Efficient Position Sharing for Location Privacy using Binary Space Partitioning

Marius Wernke, Frank Dürr, and Kurt Rothermel

Institute of Parallel and Distributed Systems
Universitätsstraße 38, 70569 Stuttgart, Germany
marius.wernke,frank.duerr,kurt.rothermel@ipvs.uni-stuttgart.de

Abstract. Millions of users use location-based applications (LBAs) to share their positions with friends, request information from points of interest finders, or get notifications from event finders, etc. Such LBAs are typically based on location servers (LSs) managing mobile object positions in a scalable fashion. However, storing precise user positions on LSs raises privacy concerns, in particular, if LS providers are non-trusted. To solve this problem, we present *PShare-BSP*, a novel approach for the secure management of private user positions on non-trusted LSs. *PShare-BSP* splits up precise user positions into position shares and distributes them to *different* LSs of different providers. Thus, a compromised provider only reveals user positions with degraded precision. Nevertheless, LBAs can combine several shares from different LSs to increase their precision.
PShare-BSP improves on our previous position sharing approaches [4, 15, 17]: It uses a deterministic share generation approach based on binary space partitioning to avoid probabilistic attacks based, for instance, on Monte Carlo simulations. Moreover, it significantly decreases the computational complexity and increases the efficiency by reducing the update costs for succeeding position updates.

Key words: Location based applications, position sharing, privacy

1 Introduction

The widespread adoption of mobile devices with integrated positioning systems such as GPS has led to a drastic increase of the usage of location based applications (LBAs). For instance, points of interest finders such as Qype can be used to determine the next restaurant or gas station based on the current user position. Friend finder applications such as Loopt notify users when friends reach their vicinity. Moreover, geosocial networks such as Facebook Places, Foursquare, and Yelp let users “check-in” at locations to share their positions with friends.

LBAs typically make use of so-called location servers (LSs) to manage position information of mobile objects. Mobile objects send their position to the LS, and LBAs act as *clients* to query the LS for mobile object positions. LSs allow for the efficient and scalable management of mobile object positions, in particular, if position information is required by several clients since the LS relieves the mobile objects from sending positions to each client individually. A number of LSs are already provided on the Internet today, for instance, by Google (Latitude), Yahoo (Fire Eagle), and other service providers.

However, providing precise user positions to LSs raises privacy concerns. These concerns are intensified by a number of incidents where private data was revealed [3] and where even providers that were deemed to be trustworthy did not succeed to protect private user data. Such incidents include attacks, leaking or losing personal information. As a consequence, we cannot trust any provider to protect our data. Thus, security mechanisms for the secure management of position information taking non-trusted providers into account are needed.

Spatial obfuscation [2, 5] is a common principle to protect user location privacy in non-trusted systems. Instead of providing precise user positions to an LS, users degrade the precision of their positions and only provide this degraded information to the LS. However, such spatial obfuscation approaches limit the maximum allowed precision of a user position that can be provided to the clients of the LS by the trustworthiness of the LS. Consequently, we cannot provide a client with more precise positions than stored by the LS, no matter how much we trust the client. This might have severe impact on LBAs since usually the quality of applications might also degenerate with the quality of the provided position information. Furthermore, we cannot provide different precisions to different clients with different quality of service demands and trust levels.

Our position sharing approaches presented in [4, 15, 17] overcome this problem. Using position sharing, the mobile object splits up its precise position into a set of *position shares*, where each share represents an imprecise position, and distributes the generated shares to *different* LSs of different providers. Therefore, a compromised LS only reveals degraded position information, while clients can combine several shares to increase their precision. Thus, we can provide different precisions to different clients without storing precise positions at the LSs.

In this paper, we present *PShare-BSP*, a new position sharing approach based on binary space partitioning. We improve our previous position sharing approaches developed in our *PriLoc*-Project [11] as follows. In comparison to [4, 15], *PShare-BSP* uses a deterministic instead of a probabilistic approach to increase robustness against attacks based on Monte Carlo simulations. Compared to [17], which is based on multi-secret sharing, *PShare-BSP* reduces the computational complexity by avoiding complex cryptographic operations. Moreover, we optimize the communication costs for position updates significantly by avoiding updating all LSs for each new position.

The rest of this paper is structured as follows: First, we present an overview of the related work in Sec. 2 and describe our system model in Sec. 3. Then, we introduce our new position sharing approach in Sec. 4 and analyze its security in Sec. 5. In Sec. 6, we present our evaluation results, before we summarize the paper in Sec. 7 together with an outlook onto future work.

2 Related Work

Approaches providing location privacy can be categorized whether they require a trusted third party or not.

