

## STAMP: Enabling Privacy-Preserving Location Proofs for Mobile Users

B.Naveenkumar<sup>1</sup>, Dr.A.P.Janani M.E.,Ph.D.,<sup>2</sup>

<sup>1</sup>Department of Computer Science, Sri Subramanya College of Engineering and Technology, Palani

<sup>2</sup>Department of Computer Science, Sri Subramanya College of Engineering and Technology, Palani

<sup>1</sup>[navinbnp@gmail.com](mailto:navinbnp@gmail.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. We present the spatial temporal provenance assurance with mutual proofs (STAMP) scheme. STAMP is designed for ad-hoc mobile users generating location proofs for each other in a distributed setting. However, it can easily accommodate trusted mobile users and wireless access points. STAMP ensures the integrity and non-transferability of the location proofs and protects users privacy. A semi-trusted certification authority is used to distribute cryptographic keys as well as guard users against collusion by a light entropy-based trust evaluation approach. Our prototype implementation on the Android platform shows that it is how cost in terms of computational and storage resources. Extensive simulation experiments show that our entropy-based trust model is able to achieve high collusion detection accuracy.

**Keywords** : entropy-based trust model, cryptographic key, collusion detection.

### 1. Introduction

It requires 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 in literature 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, provenance proof, and location alibi. 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. 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.

## 1. System Analysis

### 1.1 Existing System

Everyplace 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 information), 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.

### Disadvantages

- Even more intense requirement for self-direction.
- ✓ Limitations of process in existing system.
- ✓ Low efficiency and delay in result.

### 1.2 Proposed System

A prover and a witness communicates with each other via Bluetooth or Wi-Fi in ad hoc mode. A peer discovery mechanism for discovering nearby witness is required and preferably provided by underlying communication technology instead of our protocol. The proof generation system of prover is presented a List of available witnesses. When there are multiple witnesses willing to cooperate, the prover initiate protocol with them sequentially. STP claims are sent to verifiers from prover 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 bound 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.

