PROPS: A PRivacy-Preserving Location Proof System

Sébastien Gambs, Marc-Olivier Killijian, Matthieu Roy, Moussa Traoré

PROPS : A PRivacy-preserving lOcation Proof System

Sébastien Gambs[1] Marc-Olivier Killijian[2,3] Matthieu Roy[2,3] Moussa Traoré[2,4]
[1]Université de Rennes 1 – Inria, avenue du général Leclerc, 35042 Rennes, France
[2]CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[3]Université de Toulouse, INSA, LAAS, F-31400 Toulouse, France
[4]Université de Toulouse, INP, LAAS, F-31400 Toulouse, France

***Abstract*—A secure location-based service requires that a mobile user certifies his position before gaining access to a resource. Currently, most of the existing solutions addressing this issue assume a trusted third party that can vouch for the position claimed by a user. However, as computation and communication capacities become ubiquitous with the large scale adoption of smartphones by individuals, we propose to leverage on these resources to solve this issue in a collaborative and private manner. More precisely, we introduce PROPS, for PRivacy-preserving lOcation Proof System, which allows users to generate proofs of location in a private and distributed way using neighboring nodes as witnesses. PROPS provides security properties such as unforgeability and non-transferability of the proofs, as well as resistance to classical localization attacks.**

I. Introduction

A Location-Based Service (*LBS*) takes advantage of the position of its users to deliver a service tailored to their current or past geolocated context. In practice, the position that a user transmits to an LBS is often computed determined by his own device. Thus, a malicious user can lie about his position by having his device transmitting a location of his choice. This type of attack can have a severe impact on applications such as real-time traffic monitoring, location-based access control, discount tied to the visit of a particular shop or local electronic election, to name a few.

To counter this threat, an LBS should require its users to prove their actual or past position before granting them access to resources. This notion has been formalized through the concept of *location proof* (LP), which is a digital certificate attesting the position of a user at a specific moment in time. A location proof architecture is a trusted architecture that users can interact with to acquire LPs in a secure manner.

However, relying on a dedicated architecture to certify the position of users raises important privacy concerns. First, the location privacy of users can be breached due to their regular interactions with the infrastructure (*traceability* issue). Another issue is for a malicious user to collect proof on behalf of another user with whom he colludes. This problem is known as the *terrorist fraud* in the literature of distance-bounding protocols. Furthermore, since often a LP is actually mainly a timestamped signature of a position, there is no mean for a user to change the granularity of the position endorsed by a LP without risking to tamper with its integrity. This property would be particularly interesting with respect to data minimization as it would enable a user to reveal only the granularity of his position (street, district, town, ...) needed by the LBS to ensure its functionality. Finally, in most of the current architectures [16], [27], [24], LPs are stored on centralized servers (although sometimes encrypted [19]) resulting in users losing control of their own location data. As a result, their *location privacy* might be compromised by hackers or simply abused by the LBS provider. In addition, the replication of storage servers is classically used to ensure the reliability of the system, but this increases at the same time both the risk of leakage and the deployment cost.

Contributions. In this paper, we introduce PROPS, a PRivacy-preserving lOcation Proof System, that addresses the above mentioned challenges. More precisely, our contributions are the following.

- We give a complete description of properties that are required to build a secure and privacy-preserving location proof architecture.
- We describe a distributed and collaborative location proof architecture preserving the privacy of users without relying on trusted or semi-trusted parties contrary to previous works. Our solution is based on the notion of *location proof share* (LPS), which denotes a timestamped digital signature of the position of a user generated by a nearby user. A collection of LPSs is then used to generate a LP that can reveal the user's position at particular granularity to a service. By relying on zero-knowledge proofs, our solution allows a user to prove that he is the legitimate owner of a LPwithout disclosing his identity. In addition, the proposed architecture is resistant against to classical attacks against location proof architectures.
- We provide an implementation of PROPS on a mobile platform and analyze its performances.

The outline of this paper is as follows. First in Section II, we describe the entities participating to the location proof architecture, the concept of location proof, the system assumptions as well as the adversary models we consider. Then in Section III, we define the privacy and security requirements that a location proof architecture should fulfill before briefly presenting the building blocks upon which PROPS is constructed in Section IV. Afterwards, we present the different phases of PROPS in Section V, before analyzing its security and privacy in Section VI and reviewing the implementation and its performance in Section VII. Finally, in Section VIII, we compare PROPS with existing location proof architectures before concluding.

II. Location Proof Specification

In this section, we present the concept of location proof, the different actors of our architecture, the system assumptions as well as the adversary models that we consider.

A. Location Proofs and Interacting Entities

Definition 1: (Location proof — LP) A *location proof* is a digital certificate attesting the position of a user at a specific time.

Definition 2: (Location proof share — LPS) A *location proof share* is a piece of information issued by a user attesting of the position of another user at a certain time.

A LP is generated from a collection of LPSs, and can be manipulated to disclose the location information at different levels of granularity.

A *user* corresponds to an entity using the location proof system. It typically refers to both the device carried out by an individual and the individual himself. A user can take one or several of the following roles: prover, witness or verifier. Figure 1 gives an overview of interactions between users.

A *prover* is a mobile user of the system that periodically collects LPs. In contrast to some previous works, we do not assume that the prover is required to know in advance the LBS he will interact with when collecting LPs. The prover stores all LPs collected on a personal device (*e.g.*, his smartphone) under his control to use them at a later time.

A *witness* is located in the vicinity of a prover and accepts to participate to the generation of a LPS for this prover. The identity of a witness must be kept secret from the other users of the system, with the exception of the Anonymity Lifter (defined later).

A *verifier* checks the validity of the position claimed by a prover through a LP. This role can be played by publicly known entities (*e.g.*, bank, store, social networking site, police authority, LBS provider...) or another user.

The *Certification Authority* (CA) is a trusted third party responsible for issuing the credentials to newly registered users. These credentials can be considered as being the "identity" of these users. This authority is only used to register new users and is not involved in the generation of LP.

The *Anonymity Lifter* (AL) is a trusted third party that has the capacity to lift the anonymity of a particular user when needed (for instance upon request from a judge).

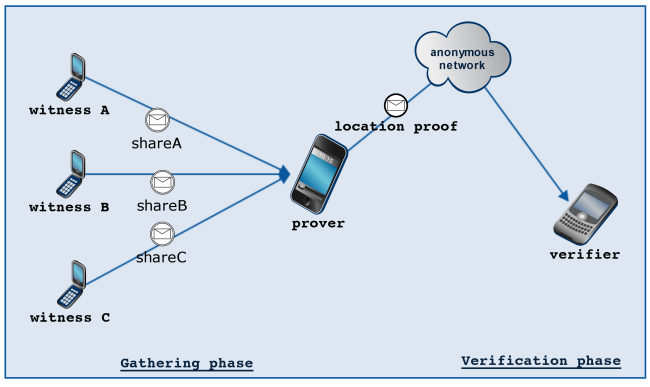

Fig. 1. Global overview and data flow of PROPS.