Approaches with trusted third party: The most prominent approach providing user privacy is *k-anonymity* [7] guaranteeing that a user is indistinguishable from $k-1$ other users. However, *k*-anonymity and its extensions such as *l-diversity* [9] and *t*-

closeness [8] usually require a trusted anonymizer, whereas we aim to provide user privacy without any trusted third party.

Approaches without trusted third party: A simple approach to protect private positions is to store encrypted positions on LSs. This approach does not rely on any security mechanism of the LSs. However, the LSs cannot perform essential computations such as nearest neighbor or range queries on the server side.

Existing *dummy-approaches* such as [12] generate several false positions (dummies) and send them together with the true position of the user to an LS. However, the provided privacy of these approaches is reduced if dummies can be identified. As shown in [10], this is possible even if sophisticated algorithms are used for dummy generation.

Spatial obfuscation approaches such as [2, 5] provide user privacy by sending positions of degraded precision to the LS. In general, obfuscation does not require a trusted third party. However, the maximum precision that can be provided to the clients is limited by the trustworthiness of the LS. Furthermore, an incremental precision increase for different clients as supported by our position sharing approach is not supported.

Our *position sharing* approach presented in [4] and its extension to maps [15] provide different precisions to clients by combining several position shares from different LSs based on random geometric transformations. Although these approaches provide sufficient privacy in many scenarios, an attacker can use a Monte Carlo simulation to derive positions of higher precision than intended with a certain probability. In contrast, we propose a deterministic approach in this paper that makes such attacks impossible.

In [17] we presented a position sharing approach using a multi-secret sharing scheme. In this paper, we present *PShare-BSP* that is based on binary space partitioning rather than cryptographic operations, which are less complex and thus increase computational efficiency. Finally, *PShare-BSP* also increases communication efficiency by implementing a protocol for optimized share updates.

3 System Model

The system model consists of three different components as depicted in Fig. 1.

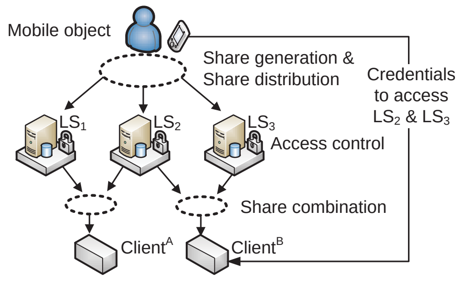

Fig. 1. System Components

The **mobile object** (MO) uses an integrated positioning system, such as GPS, to determine the current MO's position π. We assume that no malicious software component is running on the MO that can access π, e.g., using existing mobile trusted computing approaches such as [6]. We assume π to be a Cartesian point coordinate that is perfectly accurate and precise. To map ellipsoidal longitude and latitude coordinates to Cartesian coordinates, we can use a common map projection such as the Universal Transverse Mercator (UTM) projection, which divides the Earth into sixty zones, each representing a six degree band of longitude. In case the MO travels from one zone to another for a new check-in, the corresponding zone is changed as soon as the obfuscation area of the MO is completely covered by the new zone.

The MO executes a local software component for share generation splitting up π into a master share m_π, called m-share, and a set of refinement shares $S = \{r_{\pi,1}, \ldots, r_{\pi,l_{max}}\}$, called r-shares, by using the function

$$generate(\pi, l_{max}) = (m_\pi, \{r_{\pi,1}, \ldots, r_{\pi,l_{max}}\}).$$

Parameter l_{max} is defined by the MO and defines the number of $l_{max}+1$ different precision levels offered to clients. The MO's position of precision level l with $0 \leq l \leq l_{max}$ is denoted as $p(\pi, l)$. The m-share m_π defines position $p(\pi, 0)$ with a precision which is low enough to be published without raising privacy concerns. The r-shares can be used to increase the precision of the m-share ($p(\pi, 0)$) by providing refinement information stored in the r-shares. After shares are generated, they are distributed to different LSs of different providers. The communication between the MO and the LSs must take place over secure channels to avoid modification, sniffing, and message injection.

Location servers (LSs) store position shares and answer queries from different clients by returning the corresponding shares. Each LS implements an access control mechanism, as presented for example in [1], to manage the access of different clients to shares. The access rights are defined by the MO and provided to the clients as credentials to access a certain number of shares.

Clients query several shares from different LSs and use *share combination*

$$combine(m_\pi, \{r_{\pi,1}, \ldots, r_{\pi,l}\}) = p(\pi, l)$$

to increase the provided precision of $p(\pi, 0)$ to $p(\pi, l)$ with $0 < l \leq l_{max}$ by combining the m-share m_π and l r-shares from different LSs. The communication between clients and the LSs must also be protected by secure channels.