### Advantages

- ✓ The intrusion detection is high
- ✓ The performance is high

### 3. System Design

#### 3.1 Flow diagram

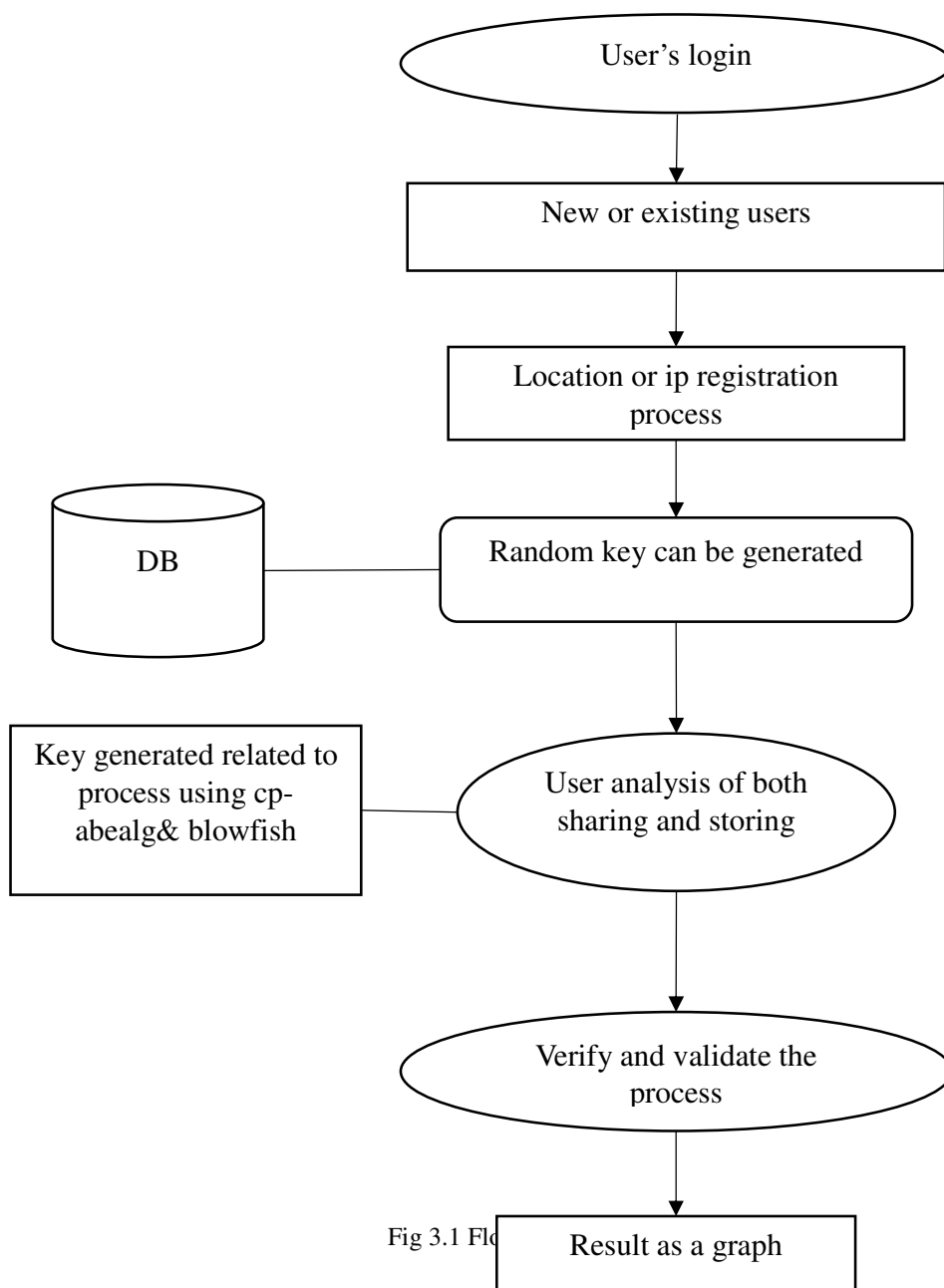


Fig 3.1 Flow

### 3.2 Use case diagram

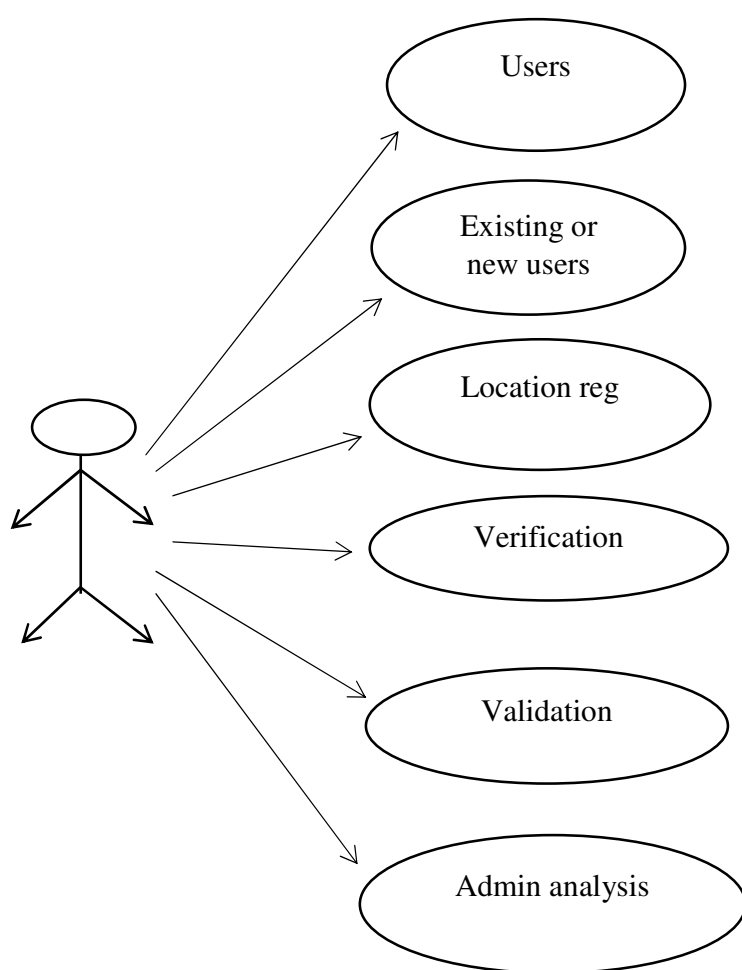


Fig 3.2 Use Case Diagram

### 3.3 Class Diagram

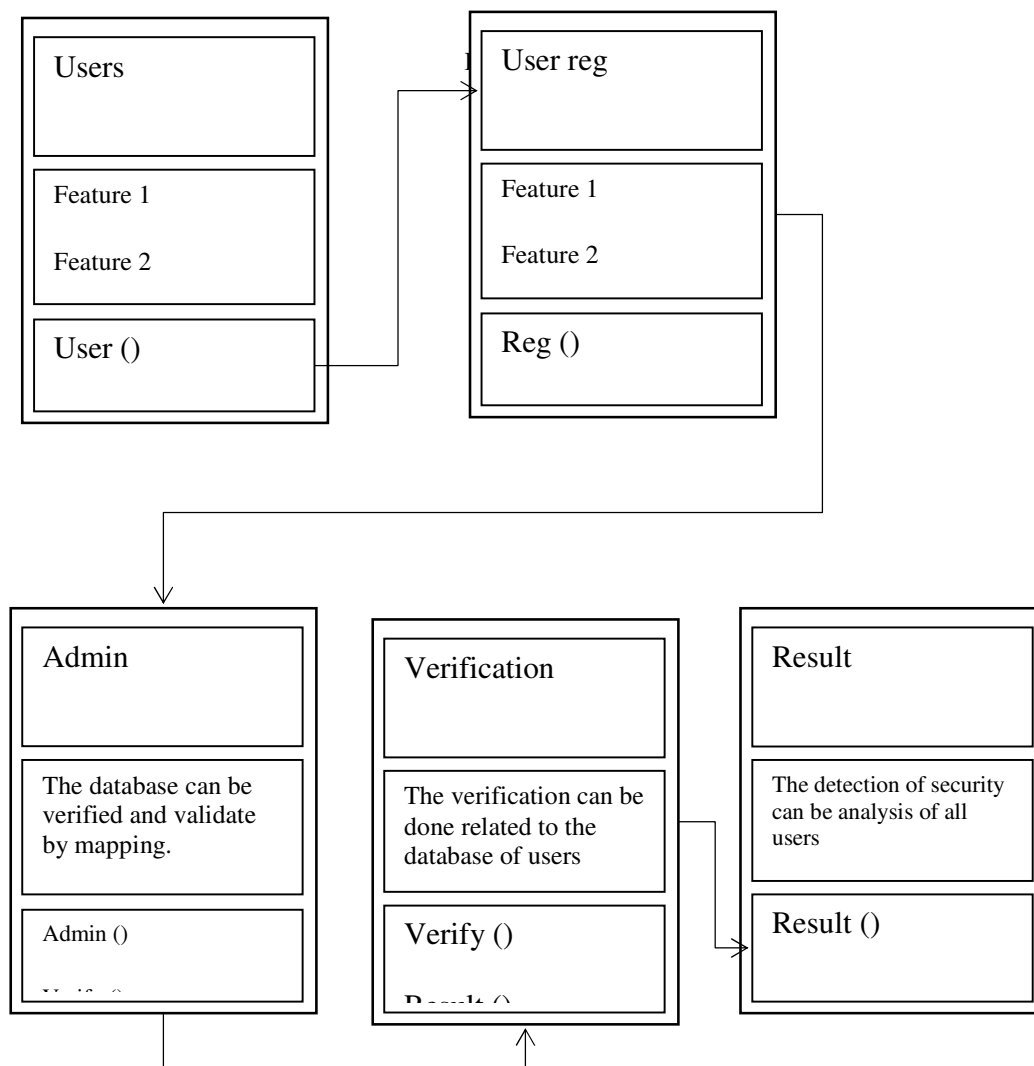


Fig 3.3 class diagram

### 3.4 Sequence diagram

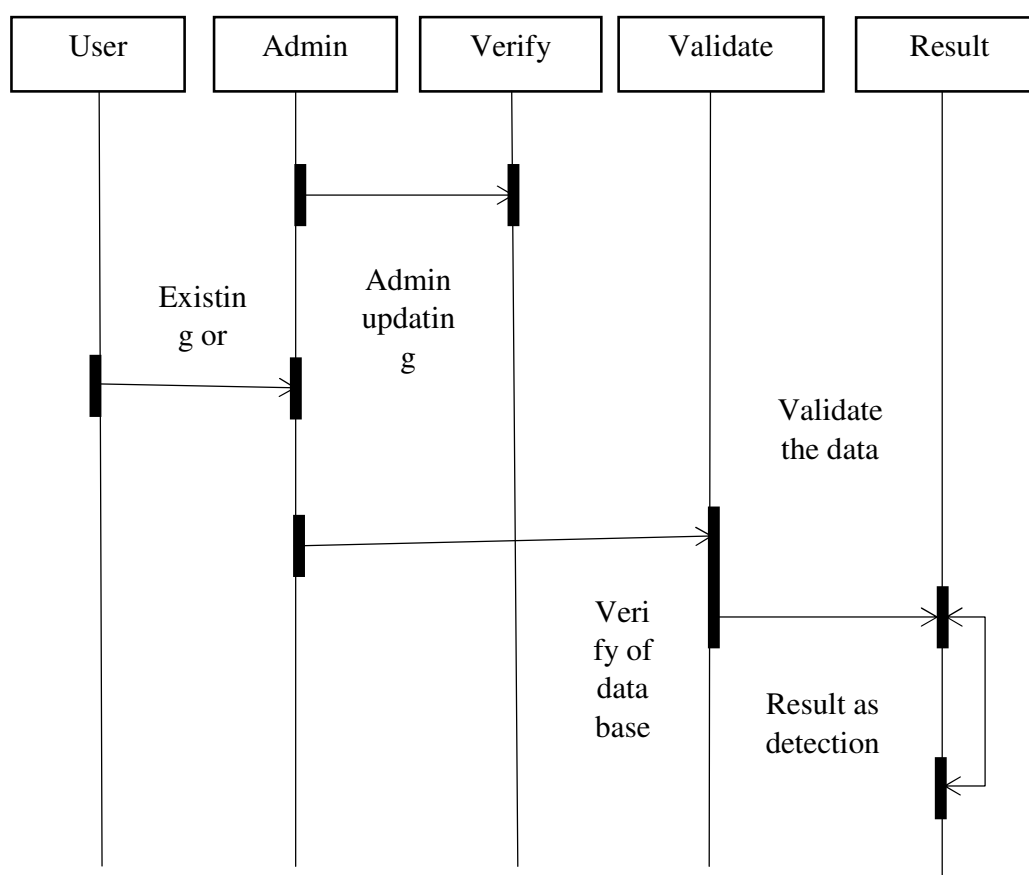


Fig 3.4 Sequence Diagram

#### **4. System Testing**

System testing is the stage of implementation, which aimed at ensuring that system works accurately and efficiently before the live operation commence. Testing is the process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an error. A successful test is one that answers a yet undiscovered error.

Testing is vital to the success of the system. System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. The candidate system is subject to variety of tests-on-line response, Volume Street, recovery and security and usability test. A series of tests are performed before the system is ready for the user acceptance testing. Any engineered product can be tested in one of the following ways. Knowing the specified function that a product has been designed to from, test can be conducted to demonstrate each function is fully operational. Knowing the internal working of a product, tests can be conducted to ensure that “all gears mesh”, that is the internal operation of the product performs according to the specification and all internal components have been adequately exercised.

