Encrypting Wireless Network Traces to Protect User Privacy: A Case Study for Smart Campus

Luqiao Zhang[†], Ozgur Oksuz[*], Levon Nazaryan[*], Chaoqun Yue[*], Bing Wang[*], Aggelos Kiayias[‡], Athanasios Bamis[§]
[*]Department of Computer Science & Engineering, University of Connecticut, Storrs, CT, USA
[†] School of Network Engineering, Chengdu University of Information Technology, Chengdu, China
[‡] National and Kapodistrian University of Athens, 15784 Athens, Greece
[§]Seldera LLC.

***Abstract*—Wireless network traces have been widely used to understand human behaviors and provide value-added services. Sanitization based techniques have been shown to be severely lacking in protecting sensitive user information embedded in such traces. In this paper, we take an encryption based approach that provides much stronger protection of user privacy. One challenge in encrypting wireless network traces is how to encrypt time range while maintaining the utility of the traces. We propose two practical encryption techniques to support queries that involve time range. These two techniques provide much stronger security guarantee than existing order preserving encryption schemes, and present different tradeoffs in complexity, as well as storage and network bandwidth requirement. Last, we quantify the performance of the proposed approach using a smart campus prototype. The results show that our approach only leads to moderate increase in storage, network bandwidth and computation overhead, demonstrating the practicality of our approach.**

***Index Terms*—privacy, smart campus, wireless network**

I. Introduction

Mobile devices, especially pocket-size devices such as smartphones, are effective "human sensors" because they are physically small and often carried by their owners. As such, wireless network traces that record the movement of mobile devices have been widely used to study human behaviors. For instance, they have been used to understand user mobility patterns [28], [10], [23], inter-meeting times [19] and group behaviors [18], leading to realistic models and much insight that are valuable for scientific research. They have also been used for value-added services, for instance, detecting anomaly in physical space [4], actuating HVAC systems in smart buildings [2], and locating emergencies in campuses [14].

The intimate relationship between mobile devices and their owners also means that mobile wireless network traces often contain sensitive user information (e.g., trajectory of a user over a period of time). To protect user privacy, a standard practice is sanitizing the traces. For instance, the identities of the users (e.g., their user names or MAC addresses) are anonymized through a one-to-one mapping. Recent studies, however, have shown that sanitization based techniques are severely lacking in preserving user privacy [30], [22]. The intuition is that human mobility is rather unique and predictable [28], [10]. Therefore, even when the identity is anonymized, a user can still be identified easily based on her unique mobility signature. Remedy solutions such as generalization (e.g., reducing the resolution in location information) and perturbation (e.g., changing time information) provide better privacy protection at the cost of potentially reducing the utility of the network traces.

In this paper, we take an *encryption* based approach that differs drastically from existing approaches. Specifically, we propose a framework where data are encrypted once being captured and their processing is performed in a privacy-preserving fashion without them being ever revealed in plaintext to the data processing server. This design provides much stronger protection of user privacy compared to various anonymization techniques. One challenge in encrypting wireless network traces is how to encrypt time range (e.g., the time range that a device is associated with a WiFi access point), where the order in time must be preserved to maintain the utility of the traces. While order preserving encryption schemes [1], [5], [26] can preserve the order in time, they reveal the order of the data, which is a significant amount of information. In this paper, we develop an effective solution to resolve this challenge. Our main contributions are as follows:

- We formulate a secure time range matching problem and propose two practical techniques to solve it. One is a simple quantization based technique and the other is based on a novel application of Delegatable Pseudorandom Function (DPRF) [20]. We rigorously analyze the security of these two techniques, and show that both techniques provide much stronger security than order preserving encryption schemes. They also present different tradeoffs in complexity, security, as well as storage and network bandwidth requirement.
- We quantify the performance of our proposed approach in a case study of smart campus, where we explore two representative applications that are supported by querying encrypted campus WiFi network traces to make a campus "smarter." The evaluation shows that our approach only leads to moderate increase in storage, network bandwidth and computation overhead, demonstrating the practicality

The work of the first author was done while he was visiting UConn. The first two authors share lead authorship.

TABLE I
A SNIPPET OF A WIFI NETWORK TRACE.

ID	Location	Start time	Duration (s)
1023	AP_1	09:01 October 16, 2015	300
2034	AP_2	16:10 October 16, 2015	4200

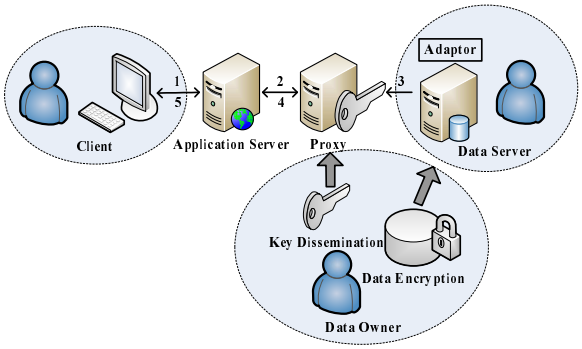


Fig. 1. System architecture and workflow.

of our approach.

The rest of the paper is organized as follows. Section II describes the problem setting and our high-level approach. Section III presents the two techniques for secure time range matching and security analysis. Sections IV and V present the prototype implementation and performance evaluation, respectively. Section VI briefly describes related work. Last, Section VII concludes the paper and outlines future directions.

II. PROBLEM SETTING

Consider a wireless network trace, e.g., a WiFi or cellular network trace. We assume the trace contains identity, spatial and temporal information. For instance, in a WiFi network, the information can describe when a mobile device is associated with an access point and the duration of the association; in a cellular network, this can describe when a phone is connected to a cellular tower and the duration of the connection. For simplicity, we assume it is of the form depicted in Table I.