4 Position Sharing Approach

In this section, we present our position sharing approach *PShare-BSP* implementing the functions for share generation and combination introduced above. Then, we present an optimization reducing the number of required updates.

4.1 Basic Approach

Both share generation and combination depend on the concept of how to translate the exact MO's position π into an obfuscated position $p(\pi, l)$ of a certain precision level l.

Geometrically, an obfuscated position $p(\pi, l)$ of a given precision level l is defined as $p(\pi, l) = ((x_l, y_l), 2^{l_{max}-l})$ representing a square area that contains π (cf. Fig. 2). The tuple (x_l, y_l) denotes the coordinates of the lower left (south-west) corner of the square area, and $2^{l_{max}-l}$ denotes the side length measured in meters. The side length $2^{l_{max}-l}$ defines the precision of $p(\pi, l)$. We assume that a maximum precision of 1 meter, which is well below the precision of common positioning systems such as GPS, is sufficient for every practical application. Therefore, we set the precision of the position $p(\pi, l_{max})$ of the highest precision level l_{max} to 1 meter. What remains to be defined is how the coordinates (x_l, y_l) defining $p(\pi, l)$ are chosen based on the MO's precise position π with the coordinates $\pi.x$ and $\pi.y$. Numerically, $\pi.x$ and $\pi.y$ can be expressed in the binary numeral system with a system-defined bit length n as

$$\pi.x = \sum_{k=0}^{n-1} \alpha_k * 2^k = (\alpha_{n-1} \cdots \alpha_1 \alpha_0)_2 \text{ and } \pi.y = \sum_{k=0}^{n-1} \beta_k * 2^k = (\beta_{n-1} \cdots \beta_1 \beta_0)_2.$$

For $p(\pi, l)$ we define the coordinates (x_l, y_l) as the coordinates of $\pi.x$ and $\pi.y$ with the $l_{max} - l$ least significant bits set to zero. These bits are the undefined bits of $p(\pi, l)$. The position of the i-th undefined bit of a coordinate is equal to its $(l_{max} + 1) - i$ least significant bit. The precision value of $p(\pi, l)$ is set to $2^{l_{max}-l}$ defining the range of the $l_{max} - l$ undefined bits. The less undefined bits exist, the higher is the precision of a position. As an example consider Fig. 2 where the undefined bits of $p(\pi, l)$ that were set to zero are underlined for each level l and $l_{max} = 3$.

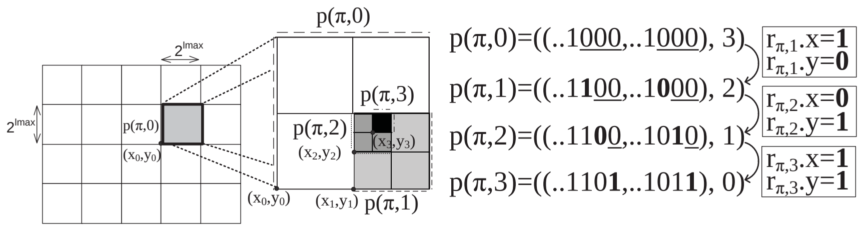

Fig. 2. Grid and refinement example for $p(\pi, 0)$ based on $l_{max} = 3$

The m-share m_π represents the coarsest obfuscation area $p(\pi, 0)$ with the coordinates (x_0, y_0) and the precision value l_{max}. By replacing the l_{max} least significant bits of $\pi.x$ and $\pi.y$ by zero when calculating $p(\pi, 0)$, l_{max} deterministically defines a partitioning of the movement area into a grid of cells with a side length of $2^{l_{max}}$ meter.

To increase the precision of the m-share m_π, the undefined bits of $p(\pi, 0)$ have to be defined. To this effect, we use a set $S = \{r_{\pi,1}, \ldots, r_{\pi,l}\}$ of $l \leq l_{max}$ r-shares, where each r-share $r_{\pi,i}$ defines the i-th undefined bit of the x and y value of $p(\pi, 0)$. At the same time the precision is improved by decreasing the side length value of $p(\pi, 0)$. Thus, position $p(\pi, 0)$ can be refined up to $p(\pi, l)$ by incrementally substituting the undefined bits of $p(\pi, 0)$ by the corresponding bits of the r-shares. The number of generated r-shares for each position update is defined by the value l_{max} such that the precision of $p(\pi, 0)$ can be increased up to l_{max} different precision levels from $p(\pi, 1)$ to $p(\pi, l_{max})$. An example is shown in Fig. 2 on the right.