B. System Assumptions

Within the architecture of PROPS, we make the following system assumptions.

Positioning capability. Users are mobile entities capable of positioning themselves into space. For instance, a user might be able to localize himself by using his GPS or through the help of a dedicated positioning infrastructure. We assume the imprecision of the location computed to be negligible.

Synchronized clocks. Each user possesses a local clock on his device that is synchronized with the clocks of other users. Thus, when two users communicate together, they rely on the same time referential. In practice, users can use the clocks of their GPS or GSM device.

Anonymous communication channel. Provers can broadcast message to neighboring witnesses without disclosing identifying information (*e.g.*, MAC or IP addresses).

C. Adversary Models

When reasoning about the security and privacy of the architecture, we will consider the following types of adversary. Each of these adversaries is assumed to be computationally-bounded (*i.e.*, he cannot break the cryptographic assumptions on which the security of cryptographic primitives rest).

Local eavesdropper. This adversary has the capacity to wire-tap on the communications exchanged between users in his vicinity. His main objective is to break their anonymity.

Malicious prover. A malicious prover aims at obtaining LPs without physically being present at a location. For instance, he can try to modify the time and position information endorsed in a LPS already issued by a witness, or lie to a verifier about the position enclosed in a LP.

Malicious verifier. A malicious verifier may want to extract the identity of the prover out of one of his LP, to breach his location privacy by obtaining more precise information than the one disclosed by the prover or even impersonate him in front of another verifier by pretending to be the owner of the LP.

Malicious witness. This witness may fool an honest prover by endorsing a different spatio-temporal information than the one requested by the prover or collude with a prover by generating more than one LPS for him.

Collusion of users. The main type of collusion attack against location proof systems is known in the literature as *collusion P-P* [25], *terrorist fraud* [13] or *wormhole attack*. This attack involves a prover A at position P_A, a prover B located at position P_B and a witness W in the vicinity of B. A and B collude to obtain a LPS from W attesting that A is at P_B.

III. PROPERTIES OF LOCATION PROOF SYSTEMS

In this section, we describe the fundamental properties that a secure and private location proof system should fulfill. These properties extend the seven design goals governing the construction of a location proof architecture as proposed by Luo and Hengartner [19].

Correctness. A LP presented by a user and generated genuinely in a collaborative manner with honest witnesses should

always be considered accepted by an honest verifier provided that LP is authentic (*completeness property*). A malicious prover should not be able to generate a proof for a location in which he has never been (*spatial and temporal soundness property*). This property has to remain valid despite possible attacks such as the *distance fraud* [2] or the *mafia fraud* [13] (*i.e.*, man-in-the-middle attack).

Proof of ownership and non-transferability. There should be a strong binding between one LP and the identity of its owner (*proof of ownership* property), while still preserving the anonymity of the entities that participated in its creation. The non-transferability property, implicitly resulting from the previous, states that only the genuine owner of a LP should be able to convince a verifier of its correctness. This property is generally enforced by ensuring that the transfer of a LP would also require the transfer of the identity (*i.e.*, the credentials obtained from the CA) to another user. This notion was first introduced in [19]. In the case of PROPS, we extend this definition to encompass the collusion $P-P$ by stating that a user should not be able to collect a LP on behalf on the users with whom he colludes. As pointed out later in the state of the art section, this attack can easily be mounted against the system proposed by Luo and Hengartner [19].

Authenticity and unforgeability. The location information endorsed by a LPS must not be modifiable by an adversary. Within the context of PROPS, a location proof is generated using LPSs collected from witnesses. The trust that a verifier has on a particular LP is proportional to the number of LPSs forming this proof. A malicious witness may collaborate with a prover to grant him more than one LPSs for his position, thus cheating the system by inflating the level of trust provided by a particular proof. Therefore, even if m malicious witnesses collude together, they should not be able to generate a LP that seems to have been created by more than m users.

Prover location privacy. A LPS should protect the location privacy of a prover, by ensuring that the location information does not appear in clear but also that the adversary cannot deduce it indirectly from the knowledge of the LPS alone.

Witness location privacy. A LPS should not reveal any information about the witnesses involved in its generation. Indeed while it is obvious that witnesses are in the communication range of the prover during the proof gathering, the LPS itself should not reveal more fine-grained location information about witnesses and their identities.

Anonymity and unlinkability of prover and witnesses. In order to ensure a high level of privacy, the prover and the witnesses participating to the creation of a location proof must remain anonymous both during the generation of the proof but also when this proof is used. Thus, the proof gathering process as well as the LP itself should not leak any information that can be used to identify or track the participants involved. Moreover, it must be impossible to decide if two LPs are associated to the same prover or if two LPSs have been generated by the same witness during two different LP requests. However, the triple {witness, prover and LPS} should be uniquely identified to prevent possible frauds (to be detailed later).

Location sovereignty. A user should keep the sovereignty on his location data by storing his LPs on his own personal device rather than on a centralized server. By doing so, he can selects the ones that he wants to show to a verifier without exchanging further information with the infrastructure. In addition, each user of the system should have the ability to control the granularity of the location information revealed by a LP without invalidating the authenticity of the underlying LPSs.

IV. BUILDING BLOCKS

In this section, we review the building blocks that we use, and we provide some intuition of how we combine them to build PROPS.

A. Unique Group Signature

Group signature schemes has been introduced by Chaum and van Heyst to provide anonymity to the signer of a message [10]. Such signature scheme relies on a single public verification key for the group but a different private key for each signer (*i.e.,* member of the group). Each member of the group can issue a signature on a message using his private key and the authenticity of the signature is checked using the group verification key. Thus, for any computationally-bounded adversary, it is impossible to identify the actual signer of a message. The main operations of a group signature scheme with optional anonymity lifting are the following ones.

