The following publication H. Hu, Q. Chen, J. Xu and B. Choi, "Assuring Spatio-Temporal Integrity on Mobile Devices with Minimum Location Disclosure," in IEEE Transactions on Mobile Computing, vol. 16, no. 11, pp. 3000-3013, 1 Nov. 2017 is available at https://doi.org/10.1109/TMC.2017.2683492.

Assuring Spatio-Temporal Integrity on Mobile Devices with Minimum Location Disclosure

Haibo Hu, *Member, IEEE,* Qian Chen, *Student Member, IEEE,* Jianliang Xu, *Senior Member, IEEE,* and Byron Choi, *Member, IEEE*

Abstract—Since the boom of smartphones and location-based services, spatio-temporal data (i.e., user locations with timestamps) have become increasingly essential in many real-life applications. To ensure these data are faithfully extracted from the underlying location tracking hardware and not altered by any malicious party or the user himself/herself, integrity assurance schemes such as digital signatures or message authentication codes (MAC) must be adopted. However, these conventional schemes disclose to the verifier the complete plaintext location and thus jeopardize users' privacy. In this paper, we propose an integrity assurance scheme with minimum location disclosure. That is, the granule of the disclosed location is just small enough to prove the user is/has been to a certain place, and the verifier cannot learn anything beyond it. To this end, we propose a new MAC scheme called Prefix-verifiable MAC (PMAC), based on which we design indexes and protocols to authenticate both spatial and spatio-temporal predicates. Security analysis and experimental results show our scheme is both secure and efficient for practical use.

Index Terms—Integrity assurance, spatio-temporal data, privacy protection

1 Introduction

Location-based services (LBS) have become increasingly popular in recent years, thanks to the intensive penetration of GPS-enabled smartphones and tablet computers. As more businesses and public services go mobile, spatio-temporal data (i.e., user locations with timestamps) become an essential input for many real-life applications. However, while location privacy has been under the spotlight in LBS research, the growing necessity for location integrity is often overlooked. In many real-life applications as listed below, the service provider must be assured of the genuineness of a mobile user's input location with respect to some *spatio-temporal predicate*. While a spatial predicate returns true or false about a relation (e.g., "inside") between the user location and a spatial geometry (e.g., a rectangular region), a spatio-temporal predicate augments it with a time interval, for example whether or not the user "had been in a rectangular region in this morning".

- **Auditing and compliance.** The location of a subject needs to be checked over time against some regulation. For example: (1) a taxi should not leave its operating area designated in its license; (2) a car rental requires the customer not to drive away from its service area for insurance coverage; (3) a field engineer is supposed to visit a service site during working hours; and (4) during a fishing or mining moratorium, a registered fishing vessel or a mining machine should not enter the region of moratorium.
- **Access control.** Some businesses or services need to verify the user's geographic location before authorizing access or providing services. For example: (1) a mobile ad network gives away coupons of a shop only to those users who are visiting a shop of its competitors [4]; (2) an online casino must not accept customers from states where online gambling is illegal [29]; and (3) an insurance company needs to assure that an online policy is physically signed in a jurisdiction where all clauses in this policy are legal [30].
- **Testimony.** A subject makes a claim of his/her historical location, which needs to be verified. For example, a passenger on a flight from a Zika-active country can waive further inspection by quarantine officers if there is a proof that he/she has not been to those outbreak zones [9].

Location integrity and provenance have been studied as a byproduct of localization in mobile computing [25], [13]. As such, most existing works assume trusted wireless infrastructure or third-party witnesses, such as nearby wireless access points [24] and co-located Bluetooth or WiFi devices in ad hoc mode [47], [48], [38]. However, these solutions cannot work outside of a well-controlled environment, which accounts for most location integrity use cases. Thanks to recent advances in CPU security (e.g., ARM's TrustZone™ and Intel's Trusted Execution Technology™), mobile OS nowadays can setup and bootload a trust environment for enterprise-level security tasks, e.g., Samsung KNOX™. As such, it is not only practical but also timely and of real potential to study location integrity schemes that purely rely on mobile devices themselves.

A naive method of authenticating a user location against a predicate is to disclose to the verifier the complete plaintext location, together with a proof (a digital signature or a message authentication code) generated by a trusted party (also known as the "authenticator") who has access to the genuine data. Unfortunately, this completely exposes

• *Haibo Hu is with the Department of Electronic and Information Engineering, Hong Kong Polytechnic University. Qian Chen, Jianliang Xu and Byron Choi are with the Department of Computer Science, Hong Kong Baptist University.*
E-mail:haibohu@polyu.edu.hk,{qchen, xujl,bchoi}@comp.hkbu.edu.hk

the user location and hence jeopardizes his/her privacy, **although all the verifier needs to authenticate is whether he/she has been to that region**.

In this paper, we study the problem of spatio-temporal integrity assurance that incurs minimum location disclosure. Specifically, the disclosed granularity of the location is just precise enough to prove the spatio-temporal predicate is true, and the verifier learns nothing beyond this. Further, to support a wide range of (future) applications (e.g., a sudden Ebola outbreak and emergent quarantine order), **the integrity proof should not assume any predicate a priori**. That is, a single proof, once generated, can authenticate the integrity against any upcoming predicates with variable region sizes and positions.

In cryptography, there are two mechanisms of integrity proof for a message string to a verifier, namely, digital signatures and message authentication codes (MAC). While digital signatures are based on asymmetric keys, MACs use symmetric keys — the authenticator and the verifier share the same key. As such, MAC is generally more efficient than digital signature scheme.

In this paper, we adopt MAC as the scheme for integrity assurance, and the major challenges are twofold: (1) how to convert the problem of spatio-temporal authentication to integrity assurance on message strings; and (2) how to assure string integrity without disclosing the strings. For the former, we encode locations and regions into strings and thus a spatio-temporal predicate becomes a prefix-matching predicate, i.e., location string x has region string q as its prefix. For the latter, we design a new MAC called *Prefix-verifiable Message Authentication Code* (PMAC). It has two unique properties: (1) it can prove a query string q is a prefix of a message string x, without disclosing x; (2) a single PMAC can authenticate against any query string q.

Based on PMAC, we propose an authentication scheme for spatial predicates and then extend it to spatio-temporal predicates. To speed up the verification process of the latter, we design two PMAC indexes, namely, the PS-tree and PS*-tree, which aggregate individual values for block verification. Two optimization techniques, one based on computational simplification and the other on space recoding, are also proposed to further improve the computation and communication costs. To summarize, our contributions made in this paper are as follows:

1) To the best of our knowledge, this is the first work that addresses spatio-temporal integrity assurance in a privacy-preserving manner without wireless infrastructure or third-party witnesses. The problem is critical in database research community, and has wide applications in mobile computing industry.
2) We design a prefix-verifiable message authentication code, based on which we develop authentication schemes for spatial and spatio-temporal predicates.
3) We design two PMAC indexes and two optimization techniques that reduce the computation and communication costs.
4) We conduct a rigorous security analysis and extensive experiments, which show the proposed schemes are both efficient and robust under various system settings.

The rest of this paper is organized as follows. Section 2 formally defines the problem and security model, followed by the prefix-verifiable message authentication code in Section 3. Section 4 presents the authentication scheme on static spatial predicates, and Section 5 extends it to spatio-temporal authentication with two indexing schemes, followed by the two optimization techniques in Section 6. Section 7 shows the experimental results, followed by a literature review in Section 8 and a conclusion in Section 9.

2 Problem Definition

In this paper, we study how a user authenticates his/her location to a third-party service against a spatial or spatio-temporal predicate while exposing the minimum location information. A *user location* is a 3-ary (a, b, t), where a, b are the user's longitude and latitude, and t is the location timestamp. As coordinates and timestamps have finite precision in practice, we assume they are all integers. The user needs to authenticate historical or current location to a verifier against some spatio-temporal predicates. According to [10], a *spatial predicate* returns true or false about a relation between the user location and a spatial geometry. In this paper, we focus on the *containment predicate*, i.e., whether the user "is" inside a rectangular window. A *spatio-temporal predicate* is augmented with a time interval, i.e., whether the user "has been" in this window. The meaning of "minimum location information" is twofold: 1) the user agrees to disclose to the verifier whether he/she is in the window or not; and 2) the verifier cannot learn anything about the user location beyond that. Note that in case of multiple predicates with different window sizes but the same timestamp, the user's location is disclosed as the intersection of these windows.

The system consists of three parties (shown in Fig. 1): (i) a location authenticator, (ii) a client (i.e., the prover), and (iii) a service provider (i.e., the verifier). The client, typically a mobile or web app, authenticates to the verifier, typically a web server, on behalf of a user. The location authenticator is a trusted software module in the mobile OS, which can be implemented in the Core Location framework of iOS and Google Location Services for Android. It accesses the location data from the underlying hardware (e.g., GPS module) and generates message authentication codes (MAC) for the client to use during verification. Note that similar to message authentication, location authentication has two phases, and the authenticator is only involved in the (offline) production phase (steps 1 and 2 in Fig. 1). In this phase, for each timestamp t, it produces a $MAC(x, t)$ for location x and stores the tuple of $(x, t, MAC(x, t))$ into an authentication data structure in (untrusted) persistent storage, such as the internal memory or an SD card. In the (online) verification phase, the verifier challenges the client with a predicate q, and the client uses the $MAC(x, t)$, timestamp t, and some additional proof to authenticate that x is inside q. In addition, there is an initialization phase where the verifier and the authenticator exchange the key for MAC (step 0 in Fig. 1), which takes place when the user first signs up that verifier (e.g., his/her company) for location authentication.[1] As for the security model, we assume the authenticator can be trusted as it is part of the OS. Thanks to recent advances in CPU security (e.g., ARM's

1. For those verifiers who only appear after the production phase, this key exchange can take place anytime before the verification phase.