Next, we describe the share generation using function $generate(\pi, l_{max})$ (cf. Alg. 1). The m-share is calculated by function $generateMShare(\pi, l_{max})$ by setting the l_{max} least significant bits of $\pi.x$ and $\pi.y$ to zero. Each r-share $r_{\pi,i}$ with $1 \leq i \leq l_{max}$ defines the two bits of $\pi.x$ and $\pi.y$ corresponding to the bits at the position of the i-th undefined bit in $p(\pi, 0)$. These bits are returned by function $getBits(\pi, i)$.

Algorithm 1
PShare-BSP: Share generation

```
Function: $generate(\pi, l_{max})$
1: $m_\pi \leftarrow generateMShare(\pi, l_{max})$
2: for $i = 1$ to $l_{max}$ do
3:     $r_{\pi,i} \leftarrow getBits(\pi, i)$
4: end for
5: $S \leftarrow \{r_{\pi,1}, \ldots, r_{\pi,l_{max}}\}$
6: return $m_\pi, S$
```

Algorithm 2
PShare-BSP: Share combination

```
Function: $combine(m_\pi, \{r_{\pi,1}, \ldots, r_{\pi,l}\})$
1: $p(\pi, 0) \leftarrow m_\pi.p(\pi, 0)$
2: for $i = 1$ to $l$ do
3:     $p(\pi, i) \leftarrow setBits(p(\pi, i-1), r_{\pi,i})$
4: end for
5: return $p(\pi, l)$
```

The share combination function $combine(m_\pi, \{r_{\pi,1}, \ldots, r_{\pi,l}\})$ is implemented as shown in Alg. 2. It takes as input the m-share m_π representing the obfuscation area $p(\pi, 0)$ and a sequence of r-shares $r_{\pi,1}, \ldots, r_{\pi,l}$. In order to be able to refine the position up to a certain precision level l, the sequence of r-shares must contain all r-shares for the levels 1 to l. The refined position $p(\pi, l)$ is calculated by replacing stepwise the i-th undefined bit of the x and y values of $p(\pi, 0)$ by the bits of r-share $r_{\pi,i}.x$ and $r_{\pi,i}.y$ for $1 \leq i \leq l$. This step is implemented by function $setBits(p(\pi, i-1), r_{\pi,i})$.

To protect multiple position updates, we use the idea of delaying updates if they would reveal additional information to an attacker. For a detailed description of an attack on multiple position updates we refer to [5]. We analyzed the counter measure of delayed updates in our previous work [17] and use the same approach for *PShare-BSP* to resist attacks on multiple position updates.

4.2 Update Optimization

A drawback of position sharing is that multiple shares have to be updated per position fix instead of one. To alleviate this problem, we present an optimization called *PShare-BSP*^{opt} reducing the number of messages required for position updates significantly.

Existing position sharing approaches [4, 15, 17] and *PShare-BSP* presented in this paper need to update the m-share and all r-shares for each new position. The general idea of *PShare-BSP*^{opt} is that, initially, the m-share and all r-shares are updated once. For the following $k-1$ position updates, we re-use the r-shares to refine the positions of several m-shares and update only the m-share for every new position. To this end, an m-share contains a partially encrypted position that can be decrypted by the bits of the corresponding key that is split up and stored within different r-shares. Therefore, only one share has to be updated per new position rather than $1 + l_{max}$ (one m-share and l_{max} r-shares). In order to allow for the re-use of r-shares for the next $k-1$ updates, we have to include additional information for each r-share. The value of k can be defined by the MO and determines the number of times the r-shares can be re-used before they must be updated. For simplicity, we limit our explanations to the x coordinate. The y coordinate is handled identically.

For each position update we use a one-time pad encryption to protect the l_{max} least significant bits of $\pi.x$. Generally, a one-time pad encryption can be used to protect a secret by XORing it with a random set of bits defining the key of the encryption. The result of the encryption is a cipher that does not provide any information about the secret without the key. We define the secret s_x as the l_{max} least significant bits of $\pi.x$. The corresponding key is of length l_{max} and called *refinement-key* (r-key). The cipher c_x is calculated by bitwise XORing s_x with the corresponding r-key as $c_x = s_x$ XOR r-key$_x$. Cipher c_x is then stored within the m-share m_π in addition to $p(\pi, 0)$ as shown in Fig. 3. The idea is now to split up r-key$_x$ into its l_{max} bits and distribute them as part of the r-shares to different LSs. The refinement of $p(\pi, 0)$ to $p(\pi, l)$ for a certain precision level l requires the l most significant bits of the r-key$_x$, denoted as r-key$_x[l]$. The refinement is done by decrypting c_x with the combined r-key$_x[l]$ of the r-shares as shown in Fig. 4. The result of the decryption defines the l most significant bits of s_x, denoted as $s_x[l]$. Finally, $p(\pi, 0)$ is refined to $p(\pi, l)$ by substituting the l most significant undefined bits of $p(\pi, 0)$ by $s_x[l]$. The $l_{max} - l$ undefined bits remain zero.