#### **4.1 Types of Testing**

##### **4.1.1 Unit Testing**

Unit testing is the testing of each module and the integration of the overall system is done. Unit testing becomes verification efforts on the smallest unit of software design in the module. This is also known as ‘module testing’. The modules of the system are tested separately. This testing is carried out during the programming itself. In this testing step, each model is found to be working satisfactorily as regard to the expected output from the module. There are some validation checks for the fields. For example, the validation check is done for verifying the data given by the user where both format and validity of the data entered is included. It is very easy to find error and debug the system.

##### **4.1.2 Integration Testing**

Data can be lost across an interface, one module can have an adverse effect on the other sub function, when combined, may not produce the desired major function. Integrated testing is systematic testing that can be done with sample data. The need for the integrated test is to find the overall system performance. There are two types of integration testing. They are:

- i) Top-down integration testing.
- ii) Bottom-up integration testing.

##### **4.1.3 White Box Testing**

White Box testing is a test case design method that uses the control structure of the procedural design to drive cases. Using the white box testing methods, we derived test cases that guarantee that all independent paths within a module have been exercised at least once. 77

#### 4.1.4 Black Box Testing

- Black box testing is done to find incorrect or missing function
- Interface error
- Errors in external database access
- Performance errors
- Initialization and termination errors

In 'functional testing', is performed to validate an application conforms to its specifications of correctly performs all its required functions. So this testing is also called 'black box testing'. It tests the external behavior of the system. Here the engineered product can be tested knowing the specified function that a product has been designed to perform, tests can be conducted to demonstrate that each function is fully operational.

#### 4.1.5 Validation Testing

After the culmination of black box testing, software is completed assembly as a package, interfacing errors have been uncovered and corrected and final series of software validation tests begin validation testing can be defined as many, but a single definition is that validation succeeds when the

software functions in a manner that can be reasonably expected by the customer.

#### 4.1.6 User Acceptance Testing

User acceptance of the system is the key factor for the success of the system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system at the time of developing changes whenever required.