As mentioned earlier, instead of anonymization, we take an encryption based approach to protect user privacy. Our goal is two-fold: providing provable security, while being simple and practical. We next describe the system architecture, the main focus of our study, and the security model.

A. System Architecture

To allow legacy client applications to be used directly in our system, we use a proxy based architecture to make the data encryption transparent to the client (as in [31], [27]). The architecture, illustrated in Fig. 1, contains a client, an application server, a proxy, and a data server. The client issues plaintext query to an application deployed at the application server. The data server stores the encrypted data, which can be stored in a database or simply a text file. For ease of exposition, we assume that the data are stored in a database in the rest of the paper. As the data are stored as ciphertext, a proxy equipped with encryption keys is introduced as a intermediary between a client application and the data server. Last, an adaptor is introduced at the data server as a middleware to handle operations that are not supported by standard database queries. Multiple clients can send queries simultaneously to the data server (through the application server and the proxy). We treat client and user interchangeably in this paper.

The workflow of our system is illustrated in Fig. 1. For initialization, the data owner encrypts the data, stores the encrypted data in the database, and sends secret keys to the proxy through a secure channel. The handling of a client query is as follows: (1) The client, after proper authentication, submits query to the application server; (2) The application server verifies that the query satisfies the access control policy, relays the query to the proxy after necessary parsing; (3) The proxy converts the query into ciphertext, and sends it to the data server; (4) The data server retrieves the data from the database, the adaptor performs necessary processing, and returns the results to the proxy; and (5) The proxy decrypts the data and returns the plaintext results to the application server, which relays to the client. We assume standard authentication technique is being used (e.g., through user name and password). Access control can be defined so that a client is restricted to submit certain queries. The detail is beyond the scope of this paper; we only briefly discuss issues related to our schemes in Section III.

B. Focus of the Study

For the wireless trace illustrated in Table I, one primary challenge is how to encrypt time range. The other fields in the table are represented as single values; how to encrypt such values while supporting various queries has been studied extensively in the literature. For instance, the study [31] presents efficient and provably secure methods for such fields. We therefore mainly focus on how to encrypt time range in this paper. While order preserving encryption schemes can preserve the order in time, the order in time that they reveal may expose the identity of a user. For instance, suppose that we can infer that a user visits buildings A, B, A and C (in the order of time) every Monday, where A is the Computer Science building, and B and C are classroom buildings. Suppose only one faculty in Computer Science teaches in building B and then in building C every Monday, then we can infer that the user is likely to be this faculty even though we do not know exactly what time the user visits these buildings. Our goal is developing practical encryption techniques for time range that do not reveal order while maintaining utility of the data.

The problem we focus on is *secure time range matching*, where the query involves a time range and each row in the database includes a time range field, and the goal is to extract the rows that have time range overlapping with the queried time range for at least a threshold value Δ (in addition to other conditions in the query). As an example, suppose the query is to find all the connections that are between time t_1 and t_2. Then the corresponding secure time range matching problem is to find all the rows in the database with the time range t'_1 and t'_2 so that $[t_1, t_2] \cap [t'_1, t'_2] \geq \Delta$ (all operations

are in ciphertext). Suppose another query is to find all the connections that are at location AP_1 between time t_1 and t_2. In this case, besides the time range matching as before, an additional condition is that the location needs to equal to AP_1 (again the comparison is in ciphertext).

A solution to the secure time range matching problem includes the following four algorithms.

- $k \leftarrow \text{Setup}(1^\lambda)$: is a probabilistic algorithm run by the data owner for setting up the system. It takes as input security parameter λ and outputs a secret key k.
- $T \leftarrow \text{Enc}(k, T_p)$: is a deterministic algorithm executed by the data owner. It takes the secret key k and the plaintext database table T_p as input, and output an encrypted database table T.
- $q \leftarrow \text{Trapdr}(k, q_p)$: is a deterministic algorithm executed by the proxy. It takes as input the secret key k and a plaintext time range query q_p, and transforms the query to trapdoor (i.e., encrypted query) q.
- $R \leftarrow \text{Search}(q, T, \Delta)$: is a deterministic algorithm executed by the data server. It takes as input the trapdoor q, searches over the encrypted database T, and outputs the set of items that overlap with the query q by at least Δ.

C. Security Model

As in [31], [27], we focus on privacy; data integrity and availability are beyond the scope of this paper. Specifically, we consider a semi-honest adversary that has complete access to the data server. The adversary is curious, but does not change any data inside the database, does not change queries issued by the proxy or query results, and does not deviate from the protocol. In addition, we assume that the adversary also controls the adaptor at the data server.

The application server and the proxy are fully trusted. The authorized users are fully trusted by the data owner, and are authorized to access the database through queries. Our model does not allow collusion between the data server and any of the users. The access control between the data owner and the users is out of the scope of this paper. We aim to achieve two privacy goals: *data privacy* and *query privacy*, defined as follows.

Data Privacy. Roughly, data privacy means that the sensitive information (plaintext values) for both data records and queries are not disclosed. As in [31], we define a leakage function $\mathcal{L}$ consisting of $\mathcal{L}_1$ and $\mathcal{L}_2$, which correspond respectively to leakage from the data and the queries. Specifically,

$\mathcal{L}_1$. Everything in the encrypted database T, including the number of rows and columns, and the relationship of the values.

$\mathcal{L}_2$. The positions of the responses to encrypted queries, denoted as $P(q_1), \ldots, P(q_t)$, where $q_1, \ldots, q_t$ are the encrypted queries.