- $\mathsf{Init}(1^\lambda)$. This procedure generates the parameters of the group (λ is a security parameter). It outputs the following keys: gpk the public verification key of the group signature, ok the opening key needed to lift the anonymity of a user and ik the issuing key needed to dynamically add user to the group of signers.
- $\mathsf{Join}(user_i, ik)$. This procedure takes as input the issuing key ik and a user's identity $user_i$. At the end of the procedure, $user_i$ receives $gsk[i]$, his private group signature key and becomes officially a member of the group.
- $\mathsf{GroupSign}(m, gsk[i])$. This procedure takes as input a message m and a signature key $gsk[i]$, and then produces a group signature σ_i of the message m.
- $\mathsf{GroupVerif}(m, \sigma_i, gpk)$. This operation enables to check the authenticity of a group signature. It requires as input the group verification key gpk, a message m and a group signature on this message σ_i. GroupVerif returns either accept or reject depending on the validity of the signature.
- $\mathsf{LiftAnonymity}(m, \sigma_i, ok)$. This procedure retrieves the identity of a particular signer from a signature σ_i. This operation takes as input a message m, a group signature on this message σ_i, the opening key ok and produces as output the identity $user_i$ of the signer.

As explained above, group signature schemes are designed to ensure the anonymity to a signer of the message and multiple-show unlinkability. Indeed, it is impossible to distinguish if two signatures originate from the same member.

In the context of location proof system, we would like to avoid potential abuses such as the unforgeability property and the possibility for one witness to provide more than one LPS per request of a prover. This issue can be solved by relying on *unique group signature scheme* [14]. More precisely in a unique group signature scheme, if a signer produces two signatures on the same message (*i.e.*, two LPSs for the same location information), then there exists an efficient algorithm to detect this:

- $\mathsf{Detect}(m, \sigma, \sigma')$. This procedure is a detection algorithm that can tell if two signatures σ and σ' are signatures on the same message m by the same user. If this situation occurs, this procedure returns true while otherwise it returns false.

Unique group signature scheme can be implemented for instance using CL-signature proposed by Camenisch and Lysyankaya [6]. This efficient group signature scheme relies on bilinear maps and its security is solely based on the *LRSW assumptions* [20]. We refer the reader to [8], [5], [6] for more details about the possibilities offered by this framework and the implementation of this group signature.

B. Commitment

A *commitment scheme* [11], [17] is a cryptographic primitive enabling a prover to hide a value of his choice such that he can decide later to reveal it. In a nutshell, a commitment scheme consists of two algorithms. First, the Commit algorithm takes as input a value m and a random string r, and then outputs a commitment C. Second, the opening algorithm VerifyCommit takes as input C, m and r, and then outputs accept if $C = \mathsf{Commit}(m, r)$ and reject otherwise. A cryptographically secure commitment scheme is *hiding* in the sense that it is computationally hard to infer m given C and *binding* meaning that it is hard to find $m' \neq m$ such that $C = \mathsf{Commit}(m', r)$. In the rest of the paper, all the commitments used refer to *Pedersen commitment* [21]. The Pedersen commitment works in the following manner. Given a group G of prime order q with generators g and h, a commitment to $x \in \mathbb{Z}_q$ is formed by choosing a random $r \in \mathbb{Z}_q$ and setting the commitment $C = g^x \times h^r$. This commitment scheme is information-theoretically hiding, and is binding under the discrete logarithm assumption, which is actually directly implied by the LRSW assumption [6].

C. Zero-knowledge Proof

A *Zero-knowledge proof* [15] is a protocol (interactive or non-interactive) that enables a prover to convince a verifier that he possesses a proof of the veracity of a mathematical statement without leaking any information about the proof itself. Using the notation of Camenisch and Stadler [7], a zero-knowledge protocol can be written as $\mathsf{ZKProof}\{(w) : F(w) = 1\}$, in which F denotes a mathematical statement (a language) and w represents a proof of this statement. Zero-knowledge proof can be used to prove various properties such as the knowledge of a discrete logarithm or of quadratic residues [1], [23], [5]. Over the years, zero-knowledge proofs have been widely used to develop anonymous credentials systems. The most popular of these systems include the *Direct Anonymous Attestation (DAA) protocol* [3], the *identity mixer anonymous credential system* [9], [8] and the CL-signature [5], [6]. In our context, we are interested in a zero-knowledge proof to convince that two Pedersen commitments are commitments to the same value. We call this procedure EqualityCommitment, and it is ran interactively between the prover and a witness. In the following, we give the intuition of how this procedure works. Without loss of generality, consider $C_1 = g^x \times h^{r_1}$ and $C_2 = g^x \times h^{r_2}$ the commitments that the prover needs to convince that they correspond the same value x. The prover picks randomly $\rho_1, \rho_2 \in \mathbb{Z}_q$ and computes $\omega = g^{\rho_1} \times h^{\rho_2}$ before sending it to the witness. The witness chooses a challenge $e \in \mathbb{Z}_q$ and forwards it to the prover. Then, the prover sends back $s_1 = \rho_1 + e \times x, s_2 = \rho_2 + e \times r_1$ and $s_3 = \rho_2 + e \times r_2$. If $g^{s_1} \times h^{s_2} = \omega \times C_1^e$ and $g^{s_1} \times h^{s_3} = \omega \times C_2^e$, then the procedure EqualityCommitment returns accept and the witness accepts the proof, otherwise it returns reject and the witness rejects the proof.

D. Proximity Testing

In PROPS, we assume the availability of a *proximity testing procedure* that is used by a prover to convince a witness that he is close to him. The success of this procedure is a prerequisite to the issuance of a LPS using a valid pseudonym. In practice in PROPS, the pseudonym is a commitment over the long term secret of the prover. C_1. We construct such a proximity testing procedure from distance-bounding (DB) protocols originally introduced by Brands and Chaum [2]. DB protocols existing in the literature such as the one from Bussard and Bagga [4] can easily be adapted for our needs but our architecture is actually agnostic to the DB protocol used. In the following, we describe the execution of the proximity testing procedure that is inspired from the Bussard-Bagga protocol.

The proximity testing procedure consists of three phases. The first phase is the preparation one, in which the prover encrypts his private key S_U with a random symmetric key k and gets the corresponding encrypted message e. Then, the prover commits individually to each bit of e and k, which results in two sequences of bit commitments R_0 and R_1. This phase can be performed offline by the prover to save time.

During the second phase, the prover sends R_0 and R_1 to the witness, which then starts a multi-round fast-bit-exchange with the prover. In each round i, the witness sends a challenge bit $b_i \in \{0, 1\}$, to which the prover replies with the i-th bit of R_b. Since the witness never learns both bit values, he will also never learn the secret S_U. After the multi-round fast-bit-exchange, the witness verifies the corresponding bit commitments of R_0 and R_1 (only for the received bits) by asking the prover to provide the opening information for these commitments.

During the third phase, the values R_0 and R_1 are used by the witness to derive a pseudonym C_2. Finally, the prover convinces the witness that C_2 and C_1 correspond to Pedersen commitments on the same value through a zero-knowledge

proof [5]. For more details about how the pseudonym C_2 is constructed, we refer the reader to [4].

