STAMP: Ad Hoc Spatial-Temporal Provenance Assurance for Mobile Users

Xinlei (Oscar) Wang*, Jindan Zhu*, Amit Pande*, Arun Raghuramu*, Prasant Mohapatra*,
Tarek Abdelzaher†, Raghu Ganti‡

* Computer Science Department, University of California, Davis
{xlwang, jdzhu, pande, araghuramu, pmohapatra}@ucdavis.edu
† Computer Science Department, University of Illinois at Urbana Champaign
zaher@cs.uiuc.edu
‡ IBM T J Watson Research Center
rganti@us.ibm.com

***Abstract*—Location-based services are quickly becoming immensely popular. In addition to services based on users' current location, many potential services rely on users' location history, or their *spatial-temporal provenance*. Malicious users may lie about their spatial-temporal provenance without a carefully designed security system for users to prove their past locations. In this paper, we present the Spatial-Temporal provenance Assurance with Mutual Proofs (STAMP) scheme. In contrast to most existing location proof systems which rely on infrastructure like wireless APs, STAMP is based on co-located mobile devices mutually generating location proofs for each other. This makes STAMP desirable for a wider range of applications. STAMP ensures the integrity and non-transferability of the location proofs and protects users' privacy. We also examine different collusion scenarios and propose a light-weight entropy-based trust evaluation approach to detect fake proofs resulting from collusion attacks. Our prototype implementation on the Android platform shows that STAMP is low-cost in terms of computational and storage resources. Extensive simulation experiments show that our entropy-based trust model is able to achieve high (> 0.9) collusion detection accuracy.**

I. Introduction

Most of the current location-based services for mobile devices are based on users' current location. In addition to users' current locations, there is an increased trend and incentive to prove/validate mobile users' past geographical locations. This opens a wide variety of new location-proof based mobile applications. Saroiu et. al described several such potential applications in [1]. Let us consider three examples: (1) A store wants to offer discounts to frequent customers. Customers must be able to show evidence of their repeated visits in the past to the store. (2) A company which promotes green commuting and wellness may reward their employees who walk or bike to work. The company may encourage daily walking goals of some fixed number of miles. Employees need to prove their past commuting paths to the company along with time history. (3) On the battlefield, when a scout group is sent out to execute a mission, the commanding center may want every soldier to keep a copy of their location traces for investigation purpose after the mission.

The above applications require users to be able to obtain proofs from the locations they visit. Users may then choose to present one or more of their proofs to a third-party verifier to claim their presence at a location at a particular time. In this paper, we define the past locations of a mobile user at a sequence of time points as the *spatial-temporal provenance* (STP) of the user, and a digital proof of user's presence at a location at a particular time as an *STP proof*. Many works [1]–[3] in literation have referred to such a proof as *location proof*. In this paper, we consider the two terms interchangeable. We prefer "STP proof" because it indicates that such a proof is intended for past location visits with both spatial and temporal information. Other terminologies have been also used for similar concepts, such as location claim [4], provenance proof [5], and location alibi [6].

Today's location-based services solely rely on users' devices to determine their location, e.g., using GPS. However, it allows malicious users to fake their STP information. Therefore, we need to involve third parties in the creation of STP proofs in order to achieve the integrity of the STP proofs. This, however, opens a number of security and privacy issues. First, involving multiple parties in the generation of STP proofs may jeopardize users' location privacy. Location information is highly sensitive personal data. Knowing where a person was at a particular time, one can infer his/her personal activities, political views, health status, and launch unsolicited advertising, physical attacks or harassment [7]. Therefore, mechanisms to preserve users' privacy and anonymity are mandatory in an STP proof system. Second, authenticity of STP proofs should be one of the main design goals in order to achieve integrity and non-transferability of STP proofs. Moreover, it is possible that multiple parties collude and create fake STP proofs. Therefore, careful thought must be given to the countermeasures against collusion attacks.

In this paper, we propose an STP proof scheme named **S**patial-**T**emporal provenance **A**ssurance with **M**utual **P**roofs (STAMP). STAMP aims at ensuring the integrity and non-transferability of the STP proofs, with the capability of protecting users' privacy. Most of the existing STP proof schemes rely on wireless infrastructure (e.g., WiFi APs) to create proofs for mobile users. However, it may not be feasible for all types of applications, e.g. STP proofs for the green commuting and battlefield examples certainly cannot be obtained from wireless APs. To target a wider range of applications, STAMP is based on a distributed architecture. Co-located mobile devices mutually generate and endorse STP proofs for each other. In addition, in contrast to most of the existing schemes which require multiple trusted or semi-trusted third parties, STAMP

978-1-4799-1270-4/13/$31.00

requires only a single semi-trusted third party which can be embedded in a Certificate Authority (CA). We design our system with an objective of protecting users' anonymity and location privacy. No parties other than verifiers could see both a user's identity and STP information (verifiers need both identity and STP information in order to perform verification and provide services). Users are given the flexibility to choose the location granularity level that is revealed to the verifier. We examine two types of collusion attacks: (1) A user A who is at an intended location masquerades as another colluding user B and obtains STP proofs for B. This attack has never been addressed in any existing STP proof schemes. (2) Colluding users mutually generate fake STP proofs for each other. There have been efforts to address this type of collusion. However, existing solutions suffer from high computational cost and low scalability.

The **contributions** of this paper can be summarized as:

1) A distributed STP proof generation and verification protocol (STAMP) is introduced to achieve integrity and non-transferability of STP proofs. No additional trusted third parties are required except for a semi-trusted CA.
2) STAMP is designed to maximize users' anonymity and location privacy. Users are given the control over the location granularity of their STP proofs.
3) STAMP is collusion-resistant. The Bussard-Bagga distance bounding protocol [8] is integrated into STAMP to prevent a user from collecting proofs on behalf of another user. An entropy-based trust model is proposed to detect users mutually generating fake proofs for each other.
4) A prototype application is implemented on the Android platform. Experiments show that STAMP requires preferably low computational time and storage.
5) Simulation experiments validate that our entropy-based trust model is able to achieve over 0.9 collusion detection accuracy with fairly high percentage ($\sim$5%) of colluding attackers.

The rest of the paper is organized as follows: Section II discusses related work. Section III describes our system model. In Section IV, we discuss the security requirements in detail and describe the threat model of this work. In Section V, we present the details of the STAMP protocol. A security analysis of STAMP against different types of attacks is provided in Section VI. In Section VII, we describe our implementation and simulation and present our experimental results on the performance evaluation. We give a discussion and outline our future work in Section VIII. Finally, Section IX concludes the paper.

II. Related Work

The notion of unforgeable location proofs was discussed by Waters et al. [9]. They proposed a secure scheme which a device can use to get a location proof from a location manager. However, it requires users to know the verifiers as a prior. Saroiu et al. [1] proposed a secure location proof mechanism, where users and wireless APs exchange their signed public keys to create timestamped location proofs. These schemes are susceptible to collusion attacks where users and wireless APs may collude to create fake proofs.