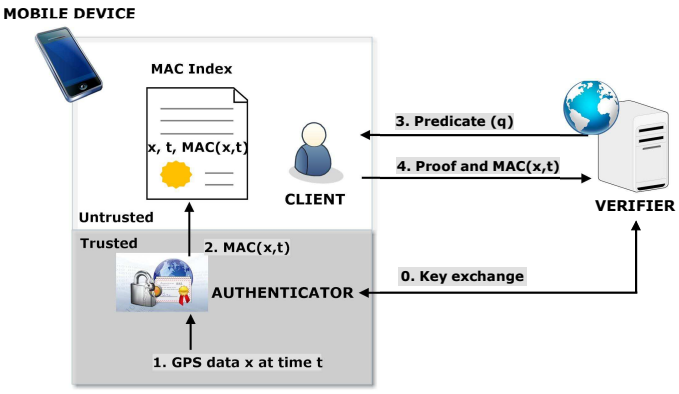


Fig. 1. System Model

TrustZone™ and Intel's Trusted Execution Technology™), modern mobile OS typically setups and bootloads a trust environment (e.g., Samsung KNOX™) for enterprise-level security tasks. We also assume that the MAC key exchange process, either through a trusted key distribution center (KDC) or through a secure protocol such as Diffie-Hellman, is secure. In addition, since a key is specific to a verifier, the compromise of one key (e.g., by a malicious verifier who colludes with the user) does not comprise the location integrity for other verifiers. In the verification protocol, we assume that both the client and the verifier are semi-honest. As such, the security threat of this system is twofold:

- **Location integrity.** The client may attempt to alter the predicate result by generating a fake location $x' \neq x$ and forging a MAC value for x'.
- **Location privacy.** The verifier may attempt to learn more information about the client's location than what is implied by the result.

In the following sections, we design the integrity proofs in the production phase and then present the client-verifier protocols to verify spatio-temporal predicates based on these proofs. **Our key idea is to encode the space by an alphabet and convert a containment predicate in space to a prefix matching predicate on strings.** In the next section, we first show how the latter can be verified without disclosing the complete string to the verifier. Then in Section 4 we show how the space is encoded and the conversion is made. There are two remarks regarding prefix matching. First, as prefixes of a location string are spatially non-overlapping, the disclosed user location after a series of authentication is always the one with the longest prefix (we will relax this in the fourth paragraph of Section 4). Therefore, we can treat predicates individually. Second, while prefix matching definitely works for containment predicates of window shapes, it can also be extended to containment predicates of irregular shapes and even other predicate types defined in the Dimensionally Extended nine-Intersection Model (DE-9IM), as long as these shapes can be encoded into strings and the predicates can be converted into prefixes (see Section 4 for more details).

3 Prefix-Verifiable Message Authentication Code (PMAC)

In this section, we design a MAC scheme that can verify that q is a prefix of string x without disclosing x. The challenge is that q **has an arbitrary length and is unknown to the authenticator in advance**. As such, this MAC scheme must also be variable-length verifiable so that a single MAC value can be used to authenticate the integrity of any prefix of x. In what follows, we first introduce HMAC, a standard MAC scheme, followed by the properties required for the proposed PMAC. Finally, we present the PMAC scheme. Table 1 summarizes the notations used in this paper.

Notation	Definition
x	encoded string of user location
m	the length of the encoded string
c	the size of the string alphabet
q	spatial window of predicate for authentication
l	the lower bound of window q
u	the upper bound of window q
t	the timestamp of location x
T	the time interval in a spatio-temporal predicate
n	the length of a trajectory, i.e., number of pairs of location and timestamp
$pre()$	the prefix of a string
$su()$	the suffix of a string
d	the length of the suffix string
$\pi(ch, i)$	the mapping function from ch, the i's character of x, to a prime number
p	order of the cyclic group $\mathcal{G}$
g	generator of the cyclic group $\mathcal{G}$
(α, k)	the symmetric key shared between the authenticator and the verifier
$\psi_k()$	a pseudorandom function with key equal to k
r	a random nonce in $\mathcal{Z}_p$
τ	number of additional grids in the overlay

TABLE 1
Notations and Definitions

3.1 Preliminary: HMAC

Keyed-hash message authentication code (HMAC) is an international standard of message authentication code with a secret cryptographic key. Given a key k and message block x, the output of HMAC is defined as [18]:

$$HMAC(k, x) = h((k \oplus opad)||h((k \oplus ipad)||x)),$$

where h is a cryptographic hash function, $opad$ and $ipad$ are two constants of one block size each.

3.2 Properties of PMAC

The PMAC of string x must satisfy the following properties. First, while the prefix of x, denoted by $pre(x)$, should be presented to the verifier as a plaintext, to protect the suffix of x from the verifier, denoted by $su(x)$, this part must be presented in an irreversible manner. As such, the PMAC should be in the following form.

$$PMAC(x) = PMAC(pre(x), T(su(x))),$$

where T is a one-way transformation, and the second $PMAC$ denotes its reconstruction out of $pre(x)$ and $T(su(x))$.

Second, to be prefix- and variable-length verifiable, the PMAC must satisfy the property that for any two prefixes of x, $pre(x)$ and $pre'(x)$ (which correspond to suffixes $su(x)$ and $su'(x)$, respectively), the PMAC values are the same. Formally,

$$PMAC(pre(x), T(su(x))) = PMAC(pre'(x), T(su'(x))).$$

Third, to prevent the client from forging the same MAC of x with a different string, the PMAC scheme must be

collision free. That is, it is hard for the client to find a collision string $x' \neq x$ such that

$$PMAC(pre(x), T(su(x))) = PMAC(pre(x'), T(su(x'))).$$

Unfortunately, HMAC and all existing MAC schemes do not satisfy the first property, where the suffix of x needs to be sent to the verifier as plaintext. To satisfy all properties above, a naive idea is to hash each character of the string and concatenate them in the same order:

$$PMAC(x) = h[h(x_1)|h(x_2)|...|h(x_m)],$$

where x_i is the character in the i-th position of string x. Suppose a prefix has length $|pre(x)| = d$, so the client has to send characters x_1, x_2, ... x_d as well as hash values $h(x_{d+1})|h(x_{d+2})|...|h(x_m)$ to the verifier. Obviously, the disadvantage of this scheme is that the total number of hash values to be received is $m - d$, and can be as many as $m - 1$ (when $d = 1$).

3.3 The Proposed PMAC

In what follows, we present the design of our PMAC scheme. Recall that a *client* wants to authenticate its location to a *verifier* without disclosing the location. The authenticator is a trusted software module in the client which is responsible for PMAC generation.

Definition 3.1. Let $g \in \mathbb{G}$ be a generator of a multiplicative cyclic group $\mathbb{G}$ of order p where the Decisional Diffie-Hellman (DDH) assumption holds and g, p are public. $\Pi(x)$ is a public hash function of string x that satisfies the following properties: (1) character-wise associative, i.e., $\Pi(x) = \prod_{i=1}^{m} \pi(x_i, i)$, where $\pi(x_i, i)$ is a function of x_i, the i-th character of x, and $\prod$ uses an associative operator; and (2) collision-resistent, that is, it is hard to find $x' \neq x$ such that $\Pi(x) = \Pi(x')$.

Key Generation A symmetric key (α, k) is generated and securely shared between the authenticator and the verifier, where $\alpha \in \mathcal{Z}_p$, and k is the key for $\psi()$, a keyed pseudorandom function.

PMAC Generation The authenticator generates a random nonce $r \in \mathcal{Z}_p$. The PMAC value of string x at timestamp t with respect to key (α, k) is defined as:

$$PMAC(x, t) = g^{\alpha(\Pi(x)r + \psi_k(t))} \mod p. \quad (1)$$

The tuple $(r, PMAC(x, t))$ together with string x and timestamp t are then stored on the client for future verification.

Verification It involves a protocol between the client and the verifier:

- Given a prefix $pre(x)$, the client obtains the suffix $su(x)$, computes $\sigma = g^{\Pi(su(x))r} \mod p$, and sends it together with t and PMAC(x,t) to the verifier.
- The verifier computes

$$(\sigma^{\Pi(pre(x))} g^{\psi_k(t)})^{\alpha} = g^{\alpha(\Pi(x)r + \psi_k(t))} \mod p, \quad (2)$$

and compares it with the received PMAC(x,t) — if they are the same, $pre(x)$ is the genuine and unmodified prefix of x.

Obviously, any PMAC defined as above satisfies the three desired properties. First, $T(su(x)) = \sigma = g^{\Pi(su(x))r} \mod p$ is irreversible, so sending it to the verifier does not disclose anything about $su(x)$. This is guaranteed by the hardness of the discrete logarithm problem — given $g^{\Pi(su(x))r}$, the verifier is unable to learn $\Pi(su(x))r$ and thus $su(x)$. The random r further guarantees confidentiality even when the domain of valid $su(x)$ is very small (i.e., the suffix contains very few characters). The second and the third properties are guaranteed by the character-wise associative and collision-resistant properties of $\Pi(x)$, respectively.[2] A practical choice of $\Pi(x)$ is to set the $\prod$ operator as modular multiplication, which is obviously associative, and to map $\pi(x_i, i)$ to a unique prime number so as to satisfy the collision-resistant property. Since all prime numbers in this mapping function are different, for a string domain of length m and an alphabet of c characters, this means that we need a total of mc prime numbers.

We will formally prove the integrity of PMAC and the confidentiality of $su(x)$ against the verifier in Section 4.2. In particular, we prove that σ cannot be forged by the client because of the irreversibility from $g^{\Pi(x)}$ to $g^{\Pi(x)/pre(x)}$, in which the client needs to extract the discrete $pre(x)$ root of $g^{\Pi(x)}$. This problem is hard as the discrete logarithm problem can be reduced to it [1].

4 Authenticating Spatial Predicate

Recall that a window containment spatial predicate q returns true if a user location x (without temporal dimension) is inside q. Throughout this paper, we only consider the predicate whose result is "true", i.e., x is inside q. In this section, we design a space encoding scheme to map x in each dimension into a string $\bigcup_{i=1}^{m} x_i$, and similarly q into another string $\bigcup_{i=1}^{d} q_i$, where q_i is the i-th character of string q, and $1 \leq d \leq m$. As such, authenticating a containment predicate q on x is equivalent to authenticating that q is a prefix of x in each dimension, the latter of which can be verified using PMAC.

To achieve the above equivalency, the encoding scheme must satisfy the following two properties. First, it must be a space partition, i.e., any user location or window must have a unique string encoding. In essence, a user location is a window of the finest granule. Second, if window A encloses window B, then the encoded string of A must be a prefix of that of B in each dimension. This latter property essentially requires the encoding has a **trie** hierarchy, where each node corresponds to a prefix string.

Given these requirements, the most intuitive encoding scheme is to adopt a hierarchical grid. Given an alphabet of size c and the longest string length m in each dimension, a grid system partitions this dimension into c uniform intervals and do it recursively for m times. An interval corresponds to a node in the trie which has m levels and exactly c children for each node. A user location has the finest granule and corresponds to a leaf node whose interval length is c^{-m} (assuming the entire length of this dimension is normalized to 1). Fig. 2 illustrates two possible grids and

2. The third property is also due to the collision-resistant property of modular exponentiation.

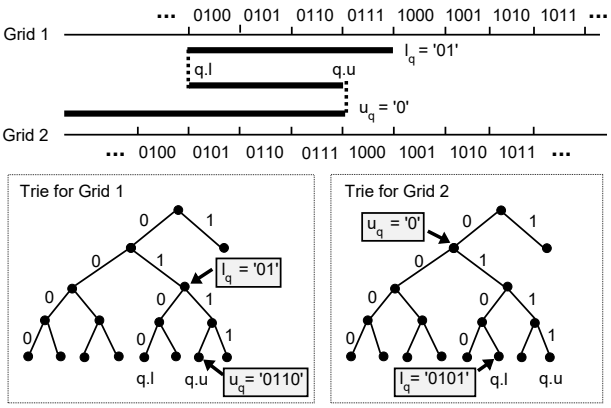


Fig. 2. Overlayed Grid System for Encoding q

their tries that deviate from each other by 1 leaf interval where $c = 2$. The encoded strings of a window q are simply the strings of nodes that correspond to q in each dimension.