In contrast to the original protocol from [4] and its implementation in STAMP [25], the witness and the verifier do not need to have a public key to authenticate the prover. Instead the authentication is performed using pseudonyms and zero-knowledge proofs to preserve the privacy of the prover. Revealing the values k and e to a colluder also disclose the long-term secret of the prover, thus ensuring that the proximity testing procedure is resistant to terrorist frauds.

Thereafter, we denote the proximity testing procedure by ProximityTesting(δ, C_1), in which δ represents the distance threshold used to verify the proximity of the prover and C_1 is the current pseudonym used by the prover at the initialization of the protocol.

E. Hash Chains

As mentioned previously, the user of a location proof system should have the possibility to reveal different granularities of the positions contained in the LPSs he collected. Zuo and co-authors propose to solve this problem by the use of multiple encryption. More precisely when creating an LPS, each witness generates five different granularities of the location of the prover. The granularities are then encrypted with different keys using a symmetric encryption algorithm such as AES. The encrypted values are then endorsed by the witness and put in the LPS. When a prover reveals his location up to a particular granularity to a verifier, he simply sends the decryption key corresponding to the granularity he wishes to disclose (this method is used for instance in STAMP). We choose to rely on another approach, proposed originally by Lenzini, Maw and Pang [18], to encode location information using hash chains thus minimizing the size of a LPS. In this approach, the digits of a GPS coordinates are hidden into hash chains. Then, revealing the leftmost digits can generate a new granularity of the position. Without loss of generality, we assume that each GPS coordinates pos, such as latitude and longitude, is represented by d digits in decimal (*i.e.*, $x_i = {x_i}^d {x_i}^{d-1} \dots {x_i}^1$).

The hash chain is composed of the following operations.

- Hide$(pos, seed)$. This procedure is run by the prover and takes as input his precise position pos, a secret string $seed$ and outputs K_{pos}, which is the encoding of pos under the secret string $seed$. In practice, K_{pos} is the last value of a hash chain and corresponds to the information that will be certified by the witness and included in the LPS instead of pos.
- Reveal$(pos, p, seed)$. This procedure is called by the prover in order to partially reveal his position at granularity p of his choice. Reveal takes as input the previous position pos, a granularity p such that $p \leq d$ and the same secret $seed$ used to encode pos with the algorithm Hide$(pos, seed)$ and outputs the pair (L_{pos}, aux_{pos}) in which L_{pos} represents the position pos revealed up to the granularity p and aux_{pos} is an auxiliary information needed to prove that L_pos is well formed.
- Check$(K_{pos}, aux_{pos}, L_{pos})$. This procedure is used by the verifier to check that the pair (L_{pos}, aux_{pos}) revealed by the prover corresponds to the location information contained in K_{pos}. This procedure takes as input K_{pos}, aux_{pos} and L_{pos} and returns accept if the location information claimed by the prover is verified, and reject otherwise.

To encode the temporal information, the prover format the current time into five values (*i.e.*, $\{x_1^{'}, \dots, x_5^{'}\}$) that correspond to the time (hh:mm:ss), period of the day (morning, afternoon or night), day, month and year. Then, he also relies on a hash chain to encode the temporal information in the LPS. The uncertainty of the position revealed depends of the number of digits hidden by the function Hide has illustrated in Table I. A GPS coordinate relies on seven digits for the precision, thus the prover can hide a maximum of six digits.

Number of hidden digits	1	2	3	4	5	6
Radius in meters	0,1	1,3	13	136	1 369	13 701

TABLE I
RADIUS OF UNCERTAINTY WITH RESPECT TO THE HIDDEN DIGITS.

V. A PRIVACY-PRESERVING LOCATION PROOF SYSTEM

Thereafter, we describe the four different phases of PROPS, namely Initialization, Join,LocationProofGathering and LocationProofVerification. We use the notation $\mathcal{UGS}$ to refer to unique group signature scheme and $\mathcal{CLS}$ to denote a CL-signature scheme (*cf.* Section IV), while $\bigwedge_{i=1}^{k} f_i(x)$ is used as a shortcut to $f_1(x) \wedge \ldots \wedge f_k(x)$ in which f_i is a boolean function and $\wedge$ represents the logic operator AND. The symmetric encryption is denoted by Encrypt$(\bullet, k_{AB})$, in which the bullet represents the data to be encrypted and k_{AB} the key used.

A. Initialization

During the initialization phase, the CA is responsible for setting up the parameters of the location proof system. More precisely, the CA calls $\mathcal{UGS}$.Init(1^λ), which corresponds to the initialization procedure of a unique group signature. This procedure generates the credentials (sk, ik, ok, gpk) and (sk_{cert}, pk_{cert}) a private/public key compatible with a CL-signature. The public parameters of the system are defined as $pk = (gpk, pk_{cert})$. The CA keeps ik secret and will use it to register new users to the system while ok is sent to the Anonymity Lifter (AL) in order to be able to lift the anonymity of a user if needed.

B. Join

When a $user_i$ joins the system, he runs this procedure together with the CA in order to get his credentials. More precisely, the CA adds the $user_i$ to the group by running $\mathcal{UGS}$.Join$(user_i, ik)$. The CA also generates a random string s_u compatible with zero-knowledge proofs and computes a signature $cert_u = \mathcal{CLS}$.sign(s_u, sk_{cert}) on this random string. As a result, the $user_i$ receives his credentials, which consist of $gsk[i]$ his private group signature key, s_u his secret random string and $cert_{s_u}$ a CL-signature [5] over s_u done using sk_{cert}.

The CL-signature will be used by a prover to demonstrate the possession of a signature without revealing the signature itself.

C. LocationProofGathering

When a prover wants to obtain a proof of his current location, he runs this procedure in collaboration with neighboring users (*i.e.*, witnesses) to obtain LPSs. After collecting at least k LPSs from different witnesses, the prover can generate a LP (*cf.* Section V-C).

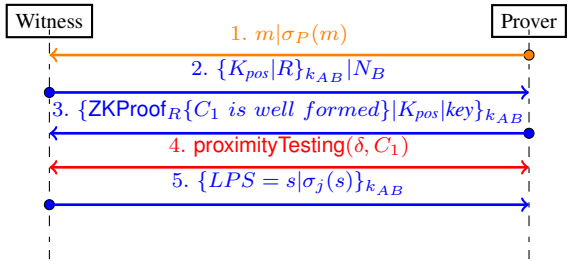

Fig. 2. The location proof gathering phase.