VeriPlace [2] is a location proof architecture which is designed with privacy protection and collusion resilience. However, it requires three different trusted entities to provide security and privacy protection: a TTPL (Trusted Third Party for managing Location in formation), a TTPU (Trusted Third Party for managing User information) and a CDA (Cheating Detection Authority). Each trusted entity knows either a user's identity or his/her location, but not both. VeriPlace's collusion detection works only if users request their location proofs very frequently so that the long distance between two location proofs that are chronologically close can be considered as anomalies. This is not a realistic assumption because users should have the control over the frequency of their requests.

Hasan et al. [5] proposed a scheme which relies on both location proofs from wireless APs and witness endorsements from Bluetooth-enabled mobile peers, so that no users can forge proofs without colluding with both wireless APs and other mobile peers at the same time. It eliminates the necessity of multiple trusted parties.

All the above systems are centralized, that is, they all require central infrastructures (wireless APs) to act as the location authorities and generate location proofs.

In Davis et al.'s alibi system [6], their private corroborator scheme relies on mobile users within proximity to create alibi's (i.e., location proofs) for each other. The security and privacy of the system is achieved based on a cryptographic commitment scheme. However, they do not deal with any collusion attacks. Also, multi-level location granularity is not considered in their work.

The system that is most closely related to our work is Zhu et al.'s APPLAUS [3]. It is a location proof system that is also based on co-located mobile devices mutually generating location proofs. In order to protect privacy, the knowledge of private information is separately distributed to three parties: a location proof server, a CA, and the verifier. Periodically changed pseudonyms are used by the mobile devices to protect their real identities from each other, and from the location proof server. We believe the location proof server is redundant for accomplishing the goals. Periodically changed pseudonyms incurs high operational overhead because of the requirement for highly cautious management and scheduling. Dummy proofs have to be regularly generated in order to achieve the privacy properties, which also incurs high communication and storage overhead. The collusion detection in APPLAUS is based on betweenness ranking and correlation clustering. These approaches require the location proof server to have access to at least the majority of the concurrent (within a short delay) location proofs at the same location (within a small region). This needs users to submit their location proofs right after generating them, which is infeasible when there is no Internet connection on-the-spot. Moreover, these approaches cost large computing power to run the detection ($>$200 seconds for 5000 pseudonyms) and their successful detection ratio is high ($>$0.9) only when the percentage of the colluding attackers is rather low ($<$0.1%).

III. System Model

As we explained, wireless infrastructure may not be available everywhere and hence a system based on wireless APs creating STP proofs would not be feasible for all scenarios. In addition, the deployment cost would be high if we require a large number of wireless APs to have the capability of generating STP proofs. Therefore, we think a distributed STP proof architecture, i.e., mobile users obtaining STP proofs from nearby mobile peers, would be more feasible and appropriate for a wider range of applications.

Figure 1 illustrates the architecture of our system. There are four types of entities based on their roles:

- **Prover**: A *prover* is a mobile device which tries to obtain STP proofs at a certain location.

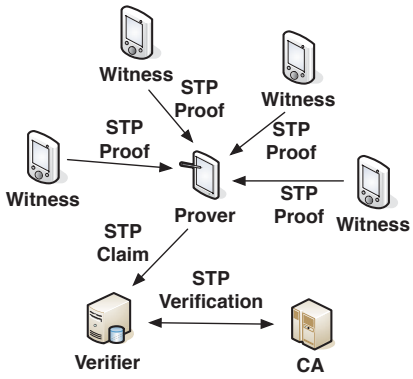


Fig. 1: An illustration of system architecture

- **Witness**: A *witness* is a mobile device or stationary wireless AP which is in proximity with the prover and is willing to create an STP proof for the prover upon receiving his/her request.
- **Verifier**: A *verifier* is the party that the prover wants to show one or more STP proofs to and claim his/her presence at a location at a particular time.
- **Certificate Authority (CA)**: The *CA* is a semi-trusted server (untrusted for privacy protection, see Section IV-C for details) which issues, manages cryptographic credentials for the other parties. CA is also responsible for proof verification and trust evaluation.

A prover and a witness communicates with each other via Bluetooth or WiFi in ad hoc mode, STP claims are sent to verifiers from provers via a LAN or Internet, and verifiers are assumed to have Internet connection with CA. Each user can act as a prover or a witness, depending on their roles at the moment. We assume the identity of a user is binded with his/her public key, which is certified by CA. Users have unique public/private key pairs, which are established during the user registration with CA and stored on users' personal devices. There are strong incentives for people not to give their privacy away completely, even to their families or friends, so we assume a user never gives his/her mobile device or private key to another party.

IV. Requirements and Challenges

Before introducing the details of our protocol, we first present and discuss the important issues and design challenges involved, in order to give an intuition of our objectives of constructing the protocol.

A. Security

The security of STP proofs are two fold: *integrity* and *non-transferability*. The integrity property requires that no prover can create fake STP proofs by himself/herself or by collaborating with one or more other untrusted parties in the system. The non-transferability property requires that no prover can claim the ownership of another prover's legitimate STP proofs.

B. Privacy

Anonymity: Location privacy is an extremely important factor that needs to be taken into consideration when designing any location based systems. Revealing both identity and location information to an untrusted party poses threats to a mobile users. First, a prover should be able to hide his/her identity from a witness. In addition, it is not only the prover's anonymity that we should pay attention to, a witness's anonymity should also be preserved. Since a witness who agrees to create an STP proof is co-located with the prover, his/her identity should not be revealed to the prover, either.

Pseudonyms: Pseudonyms are often used to provide anonymity. Nevertheless, if the same pseudonym is used by a mobile user, it is possible for an adversary to link multiple locations of the same pseudonym. By profiling and analyzing the user's location trace, the adversary which could reveal the identity of the user or at least significantly reduce the anonymity set. True anonymity requires unlinkability [10]. Anonymity can be effectively enhanced if a user is assigned with multiple pseudonyms, and pseudonyms are carefully chosen when communicating with another party. The APPLAUS scheme [3] adopts such an approach. However, this incurs high operational overhead because of the management of identities and their corresponding pseudonyms. This also requires a deliberate pseudonym scheduling algorithm which statistically eliminates the possibility of linking multiple pseudonyms or user profiling based on a single pseudonym. In addition, the pseudonym manager (e.g., CA) has to be completely trusted. Otherwise, it could be the single point of failure. If an adversary breaks into the pseudonym manager and obtains a copy of the pseudonym mapping, the whole system would break down. Therefore, we do not design STAMP based on pseudonyms. Instead, we use cryptographic encryption and commitment techniques to hide users' identities in the STP proof generation process.

Location granularity: An STP proof system needs to be flexible in terms of location granularity. The location of a prover could be represented by different levels of granularity, for example, a city, a neighborhood, or an exact geo-coordinate point. Though a prover needs to reveal both his/her identities and STP information in order to get services from a verifier, the prover does not necessarily trust the verifier completely. When a prover tries to claim his/her location at a particular time to a verifier, he/she should not be obligated to reveal his/her most accurate location to the verifier. Depending on the requested service, a prover should have control over the granularity level of his/her location that is revealed to the verifier.

C. Threat Model

Prover: A malicious prover seeks to create fake STP proofs without physically being present at a location. This includes creating fake STP proofs by himself/herself, lying to a witness about his/her location, tampering with the spatial-temporal information in his/her existing proofs, as well as stealing and using another user's STP proofs. Moreover, a malicious prover also attempts to obtain a witness's identity information in the entire process of STP proof generation.