We say our system satisfies data privacy if, after a polynomially bounded number of queries, what the adversary has observed can be simulated by a probabilistic polynomial-time (PPT) algorithm $\mathcal{S}$ using only the leakage function $\mathcal{L}$ as input.

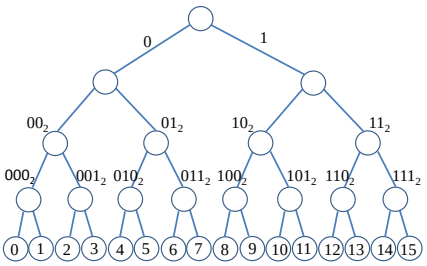


Fig. 2. Illustration of a 4-level GGM tree.

In other words, the adversary is not able to learn any useful information other than the leakage. This notion is defined and used in [31]. More formally, for any polynomial and all sufficiently large λ (security parameter), data privacy means that there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ (simulator) such that for any $t < \text{poly}(\lambda)$, any adversary $\mathcal{A}$,

$$|\Pr[\mathcal{A}_\lambda(q_1, \ldots, q_t, T) = 1] - \Pr[\mathcal{A}_\lambda(\mathcal{S}(P(q_1), \ldots, P(q_t), T)) = 1]| < 1/\text{poly}(\lambda).$$

where T is the encrypted database and poly() is a polynomial-time function.

Query Privacy. Query privacy means that the adversary cannot deduce any information about the queries that an authorized user makes. More specifically, any two different but the same size range queries remain indistinguishable by the adversary. We formalize query privacy through an indistinguishability-based game that runs in three phases: the first learning phase, the challenge phase, and the second learning phase. In other words, query privacy says that even the adversary has some information about some ranges and their corresponding trapdoors, he is not able to learn more information about the ranges that he does not know. Similar notions of query privacy are used in [20].

III. Secure Time Range Matching Schemes

We propose two practical schemes for secure time range matching. One is a basic scheme based on quantization and the other is based on a novel application of Delegatable Pseudorandom Function (DPRF) [20]. Both schemes consider a time range discretized by a given granularity. For instance, for a time range of 24 hours, at the granularity of one minute, the discretized time range is $[0, 1439]$[1], where a time point is mapped to a value corresponding to minutes that have elapsed since 00:00; a time point with finer granularity than minute is rounded to the closest minute.

A. Basic Scheme

To encrypt a time range $[a, b]$, the basic scheme simply lists all the values between a and b, and maps each value to a random value that is indistinguishable from other values.

[1]For ease of notation, we use $[a, b]$ to represent the integer interval of $\{a, a+1, \ldots, b\}$.

This can be achieved through a pseudorandom function (PRF). Specifically, we use the well-known tree-based GGM PRF family [15], proposed by Goldreich, Goldwasser and Micali. This family defines a PRF that takes a key k and a preimage x, and assigns it an image $f_k(x)$, such that $f_k(x)$ (for randomly chosen k) is indistinguishable from a uniformly random string of the same length. This PRF is based on the hierarchical application of any length-doubling Pseudorandom Generator (PRG) according to the structure induced by a tree, where input values are uniquely mapped to root-to-leaf paths. Specifically, let G be a publicly known PRG that takes a n-bit secret string $k \in \{0,1\}^n$ as input, and outputs a $2n$-bit string, $G(k)$. Let $G_0(k)$ and $G_1(k)$ denote respectively the first and second half of $G(k)$. The GGM pseudorandom function family [15] is defined as $\mathcal{F} = \{f_k : \{0,1\}^n \to \{0,1\}^n\}_{k\in\{0,1\}^n}$ such that $f_k(x) = G_{x_0}\left(G_{x_1}\left(\cdots\left(G_{x_{n-1}}(k)\right)\right)\right)$, where $(x_{n-1}\ldots x_0)_2$ is the binary representation of x.

As an example, Fig. 2 depicts a GGM tree with 4 levels. The leaves are labeled with a decimal number from 0 to 15, sorted in ascending order. Every edge is labeled with 0 (resp. 1) if it connects a left (resp. right) child. Every internal node is labeled with the binary string determined by the labels of the edges along the path from the root to this node.

The basic scheme is summarized as follows.

- $k \leftarrow \text{Setup}(1^\lambda)$: output a secret key k and the corresponding GGM tree.
- $T \leftarrow \text{Enc}(k, T_p)$: For a given secret key k and a plaintext database table T_p, output encrypted database table T, where the ciphertext for a time range record $[a,b]$ in T_p is $\cup_{x\in[a,b]} f_k(x)$.
- $q \leftarrow \text{Trapdr}(k, q_p)$: For a given secret key k and a plaintext time range query q_p, output trapdoor $q = \cup_{x\in q_p} f_k(x)$.
- $R \leftarrow \text{Search}(q, T, \Delta)$: For an input trapdoor q and every time range record r in the encrypted database T, if $r \cap q \geq \Delta$, then r is a matching record, and add r to R.

The data owner needs to encrypt a large number of time ranges, which can be accelerated through a lookup table. Specifically, we can create a lookup table that stores the PRF value for each leaf node in a GGM tree. Specifically, for a GGM tree with 2^n nodes, for leaf node $x \in [0, 2^n - 1]$, we store its PRF value $f_k(x)$ as the xth entry of the lookup table. Similarly, since the proxy is fully trusted, it can also store the table and use the table to speed up the conversion of a time range query to the corresponding ciphertext.