#### 4.1.7. Output Testing

After performing the validation testing, the next step is output asking the user about the format required testing of the proposed system, since no system could be useful if it does not produce the required output in the specific format. The output displayed or generated by the system under consideration. Here the output format is considered in two ways. One is screen and the other is printed format. The output format on the screen is found to be correct as the format was designed in the system phase according to the user needs. For the hard copy also output comes out as the specified requirements by the user. Hence the output testing does not result in any connection in the system.

### 5. System Implementation

Implementation of software refers to the final installation of the package in its real environment,



to the satisfaction of the intended users and the operation of the system. The people are not sure that the software is meant to make their job easier.

- The active user must be aware of the benefits of using the system
- Their confidence in the software built up
- Proper guidance is impaired to the user so that he is comfortable in using the application

Before going ahead and viewing the system, the user must know that for viewing the result, the server program should be running in the server. If the server object is not running on the server, the actual processes will not take place.

## 5.1 User Training

To achieve the objectives and benefits expected from the proposed system it is essential for the people who will be involved to be confident of their role in the new system. As system becomes more complex, the need for education and training is more and more important.

Education is complementary to training. It brings life to formal training by explaining the background to the resources for them. Education involves creating the right atmosphere and motivating user staff. Education information can

make training more interesting and more understandable.

## 5.2 Training on the Application Software

After providing the necessary basic training on the computer awareness, the users will have to be trained on the new application software. This will give the underlying philosophy of the use of the new system such as the screen flow, screen design, type of help on the screen, type of errors while entering the data, the corresponding validation check at each entry and the ways to correct the data entered. This training may be different across different user groups and across different levels of hierarchy.

## 5.3 Operational Documentation

Once the implementation plan is decided, it is essential that the user of the system is made familiar and comfortable with the environment. A documentation providing the whole operations of the system is being developed. Useful tips and guidance is given inside the application itself to the user. The system is developed user friendly so that the user can work the system from the tips given in the application itself.

## 5.4 System Maintenance

The maintenance phase of the software cycle is the time in which software performs useful work. After a system is successfully implemented, it should be maintained in a proper manner. System maintenance is an important aspect in the software development life cycle. The need for system maintenance is to make adaptable to the changes in the system environment. There may be social, technical and other environmental changes, which affect a system which is being implemented. Software product enhancements may involve providing new functional capabilities, improving user displays and mode of interaction, upgrading the performance characteristics of the system. So only thru proper system maintenance procedures, the system can be adapted to cope up with these changes. Software maintenance is of course, far more than “finding mistakes”.

#### 5.4.1 Corrective Maintenance

The first maintenance activity occurs because it is unreasonable to assume that software testing will uncover all latent errors in a large software system. During the use of any large program, errors will occur and be reported to the developer. The process that includes the diagnosis and correction of one or more errors is called Corrective Maintenance.

#### 5.4.2 Adaptive Maintenance

The second activity that contributes to a definition of maintenance occurs because of the rapid change that is encountered in every aspect of

computing. Therefore Adaptive maintenance termed as an activity that modifies software to properly interfere with a changing environment is both necessary and commonplace.

#### 5.4.3 Perceptive Maintenance

The third activity that may be applied to a definition of maintenance occurs when a software package is successful. As the software is used, recommendations for new capabilities, modifications to existing functions, and general enhancement are received from users. To satisfy requests in this category, Perceptive maintenance is performed. This activity accounts for the majority of all efforts expended on software maintenance.

#### 5.4.4 Preventive Maintenance

The fourth maintenance activity occurs when software is changed to improve future maintainability or reliability, or to provide a better basis for future enhancements. Often called preventive maintenance, this activity is characterized by reverse engineering and re-engineering techniques.

### 6 Modules

1. Login or location register
2. Registration and generation data
3. Security process
4. Validation

5. Verification

6. Analysis

## 6. Module description

### 6,1 Login Module

In this module the user can be verified related to the existing users or new users or the admin login process. Then the users can be enter the details for new users and then the location can be register related to the users data transmission process