The location proof gathering protocol between a prover and a witness is summarized in Figures 2 and 3. This protocol consists of the following phases.

Session initialization. The gathering process starts with an initialization phase in which the prover P generates random values *key* and c_1 and uses them to compute the following values:

- $K_{pos} = \mathsf{hide}(pos, key)$ and $K_{time} = \mathsf{hide}(time, key)$, which corresponds to the encryption of pos and $time$ under *key*. P also computes N_A, a random share for a Diffie-Hellman key agreement protocol.
- $C_1 = \mathsf{Commit}(S_U, rand_1)$, which corresponds to the commitment of his secret string s_u under $rand_1$.

Then, P anonymously broadcasts (*cf.* Section II-B) a LPS request to neighboring users and wait for their responses:

$$(1) \quad P \rightarrow * : \{LPSReq = m|\sigma_{G,P}(m)\ ,\ m = pos|time|C_1|K_{pos}|K_{time}|N_A\}$$

The request $LPSReq$ contains the following information: the concatenation of *pos*, the current position of the user (which also contains the current time), C_1 a commitment acting as a pseudonym, K_{pos} the encoding of the prover's position, K_{time} the encoding of the current time, N_A the share of the Diffie-Hellman key agreement and finally $\sigma_{G,P}(m)$ a group signature to authenticate the request.

Processing of the request. Upon reception of $LPSReq$, a witness W checks the validity of the group signature $\sigma_{G,P}(m)$ and that pos and $time$ correspond to the current geolocated context. This verification is done by verifying that pos falls in the circle of radius δ centered at the position of the witness and that $time$ corresponds to the actual time (*cf.* Section II-B). If these verifications succeed, the witness W computes the following values:

- N_B a random share of a Diffie-Hellman key agreement, which is then combined with a in order to generate a session key k_{AB}. This session key k_{AB} will then be used to encrypt all subsequent communications between P and W by relying on a symmetric cryptosystem, thus ensuring the confidentiality and integrity of communications (*cf.* Section III).
- R a fresh challenge for the zero-knowledge proof.

Afterwards, the witness W notifies the prover that he accepts to continue the generation process:

$$(2) \quad W \rightarrow P : \{LPSReply = \mathsf{Encrypt}((K_{pos}|R), k_{AB})|N_B\}$$

The request $LPSReply$ contains the value of N_B in clear, the value K_{pos} and a challenge R encrypted as an acknowledgement to the previous phase. At the reception of $LPSReply$, P can reconstruct the session key k_{AB} using N_B and retrieves the challenge R of the zero-knowledge proof. P replies to W with the following zero-knowledge proof:

$$(3)\ P \rightarrow W : \{ZK = \mathsf{Encrypt}((\mathsf{ZKProof}_R(C_1\ well formed)|K_{pos}|key), k_{AB})\}$$

The message ZK contains a zero-knowledge proof that the pseudonym C_1 is a valid commitment over a value S_U certified by the CA. It also contains the value key to convince the witness that K_{pos} and K_{time} are valid encoding of pos and $time$.

$$\mathsf{ZKProof}\{(cert_{S_U}, S_U, rand_1) : C_1\} \leftarrow \mathsf{ZKProof}\{(cert_{S_U}, S_U) : \mathsf{VerifyCommit}(C_1, S_U, rand_1) = 1 \ \wedge \ \mathcal{CLS}.\mathsf{VerifySign}(S_U, cert_{S_U}, pk_{cert}) = 1\}.$$

In order to ensure the freshness of the proof, the zero-knowledge proof will also depend on R, the nonce generated by the witness at the previous step. This is made possible by the use of zero-knowledge proofs based on CL-signatures.

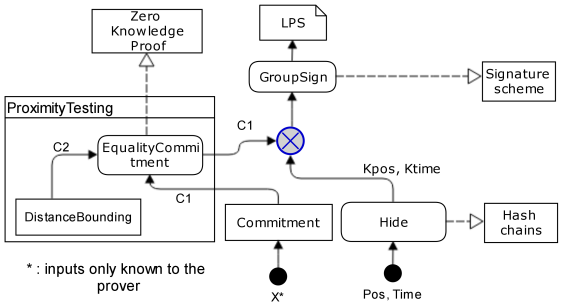

Fig. 3. Overview of the gathering procedure.

Proximity testing. Afterwards, P starts the proximity testing protocol with W.

$$(4) \quad P \leftrightarrows W : \{\mathsf{ProximityTesting}(\delta, C_1)\}$$

This protocol consists in a sequence of fast bit exchanges between the prover and the witness. More precisely, the witness selects a bit $b \in \{0, 1\}$ and sends it to the prover. Then, the prover replies with a response r. W records the time needed by the prover to produce the response to each of the challenges. These timings are used to estimate the proximity of the prover and this process is repeated several

times in order to increase the robustness of the proximity testing protocol. If the timings are close enough to δ, then the $\mathsf{ProximityTesting}(\delta, C_1)$ procedure outputs accept, while otherwise it outputs reject.

At this point, W is convinced that P is in his proximity. However, he also needs to be convinced that P has not cheated during the running of ProximityTesting. To achieve this, W constructs a commitment C_2 using the responses received during the proximity testing. More precisely provided that the proximity testing was ran correctly, C_2 is a commitment on the identity S_U of the prover under a new random string $rand_2$ only known by the prover $C_2 = \mathsf{Commit}(s_u, rand_2)$. Afterwards, P sends to W a zero-knowledge proof that the value committed value in C_1 and C_2 are equal.

LPS generation. Finally, W creates a LPS $s = \{C_1|K_{pos}|K_{time}\}$ and a signature $\sigma_W(s) = \mathcal{UGS}.\mathsf{GroupSign}(S, gsk_W)$. P receives from W:

$$(5) \quad W \rightarrow P : \{\mathsf{Encrypt}(LPS, k_{AB})\}, LPS = s|\sigma_W(s)$$

Afterwards, the prover locally stores the LPSs collected from surrounding witness, the associated variables (the encoding key*key*) and the current spatio-temporal context ($time$ and pos).

D. Location Proof Verification

The proof verification phase described in Figure 4 enables a prover to convince a verifier that he was at a specific location at a particular moment in time. The verification process requires a proof of ownership by the current prover and also a check from the verifier that the location proof contains LPSs from different witnesses. If this last verification fails, then the verifier will forward the evidences of the fraudulent LPS to the AL who has the capacity to reveal the identity of the cheater using the opening key ok.