B. Improved Scheme

The basic scheme can lead to significant communication and storage overhead when the time range is large and the granularity is small since it maps every value in a time range to a PRF value individually. To reduce the overhead, we propose an improved scheme based on DPRF [20]. DPRF is a cryptographic primitive for delegating the evaluation of a pseudorandom function to an untrusted party, which enables the untrusted party to evaluate a PRF on a strict subset of its domain using a trapdoor derived from the DPRF secret key. The trapdoor is constructed to respect certain *policy predicate* that determines the subset of input values that the proxy is allowed to compute. When the policy predicate is described as (1-dimensional) ranges, the study [20] proposes one DPRF construction that provides query privacy, which we adopt for encrypting time ranges in our context (see details below). The DPRF values are computed using a GGM tree.

Specifically, for a given GGM tree, to encrypt a time range, the improved scheme finds a set of internal nodes in the tree whose subtrees (leaves) collectively covers the time range, and sets the ciphertext of the time range as the union of the ciphertext of the internal nodes as well as their corresponding heights. There may exist multiple sets of internal nodes that can cover a time range. One way is to use *Best Range Cover* (BRC) [20] that finds a set that contains the minimum number of internal nodes that covers the time range. While BRC is optimal in storage space, it may leak information. As an example, two ranges $[2,7]$ and $[9,14]$ have the same length of 6, while BRC uses two internal nodes 001_2, and 01_2 (with the heights of 1 and 2 respectively) for $[2,7]$, and uses four internal nodes 101_2, 1001_2, 110_2, 1110_2 (with the heights of 1, 0, 1, 0 respectively) for $[9,14]$, which are distinguishable. *Uniform Range Cover* (URC) [20] solves the above problem. It modifies the output of BRC to generate a uniform output for a given range size r (see more details in [20]). Using URC, the internal nodes for $[2,7]$ are 010_2, 0010_2, 011_2, 0011_2 (with the heights of 1, 0, 1, 0 respectively), the internal nodes for $[9,14]$ remain the same as before. Therefore the representations for these two intervals have the same number of elements with pairwise equal heights, and thus indistinguishable. While URC is less efficient than BRC, interestingly, it preserves the same storage overhead $O(\log \ell)$, where ℓ is the length of the interval. We adopt URC in this paper.

As in the basic scheme, the data owner or proxy can also use a lookup table to speed up data encryption when using the improved scheme. The lookup table needs to store the PRF values for both the internal and leaf nodes. We index a node of value x and depth (level) d as $I(d,x) = 2^d - 2 + x$, and store its PRF value in the $I(d,x)$th entry of the lookup table. For instance, in Fig. 2, node 0 (i.e., the left child of the root) is indexed as 0, node 1 is indexed as 1, node 00_2 is indexed as 2, node 01_2 is indexed as 3, and their PRF values are stored correspondingly in the lookup table.

A limitation of the above DPRF scheme is that, for a given internal node in the GGM tree, it leaks the underlying subtree structure. We use the following approach to limit the amount of leakage to individual subtrees of height h, while the relative order of the subtrees is not leaked. Specifically, we divide any range $[a,b]$ into a set of sub-intervals of $[a, i\times 2^h], [i\times 2^h + 1, (i+1)\times 2^h], \ldots, [(i+m)\times 2^h + 1, b]$, where $i \times 2^h$ is the smallest multiple of 2^h that is larger than a, and $(i+m)\times 2^h$ is the largest multiple of 2^h that is smaller than b. Then the ciphertext of $[a,b]$ is the union of the ciphertext of the set of sub-intervals (the order of the sub-intervals is randomly permuted before transforming them to ciphertext). For an interval of length ℓ, the storage overhead

is $O\left(\lceil \ell/2^h \rceil \log 2^h\right) = O\left(h\lceil \ell/2^h \rceil\right)$. The choice of h presents a tradeoff: a larger h leads to smaller storage overhead at the cost of larger leakage.

Algorithm 1 Query search in the improved scheme: check whether $r \in T$ matches query q

```
1: overlap = 0
2: for each (f_k(x), h_x) ∈ q do
3:   for each (f_k(y), h_y) ∈ r do
4:     if (h_x = h_y) then
5:       if (f_k(x) = f_k(y)) then
6:         overlap = overlap + 2^{h_x}
7:       end if
8:     else
9:       if (h_x > h_y) then
10:        Let S represent the set of partial PRF values
           when expanding f_k(x) to the height of h_y
11:        if (f_k(y) ∈ S) then
12:          overlap = overlap + 2^{h_y}
13:        end if
14:      else
15:        Let S represent the set of partial PRF values
           when expanding f_k(y) to the height of h_x
16:        if (f_k(x) ∈ S) then
17:          overlap = overlap + 2^{h_x}
18:        end if
19:      end if
20:    end if
21:    if overlap ≥ Δ then
22:      return true
23:    end if
24:  end for
25: end for
26: return false
```

We next describe how to query a time range when using the improved scheme. For an input trapdoor q and every time range record r in the encrypted database T, a naive way to check whether r overlaps with q is to expand both of them to the leaf level, and use the same approach as that used for the basic scheme. We next describe a more efficient scheme by comparing partial PRF values, as shown in Algorithm 1. The algorithm considers each element $(f_k(x), h_x) \in q$ and each element $(f_k(y), h_y) \in r$. If $h_x = h_y$, it compares $f_k(x)$ and $f_k(y)$ directly. If $h_x > h_y$, it expands $f_k(x)$ to the level of h_y, and then compares it with $f_k(y)$. Similarly, if $h_x < h_y$, it expands $f_k(y)$ to the level of h_x, and then compares it with $f_k(x)$. The variable, *overlap*, keeps track of the amount of overlap between q and r. If *overlap* $\geq \Delta$, then r is a matching record for q. Otherwise, r is a not matching record for q.