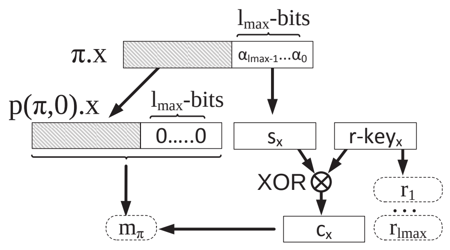

Fig. 3. Share generation overview

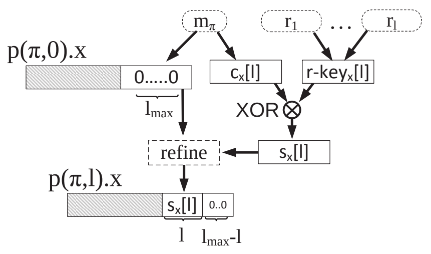

Fig. 4. Share combination overview

To fulfill the optimization goal, we provide within each r-share k bits that can be used to reconstruct the r-keys of k updates. Therefore, the r-share r_i stores the i-th bit of any of the k generated r-keys. The different r-keys are referenced by an id, denoted as r-key^{id}. The correlation of r-shares and r-keys is shown in Fig. 5, where LS_i stores the r-share r_i representing a secure random set of k bits.

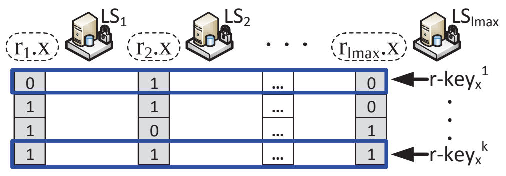

Fig. 5. Correlation of r-shares and r-keys

Next, we describe the detailed algorithms for share generation and combination. The share generation presented in Alg. 3 checks whether a distributed r-key can be used for the next update. If no unused r-key is available, a set of k new r-keys and l_{max} new r-shares is generated. Then, the id of the r-key to use is set. Finally, Alg. 4 generates the m-share m_π including cipher c_x. After generation, the shares are distributed to the LSs.

Share combination presented in Alg. 5 reconstructs position $p(\pi, l)$ of precision level l by combining the m-share m_π with l r-shares $r_1, \ldots, r_l$. The l r-shares can be used to compose the corresponding *r-key*$_x^{id}[l]$ of length l. Then, cipher c_x of m_π is XORed with *r-key*$_x^{id}[l]$. The reconstructed refinement bits are then used as substitute of the corresponding undefined bits. The remaining $l_{max} - l$ bits are still undefined. Without knowing the missing $l_{max} - l$ r-shares and thus the corresponding parts of the *r-key*, it is not possible to further increase the precision of $p(\pi, l)$.

Algorithm 3 *PShare-BSP*^{opt}: Share generation

```
Function: $generateSharesOPT(\pi, l_{max})$
 1: if $noUnusedRKeyAvailable()$ then
 2:     for $id = 1$ to $k$ do
 3:         $r\text{-}key_x^{id} \leftarrow getRandBits(l_{max})$
 4:     end for
 5:     for $i = 1$ to $l_{max}$ do
 6:         $r_i.x \leftarrow getBitsFromRKeys(i)$
 7:     end for
 8:     $S \leftarrow \{r_1, \ldots, r_{l_{max}}\}$
 9: end if
10: $id \leftarrow getUnusedRKeyID()$
11: $m_\pi \leftarrow generateMShareOPT(\pi, l_{max}, r\text{-}key_x^{id}, id)$
12: return $m_\pi, S$
```

Algorithm 4 *PShare-BSP*^{opt}: m-share generation

```
Function: $generateMShareOPT(\pi, l_{max}, r\text{-}key_x^{id}, id)$
 1: $m_\pi \leftarrow generateMShare(\pi, l_{max})$
 2: $s_x \leftarrow getXRefinement(\pi.x, l_{max})$
 3: $m_\pi.c_x \leftarrow s_x$ XOR $r\text{-}key_x^{id}$
 4: $m_\pi.rKeyID \leftarrow id$
 5: return $m_\pi$
```

Algorithm 5 *PShare-BSP*^{opt}: Share combination

```
Function: $combineOPT(m_\pi, \{r_1, \ldots, r_l\})$
 1: $r\text{-}key_x^{id}[l] \leftarrow getRKey(m_\pi.rKeyID, \{r_1.x, \ldots, r_l.x\})$
 2: $p(\pi, l).x \leftarrow setXBits(m_\pi.p(\pi, 0).x, m_\pi.c_x[l]$ XOR $r\text{-}key_x^{id}[l])$
 3: return $p(\pi, l)$
```

To guarantee the security of *PShare-BSP*^{opt}, it is essential that each r-key is only used once. Otherwise the encryption could possibly be broken. Therefore, we use each r-key only once and renew the r-shares after all k r-keys have been used.