The grid scheme imposes limitations on the window q — in each dimension, the beginning and ending positions of q, denoted by $q.l$ and $q.u$, must share the same prefix and their suffixes are all '0's and '1's, respectively. There are two consequences. First, for any two windows $q' \neq q$, either q fully contains q' or vice versa, and thus they are non-overlapping. Second, the length of any window q must be in a power of c of a leaf interval, i.e., $c^{m/d}$, $c^{m/d+1}$, ..., c^1, 1. To refrain from this limitation, we propose an *overlayed grid system*. The key idea is to find two intervals from different grids whose intersection is exactly q. To achieve this, one interval should begin with $q.l$ in its grid and the other should end with $q.u$ in its grid. For example, in Fig. 2, interval '01' begins with $q.l$ in grid 1 and interval '0' ends with $q.u$ in grid 2. So q can be encoded by a pair of strings (l_q, u_q), where l_q is '01' and $q'.u$ is '0'. Predicate "x is inside q" is equivalent to two predicates: (1) l_q is a prefix of x in grid 1, and (2) u_q is a prefix of x in grid 2.

In general, given a window q and a set of grids, for each dimension we first find l_q and u_q, and then authenticate location x on them as follows. To find l_q, on the trie of each grid we first locate $q.l$ in the leaf level and then traverse upward until the label on the edge is no longer '0'. The corresponding subtree denotes the string of l_q. In Fig. 2, there are two '0's to traverse, so the resulted candidate l_q is '01'. Similarly, we can find the candidate u_q on this trie by locating $q.u$ in the leaf level and traverse '1's. The result is '0110'. After we enumerate all tries, we choose l_q and u_q from those candidates who have the longest interval length. In Fig. 2, since in grid 2 the candidate l_q is '0101' and the candidate u_q is '0', the final l_q is '01' in grid 1, and the final u_q is '0' in grid 2. Note that if u_q (or l_q) has a length shorter than that of q, it means we cannot find a valid u_q (resp. l_q) interval whose intersection with l_q (resp. u_q) is q. As such, the verifier has to shift $q.u$ to where a valid u_q (resp. l_q) exists, which is considered as a limitation of this overlayed grid system (but less stringent than without it). Afterwards, we invoke the PMAC verification protocol on x and l_q, and on x and u_q in their corresponding grids. Algorithm 1 illustrates the complete procedure of spatial authentication in an overlayed grid system.

Algorithm 1 Privacy-Preserving Spatial Authentication in Overlayed Grid System

Input: **q**: the window for authentication
x: the client-side user location
$PMAC(i, j)$: the PMAC value of the i-th dimension in the j-th trie
Output: true after the authentication succeeds
Procedure:

1: **for** each dimension i **do**
2: set l_q and u_q as null
3: **for** each trie j **do**
4: compute candidate l_q and u_q
5: **if** the new candidate is longer than the current l_q or u_q **then**
6: update l_q or u_q, and set j as its grid
7: **if** l_q or u_q is shorter than q **then**
8: shift $q.l$ or $q.u$ to the closest location that has a valid l_q or u_q
9: invoke PMAC verification protocol with x and l_q in the grid of l_q
10: invoke PMAC verification protocol with x and u_q in the grid of u_q
11: **if** all verifications are passed **then**
12: return true
13: **else**
14: return false

4.1 Optimal Grid System Overlay

Obviously, having more grids in the overlayed system makes it more likely to authenticate q without shifting it. However, this is at the cost of generating more PMAC values. As such, the final problem in this section is that given a cost budget, i.e., τ grids in addition to the first grid, a.k.a, the "master" grid, how they should be placed to minimize the average shift of $q.l$ and $q.u$. In this subsection, we show that for $c = 2$, a minimum average shift can be achieved by placing these grids $2, ..., 2^\tau$ leaf intervals from the master grid.

Theorem 4.1. Given uniform distribution of x, a minimum average shift of $q.l$ or $q.u$ can be achieved when additional grids are placed $2, ..., 2^\tau$ leaf intervals from the master grid.

PROOF. We prove this by mathematical induction. Let us assume this theorem hold for $\tau = k$, and then prove it also holds for $\tau = k + 1$. We first sort these $k + 1$ grids in ascending order of their deviation from the master grid. Without loss of generality, the first k grids in an optimal solution for $k + 1$ grids must be deviated by at most 2^k leaf intervals from the master grid. Since they themselves form an optimal placement for k grids, according to the assumption, they must be deviated $2, ..., 2^k$ leaf intervals from the master grid. So the remaining task is to find the optimal placement for the $k + 1$-th grid.

We then prove that if the deviation of this grid is between 2^{k+1} and $2^{k+2} - 1$ (both inclusive) leaf intervals, it will reduce the same amount of average shift of $q.l$ or $q.u$. In fact, this grid is selected only when q is longer than 2^k and the $k + 1$-th least significant bit of $q.l$ (resp. $q.u$) is 1 (resp. 0). By selecting this grid, the $k + 1$-th bit of $q.l$ (resp. $q.u$) is flipped to 0 (resp. 1), reducing the average shift by 2^{k+1} leaf intervals. On the other hand, the first k least significant bits of $q.l$ (resp. $q.u$) will be randomly flipped by selecting this grid, and thus contributing 0 to the average shift of $q.l$ or $q.u$.

Finally, we show that if the deviation of this $k+1$-th grid is larger than or equal to 2^{k+2} leaf intervals from the master grid, it will always reduce a smaller amount of average shift

of $q.l$ or $q.u$. We prove this for deviation between 2^{k+2} and $2^{k+3}-1$ (both inclusive), and the proof follows for other values. This $k+1$-th grid is selected in two cases. In the first case, q is longer than 2^{k+1} and the $k+1$-th and $k+2$-th least significant bits of $q.l$ (resp. $q.u$) are both 1 (resp. 0). By selecting this grid, the $k+2$-th bit of $q.l$ (resp. $q.u$) is flipped to 0 (resp. 1), reducing the average shift by 2^{k+2} leaf intervals. In the second case, q is longer than 2^k but shorter than 2^{k+1} and the $k+1$-th and $k+2$-th least significant bits of $q.l$ (resp. $q.u$) are both 1 (resp. 0). However, in this case, the average shift is reduced by at most 2^{k+1} leaf intervals because q is shorter than 2^{k+1} (that is, flipping the $k+2$-th bit is not necessary). As such, the reduction of average shift of $q.l$ or $q.u$ is smaller than

$$\frac{1}{4}2^{k+2}P(q \geq 2^{k+1}) + \frac{1}{4}2^{k+1}P(2^k < q < 2^{k+1}) < \frac{1}{2}2^{k+1}P(q > 2^k),$$

where $P()$ denotes the probability of q, and obviously $P(q \geq 2^{k+1}) + \frac{1}{2}P(2^k < q < 2^{k+1}) < P(q > 2^k)$. □

4.2 Security Analysis

We analyze the security of spatial predicate authentication. Specifically, we prove it achieves both aspects of our security model: location confidentiality against the verifier, and location integrity against the client. Further, it suffices to prove them for the PMAC verification protocol only since it is the only interaction between the client and verifier during authentication.

4.2.1 Location Confidentiality against Verifier

Equivalently, we prove the suffix $su(x)$ of an encoded location string x is secret to the verifier. Recall that in PMAC, $g \in \mathbb{G}$ is a generator of a multiplicative cyclic group $\mathbb{G}$ of order p. First, we show that g^r does not disclose any information about g.

Lemma 4.2. For any random $r \leftarrow [0, |\mathbb{G}|)$, g^r has equal probability of being any element in $\mathbb{G}$. Formally, for any $\hat{g} \in \mathbb{G}$

$$Pr[g^r = \hat{g}] = 1/|\mathbb{G}|.$$

PROOF. Let $\log_g()$ denote the discrete logarithm of base g in group $\mathbb{G}$.

$$Pr[g^r = \hat{g}] = Pr[r = \log_g(\hat{g})].$$

Since r is random, the probability of r being a fixed element $\log_g(\hat{g})$ equals to $1/|\mathbb{G}|$. □

As the verifier can only observe $g^{\Pi(su(x))r}$, we prove the verifier learns nothing about $su(x)$ from it.

Theorem 4.3. The $g^{\Pi(su(x))}$ is indistinguishable under chosen plaintext attacks.

PROOF. Since $g^{\Pi(su(x))r} = (g^{\Pi(su(x))})^r$, and according to the above lemma, $g^{\Pi(su(x))}$ has equal probability of being any element in $\mathbb{G}$. As such, the verifier learns nothing about $\Pi(su(x))$ and thus $su(x)$. Further, since r is random for each x, $g^{\Pi(su(x))r}$ is indistinguishable under chosen plaintext attacks. □

4.2.2 Location Integrity against Client

We prove it is impossible for the client to forge a valid PMAC(x,t). The following proof is in two steps: (1) it is hard for the client to forge any $g^{\Pi(x)r}$ and PMAC(x,t) that satisfy Eqn. 2; (2) given $g^{\Pi(x)r}$ and q as the predicate, it is hard for the client to forge σ such that $\sigma^{\Pi(pre(q))} = g^{\Pi(x)r}$. Recall that in the PMAC scheme, p and g are public, and (α, k) is the symmetric key of the verifier.