We now illustrate the above algorithm using an example. In Fig. 2, suppose the encrypted query is $\{(f_k(001_2), 1), (f_k(0001_2), 0)\}$ (corresponding to plaintext query $[1, 3]$), and the time range in a record is $\{(f_k(011_2), 1)\}$. The naive approach incurs 6 equality checks, while Algorithm 1 requires only 3 equality checks: one for comparing $(f_k(001_2), 1)$ and $(f_k(011_2), 1)$ since they are both at level 1, and two for comparing $(f_k(0001_2), 0)$ and the two expanded leaf nodes from $(f_k(011_2), 1)$.

C. Security Analysis

We next prove that the basic and improved schemes for encrypting time range preserve both data privacy and query privacy.

Theorem 1 (Data Privacy): The basic and improved schemes are data private in that they do not leak any more information than the leakage $\mathcal{L}_1$ and $\mathcal{L}_2$.

Proof: To prove this theorem, we need to show that for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that takes the input and output of the corrupted party (i.e., the server), and outputs a view that cannot be distinguished from the server's real protocol view. In other words, the adversary is not able to distinguish whether the given view is formed by interacting with a simulator or a real party. $\mathcal{S}$ takes $\mathcal{L}_1$ and $\mathcal{L}_2$, which consist of the encrypted table T, and t encrypted queries where t is polynomially bounded (since our main focus is on time range, we only consider time ranges in the table) that are input and output of the corrupted party (i.e., the server). Specifically, let $P(q_i)$ denote the set of entries and their corresponding positions that are hit (or matched) from table T corresponding to the ith queried range, $q_i = [a, b]$, where $i = 1, \ldots, t$. Assuming the size of the range is one (i.e., it contains only a single element, $a = b$) for each query. Since the simulator has the leakage $\mathcal{L}_2$, it assigns a random value in the positions that satisfy the query in table T to the same positions in the T' (a new table). Then, the simulator assigns random values to fill in the entire table using $\mathcal{L}_1$ and outputs corresponding $q'_1, \ldots, q'_t$. The case when the size of the range is larger than one can be shown in a similar way.

When time ranges are encrypted using the improved scheme, the simulation is a little bit more involved since the derivation should be consistent from each ith level to $(i-1)$th level in all records, where $i = 1, .., \alpha$ (α is the height of the tree). In this case, $\mathcal{L}_1$ is the same as that in the basic construction, which consists of the encrypted table T. Specifically, the size of the table, the number of unique partial PRF values for each row, their levels and positions. The leakage $\mathcal{L}_2$ consists of the observation of the t encrypted query results, which consist of the positions of the rows that satisfy the ith query, $i = 1, \ldots, t$. $\mathcal{L}_2$ also consists of which positions and levels of partial PRF values that satisfy the query in each row in the encrypted table T. To begin with, the simulator can create a GGM tree using a random value m' as the secret seed, and establish a one-to-one mapping between the nodes in the original GGM tree and the nodes in the new GGM tree. The simulator knows the total number of distinct PRF values in T. It also knows the height of the original GGM tree (e.g., by deriving all partial PRF values to the leaf level). Therefore, it can create a tree using m' so that the new tree is of the same height of the original tree. The simulator first simulates the entire table using $\mathcal{L}_1$ as follows: the simulator performs the following steps by first marking all the nodes in

the new tree as not used. It then performs the followings two steps: (1) Among all the PRF values in T, find the PRF value that is associated with the maximum height. Suppose that this PRF value is v and the height is h. The simulator finds the leftmost unused node at height h in the new tree. Let v' be its PRF value. Then the simulator marks the node (in the new tree) as used, maps v to v', and changes all instances of v with height h in table T and the t queries to v'. After that, the simulator maps the left (resp. right) child of v to the left (resp. right) child of v' in the new tree, marks these nodes in the new tree as used, and changes the values in T according to the mapping. The procedure repeats sequentially until all the children of v have been replaced. (2) Of the remaining PRF values in T, the simulator repeats step (1) until a one-to-one mapping is established for each of these values between the original and the new tree, and the values in both table T. In order to simulate encrypted queries, the simulator does the following. First assume that the size of the range is one (only one single element) for each query. Since the simulator has the leakage $\mathcal{L}_2$, it assigns the value of the same position in T' for each query. Then, the simulator outputs $T', q'_1, \ldots, q'_t$. The case when the size of the query is bigger than one can be shown in a similar way. ■

Query Privacy. Recall that query privacy means two different but the same size range queries are not distinguishable by the adversary. For the improved scheme, the proof of query privacy follows from the security proof of URC in [20] (Theorem 4). For the basic scheme, query privacy can be proved similarly since the basic scheme is a special construction of the improved scheme.

IV. Case Study: Prototype for Smart Campus

As a case study, we implement a prototype that encrypts university campus WiFi network traces for smart campus applications.

Smart Campus Case Study. In this case study, network traces collected from a university campus WiFi network are used to make the campus "smarter". In university campuses, people often prefer to connect their smartphones to campus WiFi networks (instead of cellular networks) for Internet access, because of their higher bandwidth, lower delay, lower cost, and lower energy consumption [3], [13], [21]. Therefore, the smartphones (and hence their owners) can be tracked through the access points (APs) that they are associated with. The knowledge of people's locations on a campus in realtime can make the campus "smarter", as has already been shown in several recent studies [4], [2].

We consider two representative applications in the context of smart campus. The first is *user presence check*, namely, checking whether a user is in a building or not during a time range. This can be used for forensic purpose, e.g., finding a missing person. The second application is *people counting*, namely counting the number of people in a building during a time range. This can be useful for indoor environment conditioning [2] or detecting anomaly in physical space [4].

Prototype Implementation. Since in practice many applications are web based, we implement the application server, proxy and adaptor as Servlets that run on web servers. Specifically, we use Apache Tomcat 7.0 web servers in the prototype. The encrypted data are stored in a MySQL server 5.6 database.