Without loss of generality, consider that the prover has collected k LPSs before sending a LP verification request to a verifier. A LP is generated as follows:

$$P \rightarrow V : LP = s|\sigma_{W_1}(s)| \ldots |\sigma_{W_k}(s)|aux_{pos}|L_{pos}|aux_{time}|L_{time}$$

in which L_{pos} (*resp.*L_{time}) represents the granularity of the position (respectively the time) to be revealed. The values aux_{pos} and aux_{time} are used to check that these granularities are conformed with the values K_{pos} and K_{time} of LP. These values are computed using the function Reveal (*cf.* Section IV-E).

Once he has received the LP , the verifier V runs sequentially the following verifications steps.

1) First, V checks for each LPS $\sigma_i(s)$ in $certificate$ whether $\mathcal{UGS}.\mathsf{GroupVerif}(s, \sigma_i(s), gpk)$ returns accept. More precisely, the function $\mathcal{UGS}.\mathsf{GroupVerif}(s, \sigma_i(s), gpk)$ verifies the validity of the group signature of each of the k LPSs $\sigma_i(s)$:

$$\bigwedge_{i=1}^{k} (\mathcal{UGS}.\mathsf{GroupVerif}(s, \sigma_i(s), gpk)) = \mathsf{true}.$$

2) Then, the verifier validates the uniqueness of the LPS in LP by verifying that *all these LPSs* have been generated by different witnesses:

$$\bigwedge_{i=1}^{k} \bigwedge_{j=i+1}^{k} (\mathcal{UGS}.\mathsf{Detect}(s, \sigma_i(s), \sigma_j(s))) = \mathsf{false}.$$

3) Next, V anonymously authenticates the prover P as the legal owner of LP by running a zero-knowledge proof protocol with him. At the end of this step, V is convinced that P knows the secret S_U used to generate the pseudonym C_1 and that this secret has been certified by the CA.
4) Finally, V uses the K_{pos} and K_{time} contained within LP to evaluate the validity of the spatio-temporal informatin (L_{pos}, L_{time}) claimed by the prover:

$$\mathsf{Check}(K_{pos}, aux_{pos}, L_{pos}) \wedge \mathsf{Check}(K_{time}, aux_{time}, L_{time}).$$

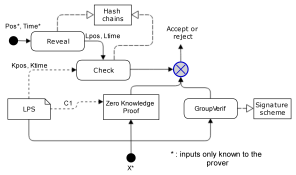

Fig. 4. Overview of the verification procedure.

VI. Security and Privacy Analysis

In this section, we analyze how PROPS fulfils the security and privacy properties described in Section III.

Correctness. The completeness property is trivial. Once an LPS is received by the prover, he can verify that the spatio-temporal information contained within it is valid. The *spatial and temporal soundness* are ensured because revealing a geolocated context that does not match the one contained in the LP will be detected during the Step 4 of the verification process. Thus, a malicious prover cannot alter the integrity of a LPS and fool the verifier by claiming a different location than the one contained in the LPS. In the following, we give more details about how PROPS ensures the spatial soundness property (*cf.* Section III) by proving its resistance to the distance fraud and mafia fraud.

Resistance to distance fraud. In a distance fraud, a malicious prover tries to convince an honest witness that he is closer than in reality. By assumption in PROPS, the distance fraud is prevented by the use of the $\mathsf{ProximityTesting}(\delta, C_1)$ protocol.

Resistance to mafia fraud. Let P be an honest prover located at position L_p and W an honest witness located at position L_w such that $\mathsf{dist}(L_p, L_w) > \delta$. Consider $\bar{W}$ and $\bar{P}$, which are two different colluding users or the same malicious user playing two different roles. In the mafia fraud, the objective of an adversary is to replay a session that involved a honest prover P to fool W and make him believe that P is closer than he really is. Thus, two sessions need to be run, the first one involving $\bar{W}$ and P with the objective to get commitments

from P followed by a second one performed between $\bar{P}$ and W in which the commitments of the first protocol are replayed in order to obtain a location share on behalf of P. In the second session $\bar{P}$ will need to compute a fresh zero-knowledge proof using r from W to prove knowledge of the identity s_u of P, which is impossible without the knowledge of s_u.

Proof of ownership and non-transferability. During the verification phase, the verifier checks that the current prover is effectively the legitimate owner of the LP by running a zero-knowledge protocol over the pseudonym C_1 included in the proof. Within PROPS, the non-transferability property is equivalent to the resistance to the collusion $P - P$. The resilience to the collusion $P - P$ follows directly from the resilience to the *terrorist fraud* of the DB protocol used.

Unforgeability. The unforgeability is ensured partially by the uniqueness property provided by unique group signature, which prevents the adversary controlling a collusion of m malicious users to gather enough LPSs as long as the size of the collusion is less than the number of shares needed (*i.e.*, $k > m$).

Resistance to distance hijacking. In a distance hijacking attack, a malicious user M tries to hijack the gathering session of an honest prover P. More precisely, M waits until P has successfully proved that he is in the vicinity of an honest witness W and then hijacks P's session to collect its LPS. However in PROPS, a witness verifies that the entity who ran $\mathsf{ProximityTesting}(\delta, C_1)$, is the same as the one that computes the commitment C_1. Therefore, such an attack will be detected by W and the gathering process will be aborted before the malicious prover receives the LPS, thus avoiding the possibility of hijacking.

Anonymity and unlinkability of prover and witnesses. Due to the use of commitments and zero-knowledge proofs in PROPS, users can remain anonymous in the system as long as they behave honestly. Furthermore, the prover creates periodically nonces to generate new pseudonyms. Therefore, the pseudonyms are unlinkable provided the nonces are chosen at random and independently. Moreover due to the hiding property of the commitments, the pseudonyms do not disclose any information that can be used to trace back to the identity of the prover. In addition, the anonymity and unlinkability of witnesses are ensured by the use of group signature. Finally, the use of a zero-knowledge proof enables the prover to anonymously authenticate himself to the verifier as the owner of a LP.

Witness location privacy. When establishing a LP, a witness never discloses his exact position but rather checks that the position claimed by the prover is in the vicinity using the proximity testing protocol. Therefore, a local eavesdropper can only infer that the witness is in the proximity of the prover but does not learn his exact position.

Prover location privacy and authenticity. The location information is first encoded into the hash chains before being endorsed by the witness. This information cannot be modified later by the prover and it does not appear in clear in the LP.

Location sovereignty. Within PROPS, the LPs gathered by a user are saved locally on his device in contrast with other schemes in which the proofs are stored and controlled by remote servers. Finally, due to the use of hash chains the prover can decide the granularity of the information he wants to disclose (*cf.* Section IV-E).