Witness: A malicious witness's goals include acquiring a prover's identity information and repudiating an STP proof that is generated by him/her.

Verifier: A verifier is often a service provider or an authority that is trying to validate a prover's STP claim. A prover has to present both his/her identity and STP information to the verifier in order to get a service or simply prove his/her alibi. However, a prover should be able to only give a verifier his/her STP information that is necessary. In other words, a prover should have the control over which STP proofs and what location granularity of the STP proofs are revealed to a verifier.

CA: We assume CA is semi-trusted, in the sense it is only trusted in term of correctly performing its functions, i.e., user registration, key and credential management, and trust assessment for STP proofs. However, we consider CA not trusted in terms of protecting users' location privacy. CA may

intend to use any information it learned to profile user's spatial-temporal history and thus a potential privacy abuse may happen at CA.

Collusion: We specifically tackle two different collusion scenarios in this work: (1) A witness can collude with a prover by creating an STP proof for him/her even though one or both of them are not at the location as claimed in the STP proof. We name this collusion scenario as *W-P collusion*. To the best of our knowledge, there is no good solution to detect this type of collusion yet. (2) A prover A who requires a colluding prover B who is at a specific location to masquerade as him/her and generate a fake STP proof. Though we assume A does not give his/her private key to B, it is possible for A and B to have a hidden communication tunnel during the STP proof generation process, so that B could relay messages to A, A signs on them and returns them to B in real time. This kind of collusion attack is a type of *Wormhole* attack [11], which has been more commonly referred to as the *Terrorist Fraud* attack [12] in location verification. It is one of the most challenging attacks to protect against in location verification. Applied to our context, we name this collusion scenario as *P-P collusion*.

In this work, we do not specifically deal with establishing an anonymous communication channel between a prover and a witness. This can be achieved by various existing anonymous communication protocols [13]. Attacks via communication links and DoS attacks (e.g., jamming and flooding) are out of the scope of this paper.

V. The STAMP Scheme

A. Preliminaries

1) Location Granularity Levels: We assume there are n granularity levels for each location, which can be denoted by $L_1, L_2, \ldots, L_n$, where L_1 represents the finest location granularity (e.g., an exact Geo coordinate), and L_n represents the most coarse location granularity (e.g., a city). Hereafter, we refer to location granularity level as *location level* for short. When a location level L_x is known, we assume it is easy to obtain a corresponding higher location level L_y where $y > x$. The semantic representation of location levels are assumed to be standardized throughout the system.

2) Cryptographic Building Blocks: STAMP uses the concept of *commitments* to ensure the privacy of provers. A commitment scheme allows one to commit to a message while keeping it hidden to others, with the ability to reveal the committed value later. The original message cannot be changed after it is committed to. A commitment to a message M can be denoted as $C(M, r)$ where r is a nonce used to randomize the commitment so that the receiver cannot reconstruct M, and the commitment can later be verified when the sender reveals both M and r. A number of commitment schemes [14]–[17] have been proposed and commonly used. Our system does not require a specific commitment scheme.

One-way hash functions have the similar binding and hiding properties as commitment schemes. However, for privacy protection purpose, we do not use hash functions because they are vulnerable to *dictionary* attacks. An adversary who has a full list of possible inputs could run an exhaustive scanning over the list to crack the input of a hash function.

We assume every user has the ability to generate one-time symmetric keys. All parties have agreed upon a one-way hash function and a commitment scheme. All cryptographic notations have been summarized in Table I.

3) Distance Bounding: A location proof system needs a prover to be securely localized by the party who provides proofs. A distance bounding protocol serves the purpose. A distance bounding protocol is used for a party to securely verify that another party is within a certain distance [18]. Different types of distance bounding protocols have been studied and proposed. A most popular category is based on *fast-bit-exchange*: one party sends a challenge bit and another party replies with a response bit and vice versa. By measuring the round-trip time between the challenge and the response, an upper bound on the distance between the two parties can be calculated. This fast-bit-exchange phase is usually repeated a number of times.

TABLE I: List of notations

$M_1 \mid M_2$	Concatenation of messages M_1 and M_2
K_u^+	Public key of user u
K_u^-	Private key of user u
$E^K(M)$	Encryption of message M with key K
$H(M)$	One-way hashing of message M
$C(M, r)$	Commitment to message M with nonce r

One of the most challenging problems in distance bounding is the Terrorist Fraud attack, i.e., the P-P collusion scenario. The Terrorist Fraud attack is hard to defend against because a fast-bit-exchange process demands no processing delay (or at least extremely small processing delay) at the prover end between receiving a challenge bit and replying a response bit [18]. Thus, signing cannot be executed in the middle of a fast-bit-exchange, which means a hidden communication tunnel between two colluding parties allows them to execute fast-bit-exchange and signing separately. Thereby, one is only certain that the party who executed the fast-bit-exchange is nearby, but the party may not actually possess the private key of the identity who he/she claimed to be.

To the best of our knowledge, three existing distance bounding protocols [8], [19], [20] addressed the Terrorist Fraud attack. The schemes proposed in [19], [20] are based on pre-established shared secrets, and thus does not fit our scheme considering the anonymity requirement between a prover and a witness. The Bussard-Bagga protocol proposed in [8] is based on a zero-knowledge proof technique, and it allows the prover to be authenticated via a private/public key pair. Hence, we adopt the Bussard-Bagga protocol as our distance bounding protocol. The protocol consists of three stages. The first stage is the *preparation* stage, where the prover encrypts his/her private key K_p^- with a random symmetric key k and gets an encrypted message e. The prover then commits to each bit of e and k, resulting two sequences of bit commitments C_e and C_k. In the second *distance bounding* stage, the prover sends C_e and C_k to the location verifier (or the witness in our context), the location verifier then starts a multi-round fast-bit-exchange. In round i, the prover replies the ith bit of k or e depending on the challenge bit. Since the location verifier never learns both bit values, he/she can never learn about K_p^-. After the fast-bit-exchange, the location verifier de-commits and verifies the corresponding bit commitments in C_e and C_k (only for the received bits) by asking the prover to provide the nonces used for those commitments. In the third *zero-knowledge proof* stage, the prover convinces the verifier that he/she knows K_p^- through a zero-knowledge proof. It is not possible for a user to give away the values of k and e, which would mean that K_p^- is given away. Because of this, the protocol is not vulnerable to the Terrorist Fraud attack. In the scenario we are considering, a witness does not know the identity of a prover, we therefore cannot rely on the witness only to authenticate the prover via the zero-knowledge proof. We integrate the Bussard-Bagga protocol into STAMP by breaking up its execution and have the witness and verifier jointly authenticate the prover. The details are given in Section V-B.