The wireless network traces that we use are AP association logs collected from the University of Connecticut campus WiFi network. During preprocessing, we extract the identity (MAC address), location (represented as building name), and time range of each association record from the traces. Both IDs and locations are encrypted using AES-128 in Electronic Code Book mode (the length of an ID or building name is short). The choice of AES-128 is for simplicity; we may use the schemes in [31] for better security at the cost of slightly more resource usage. The time ranges are encrypted using the basic and improved schemes, respectively. Last, the prototype uses SHA-256 as the PRG function, and Base64 to convert between binary data and ASCII strings.

The prototype supports the two smart campus applications that are described earlier. For user presence check, a query specifies an ID, a building name, and a time range. For people counting, a query specifies a building name and a time range. For both cases, a set of matching records are returned from the data server to the proxy, which in turn decrypts them and returns them to the client. To reduce the latency from converting a PRF value back to the corresponding time range, we also encrypt staying times using AES and store them in an additional column in the database.

V. Performance Evaluation

In this section, we evaluate the performance of the smart campus prototype. We organize the AP association trace into multiple databases, each containing up to d days of data. The choice of d presents a tradeoff: larger d provides better privacy at the cost of more resources. We set $d = 10$ in the rest of the paper. The time ranges are encrypted using minute time granularity (this granularity is sufficient for the two smart campus applications). As such, we use a GGM tree with $2^{14} > 24 \times 60 \times 10$ leaves. We consider two 10-day time periods, from 4/17/2013 to 4/26/2013 and from 10/16/2015 to 10/25/2015. In the interest of space, we only report the results for the former, which contains 1,524,850 records, 222 distinct buildings and 41,293 distinct user devices. The results for the latter show similar trend. We set the threshold $\Delta = 2$ minutes to avoid categorizing a user that simply passes by a building as one that stays in the building. For the improved scheme, we set h to 6, i.e., the amount of leakage is within individual subtrees of height 6 (containing up to 64 leaves, corresponding to 64 minutes, slightly larger than one hour).

For all the results below, we conduct experiments on three virtual machines (VMs). The client and application server are on one VM that has 4GB RAM and two cores of a 2.5 GHz Intel Core i5 CPU. The proxy is on another VM that has similar configuration. The adaptor and database are on the third VM that has 8GB RAM and one core of a 2.5 GHz Intel Xeon CPU.

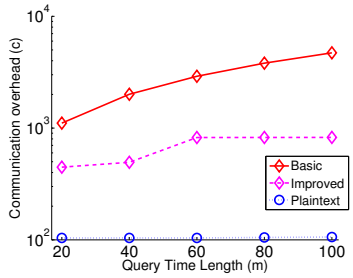


Fig. 3. Communication overhead (in characters or bytes), $d = 10$.

A. Data Encryption: Computation Time and Storage Space

The data owner encrypts the data and stores the ciphertext at the data server. On average, encrypting an ID and a location (building name) using AES takes 0.06 and 0.03 ms, respectively. Encrypting a time range using the basic and improve schemes is by using lookup tables as described in Section III. The time required to create the lookup table is 0.18 and 0.19 second for the basic and improved schemes, respectively.

The storage overhead for representing staying time under the improved scheme is significantly lower than that of the basic scheme. Specifically, a time range $[a, b]$ leads to $(b - a)$ PRF values under the basic scheme, and $O\left(h\lceil(b-a)/2^h\rceil\right)$ PRF values under the improved scheme. The average number of PRF values for one time range in our data set is 6.03 and 21.2 under the improved and basic schemes, respectively. The database sizes (including all the fields in the database as well as metadata) under the improved and basic schemes are 836 MB and 2.4 GB, respectively, $3.8\times$ and $11.7\times$ of the size of the plaintext database. For both schemes, the database size can be reduced when using more storage-efficient implementation for representing PRF values.

B. Communication Overhead

For communication overhead, we mainly quantify the size of a ciphertext query that is sent from the proxy to the data server. Fig. 3 plots the results, where the query time range increases from 20 to 100 minutes. The encrypted query under the improved scheme is a few hundred bytes, which can be included in a single packet. For the basic scheme, the encrypted query can be included in a few packets.

C. Computation Time

For a query, the computation mainly contains three parts: computation at the proxy (converting plaintext query into ciphertext, and decrypting query results), computation at the adaptor (processing data read from the database to obtain results for a query), and database reading time (reading data from the database corresponding to a query). The computation time at the proxy is negligible (below 5 ms for all the cases). We next only present the computation time at the server.