VII. PROOF-OF-CONCEPT IMPLEMENTATION

In this section, we briefly report on the current proof-of-concept implementation of PROPS. Our main objective is to demonstrate that the architecture of PROPS can be implemented with currently available technology. Our implementation relies on Idemix[1] 2.3.4, a Java library containing advanced cryptographic primitives such as CL-signatures, commitment schemes and zero-knowledge proofs (*cf.* Section IV). Moreover, we have implemented unique group signatures relying on the concept of domain pseudonym offered by Idemix. In a nutshell, a domain pseudonym can be used to link all the group signatures performed by a user within the same domain. Within the context of PROPS, we set the domain to the value r included in the location proof, thus allowing a verifier to check if several signatures have been issued by the same user on a specific domain (*i.e.*, LP).

We implemented a Java prototype client application on Android. Our experiments are carried out on two devices: (1) a Samsung Galaxy Note 2 equipped with a Quad Core 1.6 GHz processor, 2GB of RAM, a GPS and running Android OS 4.1.2, acting as a prover, and (2) a Google Nexus 7 (2012 version) equipped with a Nvidia Tegra processor and 1 GB of RAM acting as a witness. The measurements that we report for each phase have been computed by averaging over 10 independent trials. In our testbed, the witness listens to the network until he receives a location proof request from a nearby prover. As a result, the witness decides whether he accepts to serve the request or not. If the request is accepted, the witness sends an acknowledgement back to the prover. This process is denoted as the *initialization phase* in Figure 5. After the initialization phase, the witness check that the prover possesses a valid credential from the CA on the value that is contained in the commitment (*authentication phase*). Afterwards, the prover and the witness executes the *proof creation phase*. Finally, we also test the *proof sending phase* in which the witness sends the location share to the prover. During our experiments, we have measured the computational time (also an indicator of power consumption) and storage that are needed to run our current implementation PROPS on real smartphone devices.

From a memory point of view, each LPS has a size of 3444 bytes, which is higher compared to those of previous works. However, PROPS provides stronger privacy and security guarantees. The running time of the whole gathering phase conducted by a witness is on average of 2.98 ± 0.02 seconds.

For the verification phase, we have designed a realistic scenario in which a prover wants to show a LP to a remote LBS server. The server we used run on an Intel i5-2435M dual-core

[1] http://www.zurich.ibm.com/security/idemix/

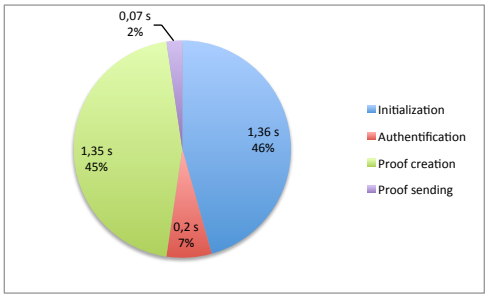

Fig. 5. Cost of the different phases of the proof generation for the witness.

processor at 2.4 Ghz with 4 GB of 1333 Mhz DDR3 SDRAM running OSX 10.8.3. The average time needed to verify one LPS is around 0.66 ± 0.03 seconds for the verifier.

VIII. Related work

Location proof architectures can broadly be categorized into two classes depending on whether a user collects a LP (1) through a bipartite interaction with a trusted node belonging to the infrastructure or (2) through a collaborative protocol performed with several other (untrusted) users.

Bipartite gathering approach. In the *bipartite gathering approach* a user cannot request a proof for his location until he is close to a reference trusted node, such as an Access Point (AP). When this situation occurs, a protocol is run between the user and the reference node in order to generate a location proof. For instance, Waters and Felten [26] introduced a system relying on distance-bounding to compute the latency between a prover and his nearest AP. Then, this latency is inserted in a proof that can be checked by a given verifier. One of the limit of this system is that an adversary can collect LPs on behalf of malicious users, thus leading to terrorist fraud. Saroiu and Wolman [22] proposed a system in which periodically broadcasted beacons are used to validate the proximity of a prover to an AP. From a privacy point of view, one of the main issue of this protocol is that a prover needs to publicly reveal his identity during the gathering process. More recently, Luo and Hengartner [19] have designed a scheme in which intermediate proofs generated by APs need to be validated by a trusted third party. This protocol can detect cheating attempts in which a prover has acquired at the same time two LPs from access points located at different places. In addition as for PROPS, a user can decide on the granularity of the revealed location information.

Cooperative gathering approach. In the *cooperative gathering approach*, users directly collaborate together in order to generate LPs. For instance, Zhu and Cao designed APPLAUS [27], in which neighboring mobile devices communicate via Bluetooth to cooperatively generate LPs, before forwarding updates of their current positions to a location proof server responsible for storing these proofs, while periodically changing their pseudonyms. Graham and Gray [16] proposed a protocol based on the use of distance-bounding protocols in which a central server monitors the whereabouts of all the users which is in direct opposition with the preservation of the location privacy of users of such system. Talasila, Curtmola and Borcea introduced LINK [24], a system in which each user is assigned a trust score reflecting his behavior in the system. LINK can thwart attacks from colluding users in which the same witnesses always certified the position of a prover but LINK have not integrated any privacy issues in its design. Davis, Chen and Franklin introduced an alibi system [12]. An alibi is a certificate providing evidences of an individual's past location, which can be critical in proving his innocence with respect to an event occurring in a location in which he was not present. In this system, the identity of the alibi owner is concealed at the time of the creation of the alibi. In this architecture, both the prover and the witnesses have to reveal their identities when the alibi needs to be shown to a judge.

Recently, Wang and co-authors propose STAMP [25], which is a location proof system in which co-located mobile devices mutually generate LPS for each other using bluetooth or WiFi in ad-hoc mode. In the same spirit as PROPS, a prover convince a verifier of his location by showing several LPSs. STAMP ensures the authenticity of LPS (*i.e*, a prover cannot modify the data included in a LPS once generated), the non-transferability, and the anonymity of prover and witnesses generating the proof. Users have also the possibility to choose the granularity to reveal to a verifier. However, in contrast to PROPS, the LPSs are encrypted under the CA public key, thus the prover cannot check himself the validity of location information endorsed by the witness. During the verification stage, the verifier needs to contact the CA to validate a LP. The collusion detection algorithm is an entropy-based trust evaluation approach like the one used in APPLAUS. STAMP incorporates the Bussard-Bagga distance-bounding protocol as a countermeasure to *wormhole attacks*. The authors also provide a prototype implementation on the Android platform with an averaging running time of 8 seconds.