5 Security Analysis

In this section, we present our attacker model and analyze various attacks.

5.1 Attacker Model

We assume malicious clients and malicious LSs that could be compromised as possible attackers. Malicious clients are a special sub-case of malicious LSs. We consider the case that the LSs storing the m-share and l r-shares are compromised and collude together such that the attacker knows an MO's position $p(\pi, l)$ of precision level l. An ideal position sharing approach will not allow an attacker knowing $p(\pi, l)$ to derive a position with a precision beyond the precision of $p(\pi, l)$.

We assume a free-space mobility model where each position π is equally likely. Different *probability distribution functions* (pdfs) of positions are part of our future work. For instance, an attacker could calculate a pdf for $p(\pi, l)$ using additional map knowledge and exclude non-reachable areas from $p(\pi, l)$.

5.2 Attacks on *PShare-BSP* and *PShare-BSP*^{opt}

PShare-BSP and *PShare-BSP*^{opt} both deterministically transform the MO's position π for a given value of l_{max} to position $p(\pi, 0)$. For each position $\pi' \in p(\pi, 0)$ it holds that the same position $p(\pi, 0)$ is calculated. This is guaranteed by mapping the l_{max} least significant bits of the x and y coordinate to zero, independent whether the bit was zero or one before. An attacker knowing $p(\pi, l)$ trying to increase his precision to level $l+1$ would have to determine the first unknown bit of the x and the y coordinate of $p(\pi, l)$. Thus, the attacker could try to analyze the undefined bits of $p(\pi, l)$ and try to calculate the inverse function of the mapping to zero values. However, as values of zero and one are both mapped to zero, no further information is revealed to the attacker without knowing the corresponding r-share r_{l+1}. The same holds for an attacker analyzing the four possible refinement areas of $p(\pi, l)$ on level $l+1$. All four areas are of the same size and share therefore the same probability to cover π. Thus, the probability of selecting the correct area on level $l+1$ is equal to randomly guessing a certain area on level $l+1$.

An attacker knowing m_π in *PShare-BSP*^{opt} also knows ciphers c_x and c_y. As proven in [13], the cipher of a one-time pad encryption provides no information about the protected secret, even if the attacker has infinite computational power. Thus, it is not possible for an attacker to increase precision from $p(\pi, l)$ to $p(\pi, l+1)$ without knowing r-share r_{l+1} defining the corresponding parts of the r-keys to decrypt c_x and c_y.

In addition to analyzing the undefined bits, an attacker could also try to analyze the generated positions. For instance, an attacker could try to use a *region intersection attack* [16] that can be successful on obfuscation-based approaches if different obfuscation areas are generated for the same position π. However, we deterministically generate for each position π and each level l always the same obfuscation area. For the MO's value of l_{max} and two different positions π' and π'' either the obfuscation areas of level l are equal, i.e., $p(\pi', l) = p(\pi'', l)$, or the areas do not intersect each other, i.e., $p(\pi', l) \cap p(\pi'', l) = \emptyset$. In both cases, an attacker cannot refine $p(\pi, l)$.

A *probability distribution attack* [14] calculates the probability that the MO is located in certain areas. It is most beneficial for probabilistic privacy algorithms that might lead to an uneven distribution of (possible) MO positions within the obfuscation area. For instance, the algorithm presented in [4] leads to a concentration in the center of the obfuscation area. We try to avoid such concentrations and strive for a uniform distribution within the obfuscation area. Our deterministic share generation algorithm guarantees that each position $\pi' \in p(\pi, l)$ would lead to the obfuscation area $p(\pi, l)$ with the same probability for a certain precision level l. Running a Monte Carlo simulation for the deterministic obfuscation and share generation algorithm over $\pi' \in p(\pi, l)$ leads to a uniform distribution over $p(\pi, l)$. Therefore, probability distribution attacks do not provide any additional information to the attacker.

By design, we only generate and update new position shares if they are not vulnerable to a *maximum velocity attack* [5] by using delayed updates as counter measure. Thus, an attacker cannot increase his precision by analyzing succeeding position updates using a maximum velocity attack.

6 Evaluation

Next, we analyze the computational efficiency of our approaches by measuring the performance of share generation and share combination using a prototype implementation of our system. Afterwards, we analyze the bandwidth efficiency of our approaches.