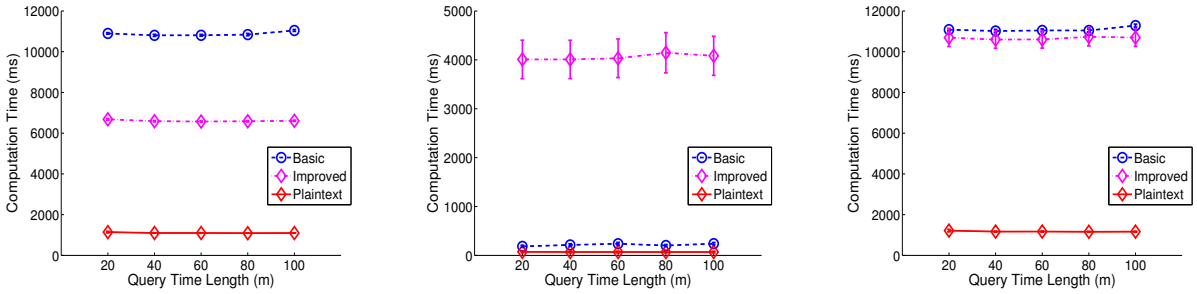
People Counting. For this application, we choose to query the two mostly visited buildings. The queried time range is randomly chosen, varied from 20 to 100 minutes. Only the queries that return non-empty set are considered when calculating the various overhead. For each setting, the results are obtained from 100 queries. The data server retrieves a set of entries that match the building name (encrypted) specified in the query from the database. Fig. 4(a) plots the average database reading time (with 95% confidence intervals). The average database reading time for the plaintext database is the lowest. For the encrypted databases under the basic and improved schemes, the average database reading time under the basic scheme is much larger, due to the much larger database size. The adaptor processes the retrieved data from the database by checking whether a retrieved record contains a time range that overlaps with the queried time range for more than $\Delta = 2$ minutes. Fig. 4(b) plots the latency at the adaptor. When using the basic scheme, both the retrieved and queried time ranges are represented by a set of PRF values of leaf nodes in a GGM tree, and hence can be compared directly through equality check. When using the improved scheme, the comparison is through Algorithm 1 (see Section III-B), which may involve deriving PRF values to a lower layer of the GGM tree, and hence incurs larger delay than that under the basic scheme. Fig. 4(c) plots the total latency (i.e., database reading time plus computation time at the adaptor). We observe the latency under the basic scheme is slightly larger than that of the improved scheme.

User Presence Check. For this application, we again vary the queried time range from 20 to 100 minutes. For each query, the data server first retrieves a set of entries that match the ID and building name (both encrypted) specified in the query. The adaptor then processes the retrieved records. Fig. 5 plots the results. The database reading time under the basic scheme is still much larger than that of the improved scheme. The latency at the adaptor under the improved scheme is only tens of milliseconds larger than that of the basic scheme since much less records are returned from the database for this application compared to that of people counting application. As a result, for the total latency, the difference between these two schemes is larger than that in the people counting application.

VI. Related Work

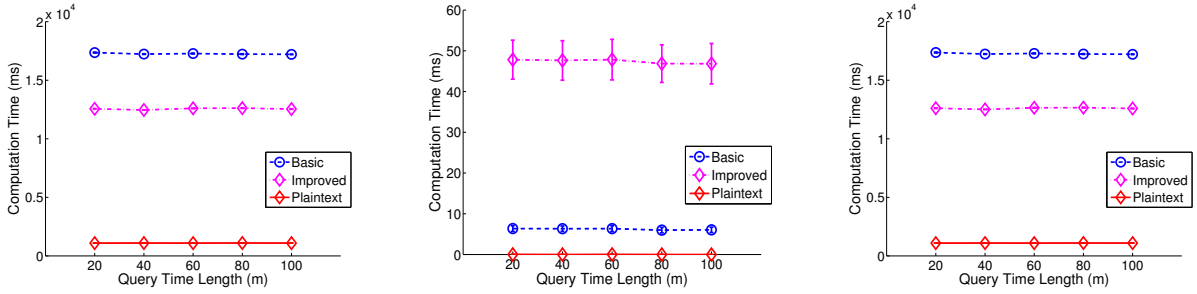
Our work is broadly related to searchable encryption, which allows search over encrypted data. A rich literature is on this topic (e.g., [29], [7], [9]). A recent study [11] points out that no off-the-shelf searchable symmetric encryption scheme supports range queries. Hence the authors reduce range search to multi-keyword search, introduce the first range searchable symmetric encryption (RSSE) framework and propose several practical RSSE schemes. Our work is independent of and in parallel with [11]. The key difference is that we formulate and solve a secure time range matching problem, where both the queries and database records are time ranges (in [11], only queries are ranges while database records are not ranges).

Many studies developed techniques to support queries over encrypted database. For instance, Yang et al. [31] proposed efficient and provable secure methods for queries on encrypted data stored in an outsourced database. Hacigumus et al. [16]



(a) Database reading time. (b) Computation time at the adaptor. (c) Total computation time.

Fig. 4. Computation overhead at the data server for people counting application, $d = 10$.



(a) Database reading time. (b) Computation time at the adaptor. (c) Total computation time.

Fig. 5. Computation overhead at the data server for user presence check application, $d = 10$.

proposed heuristic bucketization techniques that divide the domain of a column into partitions, randomly map the partitions, and then store the partition number for each data item. Hore et al. [17] explored the performance tradeoffs in setting the bucket sizes. Popa et al. [27] developed CryptDB that executes SQL queries over encrypted data using a collection of efficient SQL-aware encryption schemes. Our study differs in scope from them in that we focus on secure time range matching in wireless network traces, instead of general databases and general SQL queries.

Lu et al. [25] proposed a range query scheme that enjoys privacy preserving logarithmic complexity search using B+ structure. The scheme has several drawbacks. First, each range trapdoor size is $O(\log^2 L)$, where L is the number of leaves in the tree. The ciphertext size of a single point in a range is $O(\log L)$, also depending on L. Secondly, since the scheme uses B+, the ciphertexts are sorted in order and stored in the database server, the server can guess and figure out the order between a query value and a data value. The study in [24] proposed range query processing scheme that uses bloom filter data structure, which introduces unmatched data items (false positives). The study in [8] proposed a range query scheme where a user needs to interact with the data owner to get secure trapdoor, and hence the data owner is required to be online all the time. Our schemes do not have such a requirement.