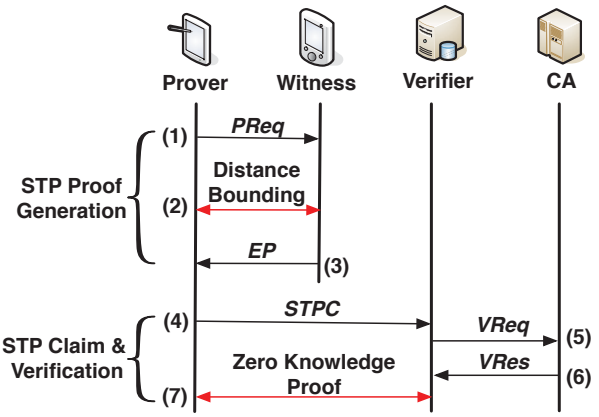


Fig. 2: An illustration of STAMP protocol

B. Protocol

1) Overview: Our protocol consists of two primary phases: *STP proof generation* and *STP claim and verification*. Figure 2 gives an overview of the two phases and the major communication steps involved.

When a prover collects STP proofs from his/her co-located mobile devices, we say an *STP proof collection event* is started by the prover. An STP proof generation phase is the process of the prover getting an STP proof from one witness. Therefore, an STP proof collection event may consist of multiple STP proof generations. The prover finally stores the STP proofs he/she collected in the mobile device.

When a prover encounters a verifier (the frequency of such encounters is specific to the application scenarios) and he/she intends to make a claim about his/her past STP to the verifier, the STP claim and verification phase takes place between the prover and the verifier. A part of the verification job has to be done by CA. Therefore, communication between the verifier and CA happens in the middle of the STP claim and verification phase.

In Figure 2, the two arrowed lines in red color represent the latter two stages of the Bussard-Bagga protocol. These stages require multiple interactions between the two involved parties, and thereby are represented by doubly arrowed lines. The preparation stage of the Bussard-Bagga protocol does not need to be executed for every STP proof generation and thus is not shown. Users could run the preparation stage before each STP proof collection event or pre-compute and store several sets of the bit commitments and primitives, and randomly choose one set of them when needed. Subsequently, we present the details of the STAMP protocol.

2) STP Proof Generation: **Prover**: Suppose a prover wants to start an STP proof collection event at time t, the prover first broadcasts an *STP proof request* (denoted as $PReq$) to other nearby mobile devices and waits for responses. A $PReq$ is constructed as follows:

$$PReq = C(ID_p, r_p)|L_1|t \tag{1}$$

where ID_p is the prover's ID, r_p is a random nonce generated by the prover for the commitment to ID_p, and L_1 is the lowest level of the current location.

Witness: A witness who receives a $PReq$ decides if he/she accepts the request. If the request is accepted, the witness sends an ACK back to the prover, after which, the two parties start the execution of the distance bounding stage of the Bussard-Bagga protocol. This enables the witness to know that the party who is requesting an STP proof is within a certain range. However, the witness has no way to verify if the party has the private key which in fact corresponds to the committed identity. The zero-knowledge proof stage cannot be carried out by the witness because it requires the knowledge of the prover's public key. As shown in Figure 2, we leave the zero-knowledge proof stage to be executed by a verifier later in the STP claim and verification phase. However, this does not mean the witness can simply ignore the zero-knowledge proof stage. Based on the Bussard-Bagga protocol, the witness is able to compute a big integer z from the bit commitments received from the prover after the distance bounding stage. The witness has to make sure that z is securely enclosed in the STP proof so that the verifier can run the zero-knowledge proof stage base on z and the prover's public key K_p^+.

If the distance bounding stage succeeds, the witness starts creating an STP proof for the prover. The witness first creates an *STP record* (denoted as $STPR$):

$$STPR = C(L_1, r_w^1)|\dots|C(L_n, r_w^n)|t \tag{2}$$

where r_w^1 is a random nonce generated by the witness and used to commit to L_1 provided in $PReq$. The higher location levels $L_2, \dots, L_n$ are also committed with different nonces $r_w^2, \dots, r_w^n$, which in turn are derived based on a *hash chain* of r_w^1:

$$r_w^x = H(r_w^{x-1}) \qquad \forall\ x = 2, 3, \dots, n \tag{3}$$

A plaintext STP proof (denoted as P) is then created as follows:

$$P = C(ID_p, r_p)|STPR|z \tag{4}$$

P is finally endorsed by the witness and encrypted using CA's public key. The *endorsed STP proof* (denoted as EP) is given by:

$$EP = E^{K_{CA}^+}(ID_w|P|E^{K_w^-}(H(P))) \tag{5}$$

where ID_w is the witness's ID. Finally, the witness sends $EP|r_w^1$ to the prover. EP is encrypted using CA's public key to protect the witness's ID from being seen by the prover. r_w^1 should not be seen by CA and thus is not included in P but sent along with EP.

Prover: Suppose the prover finally receives $EP|r_w^1$ from m witnesses (denoted as $EP_1|r_{w,1}^1, \dots, EP_m|r_{w,m}^1$) for this STP proof collection event, the prover stores them locally together with the associated primitives (i.e., r_p and the Bussard-Bagga primitives) and the spatial-temporal information (i.e., L_1 and t). We say the prover now has created an *STP proof entry* for himself/herself at location L_1 and time t.

3) STP Claim and Verification: **Prover**: At the beginning of an STP claim and verification phase, the prover extracts the necessary data from his/her corresponding STP proof entry and creates an *STP claim* (denoted as $STPC$) as follows:

$$STPC = EP_1|\dots|EP_m|r_{w,1}^x|\dots|r_{w,m}^x|ID_p|r_p|L_x|t \tag{6}$$

where L_x is the lowest location level that the prover intends to reveal to the verifier. $r_{w,1}^x, \dots, r_{w,m}^x$ are derived from the $r_{w,1}^1, \dots, r_{w,m}^1$ based on a hash chain operation.

Verifier: After receiving the prover's $STPC$, the verifier needs CA's assistance in verifying the $STPC$. The verifier now constructs a *verification request* (denoted as $VReq$) by extracting the following information from the $STPC$:

$$VReq = EP_1|\dots|EP_m|ID_p|r_p \tag{7}$$

The $VReq$ is then sent to the CA.

CA: When CA receives a $VReq$, it is able to decrypt everything in $EP_1|\dots|EP_m$ except for the committed location levels in the $STPR$s, because CA does not know any of the random numbers used by the witnesses to construct the location level commits. CA is now responsible for two tasks: *EP verification* and *P-W collusion detection*.

First, CA performs EP verification by checking the following in each EP enclosed in the $VReq$:

- Signature of K_w^- agrees with the public key of ID_w;
- $H(P)$ agrees with P;
- $C(ID_p, r_p)$ can be de-committed with ID_p and r_p.

For all the EPs that passed the verification, CA starts a trust evaluation and obtains a P-W collusion detection result. We present the details of the P-W collusion detection process separately in Section V-B4.

If all the EPs fail the verification or the P-W collusion detection returns a positive result, CA sends back a *verification response* (denoted as $VRes$) with a one-bit failure notification to the verifier. Otherwise, CA creates a $VRes$ as follows and sends it back to the verifier:

$$VRes = E^{K_{CA}^-}(STPR_1|\dots|STPR_m|z) \quad (8)$$

where $STPR_1|\dots|STPR_m$ are the $STPR$s extracted from $EP_1|\dots|EP_m$ respectively, and z is the big integer resulted from the distance bounding stage. Notice that z's value is derived from the prover's bit commitments that are prepared for the Bussard-Bagga protocol. For different witnesses, we request the prover to use the same bit commitments during a same STP proof collection event. In this case, the same z should be in each of $EP_1|\dots|EP_m$. Therefore, only one copy of z is attached in $VRes$.