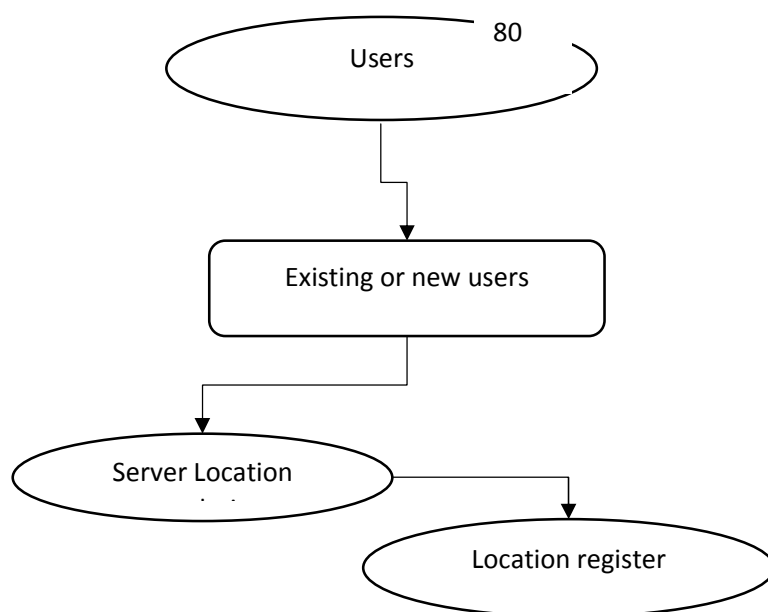


Fig 6.1 Login Module

### 6.1.2 Register and generation data

81

In this module the data can be analysed related to the users privileges and then the server can be provided and then the data transmission can be accessed related to user's authentication and access,

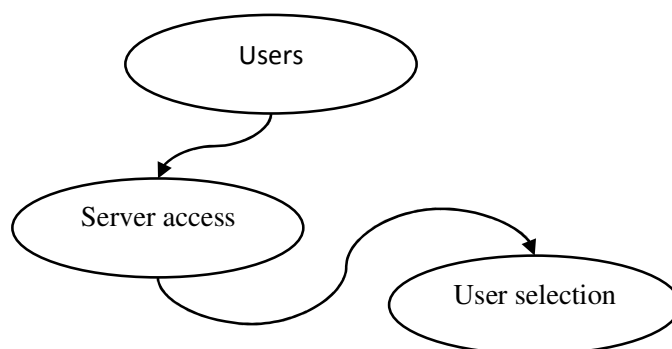


Fig 6.2 Register data

### 6.1.3 Validation

In this module the data can be analysed by the admin and then the data can be validate related to the users authentication.

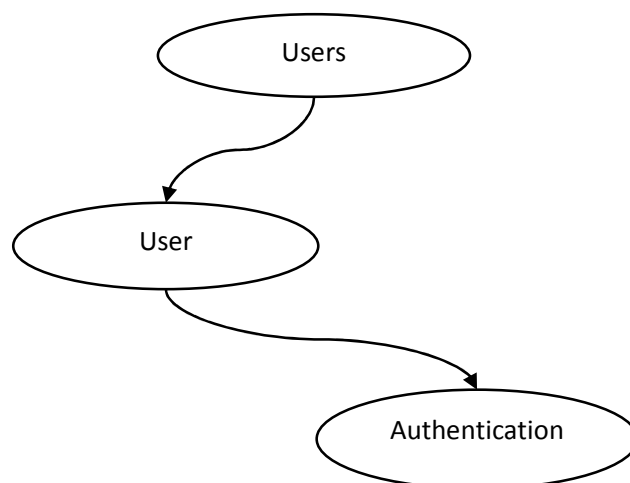


Fig 6.3 Validation

#### 6.1.4 Verification

82

In this module the data can be verified related to the users duplication process and then server can be verified related to the user's usage and then the data can be transmitted to the process.

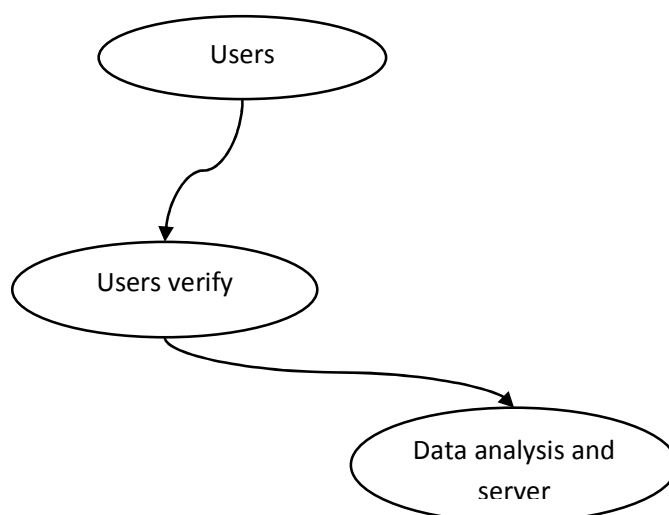
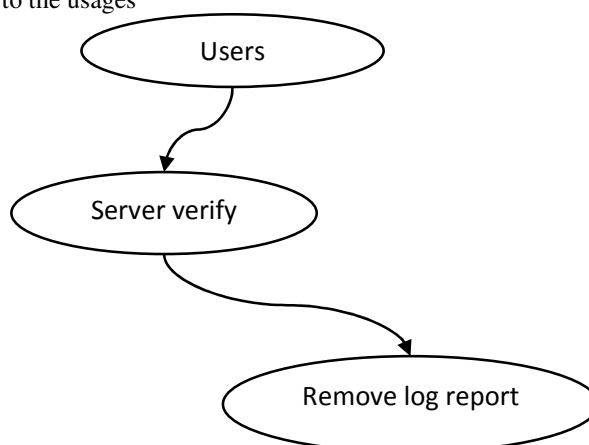