One technique for supporting range queries is through order preserving encryption (OPE) schemes, where ciphertexts preserve the numerical orders of the corresponding plaintexts. The first OPE scheme was proposed by Agrawal et al. [1]. Boldyreva et al. [5] gave the first formal treatment of OPE. The authors later found that their proposed security guarantee, however, reveals half of the plaintext besides the order of the plaintext [6]. Popa et al. [26] proposed Order Preserving Encoding, which is an interactive protocol that requires stateful encryption functions. As shown in Section II, OPE reveals significant information and hence is not suitable for our scenarios. Our basic and improved schemes provide stronger privacy than OPEs and do not require interaction.

Last, another direction for preserving privacy in wireless network traces is through differential privacy techniques (e.g., [12]), which require adding noises and perturbation to the traces. Our approach in this paper is based on encryption that does not perturb any value in the network traces.

VII. Conclusion and Future Work

In this paper, we proposed an encryption based approach to protect user privacy in wireless network traces. Specifically, we presented two practical encryption techniques to encrypt time ranges that present different tradeoffs. We quantified the performance of the proposed approach in a case study of smart campus. Our evaluation shows that our approach only leads to moderate storage overhead and speed slowdown, demonstrating the practicality of our approach. Our study provides a new direction that uses encryption instead of anonymization for preserving user privacy in wireless network traces.

As future work, we will expand our prototype to support other applications. We will also investigate an approach that uses multiple proxies that is tolerant to attacks to proxies.

REFERENCES

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proc. of ACM SIGMOD*, June 2004.

[2] B. Balaji, J. Xu, A. Nwokafor, R. Gupta, and Y. Agarwal. Sentinel: Occupancy based HVAC actuation using existing WiFi infrastructure within commerical buildings. In *Proc. of ACM Conference on Embedded Networked Sensor Systems*, 2013.

[3] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc. of ACM IMC*, 2009.

[4] K. Baras and A. Moreira. Anomaly detection in university campus WiFi zones. In *Proc. of PerCom Workshops*, 2010.

[5] A. Boldyreva, N. Chenette, Y. Lee, and A. ONeil. Order preserving symmetric encryption. In *Proc. of EUROCRYPT*, April 2009.

[6] A. Boldyreva, N. Chenette, and A. ONeill. Order preserving symmetric encryption revisited: improved security analysis and alternative solutions. In *Proc. of CRYPTO*, 2011.

[7] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proc. of EUROCRYPT*, 2004.

[8] J. Chi, C. Hong, M. Zhang, and Z. Zhang. Privacy-enhancing range query processing over encrypted cloud databases. In *WISE*, 2015.

[9] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *CCS*, 2006.

[10] Y. A. de Montjoye, C. Hidalgo, M. Verleysen, and V. Blondel. Unique in the crowd: The privacy bounds of human mobility. In *Proc. of Nature Sci. Rep.*, 2013.

[11] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis. Practical private range search revisisted. In *Proc. of ACM SIGMOD*, 2016.

[12] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Proc. of EUROCRYPT*, pages 486–503, 2006.

[13] H. Falaki and S. Keshav. Trace-based analysis of Wi-Fi scanning strategies. *SIGMOBILE Mob. Comput. Commun. Rev.*, 13(1):73–76, 2009.

[14] A. A. Farhan, A. Bamis, and B. Wang. Locating emergencies in a campus using Wi-Fi access point association data. In *Proc. of ACM Workshop on Pervasive Urban Crowdsensing Architecture and Applications (PUCAA)*, 2013.

[15] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 1986.

[16] H. Hacigumus, B. Lyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database service provider model. In *Proc. of ACM SIGMOD*, pages 216–227, June 2002.

[17] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *Proc. of VLDB*, 2004.

[18] W. Hsu, D. Dutta, and A. Helmy. Extended abstract: Mining behavioral groups in large wireless LANs. In *Proc. of ACM MobiCom*, September 2007.

[19] W. Hsu, T. Spyropoulos, K. Psounis, and A. Helmy. TVC: Modeling spatial and temporal dependencies of user mobility in wireless mobile networks. In *Proc. of IEEE/ACM Transactions on Networking*, volume 17, pages 1564–1577, October 2009.

[20] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. In *Proc. of CCS*, 2013.

[21] K.-H. Kim, A. W. Min, D. Gupta, P. Mohapatra, and J. P. Singh. Improving energy efficiency of Wi-Fi sensing on smartphones. In *Proc. of IEEE INFOCOM*, 2011.

[22] U. Kumar and A. Helmy. Human behavior and challenges of anonymizing WLAN traces. In *Proc. of GLOBECOM*, pages 1–6, 2009.

[23] U. Kumar, N. Yadav, and A. Helmy. Gender based grouping of mobile student societies. In *Proc. of MODUS, IPSN workshop*, April 2008.

[24] R. Li, A. X. Liu, A. L. Wang, and B. Bruhadeshwar. Fast range query processing with strong privacy protection for cloud computing. *Proc. VLDB Endow.*, 2014.

[25] Y. Lu. Privacy-preserving logarithmic-time search on encrypted data in cloud. In *NDSS*, 2012.

[26] R. A. Popa, F. H. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *Proc. of Security and Prvacy*, 2013.

[27] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *Proc. of SOSP*, 2011.

[28] C. Song, Z. Qu, N. Blumm, and A.-L. Barabsi. Limits of predictability in human mobility. *Science*, 2010.

[29] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.

[30] K. Tan, G. Yan, J. Yeo, and D. Kotz. Privacy analysis of user association logs in a large-scale wireless LAN. In *Proc. of INFOCOM*, pages 31–35, 2011.

[31] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving queries on encrypted data. In *ESORICS*, 2006.

[32] L. Zhang, O. Oksuz, L. Nazaryan, C. Yue, A. Kiayias, and A. Bamis. Encrypting wireless network traces to protect user privacy: A case study for smart campus. http://nlab.engr.uconn.edu/smartcampus.pdf.