Verifier: Upon receiving the $VRes$, the verifier performs two additional verification operations:

- **Zero-knowledge proof**: The zero-knowledge proof is done based on z and the prover's public key K_p^+. A multi-round interaction is executed to minimize the prover's chance of cheating.
- **STPR opening**: From $VRes$, the verifier obtains $STPR_1|\dots|STPR_m$. The temporal information t in each $STPR$ is first checked against t claimed by the prover. A de-commitment is then done for $C(L_x, r_w^x)$ in each $STPR$, with L_x and $r_{w,1}^x|\dots|r_{w,m}^x$ obtained from the $STPC$. An inconsistent t or a location commitment which cannot be de-committed nullifies the corresponding $STPR$ and thus the EP.

Now, suppose the verifier has a list of legitimate EPs passed the verifications, the verifier finally needs to determine if the prover's STP claim is successful by looking at number of legitimate EPs. Without loss of generality, we do not specify how the verifier makes such a decision. For instance, the verifier may consider the prover's STP claim successful as long as the number of legitimate EPs or the percentage of legitimate EPs among the originally received EPs exceeds a certain threshold.

4) P-W Collusion Detection: If a prover colludes with a witness, it is easy for the witness to give the prover a legitimate STP proof with fake spatial-temporal information. Since the STP proof generation process is done in an opportunistic manner and we do not assume a trusted party (e.g., a location authority or a trusted witness) in this process, a P-W collusion cannot be prevented or detected with a 100% certainty. As a countermeasure against P-W collusions, we proposed an entropy-based trust model which measures the likelihood of such an attack. The trust evaluation is done by CA, which requires CA to keep track of the STP proof transaction history between any two users. A user's STP proof transactions include both the STP proofs he/she gets as a prover and the STP proofs he/she creates as a witness.

First of all, we want to measure each user's collusion likelihood, based on his/her past STP proof transaction history. The intuition is that a legitimate user should not intentionally choose his/her witnesses, and therefore a user who gets majority of his/her STP proofs from a small set of users has a high likelihood to be colluding with these users. Based on this intuition, we define two factors which determine a user u's collusion likelihood:

- **Diversity**: This is defined as the number of different users who had STP proof transactions with u. A higher diversity indicates that u does not rely on a small group of witnesses, and thus suggests low collusion likelihood.
- **Fairness**: This is defined as the randomness in the distribution of STP proof transactions among all the different users who have had STP proof transactions with u. A highly random distribution indicates u does not manipulate the witness choosing process, and thus suggests low collusion likelihood.

We use *entropy* to measure the collusion likelihood of a user because of its capability of capturing both the above two factors. In the STAMP system, provers meet witnesses on-the-spot. Thus, the system follows the Markov property which assumes that prover-witness pairing is memoryless. This is analogous to the rationale behind the definition of entropy in information theory. Given a user, if the diversity and fairness of his/her past STP proof transactions are high, the unpredictability of the prover-witness pairing will be high. Hence, we would like consider the user has a low collusion likelihood. Entropy is a measure of such unpredictability. Assuming u has a total number of N different users who had STP proof transactions with him/her, we denote this set of users as $u_1, u_2, \dots, u_N$. Applying the definition of entropy into our context, u's entropy is given by:

$$E_u = -\sum_{i=1}^{N} p(u, u_i) \log p(u, u_i) \quad (9)$$

where $p(u, u_i)$ denotes the percentage of past STP proof transactions between u and u_i out of u's total past STP proof transactions.

The entropy measure gives an incentive to users to increase the diversity and fairness in the process of generating their STP proofs, regardless of the size of the user set. For scenarios where there is a massive number of users, each user is encouraged to reach out and interact with different users in order to maintain a high entropy. Certain applications may require users to interact with only a limited set of other users, such as the battlefield scout group example. In such applications, lacking diversity and fairness in a user's STP proof generation pattern is still undesirable and may be deemed as having a higher collusion tendency.

The trust of an EP (denoted as T) is a scalar in [0, 1] which is evaluated by CA based on the prover's and witness's entropy as well as the specific STP proof transaction history between these two users. We define T as follows:

$$T = 1 - e^{-\frac{E_p \cdot E_w \cdot \omega}{\mathbf{Q}(p,w)}} \quad (10)$$

where $\mathbf{Q}(P, W)$ denotes the number of STP proof transactions between the prover and the witness out of the total number of their distinct STP proof transactions in the past; ω is a scaling parameter which can be fine-tuned. The exponential term can be thought of as the penalty applied to the trust based on the collusion likelihood between the prover and the witness derived from their past STP transaction history with each other as well as with other users.

With the trust level of each EP extracted from a $VReq$, CA needs to determine if these EPs resulted from a collusion. Such a decision should be made by consolidating the trust values calculated for all the EPs into one measure (denoted

as $\hat{T}$):

$$\hat{T} = F(T_1, T_2, \ldots, T_m) \tag{11}$$

where $F()$ is the *trust consolidation function*. Different trust consolidation functions could be used. Some straightforward examples are max, min, and $average$. We suggest using *weighted average*, where the trust values are weighted by the number of past STP proof transactions of the corresponding EP's witness. This makes the witnesses who have more past STP proof transactions (i.e., whose entropy is expected to be more accurate) have bigger influence on $\hat{T}$. Ultimately, a decision can be made by comparing $\hat{T}$ against a trust threshold (denoted as θ).

VI. Security Analysis

In this section, we analyze the security properties of the STAMP protocol and prove that the protocol can achieve our security goals.

Proposition 1. *A prover cannot create a legitimate EP without a witness.*

Since users do not give away their private keys, a prover has no access to another user's private key. A plaintext STP proof (P) has to be signed by a legitimate witness to create a legitimate EP. If a prover uses his/her own private key or an illegitimate private key to create a signature for EP. CA will be able to detect it.

Proposition 2. *Without colluding with a witness, a prover cannot create a legitimate EP without being present at the claimed location at the claimed time.*

Based on Proposition 1, a prover has to ask a witness to create a legitimate EP. Let us now consider two attacks: (1) a prover asserts a false location/time in a $PReq$; (2) a prover establishes a hidden communication tunnel with a proxy at the intended location and ask the proxy to send a $PReq$ for him/her (i.e., P-P collusion).

When a legitimate witness receives a $PReq$, he/she can easily check if t in $Preq$ is within an acceptable range from the current time. Subsequently, the execution of the distance bounding stage enables the witness to determine if the party who sent the $PReq$ is within an acceptable distance. Since no signal travels faster than the speed of light, a prover who communicates with the witness from a distant location will be detected by the fast-bit-exchange in the distance bounding stage. Hence, Attack (1) can easily be detected by the witness.

Based on the Bussard-Bagga protocol, the zero-knowledge proof stage is able to guarantee that a party who ran the distance bounding stages with the witness in fact has the private key corresponds to the committed ID_p in a $PReq$. That means, a prover has to give his/her private key to the proxy in order to pass both the distance bounding stage with the witness and the zero-knowledge proof stage with the verifier. Assuming a user never gives away his/her private key, our protocol ensures that Attack (2) cannot succeed.