Proof of unforgeability of $g^{\Pi(x)r}$ and PMAC(x,t) For ease of presentation, let $m = g^{\Pi(x)r}$, then:

$$PMAC(x,t) = (m \cdot g^{\psi_k(t)})^\alpha \mod p.$$

Let us prove this by contradiction. If there were a probabilistic polynomial time bounded algorithm $\mathcal{A}$ for the client to forge an m and PMAC(x,t), we can design an algorithm $\mathcal{A}'$ for the RSA problem as follows.[3] Recall that the RSA problem is that, given p and integer e that is co-prime with $\phi(p)$, and an element $y \in \mathbb{Z}_p^*$, to find x such that $x^e = y \mod p$. $\mathcal{A}'$ can design a random oracle $\mathcal{O}$ like this.[4] When $\mathcal{A}$ queries q messages $m_1, ..., m_q$ from it, $\mathcal{O}$ responds with $PMAC(x,t)^e$, where PMAC(x,t) is from the PMAC generator inside $\mathcal{A}'$, except for one random message m_i where it responds with y of the RSA problem, and $e = \alpha^{-1} \mod \phi(p)$. Then $\mathcal{A}$ requests the PMAC of $m \in \{m_1, ..., m_q\}$ from $\mathcal{A}'$. If $m \neq m_i$, $\mathcal{A}'$ responds with $PMAC(x,t)$; otherwise, $\mathcal{A}'$ aborts. Now $\mathcal{A}$ can forge a PMAC x for some $m' \in \{m_1, ..., m_q\} - \{m\}$. With a probability of $1/q$, $m' = m_i$, which solves the RSA problem because $x^e = y$. This contradicts the assumption that there is no probabilistic polynomial time bounded algorithm for the RSA problem.

Proof of unforgeability of σ given $pre(q)$ and $g^{\Pi(x)r}$. Let us prove this by contradiction. If there were a probabilistic polynomial time bounded algorithm $\mathcal{A}$ for the client to forge a σ given $pre(q)$ and $g^{\Pi(x)r}$, we can design an algorithm $\mathcal{A}'$ for the RSA problem as follows. For problem $x^e = y \mod p$, $\mathcal{A}'$ simply asks $\mathcal{A}$ to forge σ with $pre(q) = e$ and $g^{\Pi(x)r} = y$. Then $x = \sigma$, which solves the RSA problem This contradicts the assumption that there is no probabilistic polynomial time bounded algorithm for the RSA problem. Combining both proofs, we reach the following theorem regarding the integrity of PMAC scheme.

Theorem 4.4. A PPT client cannot forge a valid PMAC under chosen-plaintext attack.

5 Authenticating Spatio-temporal Predicate

The integrity of spatio-temporal trajectories is essential as more and more businesses rely on the correctness of continuous user location, and there are increasing threats of forging or manipulating it. For example, Uber drivers reportedly committed fraud by completing phantom trips without passengers, using modified software [28]. In this section, we extend the location authentication scheme from a static spatial point to a spatio-temporal trajectory. Without loss of generality, we assume a trajectory $\mathcal{T}$ consists of a

3. The construction of $\mathcal{A}'$ is similar to the one in the proof of Theorem 13.11 in [19].

4. This is based on the assumption that $m \cdot g^{\psi_k(t)}$ is a random oracle.

series of pairs of user location and associated timestamp in ascending order of the timestamps. Formally,

$$\mathcal{T} = \{(x[1], t[1]), (x[2], t[2]), \cdots, (x[n], t[n])\}.$$

A spatio-temporal predicate (q, T) on a trajectory $\mathcal{T}$ returns true only if of all timestamps $t[i]$ in the time interval T, all corresponding user locations $x[i]$ are contained in window q. For ease of presentation, in the rest of this section we focus on a single dimension and our scheme can naturally be extended to any dimensionality by treating each dimension independently.

To authenticate a spatio-temporal predicate, a naive approach is to apply the same spatial authentication scheme to each location within time interval T in the trajectory. The correctness of each timestamp t can also be verified because t contributes $\psi_k(t)$ to the PMAC and therefore the client sends plaintext t to the verifier. However, to verify its completeness, that is, to prevent the client from omitting an existing location from a trajectory, the PMAC value at timestamp t must also depend on its two neighboring timestamps $t[i-1]$ and $t[i+1]$. As such, we redefine the $\psi_k(t)$ definition in PMAC(x,t) of Eqn. 1 as follows.

$$\psi'_k(t[i]) = -\psi_k(t[i-1]) + 2\psi_k(t[i]) - \psi_k(t[i+1])),$$

where $\psi_k()$ is the original keyed pseudorandom function and $\psi'_k()$ is the new function used in Eqn. 1. To avoid overnotating, in the following we still use $\psi_k()$ to denote the new function.

Therefore, given a spatio-temporal predicate (q, T), a baseline authentication scheme is as follows. The client first locates timestamps $t[s], ..., t[e]$ in $\mathcal{T}$ that fall in time interval T. For each location $x[i]$ ($s \le i \le e$), it sends $pre(x[i])$, $\Pi(su(x[i]))$, $t[i]$ and $PMAC(x[i], t[i])$ to the verifier to compute and compare two versions of $PMAC(x[i], t[i])$. If they are the same, $x[i]$ is inside window q. At the end of the day, the client also sends timestamps $t[s-1]$ and $t[e+1]$ to: (1) compute the PMACs of two boundary locations $x[s]$ and $x[e]$, and (2) prove that no location in T has been omitted.

However, the major disadvantage of this baseline approach is that both the computation and bandwidth costs are proportional to the number of locations being authenticated, and is thus inefficient when T is long. In the rest of this section, we present two indexing schemes that aim to authenticate them collectively. To start with, we first present PMAC in an aggregated form.

5.1 Building Block: Aggregated PMAC

Similar to signature aggregation, MACs of different values can be aggregated into a single MAC value to save the bandwidth cost. There is a line of research in the literature of cryptography on MAC aggregation. As pointed out by [19], a common approach to enable aggregation on a MAC scheme is to adopt the "XOR" operations. That is, for messages $x[1]$, ..., $x[n]$:

$$MAC(x[1], ..., x[n]) = \bigoplus_{i=1}^{n} MAC(x[i]).$$

Unfortunately, this does not work for PMAC scheme, because besides $PMAC(x[i])$ and $pre(x[i])$, the verifier also needs $\sigma(i) = g^{\Pi(su(x[i]))r[i]} \mod p$ for verification, which cannot be aggregated by XOR in the same way as $PMAC(x[i])$. On the other hand, modular exponentiation satisfies an alternative property:

$$g^{\alpha[\sum_{i=1}^{n}(\Pi(x[i])r[i]+\psi_k(t[i]))]} = (\prod_{i=1}^{n}\sigma(i))^{\alpha\Pi(pre(x))} \cdot g^{\alpha\sum_{i=1}^{n}\psi_k(t[i])} \mod p,$$

where $pre(x)$ is the common prefix to verify for all $x[i]$. A key observation from this equation is that during verification the client can aggregate individual $\sigma(i)$ into a single one, and correspondingly the verifier only needs to verify an aggregate PMAC value of all $x[i]$ as below:

$$PMAC(\mathbf{x}, \mathbf{t}) = g^{\alpha\sum_{i=1}^{n}\Pi(x[i])r[i]+\psi_k(t[i])} \mod p \quad (3)$$

We call $PMAC(\mathbf{x}, \mathbf{t})$ the PMAC of a trajectory $(\mathbf{x}, \mathbf{t})$. Note that

$$\sum_{i=1}^{n}\psi_k(t[i]) = -\psi_k(t[0]) + \psi_k(t[1]) + \psi_k(t[n]) - \psi_k(t[n+1]),$$

where $t[0]$ and $t[n+1]$ are the two timestamps adjacent to this trajectory.

Given a predicate (q, T), the verification procedure is as follows.

- The client finds trajectory $(\mathbf{x}, \mathbf{t})$ that corresponds to (q, T).
- The client computes $\sigma = g^{\sum_{i=1}^{n}(\Pi(su(x[i])r[i]))} \mod p$ and sends it together with $PMAC(\mathbf{x}, \mathbf{t})$, and timestamps $t[0]$, $t[1]$, $t[n]$, and $t[n+1]$.
- The verifier computes

  $$\sigma^{\alpha\Pi(pre(x))} \cdot g^{\alpha(-\psi_k(t[0])+\psi_k(t[1])+\psi_k(t[n])-\psi_k(t[n+1]))} \mod p$$

  and verifies if it matches the received $PMAC(\mathbf{x}, \mathbf{t})$.

From the above procedure, it is obvious that using a trajectory PMAC reduces the bandwidth cost from n to 1.

It is noteworthy that the above aggregate PMAC can be generated by the client itself by multiplying individual $PMAC(x[i], t[i])$ as follows.

$$PMAC(\mathbf{x}, \mathbf{t}) = \prod_{i=1}^{n} PMAC(x[i], t[i]) \mod p$$

This is an essential requirement because trajectory $(\mathbf{x}, \mathbf{t})$ depends on the predicate (q, T), which is unknown to the authenticator in advance.

5.2 PMAC Indexing

Aggregated PMAC reduces the communication cost when authenticating a set of locations. However, the computational cost, in terms of modular exponentiations on both sides, is still proportional to the size of the set, and hence can be inefficient when T is long. Based on aggregated PMAC, we propose two PMAC indexes that precompute and store aggregate PMACs in advance for future authentication. Note that both indexes take the form of a general (possibly unbalanced) search tree, because mobile device storage is less I/O bounded than desktop computers.

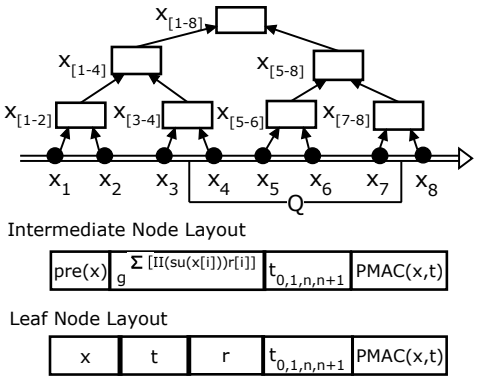


Fig. 3. PS-Tree: PMAC Search Tree

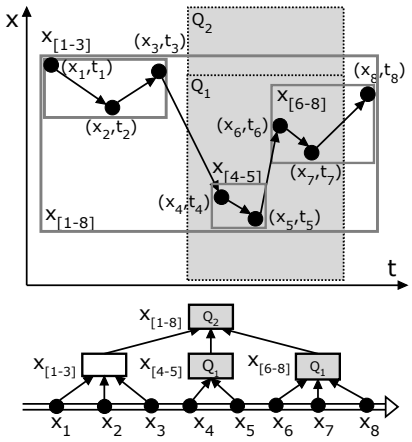


Fig. 4. PS*-Tree: PMAC Clustered Search Tree

5.2.1 PS-Tree: PMAC Search Tree

The first index treats time as a special dimension, and organizes all locations in a k-way search tree. Fig. 3 illustrates a binary search tree, where each intermediate node stores the aggregated PMAC of all its descendants. Specifically, each leaf node represents a location with a timestamp, and is threaded with adjacent leaf nodes. Each intermediate node consists of four pieces of information about its descendants: (1) their longest common prefix $pre(x)$, (2) their accumulative suffix $g^{\sum_{i=1}^{n}\Pi(su(x[i]))r[i]} \mod p$, (3) the timestamps of $t[0]$, $t[1]$, $t[n]$ and $t[n+1]$, and (4) their aggregated PMAC. Since all the information can be computed by the client, this index can be constructed by the client in a bottom-up manner without the authenticator. We illustrate this construction with the example in Fig. 3. When (x_1, t_1) arrives, the PS-tree is initiated. When (x_2, t_2) arrives, the first intermediate node $x_{[1-2]}$ will be constructed. Next, when (x_3, t_3) arrives, since (x_4, t_4) is not available yet, $x_{[3-4]}$ will be left pending. As such, the PS-tree at this moment contains the subtree of $x_{[1-2]}$ and the leaf node x_3. Upper intermediate nodes are recursively constructed in the same manner. Since each intermediate node only needs to be computed once, the total cost is proportional to the total number of nodes, which is $\lceil \frac{kn-1}{k-1} \rceil$.