6.1 Performance Evaluation

Generally, the share generation is performed on a resource-constrained mobile device with limited CPU power and battery capacity. Even on such resource-poor devices, share generation must be possible in short time, which results in a small overhead in terms of energy. To show the performance of our approaches, we measured the overall time for share generation on a state of the art mobile device (HTC Desire HD). We measured the time to create one m-share and one to 15 r-shares and plotted the overall time over the number of r-shares in Fig. 6. As reference values we used the results presented in [4] for a random share generation algorithm (denoted as *RSG*) and [17] for *PShare-GLM* based on multi-secret sharing techniques. We limited the number of generated r-shares to 15, because a precision of 32768 m should be sufficiently coarse to provide user privacy. As we can see, the share generation of both approaches stays well below 1 milliseconds even when providing $k = 184$ different r-keys within the r-shares. Thus, we can state that the share generation is highly efficient and suitable even for resource-poor devices.

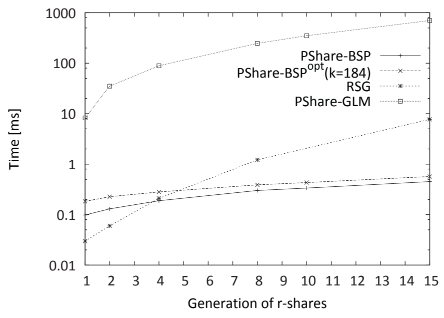

Fig. 6. Performance of share generation

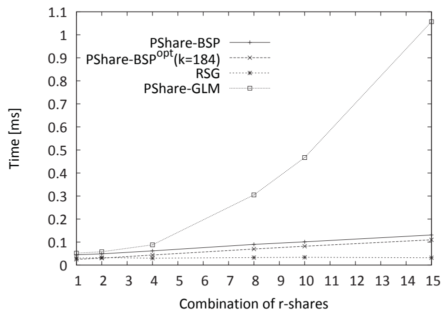

Fig. 7. Performance of share combination

In contrast to share generation, the share combination is done by clients (location-based applications) that typically run in the infrastructure with no energy restriction and high computational power. We measured the time required to combine one m-share with up to 15 r-shares on a state of the art personal computer (Intel Core 2 Duo, 2.53 GHz, 3 GB RAM). As we can see in Fig. 7, share combination is calculated very efficiently in less than 150 microseconds even for a larger number of shares.

6.2 Bandwidth Efficiency

To analyze the efficiency of our approaches in terms of communication overhead, we compare the number of required update messages for both approaches. We assume that the MO performs a number of $k' \leq k$ succeeding position updates using $l_{max} + 1$ LSs to store the generated m-share and the l_{max} r-shares. Then, for *PShare-BSP* the MO has to send in total $k' * (1 + l_{max})$ update messages, where each of the k' position updates triggers an update of the m-share and all l_{max} r-shares. For *PShare-BSP*^{opt} we update once the m-share and all l_{max} r-shares while for the next $k - 1$ updates only the m-share has to be updated. This results in a total number of $k' + l_{max}$ update messages. The r-shares are updated after $k' = k$ position updates were sent.

Next, we analyze the generated network load by analyzing the different share sizes and the overhead of lower level communication protocols. Each share has a share-ID (32 bits), a user-ID (32 bits), and a type definition (8 bits). In *PShare-BSP*, the m-share m_π adds position information (112 bits) and a list of ids defining the r-shares of m_π (l_{max} * 32 bits). An r-share r_π adds 2 bits for its refinement property. In *PShare-BSP*^{opt}, the m-share m_π^{opt} adds cipher c_x (l_{max} bits), cipher c_y (l_{max} bits), and the used r-key id (32 bits) to m_π. The additional payload of m_π^{opt} compared to m_π is denoted as Δm_π^{opt}. Each r-share r^{opt} consists of k (32 bits) and $2 * k$ bits to reconstruct r-key$_x$ and r-key$_y$.

The network load of *PShare-BSP* for k' position updates is $NL_{basic} = k' * ((size(m_\pi) + o) + l_{max} * (size(r_\pi) + o))$ bits, where o defines the protocol overhead introduced by lower level protocols for each message. The network load of *PShare-BSP*^{opt} is $NL_{opt} = k' * (size(m_\pi^{opt}) + o) + l_{max} * (size(r^{opt}) + o)$ bits. Comparing NL_{basic} and NL_{opt} leads to

$$k' \geq \frac{l_{max} * (size(r^{opt}) + o)}{l_{max} * (size(r_\pi) + o) - size(\Delta m_\pi^{opt})} \tag{1}$$