Table VIII summarizes the security and privacy properties (*cf.* Section III) enforced by the aforementioned systems and PROPS. In terms of notation, a checked cell means that the protocol ensures this property while a blank cell indicates the opposite. Note that while PROPS has been designed to meet all these security and privacy properties, it cannot defend against a collusion of malicious users whose size is unbounded. In contrast, APPLAUS [27] and LINK [24] overcome this issue by storing all LPs on a central server. By doing so, they can maintain a log of the witnesses used by each prover and use it to detect a possible collusion (*i.e*, a particular prover is always in contact with the same witnesses while there are other users that are located at the same place). While we have striven to include this security property when designing PROPS, it does not appear to be trivial to implement it without relying on a trusted third party to store the proofs, which would be in direct opposition with the spirit of PROPS, including in particular the location sovereignty property.

IX. Conclusion

In this paper, we introduced PROPS, a novel privacy-preserving location proof system based on a collaborative architecture. The main strengths of this location proof system are the following: (1) the LP collected by a prover are under his

	Properties \ Protocol	[19]	[22]	[26]	[27] APPLAUS	[16] SLVPGP	[24] LINK	[12]	[25] STAMP	PROPS
SECURITY	Correctness	✓	✓	✓	✓	✓	✓	✓		✓
	Ownership proof	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Unforgeability	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Resistance to distance fraud	✓	✓	✓		✓			✓	✓
	Resistance to mafia fraud		✓			✓			✓	✓
	Resistance to distance hijacking								✓	✓
	No single point of failure	✓	✓	✓				✓	✓	✓
	Resistance to collusion $P-P$				✓				✓	✓
	Proof share uniqueness								✓	✓
PRIVACY	Prover anonymity & unlinkability (gathering phase)	✓	✓	✓	✓			✓	✓	✓
	Prover anonymity & unlinkability (verification phase)									✓
	Witness anonymity & unlinkability (gathering phase)	✓			✓			✓	✓	✓
	Witness anonymity & unlinkability (verification phase)	✓			✓	✓	✓		✓	✓
	Witness location privacy	✓						✓	✓	✓
	Confidentiality	✓				✓			✓	✓
	Location sovereignty	✓							✓	✓

TABLE II
COMPARISON OF DIFFERENT LBS

control and does not reveal any information about his identity, (2) the prover has the ability to remain anonymous even when presenting a proof to a verifier, (3) the privacy of users is preserved with respect to a man-in-the-middle adversary, (4) a verifier can detect abuses of a malicious user that tries to produce fake LPs, and finally (5) the prover can selective disclose the spatial and temporal information to the granularity of his choice. We also demonstrate the security of PROPS to standard attacks such as *collusion $P-P$/terrorist fraud*, *distance fraud*, *mafia fraud* and *replay attack*.

In the future, we would like to extend PROPS to deal with the collusion $W-P$ in which a witness systematically reports false LPS for a colluder even though one or both of them are not at the location claimed in the LPS. Indeed, unless trusted infrastructures are deployed at each possible location, it seems quite difficult to detect that a particular LPS is a result of such a collusion [24], [25], [27]. A line of research that we would like to pursue in the future is the use of anonymous peer-to-peer reputation system as a countermeasure to frauds in this mobile environment. Finally, another research avenue is the design of a secure multiparty computation version of the protocol involving a joint interaction with the prover and multiple witnesses rather than relying on pairwise interactions between the prover and each witness.

ACKNOWLEDGEMENTS

This work is partially supported by the ANR French project AMORES (ANR-11-INSE-010) and the INRIA Project Lab CAPPRIS (Collaborative Action on the Protection of Privacy Rights in the Information Society).

REFERENCES

[1] E. Bangerter, S. Krenn, A.-R. Sadeghi, T. Schneider, and J.-K. Tsay. On the design and implementation of efficient zero-knowledge proofs of knowledge. *Software Performance Enhancements for Encryption and Decryption and Cryptographic Compilers–SPEED-CC*, 2009.

[2] S. Brands and D. Chaum. Distance-bounding protocols. In *Proceedings of EUROCRYPT '93*. 1993.

[3] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*, 2004.

[4] L. Bussard and W. Bagga. Distance-bounding proof of knowledge to avoid real-time attacks. In *Security and Privacy in the Age of Ubiquitous Computing*. 2005.

[5] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communication Networks*, Lecture Notes in Computer Science, 2003.

[6] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology–CRYPTO 2004*, 2004.

[7] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology*. 1997.

[8] J. Camenisch and E. Van Herreweghen. Identity mixer.

[9] J. Camenisch and E. Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM conference on Computer and communications security*, 2002.

[10] D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT*. LNCS 547, 1991.

[11] I. Damgard and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Advances in Cryptology - ASIACRYPT 2002*. Springer Berlin Heidelberg, 2002.

[12] B. Davis, H. Chen, and M. Franklin. Privacy-preserving alibi systems. In *7th ACM Symposium on Information, Computer and Communications Security, ASIACCS*, Seoul, South Korea, 2012.

[13] Y. Desmedt. Major security problems with the 'unforgeable' (feige)-atshamir proofs of identity and how to overcome them. In *Proceedings of SecuriCom '88*, 1988.

[14] M. Franklin and H. Zhang. Unique group signatures. In S. Foresti, M. Yung, and F. Martinelli, editors, *ESORICS*, pages 643–660. LNCS 7459, 2012.

[15] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *ACM STOC*, 1985.

[16] M. Graham and D. Gray. Protecting privacy and securing the gathering of location proofs - the secure location verification proof gathering protocol. *LNICST*, 17, 2009.

[17] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. Cryptology ePrint Archive, Report 2007 155, 2007.

[18] G. Lenzini, S. Mauw, and J. Pang. Selective location blinding using hash chains. In *Security Protocols Workshop*, LNCS 7114, 2011.

[19] W. Luo and U. Hengartner. Proving your location without giving up your privacy. In *ACM HotMobile*, 2010.

[20] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography*, 2000.

[21] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91*. Springer Berlin Heidelberg, 1992.

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

[23] C.-P. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology*, 1990.

[24] M. Talasila, R. Curtmola, and C. Borcea. Link: Location verification through immediate neighbors knowledge. *Springer, LNICST 73*, 2012.

[25] X. O. Wang, J. Zhu, A. Pande, A. Raghuramu, P. Mohapatra, T. Abdelzaher, and R. Ganti. Stamp: Ad hoc spatial-temporal provenance assurance for mobile users.

[26] B. Waters and E. Felten. Secure, private proofs of location. Technical report, Department of Computer Science, Princeton University, Tech. Rep. TR-667-03, 2003.

[27] Z. Zhu and G. Cao. Applaus: A privacy-preserving location proof updating system for location-based services. In *INFOCOM*, pages 1889–1897, 2011.