Given a predicate (q, T), the verification procedure is as follows, illustrated in Fig. 3.

- Starting from the root of the index, the client recursively checks whether the timestamps of this node are fully contained in T. If so, there is no disclosure of any location information beyond T and therefore this node can be used for authentication; otherwise, all child nodes whose timestamps overlap with T will be checked instead. The procedure terminates when no more nodes need to be checked and this leads to a minimum set of mutually exclusive nodes that jointly cover T. In this example, these nodes are x_4, $x_{[5-6]}$, and x_7. The aggregated PMACs of these nodes will be used to authenticate against q in the next step.
- For each node above, the client authenticates that q is a prefix of its $pre(x)$ by computing the following σ:

$$\sigma = \prod_{i=1}^{n} (g^{\Pi(su(x[i]))r[i]})^{\prod_{j=d'+1}^{d}(x[j].j,j)}$$

where d' and d are the length of q and $pre(x)$, respectively, and $\pi(x[i].j, j)$ is the mapped prime number of j-th character of location string $x[i]$. The client sends σ together with the PMAC value, and timestamps $t[0]$, $t[1]$, $t[n]$, and $t[n+1]$ of this node.

- For each received σ, the verifier computes

$$\sigma^{\alpha\Pi(q)} \cdot g^{\alpha(-\psi_k(t[0])+\psi_k(t[1])+\psi_k(t[n])-\psi_k(t[n+1]))} \mod p \quad (4)$$

and verifies if it matches the received PMAC value.

- By verifying $t[n+1]$ of the i-th node is equivalent to $t[0]$ of the $i+1$-th node in the above authentication, and by verifying $t[0]$ of the first node and $t[n+1]$ of the last node (in this example, t_3 and t_8) are beyond T, the verifier can guarantee no location falling in T is missing from authentication.

5.2.2 PS*-Tree: PMAC Clustered Search Tree

While preserving the temporal locality, a PS-tree does not consider the spatial locality. As a consequence, we cannot exploit the spatial predicate q to reduce the number of nodes to authenticate. For example, in Fig. 3, if q is a prefix of the root node's $pre(x)$, authenticating it alone suffices for authenticating the entire spatiotemporal predicate (q, T). However, there are two challenges to enable such pruning. First, showing the entire trajectory from t_1 to t_8 falling in q discloses more information to the verifier than he/she is supposed to know, which should only be that locations from t_4 to t_7 fall in q, and that all other timestamps are beyond T. Second, to reduce the number of nodes to authenticate, locations in a node should share as long prefix as possible so that q can be authenticated by upper-level nodes. As such, an algorithm that clusters spatial locations while still retaining their temporal order in the trajectory should be devised. In what follows, we propose a second PMAC index — PS*-tree — that addresses these two challenges.

Fig. 4 illustrates this index with the same user trajectory and predicate as in Fig. 3. The horizontal and vertical axes denote the temporal and 1D spatial dimension, respectively. Same with the threaded search tree, each leaf node in the PS*-tree represents a location with a timestamp. Since a leaf node and an intermediate node consist the same four pieces of information as in Fig. 3, the node layouts are omitted in Fig. 4. The only difference lies in the definition of

aggregated PMAC, where we replace $\psi_k(t)$ with $\Psi_k(t)$ and call it PMAC*:

$$PMAC^*(\mathbf{x}, \mathbf{t}) = g^{\alpha \sum_{i=1}^{n} [\Pi(x[i])r[i] + \Psi_k(t[i])]} \mod p, \quad (5)$$

where

$$\Psi_k(t) = \psi_k(\mathcal{G}(t - t_L)|\mathcal{G}(t_U - t)|h(t)), \quad (6)$$

where $h(t)$ is a cryptographic hashing function, L and U are the lower and upper bounds of timestamps, and $\mathcal{G}(t)$ is a digest function as defined in [31], [14] that satisfies the following properties.

- **non-negative:** The input domain of $\mathcal{G}()$ only accepts non-negative numbers.
- **addictively homomorphic:** That is,

$$\mathcal{G}(a + b) = \mathcal{G}(a) \otimes \mathcal{G}(b),$$

where $\otimes$ is a well-defined operation on $\mathcal{G}$.

By introducing $\mathcal{G}$ in the aggregated PMAC, the client can prove to the verifier that a time interval $T = [T.l, T.u]$ is fully contained in the time interval of a node without disclosing the actual interval to the verifier. To prove $T.u \leq t[n]$, the client sends $\mathcal{G}(t[n] - T.u)$ to the verifier, who then restores $\mathcal{G}(t[n] - L) = \mathcal{G}(t[n] - T.u) \otimes \mathcal{G}(T.u - L)$. By computing the $PMAC^*$ value in Eqn. 5 and matching it with the one directly from the client, the verifier can assure $T.u \leq t[n]$. Similarly, the same construct can be used to prove $T.l \geq t[1]$.

Given a predicate (q, T), the verification procedure on PS*-tree is as follows.

- Starting from the root node, the client recursively checks whether q is a prefix of this node's $pre(x)$. If so, the PMAC* value of this node will be authenticated; otherwise, all child nodes whose timestamps overlap with T will be checked instead. This procedure terminates when no more node needs to be checked. In Fig. 4, for predicate $Q_1 = (q_1, T)$, since q_1 is not a prefix of $x[1-8]$'s $pre(x)$, the PMAC* values of $x[4-5]$ and $x[6-8]$ will be authenticated; for $Q_2 = (q_2, T)$, since q_2 is a prefix of $x[1-8]$'s $pre(x)$, the PMAC* value of $x[1-8]$ will be authenticated. In either case, the number of nodes to be authenticated is reduced from 3 in the case of PS-tree.
- For each node above, the client authenticates that q is a prefix of this node's $pre(x)$, by computing the same σ as in PS-tree authentication:

$$\sigma = \prod_{i=1}^{n} (g^{\Pi(su(x[i]))r[i]})^{\prod_{j=d'+1}^{d}(x[j].j,j)}$$

The client sends σ together with the PMAC value, and timestamps $t[0]$, $t[1]$, $t[n]$, and $t[n+1]$ of this node. Note that, if any of these timestamps is beyond the time interval T, to protect them the client sends digest value $\mathcal{G}$ and $h()$ of these timestamps instead. Specifically, if $t > T.u$, the client sends $\mathcal{G}(t - T.u)$, $\mathcal{G}(U - t)$ and $h(t)$; if $t < T.l$, the clients sends $\mathcal{G}(T.l - t)$, $\mathcal{G}(t - L)$ and $h(t)$.
- For each received σ, the verifier first restores $\Psi(t)$ using Eqn. 6 for $t = t[0], t[1], t[n]$ and $t[n-1]$, then computes

$$\sigma^{\alpha\Pi(q)} \cdot g^{\alpha(-\Psi_k(t[0]) + \Psi_k(t[1]) + \Psi_k(t[n]) - \Psi_k(t[n+1]))} \mod p \quad (7)$$

and verifies if it matches the received PMAC value. Note that by successfully verifying the PMAC* values of all nodes whose timestamps overlap with T, the verifier can guarantee no location falling in T is missing from authentication.

Similar to the PS-tree, the PS*-tree can be constructed by the client in a bottom-up manner. However, there are two notable differences. First, in the leaf node information, the client is unable to generate the PMAC* value by itself, so the authenticator needs to compute it in addition to the PMAC value for an incoming leaf node. Second, the PS*-tree clusters nodes with spatial locality along with temporal locality. As such, instead of always merging k nodes together, a clustering algorithm decides whether or not to stop merging with the next node along the temporal axis. For simplicity, we propose a greedy merging algorithm as follows. It always merges the next node with the current cluster unless either the current cluster has already k nodes, or by treating the next node as a new cluster, the average length of shared prefix between this cluster and the current cluster is even longer than merging this node with the current cluster. For example, in Fig. 4, $k = 3$. When (x_4, t_4) arrives, since cluster $x_{[1-3]}$ already has 3 nodes, x_4 will start a new cluster. Then when (x_5, t_5) arrives, the shared prefix length in cluster $x_{[4-5]}$ is 6. Next, when (x_6, t_6) arrives, if it is merged with the current cluster, the shared prefix length is reduced to 3. However, if we start a new cluster with it, the current cluster will retain the length of 6 whereas the new cluster has a length of 0. Since the average length is $(6 * 2 + 0)/3 = 4$, which is larger than 3, the greedy algorithm chooses to start a new cluster with x_6 and construct intermediate node $x_{[4-5]}$.

5.3 Security Analysis

We prove the security of spatio-temporal predicate authentication. As with Section 4.2, we show below that both the aggregated PMAC and the two indexes satisfy location confidentiality against the verifier and location integrity against the client. Due to space limit, we only sketch the key points of proofs.

The confidentiality of the aggregated PMAC during verification is obvious as it directly follows Theorem 4.3 by further suppressing the disclosure of individual σ's and sending only their modular multiplication to the verifier. In addition to this, the PS-tree also discloses the accumulative suffix $g^{\sum_{i=1}^{n} \Pi(su(x[i]))r[i]} \mod p$, whose confidentiality directly follows Theorem 4.3 by further suppressing the disclosure of individual $g^{\Pi(su(x))r}$. On the other hand, the PS*-tree discloses the same information as the PS-tree except that $PMAC(\mathbf{x}, \mathbf{t})$ is replaced with $PMAC^*(\mathbf{x}, \mathbf{t})$ in each node. According to Eqn. 5 and 6, while they share the same function on x, the latter only differs from the former by wrapping t with digest function $\mathcal{G}()$ and $h()$. As such, $PMAC^*(\mathbf{x}, \mathbf{t})$ achieves the same confidentiality with respect to x as $PMAC(\mathbf{x}, \mathbf{t})$.