Fig 6.4 verification

#### 6.1.5 Analysis

In this module the data can be analysed and then the server log files can be removed related to the user's usage and the past log can be generated related to the usages



**Fig 6.5 Analysis**

## 7. SYSTEM REQUIREMENTS

83

### 7.1 Software Requirements

Operating System	:	Windows7
Technology	:	.ANDROID
Front End	:	JAVA
Back End	:	SQL LITE

### 7.2 Hardware Requirements

Processor	:	Any Processor above 500 MHz
RAM	:	512 MB
Hard Disk	:	1 GB

## 8. SOFTWARE DESCRIPTION

### Android

Android is a mobile operating system developed by Google. It is used by several

smartphones, such as the Motorola Droid, the Samsung Galaxy, and Google's own Nexus One. The Android operating system (OS) is based on the open

Linux kernel. Unlike the iPhone OS, Android is open source, meaning developers can modify and customize the OS for each phone. Therefore, different Android-based phones may have different graphical user interfaces GUIs even though they use the same OS.

The Android OS is designed for phones. Its many features include:

- Integrated browser, based on the open source Web Kit engine
- Optimized 2D and 3D graphics, multimedia and GSM connectivity
- Bluetooth
- EDGE
- 3G
- Wi-Fi
- SQLite
- Camera
- GPS
- Compass
- Accelerometer

### **Interface**

Android's default user interface is based on direct manipulation, using touch inputs, that correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate

on-screen objects, and a virtual keyboard. The response to user input is designed to be immediate and provides a fluid touch interface, often using the vibration capabilities of the device to provide haptic feedback to the user. Internal hardware such as accelerometers, gyroscopes and proximity sensors are used by some applications to respond to additional user actions, for example adjusting the screen from portrait to landscape depending on how the device is oriented, or allowing the user to steer a vehicle in a racing game by rotating the device, simulating control of a steering wheel.

Android devices boot to the home screen, the primary navigation and information "hub" on Android devices that is analogous to the desktop found on PCs (Android also runs on regular PCs, as described below). Android home screens are typically made up of app icons and widgets; app icons launch the associated app, whereas widgets display live, auto-updating content such as the weather forecast, the user's email inbox, or a news ticker directly on the home screen. A home screen may be made up of several pages that the user can swipe back and forth between, though Android's home screen interface is heavily customizable, allowing the user to adjust the look and feel of the device to their tastes. Third-party apps available on Google Play and other app stores can extensively re-theme the home screen, and even mimic the look of other operating systems, such as Windows Phone. Most manufacturers, and some wireless carriers, customize the look and feel of their Android devices to differentiate themselves from their competitors.

Present along the top of the screen is a status bar, showing information about the device and its connectivity. This status bar can be "pulled" down to reveal a notification screen where apps display important information or updates, such as a newly received email or SMS text, in a way that does not immediately interrupt or inconvenience the user. Notifications are persistent until read (by tapping, which opens the relevant app) or dismissed by sliding it off the screen. Beginning on Android 4.1, "expanded notifications" can display expanded details or additional functionality; for instance, a music player can display playback controls, and a "missed call" notification provides buttons for calling back or sending the caller an SMS message.

Android provides the ability to run applications which change the default launcher and hence the appearance and externally visible behavior of Android. These appearance changes include a multi-page dock or no dock, and many more changes to fundamental features of the user interface.