Proposition 3. *A prover cannot change the spatial and/or temporal information in an EP.*

The location levels $L_1, \ldots, L_n$ are committed by the witness in an $STPR$. The $STPR$ is in turn encrypted by CA's public key in an EP. The prover does not have CA's private key, and thus cannot decrypt an EP and see the location level commitments.

Proposition 4. *A prover cannot use an EP created for another prover.*

By the binding property of commitments, a prover's ID is binded with the $C(ID_p, r_p)$, which is in turn encrypted in an EP. A prover therefore cannot change the ID_p binded with an EP. If a prover claims to a verifier with his/her own ID_p and another prover's EP, CA will detect that the $C(ID_p, r_p)$ in the EP does not agree with the ID_p in the $VReq$ sent by the verifier. If a prover claims to a verifier with another prover's ID_p and EP, hoping to get services without showing his/her own identity, the verifier will detect that the prover does not possess the private key corresponding to ID_p via the zero-knowledge proof stage.

Proposition 5. *A witness cannot repudiate a legitimate EP created by him/her.*

A legitimate EP contains $E^{K_w^-}(H(P))$. Based on the assumption that no user gives away his/her private key, $E^{K_w^-}(H(P))$ assures the non-repudiation property of an EP.

Proposition 6. *A prover and a witness cannot find out each other's identity.*

During an STP proof generation process, the prover's identity ID_p is committed. Since r_p is not known to the witness, he/she cannot de-commit $C(ID_p, r_p)$ and obtain ID_p. The witness's identity ID_w is enclosed in EP, which is encrypted by CA's public key. Since the prover does not possess CA's private key, he/she cannot decrypt EP and obtain ID_w. Furthermore, based on the Bussard-Bagga protocol, the distance bounding stage does not reveal the two parties' identities to each other.

Proposition 7. *$PReq$s sent from the same prover for different STP proof collection events are unlinkable to a witness.*

A prover chooses different r_ps at different locations. Even a witness has received multiple $PReq$s from the same prover at different locations, there is no information that could help the witness to link these $PReq$s and thus obtain a location trace of the prover.

Proposition 8. *STP proofs generated from the same witness for different STP proof collection events are unlinkable to a prover.*

A witness chooses different r_w^1's for different STP proof collection events. The EP generated by a witness is always encrypted by CA's public key. Even the same prover has received multiple pieces of $EP|r_w^1$ from the same witness at different locations, there is no information that could help the prover to link these pieces of $EP|r_w^1$ and thus obtain a location trace of the witness.

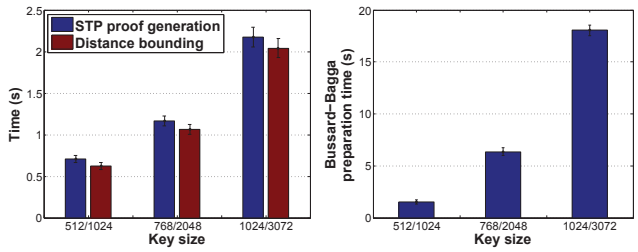
Proposition 9. *The lowest location level a verifier learns about a prover is the level that the prover intends to reveal to him/her.*

In an $STPC$, the prover sends an r_w^x to the verifier for each EP, where x is the location level that the prover intends to reveal to the verifier. Due to the one-wayness of the hash-chain applied on r_w^1, the verifier cannot obtain any r_w^y where $y < x$. Therefore, though the verifier has the access to $C(L_1, r_w^1), \ldots, C(L_n, r_w^n)$, the lowest location level commitment that can be opened by the verifier is $C(L_x, r_w^x)$. Due to the randomness introduced by the r_w's, a dictionary attack over all possible locations under the location level x is infeasible.

Proposition 10. *CA cannot learn any location information about a prover or witness from $VReq$.*

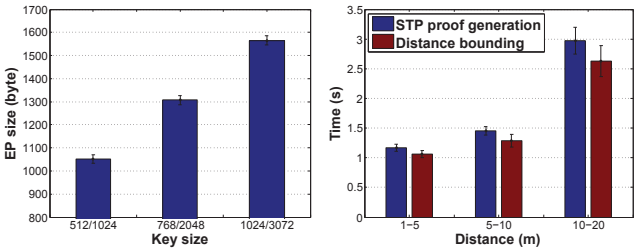
CA cannot de-commit any location level commitments from a $VReq$ and thus cannot obtain any location information about the prover and the witnesses associated with the $VReq$. Due to the randomness introduced by the r_w's, a dictionary attack over all possible locations is infeasible.

The above analysis shows that our pre-discussed security goals are achieved properly by the STAMP protocol. In



(a) Time to generate an STP proof under different key sizes

(b) Time of Bussard-Bagga preparation stage under different key sizes



(c) Size of EP under different key sizes

(d) Time to generate an STP proof under different communication distances

Fig. 3: Implementation results

summary, an STP proof's integrity and non-transferability is assured; the possibility of P-P collusion is eliminated; provers and witnesses always remain anonymous in the process of proof generations; a verifier only sees the location levels that a prover intends to reveal; CA only learns users' identities but not their locations.

VII. Experiments and Results

A. Prototype Implementation

We implemented a prototype client application on Android with Java. Our experiments are carried out on two Samsung Exhibit II 4G devices equipped with Qualcomm MSM 8255 1GHz chipset, 512MB RAM, 1GB ROM, GPS, and Bluetooth, and running Android OS 2.3. Bluetooth is used as the communication interface between mobile devices. We use DSA key pairs for signing/authentication operations because DSA is based on the discrete-log problem, which makes it possess the mathematical properties desired by the Bussard-Bagga protocol. Since DSA is not designed for encryption/decryption purpose, we use RSA key pairs as subkeys for encryption/decryption operations. We use SHA1 as the one-way hashing function and 128-bit AES as the symmetric key encryption scheme. We implemented the string commitment scheme presented in [15] and use it for ID and location commitments. We model each location with six levels: exact location, neighborhood, town/city, region/county, state and country, where each level is represented by a name string except that the lowest level also has the geo-coordinates.

With our implementation, we examine the computational time (also an indicator of power consumption) and storage that are needed to run STAMP. Since the STP verification is done by verifiers and CA where desktops or servers with high computational power can be used, we focus our testing on the STP proof generation phase that is executed on mobile phones. The results we show are obtained based on 10 runs of each test. No other background processes were running in parallel during the tests.

First, we study the impact of key size on the performance of our client application. Since both DSA and RSA are used in our implementation, we test three key size combinations representing three different security levels: (1) 512-bit DSA with 1024-bit RSA (denoted as 512/1024); (2) 768-bit DSA with 2048-bit RSA (denoted as 768/2048); (3) 1024-bit DSA with 3072-bit RSA (denoted as 1024/3072). Figure 3(a)-(c) show the computational resources required with these three key size settings.