As for the location integrity, similar to Section 4.2.2, we need to prove that: (1) it is hard for the client to forge $PMAC(\mathbf{x}, \mathbf{t})$ as defined in Eqn. 3 (for aggregated PMAC and PS-tree) or $PMAC^*(\mathbf{x}, \mathbf{t})$ as defined in Eqn. 5 (for PS*-tree); (2) given (q, T) as the predicate, it is hard for

the client to forge σ that satisfies Eqn. 4 (for aggregated PMAC), or Eqn. 4 (for PS-tree), or Eqn. 7 (for PS*-tree). The proof of the former follows the first proof in Section 4.2.2 as there is no probabilistic polynomial time bounded algorithm for the RSA problem where $\mathbf{x} = \Pi x_i$. Note that $\mathbf{t}$ is irrelevant in this proof so it applies to both $PMAC$ or $PMAC^*$. The proof of the latter takes two steps because the verifier performs a modular multiplication of $\sigma^{\alpha\Pi(q)}$ and $g^{\alpha(-\psi_k(t[0])+\psi_k(t[1])+\psi_k(t[n])\psi_k(t[n+1]))}$. In the first step, we prove that the client cannot forge σ that leads to the correct first operand of the multiplication. This directly follows the second proof in Section 4.2.2. In the second step, we further prove that the client cannot manipulate the second operand either to match the correct $PMAC$ or $PMAC^*$. This is due to the hardness of discrete logarithm problem and pseudorandomness of $\psi_k()$ or $\Psi_k()$.

6 Performance Optimizations

In this section, we propose two optimization techniques that further reduce the computational cost of our scheme.

6.1 Accelerating PMAC Verification

According to Euler's theorem [19]: for modulus p and g that is co-prime to p, $g^{\phi(p)} \mod p \equiv 1$, where $\phi(p)$ denotes p's totient number, i.e., the number of integers between 1 and p that are co-prime to p. Applying this theorem, the client can reduce the cost of computing $\sigma = g^{\Pi(su(x))r} \mod p$ by

$$g^{\Pi(su(x))r} \mod p = g^{\Pi(su(x))r \mod \phi(p)} \mod p$$

Similarly, the verifier can reduce the cost of computing PMAC by applying $\mod \phi(p)$ on the exponents:

$$(\sigma^{\Pi(pre(x)) \mod \phi(p)} g^{\psi_k(t)})^{\alpha \mod \phi(p)} \mod p$$

Note that since in practice p is the product of two large primes, disclosing $\phi(p)$ to the client may cause security implications if p is not properly chosen [19]. As such, in our implementation, only the verifier learns $\phi(p)$ and can accelerate its computation.

6.2 Density-Based Spatial Recoding

In previous sections, we adopt a hierarchical regular grid as the space encoding scheme. The disadvantage of using a regular grid, however, is that every user location has to be encoded in a full-length string. In fact, for a given alphabet of c characters and string length of m, there can be totally c^m combinations, much larger than the total number of user locations collected in a long period of time. As such, if we map distinct user locations to each of these combinations, the string length m can be greatly reduced. In this optimization, we propose *density-based spatial recoding* that uses an auxiliary index to translate a user location in a full-length string to an "acronym" with shorter length, which can reduce the computational cost of generating and verifying PMACs.

Fig. 5 illustrates the idea of density-based recoding and the auxiliary index for $c = 2$. We assume for privacy protection, the client only recodes the first $\tau = 3$ characters. In the figure, all 12 user locations are clustered in three full-length prefixes: "000", "010" and "101", which may correspond

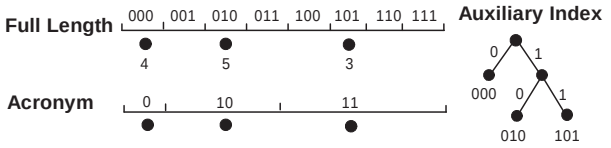


Fig. 5. Density-Based Recoding

to his/her neighborhood and workplaces. The digits "4", "5" and "3" denote the occurrences of user locations in each of these prefixes. As the objective of recoding is to minimize the average string length of user locations, this is similar to Huffman coding that minimizes the average string length of a set of symbols, except that to enable authentication, our recoding must also preserve the original order of the recoded strings. It is noteworthy that the greedy Huffman coding algorithm achieves the minimum string length while for this order-preserving variant, an optimal solution can only be achieved by dynamic programming, which is extremely costly given the size of the prefixes. In this regard, we propose the following *order-preserving Huffman coding* algorithm. Instead of merging any c symbols that have the minimum sum of occurrences, the algorithm merges the c *consecutive* symbols that have the minimum sum. In this figure, "000" "010" and "010" "101" serve as the only two sets of consecutive symbols, and the latter has the minimum sum. As such, "010" "101" will be merged first and then be merged with "000". The resulted auxiliary index is shown on the right hand side. The verifier will receive this index before the first verification request and recode a predicate into the same form of acronym. Then all PMAC generation and verification will be conducted in the recoded string space.

One remaining problem is how the verifier can authenticate the auxiliary index itself. In the literature, there are a wide range of authentication data structures for trees, in particular, the family of Merkel Hash Tree [26]. Specifically, each node will be accompanied with a *digest*: a leaf node digest is the hash value of its contents (in our auxiliary index they are the prefix string such as "010"); and an intermediate node digest is the hash value of its children digests and its contents (in our auxiliary index they are empty). As such, the digest of the root node depends on the entire tree and can be signed by the authenticator when the auxiliary index is constructed.

7 Performance Evaluation

In this section, we evaluate the experimental results of the proposed PMAC scheme and spatio-temporal authentication protocols. To test the performance in a real-life setting, we use dataset "GeoLife GPS Trajectories" from Microsoft Research [46]. This dataset collected 17,621 GPS trajectories (latitude, longitude, altitude, timestamp) from April 2007 to August 2012. We filter out those trajectories whose number of locations are fewer than 990 and convert all longitude, latitude and timestamps into 32-bit binaries. For each resulted 2D trajectory, we build both a 2-way PS-tree and a PS^*-tree on its PMACs.

The code of both client and verifier is implemented in Java. The client is set up on a Samsung Galaxy Tab S2

Parameter	Symbol	Value
total number of trajectories	N	5881
average trajectory length	n	3463
spatial predicate bit length	$\|q\|$	$[16, 32]$
temporal interval bit length	$\|T\|$	$[2, 8]$
number of additional grids in the overlay	τ	16

TABLE 2
Parameter Settings for Experiments

9.7 tablet with Exynos 5433 Octa SoC (4 ARM Cortex-A57 1.9GHz cores and 4 Cortex-A53 1.3GHz cores) and 3GB RAM, running Android 5.0, and the verifier is set up on an IBM server with Dual 6-core Intel Xeon X5650 2.66GHz CPU and 32GB RAM, running GNU/Linux and OpenJDK 1.6 64-bit. The hash function $h()$ is 160-bit SHA-1, and to enable high security, we set all security parameters in our scheme, including the modulus p, α, r, and g as 1024-bit. The pseudo random function $\psi_k()$ adopts AES-256. We use the same digest function $\mathcal{G}()$ as in [31], [14] with the base of the canonical representation set to 2. By default, the authentication algorithms adopt both optimizations proposed in Section 6.

For performance evaluation, we measure the computational cost (in terms of the client and verifier's CPU time) for authentication, and the communication overhead (in terms of the transmitted data size). The bit length of spatial predicates ranges in from 16 to 32, and the bit length of timestamps range from 2 to 8. For each measurement, 500 spatio-temporal predicates are authenticated and their average value is reported. Table 2 summarizes the parameter settings used in the experiments.

7.1 Overall PMAC Generation and Authentication Performance

In this subsection, we evaluate the overall performance of PMAC generation, PS-tree and PS*-tree construction and authentication. Table 3 shows the CPU time and size of PMAC, PS-tree, and PS*-tree, of three sample trajectories with different length n. We observe that both metrics of the PS-tree and PS*-tree increase in proportion to the length of the trajectory. Note that theoretically a PS*-tree has the same size of a PS-tree, but to speed up subsequent computation on the $\mathcal{G}()$ during verification, we pre-compute and store the canonical form of $\mathcal{G}()$ as in [31]. Therefore, in this table a PS*-tree is about 4 times larger than a PS-tree. In terms of the CPU cost, both PMAC and PS-tree are efficient to compute, whereas PS*-tree takes about twice the time of PMAC, but still fewer than 20 minutes even for the longest trajectory (whose temporal interval is over $2,000$ minutes). To demonstrate the practical use of our scheme in a trusted environment, we also deploy the authenticator in Samsung My Knox, a secure container powered by ARMs TrustZone, and measure the authenticator's CPU time. We observe that the container introduces a mild and consistent overhead of about 20% more CPU time than its non-Knox counterpart.

To evaluate the authentication cost of a spatio-temporal predicate, we first vary the spatial predicate bit length $|q|$ from 16 to 32. Figs. 6(a) and 6(b) plot the CPU and communication costs with respect to $|q|$. In all cases, as $|q|$ increases (i.e., the spatial predicate gets smaller), the costs are reduced because the authentication can terminate in higher-level nodes in the tree. PS*-tree is up to 2-orders of magnitude more efficient than PS-tree in terms of CPU cost,

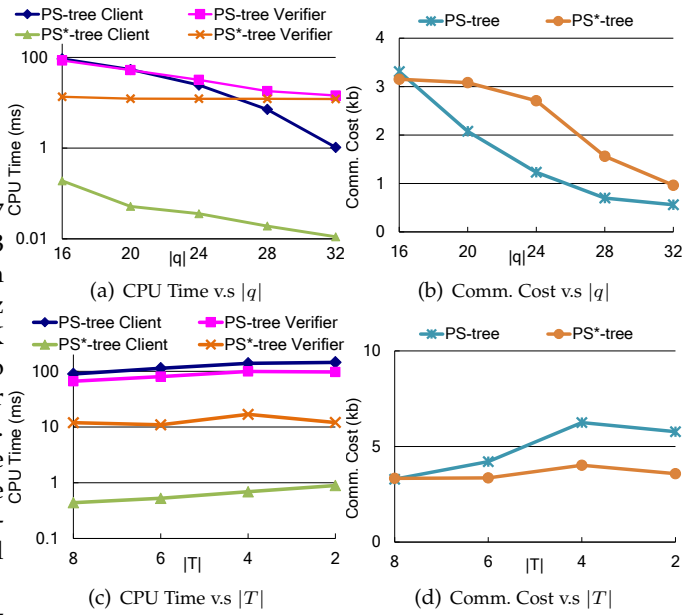


Fig. 6. Authentication Performance on PS- and PS*-trees

because it can terminate in even-higher-level nodes, thanks to the $\mathcal{G}$ function and the clustering effect. However, it is at the cost of larger communication size, due to the additional canonical form of $\mathcal{G}$ being sent.

