ciphertext values always follow some given target distribution. Therefore, the attacker can not decide the mapping relationship between plaintext and ciphertext values by the frequency of plaintext and ciphertext values. [Wang and Lakshmanan 2006] proposes an approach that could transform the original occurrence frequency distribution of plaintext into uniform distribution. [Agrawal et al. 2004] and [Boldyreva et al. 2009] propose to transform the original occurrence frequency distribution to a certain target distribution, such as Gaussian distribution. However, all of these work coped with the frequency-based attack in a centralized framework; none of the work can be applied directly to the distributed data storage of wireless networks.

Additionally, initialization vectors (IVs) [Paar and Pelzl 2010], can also defend against the frequency-based attack effectively. However, it cannot support efficient query evaluation over encrypted data. The reason is two-fold. First, the initialization vectors do not support order-preserving encryption. This makes the rewriting of range-based queries (e.g., $V > p_v$) to be extremely difficult. Second, due to the fact that each plaintext value will be encrypted to be a unique ciphertext value, every point query with value-based constraint $V = p_v$ will be rewritten as $V = C_1$ or $C_2, \cdots$ or C_k, assuming that the set of p_v values is encrypted as $c_1, c_2, ...c_k$ respectively. Such query rewriting is cumbersome. Even worse, this may enable the attacker to infer the mapping between plaintext and ciphertext values by observing a sequence of point queries, as the ciphertext values of different plaintext values in these queries never overlap. In contrast, our encryption scheme always translates the value-based constraints (either point-based or range-based) in queries to ranges. This increases the difficulty of the attacker to crack the encryption scheme by comparing the queries.

8. CONCLUSION

In this paper, we address the security problem when the data is stored in each sensor node in a distributed manner in wireless networks. We consider a sophisticated attack model that the attackers possess the knowledge of frequencies of the original data in the network and utilize such knowledge to decipher the encryption on these data, both locally and globally. To cope with such frequency-based attacks, we design a 1-to-n encryption scheme based on our proposed dividing and emulating techniques. We show that our dividing and emulating techniques not only provide provable security guarantee against frequency-based attacks but also support efficient query evaluation over encrypted data. For the advanced case when the knowledge of the global frequency is not available, we propose two frameworks, incremental collection and clustering collection, to collect the global frequency across the network under different network topologies and distribute encryption keys back to each node to cope with the global frequency based attack. Furthermore, built upon our encryption scheme we further develop two mechanisms, direct emulating and dual encryption, to handle the data updates in the distributed wireless environment. Thus, our approach covers both static and dynamic data in wireless networks. Our testbed study using real sensor nodes demonstrates the feasibility of our approach in real environments. And the extensive simulation results confirm the effectiveness and efficiency of our approach.

9. ACKNOWLEDGMENTS

The preliminary results of this project have been published in DCOSS 2010 [Liu et al. 2010]. This work was supported in part by the National Science Foundation under grant numbers CNS1016303, CNS0954020, and CCF1018270.

REFERENCES

AGRAWAL, R., KIERNAN, J., SRIKANT, R., AND XU, Y. 2004. Order preserving encryption for numeric data. In *ACM SIGMOD international conference on Management of data (2004)*.

BOLDYREVA, A., CHENETTE, N., LEE, Y., AND O'NEIL, A. 2009. An introduction to roc analysis. *Lecture Notes in Computer Science 5479*, 224–241.

BOLDYREVA, A., N. CHENETTE, Y. L., AND O'NEILL, A. 2009. Order-preserving symmetric encryption. In *"The proceeding of the International Conference on the Theory and Applications of Cryptographic Techniques (EuroCrypt)"*.

CAPKUN, S. AND HUBAUX, J. P. 2005. Secure positioning of wireless devices with application to sensor networks. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*. 1917–1928.

CHEN, Y., TRAPPE, W., AND MARTIN, R. P. 2007. Detecting and localizing wirelss spoofing attacks. In *Proceedings of the Fourth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*.

CROSSBOW TECH. INC. White paper available at http://www.xbow.com.

DESNOYERS, P., GANESAN, D., AND SHENOY, P. 2005. Tsar: A two tier sensor storage architecture using interval skip graphs. In *3rd ACM Conference on Embedded Networked Sensor Systems (2005)*.

ESCHENAUER, L. AND GLIGOR, V. 2002. A key-management scheme for distributed sensor networks. In *9th ACM Conference on Computer and Communications Security (2002)*.

FANG, Q., GAO, J., AND GUIBAS, L. 2006. Landmark-based information storage and retrieval in sensor networks. In *25th IEEE International Conference on Computer Communications (2006)*.

GHOSE, A., GROSSKLAGS, J., AND CHUANG, J. 2003. Resilient data-centric storage in wireless ad-hoc sensor networks. In *4th International Conference on Mobile Data Management*. 45–62.

GIRAO, J., WESTHOFF, D., MYKLETUN, E., AND ARAKI, T. 2007. Tinypeds: Tiny persistent encrypted data storage in asynchronous wireless sensor networks. *Ad Hoc Networks, Elsevier 5*, 1073–1089.

GREENSTEIN, B., RATNASAMY, S., SHENKER, S., GOVINDAN, R., AND ESTRIN, D. 2003. Difs: a distributed index for features in sensor networks. *Ad Hoc Networks, 1*, 2-3, 333–349.

HATCHER, A. 2004. Algebraic topology. *ACM Mobile Computing and Communications Review 8*, 2, 50–65.

JOSHI, D., NAMUDURI, K., AND PENDSE, R. 2005. Secure, redundant, and fully distributed key management scheme for mobile ad hoc networks: an analysis. *EURASIP Journal Wireless Communnication Networks* 4, 579–589.