Figure 3(a) shows the time needed for a prover to get an STP proof from a witness and for the portion of this process taken by the Bussard-Bagga distance bounding. We could observe that a majority portion of the STP proof generation is taken by the Bussard-Bagga distance bounding. It is easy to disable the distance bounding stage for application scenarios where P-P collusion is not a concern. In that case, the time for each proof generation will be significantly reduced to less than 0.2s even if large keys are used. Figure 3(b) shows the time needed for the Bussard-Bagga preparation stage. We can see this could cause long delay ($\sim$18s) if large keys are used. However, as we explained in Section V-B, to achieve best unlinkability, it could be executed only once for an STP proof collection event. Under a relaxed unlinkability requirement, users could also pre-compute and store several sets of the bit commitments and primitives, and randomly choose one set of them when needed. Figure 3(c) shows the size of an EP that needs to be stored on a prover's mobile device. Since multiple EPs could be received for each STP proof collection event, the size of an EP is the main factor that determines the storage need for an STP proof entry. We can see that each EP is less than 2000 bytes. Though several such EPs may need to be retained for each STP proof entry, the storage consumption is definitely acceptable considering the storage capacity of today's mobile devices.

Figure 3(a)-(c) tell us the choice of key size is critical. Larger keys provide stronger security, but also require more resources in terms of computational time and storage. For achieving general security requirements for a light-weight mobile application, we suggest using a small key size setting.

In addition to key size, we study the impact of communication distance between mobile devices on the STP proof generation time. Figure 3(d) shows the results of our testing with the 768/2048 key size setting. As expected, the communication distance negatively affects the STP proof generation time as well as the distance bounding time.

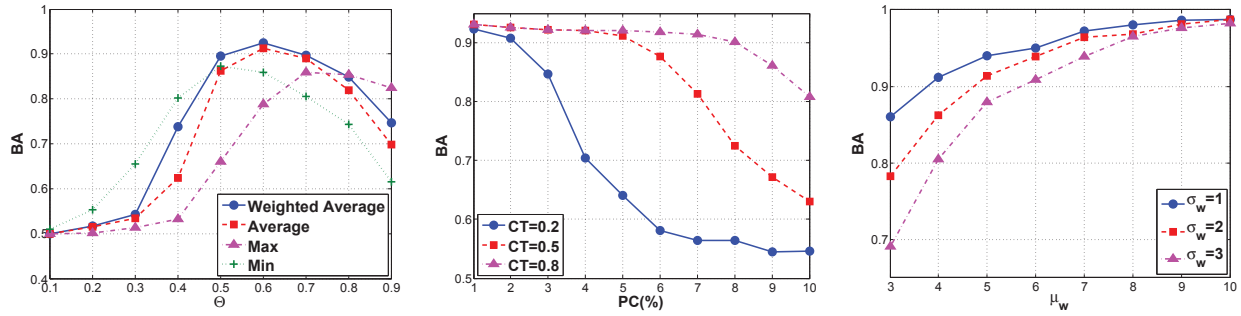
TABLE II: Default simulation settings

Parameter	Value
Percentage of colluding attackers (PC)	2%
Collusion tendency (CT)	0.2
Mean of witnesses (μ_w)	5
Standard deviation of witnesses (σ_w)	2
Trust scaling parameter (ω)	1×10^{-4}
Collusion trust threshold (θ)	0.6

B. Simulation

To measure the effectiveness and accuracy of our P-W collusion detection, we implemented our trust model with Java simulation. In this section, we present our simulation details and the performance results that we obtained from our simulation experiments.

1) Simulation Setup: In our simulation, a total number of 1000 users are deployed. All users are traveling with a random mobility model. In each STP proof collection event, a random prover is selected among all the users. The selection of witnesses is modeled by a Gaussian distribution with default mean (μ_w) of 5 and default standard deviation (σ_w) of 2. The *percentage of colluding attackers* (PC) among these users is varied from 1% to 10%. We allow all the attackers to find each other through a hidden channel and form a collusion



(a) BA under different trust consolidation functions

(b) BA under different percentage of colluding attackers and collusion tendency

(c) BA under different number of witnesses

Fig. 5: Impact of system settings and network conditions on BA

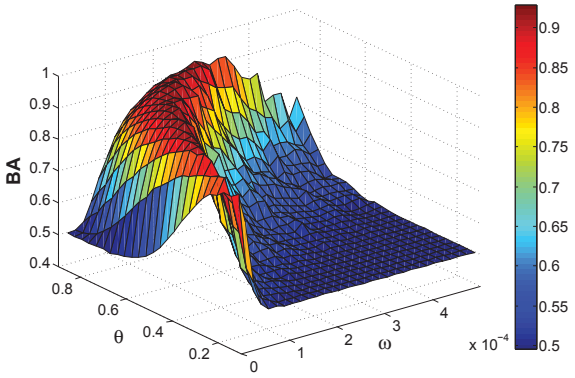


Fig. 4: BA under different ω and θ

group. Whenever an attacker needs to get a fake STP proof, he/she seeks assistance from random witnesses in the collusion group, in order to maximize his/her own entropy by making his SPT proof generation patten as unpredictable as possible within the collusion group. Each attacker is configured with a *collusion tendency* (CT) in the range of (0, 1], which represents the attacker's probability of launching a collusion for each of his/her STP proof collection event. In our experimental tests, we vary some critical parameters to see their impact on the performance. Default settings are used for parameters that are not under test. Table II summarizes our default settings.

We run a training phase with the first 10000 STP proof collection events, i.e., an average of 10 STP proof collection events for each user. Each of the data points shown in our simulation results is based on another 100000 STP proof collection events after the training phase, i.e., an average of 100 STP proof collection events for each user.

2) Performance Metric: We use the *Balanced Accuracy* (BA) [21] as our performance metric, which is a commonly used accuracy measure for classification algorithms. BA is defined as the arithmetic mean of sensitivity (true negative rate) and specificity (true negative rate). We choose BA because: (1) its interpretation is straightforward, (2) it takes both sensitivity and specificity into account, and (3) it avoids inflated performance estimates on imbalanced datasets.

3) Simulation Results: First, two crucial parameters that affect how our trust model performs are the trust scaling parameter ω and collusion trust threshold θ. We run extensive tests to see the BA distribution under different choices of ω and θ with our default setup. The results are shown in Figure 4. Our trust model performs well (BA ¿ 0.9) only when good ω and θ are chosen. From Figure 4, we observe that our trust model is very sensitive to the choice of ω, which agrees with Equation 10. For different choices of ω, different θ has to be set to achieve a high BA. We choose $\omega = 1\times10^{-4}$ and $\theta = 0.6$ because this pair yields the best result in our testing.

Subsequently, we examine how the choice of trust consolidation functions affects the accuracy of our collusion detection. Figure 5(a) shows the BA results when we used *weighted average*, *average*, *max* and *min*. The tests are carried out under the default settings except that we vary the collusion trust threshold θ, because different trust consolidation functions get their best BA with different θ. *max* gets its best BA with a higher θ than other approaches because *max* only compares the highest trust value with θ. Similarly, *min* is gets its best BA with a lower θ. Comparing the best BAs yielded by the four approaches, we can see *weighted average* outperforms the other three. In our simulation, the STP proof collection events are randomly distributed to the users, so users' amounts of past STP proof transactions are roughly balanced. We anticipate the BA of *weighted average* will be more pronounced if users' amounts of past STP proof transactions are more skewed.