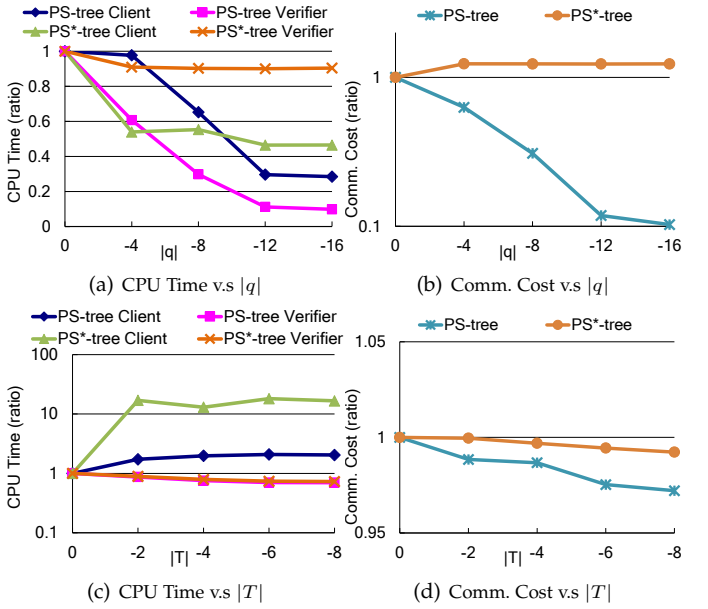
We then vary $|T|$ from 8 to 2. Figs. 6(c) and 6(d) plot the CPU and communication costs with respect to $|T|$. As $|T|$ decreases, the temporal interval of the predicate becomes longer, and therefore all verification costs increase. Nonetheless, in terms of both CPU time, PS*-tree outperforms PS-tree by at least one-order of magnitude, and it also costs less communication than PS-tree. This demonstrates that the PS*-tree is insensitive to the temporal interval of a spatio-temporal predicate and performs exceptionally well when this interval is long. The rationale is that, as the temporal interval increases, the corresponding spatial predicate must also have a shorter prefix, and therefore negates the factor of a longer temporal interval spanning more tree nodes.

7.2 Effect of Relative Authentication Performance

In the above experiments, each measurement is averaged by a variety of spatio-temporal predicates and thus can be influenced by the location distribution in the trajectories. To show how PS-tree and PS*-tree behave when a verifier enlarges or reduces a given predicate, in this subsection we create a variety of *minimum viable predicates* (MVPs), which have the smallest spatial size $|q|$ given a temporal interval or vice versa, according to the tightest bound of a trajectory. We then create other predicates by increasing $|q|$ or decreasing $|T|$ of these MVPs. Fig. 7 show the relative performance for these predicates with respect to their MVPs, with measurements for MVPs set as base. As such, the leftmost position in the x-axis denote the measurement for MVPs and always has value 1. In Figs. 7(a) and 7(b), each position reduces the bit length of $|q|$ by 4 bits, so the size of the predicate becomes 16 times larger in each dimension. In general, as $|q|$ decreases, the performance improves as the authentication can terminate at higher-level of nodes for both PS-tree and PS*-tree. PS-tree has an even sharper drop of the costs,

	CPU Time (s)			CPU Time in My Knox (s)			Size (MB)		
n	**PMAC**	**PS-tree**	**PS*-tree**	**PMAC**	**PS-tree**	**PS*-tree**	**PMAC**	**PS-tree**	**PS*-tree**
993	3.8	0.75	7.1	4.5	0.87	8.7	0.35	1.2	5.4
10019	34.2	5.7	62.4	41.4	6.8	73.6	3.4	11.6	54
92645	322	55.9	471	389	66.9	572	32	110	512

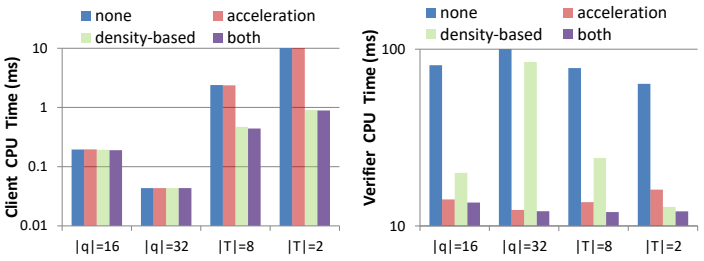
TABLE 3
PMAC and Index Construction Cost



(a) CPU Time v.s $|q|$ (b) Comm. Cost v.s $|q|$

(c) CPU Time v.s $|T|$ (d) Comm. Cost v.s $|T|$

Fig. 7. Relative Authentication Performance on PS- and PS*-tree



(a) Client CPU Time (b) Verifier CPU Time

Fig. 8. Effect of Optimizations

which means that it is more sensitive to the spatial size of the predicate while PS*-tree already achieves satisfactory performance for the MVPs and thus further improvement is less noticeable. Similar observation can be made in Figs. 7(c) and 7(d), where each position reduces the bit length of $|T|$ by 2 bits and effectively shrinks the temporal interval by $1/4$. PS*-tree leads to worse performance than PS-tree as shorter $|T|$ favors the latter more. To conclude, with reasonably higher construction and storage cost, a PS*-tree achieves better performance than a PS-tree, especially when the temporal interval is long or the spatial predicate is small.

7.3 Effect of Optimizations

In this subsection, we evaluate the effect of the two schemes, namely, PMAC accelerating and density-based spatial re-coding introduced in Section 6. We choose four settings for predicates: $|q| = 16$, $|q| = 32$, $|T| = 8$ and $|T| = 2$, and plot the client and verifier's CPU time on PS*-tree in Fig. 8. Note that for a tight security model, we disable client-side PMAC accelerating, so it has no effect on client's CPU time. Other than that, both optimizations achieve significant performance enhancement over the original scheme, by factors ranging from 2 to 8. In particular, by adopting both optimizations, the verifier's CPU time can be reduced by almost an order of magnitude.

8 Related Work

In this section, we review existing literature on integrity assurance, query authentication, privacy-preserving location publication, and location provenance.

Integrity Assurance Most existing database literature on integrity is based on digital signature, where a verifier proves the integrity of the message by the signer's public key and a signature produced by the signer's private key. Classic indexing schemes have been modified to store a signature for every data value. The VB-tree [32] augments a conventional B^+-tree with a signature in each leaf entry. By verifying the signatures of all returned values, the client can guarantee the soundness of these results. To further guarantee the completeness, Pang et al. proposed signature chaining [31], which connects a signature with adjacent data values to guarantee no result can be left out. Signature chaining was also adapted to multi-dimensional indexes, such as R-tree, by Cheng and Tan [7].

Another family of data structure for integrity assurance, namely the Merkle hash tree (MHT) [26], is based on both digital signature and one-way hash function. In an MHT, only the root node is signed, whereas any other node is protected by a *digest*, a hash value jointly determined by its own value and the digests of all its children. The notion of MHT has been adapted to various index structures. Typical examples include Merkle B-tree and its variant Embedded Merkle B-tree (EMB-tree) [21], and Merkle R-tree [40], [41].

In cryptography, message authentication code (MAC) is an alternative mathematical scheme for proving the integrity of a message. It was adopted in [3] for exact-match and range queries in outsourced database. Papadopoulos et al. used standard HMAC as a component in their pseudo-random function for linear algebraic queries in outsourced data streams [33]. While these works adopt standard MAC schemes, to the best of our knowledge, our PMAC is the first MAC designed specifically for integrity assurance on spatio-temporal data.

Query Authentication Following the above work on integrity assurance, there is a large body of research works on authenticating the integrity of results from queries more complex than selection and range queries. These works study kNN queries [6], [45], [16], top-k queries [8], shortest paths [44], join queries [41], aggregation queries [20], and subgraph queries [11]. Authenticating streaming data for various aggregation queries is also studied in [22], [43], [34], [27], [33]. As for privacy-preserving query authentication, Hu et al. presented a solution for range queries in location-based service [14], based on a cryptographic construct in [31]. They also extended it to authenticate location-based top-k queries, where ranks are based on both spatial and

non-spatial scores [5]. Our work differs from these previous works as being the first work on MAC-based authentication scheme that addresses data privacy.

Location Integrity and Provenance They have been studied in mobile computing, particularly in the field of localization. Existing works can be categorized in three contexts, namely trusted hardware based, wireless infrastructure based, and peer-to-peer based context. The first context uses trusted SoC (e.g., Trusted Platform Module) to generate location proofs in end devices. Yap *et al.* proposed a secure user-centric attestation service protocol that generates location evidence using a tamper-resistant device [42]. Liu et al. proposed software abstraction on x86 and ARM architectures to offer trusted readings from GPS sensors to mobile applications [23]. The second context signs users GPS data by co-located wireless access points. The original work by Saroiu and Wolman does not consider user privacy [36], which was later addressed by Luo and Hengartner [24] through non-colluding trusted parties holding user's identity and location separately. Pham et al. also addressed privacy by only reporting location activity summaries (e.g., total walking distance) using Wi-Fi access-point networks [35]. The third context relies on collocated peer users to generate proofs. In [12], proofs gathered from trusted neighboring devices (called proof providers) are used to verify the device location in a vehicular ad-hoc network. Zhu et al. proposed APPLAUS, which assumes the peers are not trusted and detects colluding attackers by ranking and correlation clustering approaches [47], [48]. Wang et al. presented a similar scheme STAMP with an entropy-based trust evaluation approach to detect collusion and fake proofs [38]. Khan et al. studied the security and performance of a variety of secure proof schemes for location provenance, including hash chain, bloom filter, and RSA chaining [13]. Note that except for [38], the above works focus on standalone location points and cannot address the completeness issue when applied to spatio-temporal trajectories. Recently Lyu et al. proposed a Merkle hash tree based continuous location provenance protocol [25]. It requires multiple rounds of communication between wireless APs, peer witness, raw sensors of the device, and a certificate authority. Our work belongs to the first context but differs from existing works by focusing on continuous, privacy-preserving location provenance.

Privacy-Preserving Location Publication As location analytic has become popular in data mining and mobile computing community, privacy protection has been introduced to user location or trajectory publication. Main approaches include clustering [2] and location cloaking (i.e., location generalization) [39], [15]. For trajectories, location suppressing, i.e., omitting sensitive locations, and noise sampling (for differential privacy) have been proposed in [37], [17]. Our work differs from them by focusing on integrity assurance during location publication. As such, it can work orthogonally with the above works.