denoting the number of k' updates that have to be sent until *PShare-BSP*^{opt} outperforms *PShare-BSP* and $NL_{opt} \leq NL_{basic}$ holds. The size of the message overhead measured for sending a share over TCP/IP is $o = 320\,bits$. For $l_{max} = 16$ generated r-shares and for example $k = 128$ this results in a value of $k' \geq 1.72$. This means that *PShare-BSP*^{opt} outperforms *PShare-BSP* as soon as the second MO's position is updated. By using Equation 1, we can calculate that *PShare-BSP*^{opt} outperforms *PShare-BSP* always with the second update as long as $k \leq 184$. For sending $k' = k = 184$ position updates *PShare-BSP* generates a total network load of NL$_{basic}$= 172 040 bytes when also taking the TCP/IP overhead into account. *PShare-BSP*^{opt} at the same time only generates a load of NL$_{opt}$= 27 896 bytes. This results in a reduction of 83.8% of the generated network load. By considering the additional overhead required to provide secure channels by using TLS, the overhead of each message is further increased. Thus, reducing the number of messages by *PShare-BSP*^{opt} further increases its efficiency.

In addition to optimizing the updates of shares—i.e., the communication between MO and LSs—, PShare-BSP^{opt} also optimizes the communication between the LSs and the clients. A client has to query all of its accessible r-shares only once within k updates instead of querying the r-shares every time a new position of the MO is updated.

7 Summary and Future Work

In this paper, we presented a new position sharing approach protecting user privacy in non-trusted systems of third-party location servers (LSs) and clients. *PShare-BSP* splits up a precise user position into position shares of limited precision, which are distributed to different LSs of different providers. Different clients can then combine several shares and increase their provided precision. Our approach has the advantage that a compromised server only reveals positions of degraded precision instead of precise positions.

We improve existing position sharing approaches [4, 15, 17] by providing a deterministic approach using binary space partitioning to avoid probabilistic attacks and by decreasing the computational complexity significantly by one order of magnitude compared to [4] and by more than three orders of magnitude compared to [17]. Furthermore, we presented an extension for *PShare-BSP* reducing the number of required messages for multiple position updates significantly.

As future work we will consider map knowledge and semantic knowledge, for instance, periodic behavior of users, knowledge about points of interest, etc. that could be used by an attacker to increase precision. Furthermore, we will consider replication strategies to increase the availability of shares without decreasing privacy.

References

1. P. A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security*, 10:241–271, 2002.
2. M. Damiani, C. Silvestri, and E. Bertino. Fine-grained cloaking of sensitive positions in location-sharing applications. *Pervasive Computing*, 10:64 –72, 2011.
3. DATALOSSDB. www.datalossdb.org, Apr. 2012.
4. F. Dürr, P. Skvortsov, and K. Rothermel. Position sharing for location privacy in non-trusted systems. In *Proceedings of the 9th IEEE International Conference on Pervasive Computing and Communications (PerCom '11)*, 2011.
5. G. Ghinita, M. L. Damiani, C. Silvestri, and E. Bertino. Preventing velocity-based linkage attacks in location-aware applications. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '09)*, 2009.
6. P. Gilbert, L. P. Cox, J. Jung, and D. Wetherall. Toward trustworthy mobile sensing. In *Proc. of the 11th Workshop on Mobile Computing Systems & Applications (HotMobile '10)*, 2010.
7. P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preventing location-based identity inference in anonymous spatial queries. *IEEE Transactions on Knowledge and Data Engineering*, 19:1719 –1733, 2007.
8. N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *IEEE 23rd International Conference on Data Engineering (ICDE '07)*, 2007.
9. A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1:3, 2007.
10. S. T. Peddinti and N. Saxena. On the limitations of query obfuscation techniques for location privacy. In *Proc. of the 13th Int. Conference on Ubiquitous Computing (UbiComp '11)*, 2011.
11. PriLoc-Project. www.PriLoc.de, Nov. 2012.
12. P. Shankar, V. Ganapathy, and L. Iftode. Privately querying location-based services with sybilquery. In *Proceedings of the 11th International Conference on Ubiquitous Computing (UbiComp '09)*, 2009.
13. C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, 1949.
14. R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, and J.-P. Hubaux. Quantifying location privacy. In *Proc. of the 2011 IEEE Symposium on Security and Privacy (SP '11)*, 2011.
15. P. Skvortsov, F. Dürr, and K. Rothermel. Map-aware position sharing for location privacy in non-trusted systems. In *Proceedings of the 10th International Conference on Pervasive Computing (Pervasive '12)*, 2012.
16. N. Talukder and S. I. Ahamed. Preventing multi-query attack in location-based services. In *Proceedings of the 3rd ACM Conference on Wireless network security (WiSec '10)*, 2010.
17. M. Wernke, F. Dürr, and K. Rothermel. PShare: position sharing for location privacy based on Multi-Secret sharing. In *Proceedings of the 10th IEEE International Conference on Pervasive Computing and Communications (PerCom '12)*, 2012.