Figure 5(b) shows the performance of our trust model with different percentages of colluding attackers (PC) and when the attackers have different collusion tendencies (CT). We can see that our trust model is more resistant against attackers with a higher collusion tendency. Under high collusion tendency (CT=0.8), our trust model achieves a BA over 0.9 for up to 8% of colluding attackers. This means if attackers launches collusions frequently so that 80% of their STP proofs are fake, our trust model is able to detect the collusions with BA over 0.9 even when 8% of all users are colluding. If the percentage of colluding attackers is low (PC $\leq$ 2%), Figure 5(b) shows that our trust model can achieve a BA over 0.9 even when CT is as low as 0.2. This means if 2% of all users are colluding, and every attacker is intelligent and tries to cover their collusions by having 80% of their STP proof transactions legitimate, our trust model can still detect collusions with a BA over 0.9.

Other than the above tested factors, we want to study the impact of the number of witnesses involved in an STP proof collection event. Since we model the number of witnesses for each STP proof collection event with Gaussian Distribution, we vary the mean (μ_w) and standard deviation (σ_w) and test the resulting BA when other settings are set to default. Figure 5(c) shows the BA levels we get for different μ_w and σ_w. We can conclude that our trust model performs better when there are more witnesses involved in STP proof collection events on average. Thereby, a good way to make P-W collusions hard and also enhance the collusion detect accuracy is to require more witnesses for an STP proof.

VIII. Discussion and Future Work

From our experimental results, we observe that under small key size settings, our scheme works efficiently in terms of both computational and storage resources. However, the computational latency could become rather long when large keys are desired. A major part of computational cost is caused by the Bussard-Bagga protocol, which is known for its expensive computation due to large amount of modular exponentiations [18]. Other than defending against the Terrorist Fraud attack (P-P collusion), functionalities of STAMP do not specifically rely on the Bussard-Bagga protocol. Therefore, under circumstances where P-P collusion is not a concern, we suggest to disable the Bussard-Bagga stages in STAMP, which will result less than 0.2s for each STP proof transaction (distance bounding time deducted from STP proof generation time) without the necessity of the preparation stage. Furthermore, active on-going research in the location verification field is being conducted to achieve the same security property as the Bussard-Bagga protocol with much better performance. It is also a part of our future work to investigate such possibilities. A new distance bounding scheme can be easily plugged into STAMP and replace the Bussard-Bagga protocol.

Our P-W collusion detection is supported by entropy-based trust evaluation, instead of complex graph algorithms like the ones used by the APPLAUS system. Therefore, each run of our P-W collusion detection only requires a number of cheap computations. It is much more efficient than APPLAUS where a few hundred seconds are needed to run a detection among a few thousands of users. The weakness of our detection, however, is that if attackers only launch collusions very infrequently, or if there is a large pool of users that an attacker can choose to collude with, the accuracy may drop significantly. Nevertheless, unless trusted infrastructures are deployed at every location, it is always hard to tell if an STP proof is a result of collusion or not. Our trust model serves as a good countermeasure so that malicious users are deterred from launching collusions of their own free will or with only a small group of users. In the future, we will further examine alternative approaches which could make such collusions impossible to succeed. A potential direction is to utilize camera or other sensors on mobile devices and have a prover and a witness jointly create a proof that contains a consistent signature (e.g., a picture) of the location.

STAMP does not limit the possibility of utilizing wireless APs. We can easily accommodate wireless APs in our architecture by treating them as stationary witnesses. Since we are targeting a distributed system in this paper, we do not specifically differentiate wireless APs from other mobile devices. In fact, if trusted APs are deployed, our P-W collusion detection can be further enhanced because provers who claim their presence at a location with a trusted AP should have an EP from the AP. We will study how our trust model could be improved given some trusted EPs.

IX. Conclusion

In this paper we have presented STAMP, which aims at providing security and privacy assurance to mobile users' proofs for their past location visits. STAMP relies on mobile devices in vicinity to mutually generate location proofs. Integrity and non-transferability of location proofs and location privacy of users are the main design goals of STAMP. We have specifically dealt with two collusion scenarios: P-P collusion and P-W collusion. To protect against P-P collusions, we integrated the Bussard-Bagga distance bounding protocol into the design of STAMP. To detect P-W collusion, we proposed an entropy-based trust model to evaluation the trust level of each claim of the past location visits. Our security analysis shows that STAMP achieves the security and privacy objectives. Our implementation on Android smartphones indicates that low computational and storage resources are required to execute STAMP on mobile devices. Extensive simulation results show that our trust model is able to attain a high balanced accuracy (> 0.9) with appropriate choices of system parameters.

Acknowledgment

This work was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official polices, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

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

[2] W. Luo and U. Hengartner, "VeriPlace: a privacy-aware location proof architecture.," in *ACM GIS*, 2010.

[3] Z. Zhu and G. Cao, "Towards privacy-preserving and colluding-resistance in location proof updating system," *IEEE Transactions on Mobile Computing*, 2011.

[4] N. Sastry, U. Shankar, and D. Wagner, "Secure verification of location claims," in *ACM WiSe*, 2003.

[5] R. Hasan and R. Burns, "Where have you been? Secure location provenance for mobile devices," *CoRR*, 2011.

[6] B. Davis, H. Chen, and M. Franklin, "Privacy preserving alibi systems," in *ACM ASIACCS*, 2012.

[7] I. Krontiris, F. Freiling, and T. Dimitriou, "Location privacy in urban sensing networks: research challenges and directions," *IEEE Wireless Communications*, 2010.

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

[9] B. Waters and E. Felten, "Secure, private proofs of location," tech. rep., Department of Computer Science, Princeton University, 2003.

[10] A. Pfitzmann and M. Köhntopp, "Anonymity, unobservability, and pseudonymity-a proposal for terminology," in *Designing privacy enhancing technologies*, Springer, 2001.

[11] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Wormhole attacks in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 370–380, 2006.

[12] Y. Desmedt, "Major security problems with the 'unforgeable' (feige)-fiat-shamir proofs of identity and how to overcome them," in *SecuriCom*, 1988.

[13] G. Danezis and C. Diaz, "A survey of anonymous communication channels," *Computer Communications*, 2008.

[14] M. Naor, "Bit commitment using pseudorandomness," *Journal of cryptology*, 1991.

[15] S. Halevi and S. Micali, "Practical and provably-secure commitment schemes from collision-free hashing," in *CRYPTO*, 1996.

[16] I. Damgård, "Commitment schemes and zero- knowledge protocols," *Lectures on Data Security*, 1999.

[17] I. Haitner and O. Reingold, "Statistically-hiding commitment from any one-way function," in *ACM Symposium on Theory of Computing*, 2007.

[18] D. Singelee and B. Preneel, "Location verification using secure distance bounding protocols," in *IEEE MASS*, 2005.

[19] J. Reid, J. Nieto, T. Tang, and B. Senadji, "Detecting relay attacks with timing-based protocols," in *ACM ASIACCS*, 2007.

[20] C. Kim, G. Avoine, F. Koeune, F. Standaert, and O. Pereira, "The swiss-knife rfid distance bounding protocol," in *ICISC*, 2009.

[21] K. Brodersen, C. Ong, K. Stephan, and J. Buhmann, "The balanced accuracy and its posterior distribution," in *IEEE ICPR*, 2010.