9 Conclusion

In this paper, we studied the problem of integrity assurance which discloses to the verifier no more information beyond the spatio-temporal predicate itself. The solution is based on prefix-verifiable MAC (PMAC), a cryptographic construct designed by us to verify the integrity of any prefix of a string. We then presented authentication protocols for both spatial and spatio-temporal predicates. Two indexing schemes for PMACs were proposed to pre-aggregate sub-trajectories and accelerate the verification process. We further proposed two optimization techniques to reduce the computational and communication costs. Our security analysis and experimental results show that this authentication scheme is both secure and efficient for practical use.

As for future work, we plan to study the integrity assurance schemes for more complex predicates. In particular, we are interested in the complement of a containment predicate — a user is "not in" a specific region. This problem is even harder as it is equivalent to authenticating that a string x has a prefix from any of a set of strings, instead of all of them.

10 Acknowledgments

This work was supported by Research Grants Council, Hong Kong SAR, China, under projects 210612, 12200914, 15238116, 12244916, 12232716, and C1008-16G, and was also supported by National Natural Science Foundation of China (Grant No: 61572413 and U1636205).

References

[1] Discrete root extract. http://e-maxx.ru/algo/discrete_root.
[2] O. Abul, F. Bonchi, and M. Nanni. Never walk alone: Uncertainty for anonymity in moving objects databases. In *Proc. of ICDE*, 2008.
[3] G. Amanatidis, A. Boldyreva, and A. O'Neill. Provably-secure schemes for basic query support in outsourced databases. In *Proc. the 21st annual IFIP WG 11.3 working conference on Data and applications security*, pages 14–30, 2007.
[4] D. Bradbury. Making sense of mobile marketing: it's about the customer journey. The Guardian, https://www.theguardian.com/technology/2014/jan/15/making-sense-of-mobile-marketing, January 2014.
[5] Q. Chen, H. Hu, and J. Xu. Authenticating top-k queries in location-based services with confidentiality. In *Proc. VLDB*, 2014.
[6] W. Cheng and K. Tan. Authenticating knn query results in data publishing. In *SDM*, 2007.
[7] W. Cheng and K. Tan. Query assurance verification for outsourced multi-dimensional databases. *Journal of Computer Security*, 2009.
[8] S. Choi, H. Lim, and E. Bertino. Authenticated top-k aggregation in distributed and outsourced databases. In *SOCIALCOM-PASSAT12*, pages 779–788, 2012.
[9] A. Department of Health. Zika virus information for clinicians and public health practitioners. http://www.health.gov.au/internet/main/publishing.nsf/content/ohp-zika-health-practitioners.htm.
[10] M. Erwig and M. Schneider. Spatio-temporal predicates. *TKDE*, 14(4):881–901, 2002.
[11] Z. Fan, Y. Peng, B. Choi, J. Xu, and S. S. Bhowmick. Towards efficient authenticated subgraph query service in outsourced graph databases. *IEEE Transactions on Services Computing*, 2014.
[12] M. Graham and D. Gray. Protecting privacy and securing the gathering of location proofs – the secure location verification proof gathering protocol. In *Proc. of 1st International Conference on Security and Privacy in Mobile Information and Communication Systems (MobiSec)*, 2009.
[13] R. Hasan, R. Khan, S. Zawoad, and M. M. Haque. Woral: A witness oriented secure location provenance framework for mobile devices. *IEEE Transactions on Emerging Topics in Computing*, 2015.
[14] H. Hu, J. Xu, Q. Chen, and Z. Yang. Authenticating location-based services without compromising location privacy. In *Proc. SIGMOD*, pages 301–312, 2012.
[15] H. Hu, J. Xu, S. T. On, J. Du, and K. Ng. Privacy-aware location data publishing. *TODS*, 35(3), 2010.
[16] L. Hu, W.-S. Ku, S. Bakiras, and C. Shahabi. Spatial query integrity with voronoi neighbors. *IEEE TKDE*, 25(4):863 –876, 2013.

[17] J. Huang, Y. Xue, Y. Zheng, R. Zhang, X. Xie, and Z. Xu. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *Proc. of ICDE*, 2013.

[18] IETF. Rfc 2104: Hmac: Keyed-hashing for message authentication. https://tools.ietf.org/html/rfc2104.

[19] J. Katz and A. Y. Lindell. Aggregate message authentication codes. In *Proceedings of the 2008 The Cryptopgraphers' Track at the RSA conference on Topics in cryptology*, 2008.

[20] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Authenticated index structures for aggregation queries. *ACM TISSEC*, 13(32):1–35, 2010.

[21] F. Li, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proc. SIGMOD*, pages 121–132, 2006.

[22] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios. Proof-infused streams: Enabling authentication of sliding window queries on streams. In *VLDB*, 2007.

[23] H. Liu, S. Saroiu, A. Wolman, and H. Raj. Software abstractions for trusted sensors. In *Proc. of ACM MobiSys*, pages 365–378, 2012.

[24] W. Luo and U. Hengartner. Veriplace: a privacy-aware location proof architecture. In *Proc. of ACM GIS*, 2010.

[25] C. Lyu, A. Pandea, X. O. Wang, J. Zhu, D. Gu, and P. Mohapatra. Clip: Continuous location integrity and provenance for mobile phones. In *Proc. of IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems*, 2015.

[26] R. C. Merkle. A certified digital signature. In *Proc. Crypto*, pages 218–238, 1989.

[27] S. Nath and R. Venkatesan. Publicly veriable grouped aggregation queries on outsourced data streams. In *Proc. ICDE*, 2013.

[28] B. News. One driver explains how he is helping to rip off uber in china. https://www.bloomberg.com/news/articles/2015-06-28/one-driver-explains-how-he-is-helping-to-rip-off-uber-in-china, June 29 2015.

[29] D. of Communications and A. the Arts. Internet governance: Online gambling. https://www.communications.gov.au/what-we-do/internet/internet-governance/online-gambling.

[30] C. C. of Insurance Regulators. Jurisdiction specific requirements life insurance and accident and sickness. http://www.ccir-ccrra.org/en/forms/Appendix_A_(En)_2_LifeInsurance-21Mar06_RV.pdf.

[31] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying completeness of relational query results in data publishing. In *SIGMOD*, pages 407–418, 2005.

[32] H. Pang and K.-L. Tan. Authenticating query results in edge computing. In *Proc. ICDE*, 2004.

[33] S. Papadopoulos, G. Cormode, A. Deligiannakis, and M. Garofalakis. Lightweight authentication of linear algebraic queries on data streams. In *Proc. ACM SIGMOD*, pages 881–892, 2013.

[34] S. Papadopoulos, Y. Yang, and D. Papadias. Continuous authentication on relational streams. *VLDBJ*, 19:161–180, 2010.

[35] A. Pham, K. Huguenin, I. Bilogrevic, and J. Hubaux. Secure and private proofs for location-based activity summaries in urban areas. In *Proc. of ACM International Joint Conference on Pervasive and Ubiquitous Computing (Ubicomp)*, 2014.

[36] S. Saroiu and A. Wolman. Enabling new mobile applications with location proofs. In *Proc. of ACM HotMobile*, 2009.

[37] M. Terrovitis and N. Mamoulis. Privacy preservation in the publication of trajectories. In *Proc. of MDM*, 2008.

[38] X. Wang, J. Zhu, A. Pande, A. Raghuramu, P. Mohapatra, T. Abdelzaher, and R. Ganti. Stamp: Ad hoc spatial-temporal provenance assurance for mobile users. In *Proc. of 21st IEEE International Conference on Network Protocols (ICNP)*, 2013.

[39] T. Xu and Y. Cai. Exploring historical location data for anonymity preservation in location-based services. In *IEEE Infocom, Phoenix, Arizona*, 2008.

[40] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios. Spatial outsoucing for location-based services. In *Proc. ICDE*, pages 1082–1091, 2008.

[41] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios. Authenticated indexing for outsourced spatial databases. *The VLDB Journal*, 18(3):631–648, 2009.

[42] L. Yap, T. Yashiro, M. Bessho, T. Usaka, M. Khan, N. Koshizuka, and K. Sakamura. Sucas: An architecture for secure user centric attestation in location-based services. In *Proc. of IEEE International Conference on Social Computing*, pages 760–767, 2010.

[43] K. Yi, F. Li, G. Cormode, M. Hadjieleftheriou, G. Kollios, and D. Srivastava. Small synopses for group-by query verification on outsourced data streams. *ACM TODS*, 34(3), 2009.

[44] M. L. Yiu, Y. Lin, and K. Mouratidis. Effcient verification of shortest path search via authenticated hints. In *Proc. ICDE*, pages 237–248, 2010.

[45] M. L. Yiu, E. Lo, and D. Yung. Authentication of moving knn queries. In *Proc. ICDE*, 2011.

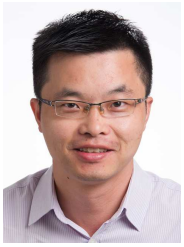
[46] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *In Proc.of International Conference on World Wild Web (WWW 2009)*, 2009.

[47] Z. Zhu and G. Cao. Applaus: A privacy-preserving location proof updating system for location-based services. In *Proc. of INFOCOM*, 2011.

[48] Z. Zhu and G. Cao. Toward privacy preserving and collusion resistance in a location proof updating system. *IEEE Transactions on Mobile Computing*, 2013.

Haibo Hu is an assistant professor in the Department of Electronic and Information Engineering, Hong Kong Polytechnic University. His research interests include information security, privacy-aware computing, and location-based services, where he has published over 60 research papers. As a principal investigator, he has received over 6 million HK dollars of external research grants. He is the recipient of ACM-HK Best PhD Paper Award, Microsoft Imagine Cup, and GS1 Internet of Things Award.

Qian Chen is a PhD student in the Department of Computer Science, Hong Kong Baptist University. His research interests include privacy-aware computing. He is a member of the Database Group at Hong Kong Baptist University. (http://www.comp.hkbu.edu.hk/~db/)



Jianliang Xu is a Professor in the Department of Computer Science, Hong Kong Baptist University. He received his PhD degree in computer science from Hong Kong University of Science and Technology in 2002. He held visiting positions at Pennsylvania State University and Fudan University. His research interests include data management, mobile/pervasive computing, wireless sensor networks, and distributed systems, where he has published more than 120 technical papers.

Byron Choi received the bachelor of engineering degree in computer engineering from the Hong Kong University of Science and Technology (HKUST) in 1999 and the MSE and PhD degrees in computer and information science from the University of Pennsylvania in 2002 and 2006, respectively. He is now an associate professor in the Department of Computer Science at the Hong Kong Baptist University.