LI, X., KIM, Y., GOVINDAN, R., AND HONG, W. 2003. Multi-dimensional range queries in sensor networks. In *1st International Conference on Embedded Networked Sensor Systems (2003)*.

LIU, D. AND NING, P. 2003a. Establishing pairwise keys in distributed sensor networks. In *10th ACM Conference on Computer and Communications Security (2003)*.

LIU, D. AND NING, P. 2003b. Establishing pairwise keys in distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*.

LIU, H., WANG, H., AND CHEN, Y. 2010. Ensuring data storage security against frequency-based attacks in wireless networks. In *Proceedings of the 6th IEEE international conference on Distributed Computing in Sensor Systems (DCOSS2010)*. 201–215.

LIU, L., WU, Y., YANG, G., AND ZHENG, W. 2008. Zettads: A light-weight distributed storage system for cluste. In *3rd ChinaGrid Annual Conference (2008)*.

LLC, B. 2010. *Network Topology: Star Network, Grid Network, Tree and Hypertree Networks, Spanning Tree Protocol, Metro Ethernet, Token Ring, Mesh Networking*. General Books LLC.

MOLLOY, I., LI, N., AND LI, T. 2009. On the (in)security and (im)practicality of outsourcing precise association rule mining. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*. ICDM '09. IEEE Computer Society, Washington, DC, USA, 872–877.

NALIN, S., YANG, C., AND ZHANG, W. 2007. Securing distributed data storage and retrieval in sensor networks. In *5th Pervasive Computing and Communications (2007)*.

NEWSOME, J. AND SONG, D. 2003. Gem:graph embedding for routing and data-centric storage in sensor networks without geographic information. In *1st ACM Conference on Embedded Networked Sensor Systems (2003)*.

PAAR, C. AND PELZL, J. 2010. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer-Verlag New York Inc.

PERRIG, A., SZEWCZYK, R., WEN, V., CULLER, D., AND TYGAR, J. 2001. Spins: security protocols for sensor netowrks. In *7th ACM International Conference on Mobile Computing and Networking (2001)*.

PIETRO, R. D., MANCINI, L. V., SORIENTE, C., SPOGNARDI, A., AND TSUDIK, G. 2008. Catch me (if you can): Data survival in unattended sensor networks. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom)*.

POLASTRE, J., HILL, J., AND CULLER, D. 2004. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. 95–107.

RATNASAMY, S., KARP, B., YIN, L., YU, F., ESTRIN, D., GOVINDAN, R., AND SHENKER, S. 2002. GHT: A geographic hash table for data-centric storage. In *ACM International Workshop on Wireless Sensor Networks and Applications*.

REN, W., REN, Y., AND ZHANG, H. 2008. Hybrids: A scheme for secure distributed data storage in wsns. In *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (2008)*.

SEITZ, L., PIERSON, J., AND BRUNIE, L. 2003. Key management for encrypted data storage in distributed systems. In *2nd IEEE International Security in Storage Workshop (2003)*.

SHAO, M., ZHU, S., ZHANG, W., AND CAO, G. 2007. pdcs: Security and privacy support for data-centric sensor networks. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*.

SHENKER, S., RATNASAMY, S., KARP, B., GOVINDAN, R., AND ESTRIN, D. 2003. Data-centric storage in sensornets. *ACM SIGCOMM Computer Communication Review archive 33*.

WANG, G., ZHANG, W., CAO, G., AND PORTA, T. L. 2003. On supporting distributed collaboration in sensor networks. In *IEEE Military Communications Conference (2003)*.

WANG, H. AND LAKSHMANAN, L. V. 2006. Efficient secure query evaluation over encrypted xml database. In *32nd International Conference on Very Large Data Bases (2006)*.

WANG, N. AND LIN, J. 2008. Network coding for distributed data storage and continuous collection in wireless sensor networks. In *4th International Conference on Wireless Communications, Networking and Mobile Computing (2008)*.

WANG, Q., REN, K., LOU, W., AND ZHANG, Y. 2009. Dependable and secure sensor data storage with dynamic integrity assurance. In *28th IEEE International Conference on Computer Communications (2009)*.

WONG, W. K., CHEUNG, D. W., HUNG, E., KAO, B., AND MAMOULIS, N. 2007. Security in outsourcing of association rule mining. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB '07. VLDB Endowment, 111–122.

XIE, M., WANG, H., YIN, J., AND MENG, X. 2007. Integrity auditing of outsourced data. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB '07. 782–793.

XU, Z. AND JIANG, H. 2009. Hass: Highly available, scalable and secure distributed data storage systems. In *12th International Conference on Computational Science and Engineering (2009)*.

YANG, H., YE, F., YUAN, Y., LU, S., AND ARBAUGH, W. 2005. Toward resilient security inwireless sensor networks. In *6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (2005)*.

YANG, J., CHEN, Y., AND TRAPPE, W. 2008. Detecting sybil attacks in wireless and sensor networks using cluster analysis. In *the Fourth IEEE International Workshop on Wireless and Sensor Networks Security (IEEE WSNS)*.

YE, F., LUO, H., CHENG, J., LU, S., AND ZHANG, L. 2002. A two-tier data dissemination model for large-scale wireless sensor networks. In *8th ACM International Conference on Mobile Computing and Networking (2002)*.

YE, F., LUO, H., LU, S., AND ZHANG, L. 2004. Statistical en-route filtering of injected false data in sensor networks. In *3rd IEEE Conference of Communications Society (2004)*.

ZHANG, W., CAO, G., AND PORTA, T. L. 2003. Data dissemination with ring-base index for wireless sensor networks. In *IEEE International Conference on Network Protocols*.

Received XXXX; revised XX; accepted XX