## Applications

Applications, which extend the functionality of devices, are written using the Android software development kit (SDK) and, often, the Java programming language that has complete access to the Android APIs. Java may be combined with C/C++, together with a choice of non-default runtimes that allow better C++ support; the programming language is also supported since version 1.4, which can also be used exclusively

although with a restricted set of Android APIs. Initially, Google's supported integrated development environment (IDE) was Eclipse using the Android Development Tools (ADT) plugin; in December 2014, Google released Android Studio, based on IntelliJ IDEA, as its primary IDE for Android application development. Android has a growing selection of third-party applications, which can be acquired by users by downloading and installing the application's APK (Android application package) file, or by downloading them using an application store program that allows users to install, update, and remove applications from their devices.

Google Play Store is the primary application store installed on Android devices that comply with Google's compatibility requirements and license the Google Mobile Services software. Google Play Store allows users to browse, download and update applications published by Google and third-party developers; As of July 2013, there are more than one million applications available for Android in Play Store. As of May 2013, 48 billion applications have been installed from Google Play Store and in July 2013, 50 billion applications were installed. Some carriers offer direct carrier billing for Google Play application purchases, where the cost of the application is added to the user's monthly bill.

Due to the open nature of Android, a number of third-party application marketplaces also exist for Android, either to provide a substitute for devices that are not allowed to ship with Google Play Store, provide applications that cannot be offered on Google Play Store due to policy violations, or for



other reasons. Examples of these third-party stores have included the Amazon Appstore, GetJar, and SlideMe. F-Droid, another alternative marketplace, seeks to only provide applications that are distributed under free and open source licenses.

### **Memory management**

Since Android devices are usually battery-powered, Android is designed to manage memory (RAM) to keep power consumption at a minimum, in contrast to desktop operating systems which generally assume they are connected to unlimited mains electricity. When an Android application is no longer in use, the system will automatically suspend it in memory; while the application is still technically "open", suspended applications consume no resources (for example, battery power or processing power) and sit idly in the background until needed again. This brings a dual benefit by increasing the general responsiveness of Android devices, since applications do not need to be closed and reopened from scratch each time, and by ensuring that background applications do not consume power needlessly.

Android manages the applications stored in memory automatically: when memory is low, the system will begin killing applications and processes that have been inactive for a while, in reverse order since they were last used (oldest first). This process is designed to be invisible to the user, so that users do not need to manage memory or the killing of applications themselves. As of 2011, third-party task killers were reported by Life hacker as doing more harm than good.

86

### **Hardware**

The main hardware platform for Android is the ARM architecture (ARMv7 and ARMv8-A architectures), with x86 and MIPS architectures also officially supported. Since Android 5.0 "Lollipop", 64-bit variants of all platforms are supported in addition to the 32-bit variants. Unofficial Android-x86 project used to provide support for the x86 and MIPS architectures ahead of the official support. Since 2012, Android devices with Intel processors began to appear, including phones and tablets. While gaining support for 64-bit platforms, Android was first made to run on 64-bit x86 and then on ARM64. Android devices incorporate many optional hardware components, including still or video cameras, GPS, orientation sensors, dedicated gaming controls, accelerometers, gyroscopes, barometers, magnetometers, proximity sensors, pressure sensors, thermometers, and touchscreens. Android was developed initially as a phone OS, hardware such as microphones were required, while over time the phone function became optional. Android used to require an autofocus camera, which was relaxed to a fixed-focus camera if it is even present at all, since the camera was dropped as a requirement entirely when Android started to be used on set-top boxes. In addition to running on smartphones and tablets, several vendors run Android natively on regular PC hardware with a keyboard and mouse. Chinese companies are building a PC and mobile operating system, based on Android, to compete directly with Microsoft Windows and Google Android.

### **Development**

Android is developed in private by Google until the latest changes and updates are ready to be released, at which point the source code is made available publicly. Android's source code does not contain the often proprietary device drivers that are needed for certain hardware components. The green Android logo was designed for Google in 2007 by graphic designer Irina Blok. The design team was tasked with a project to create a universally identifiable icon with the specific inclusion of a robot in the final design. After numerous design developments based on science-fiction and space movies, the team eventually sought inspiration from the human symbol on restroom doors and modified the figure into a robot shape. As Android is open-sourced, it was agreed that the logo should be likewise, and since its launch the green logo has been reinterpreted into countless variations on the original design.

### Licensing

The source code for Android is open source; it is developed in private by Google, with the source code released publicly when a new version of Android is released. Google publishes most of the code (including network and telephony stacks) under the non-copy left Apache License version 2.0, which allows modification and redistribution. Only the base Android operating system (including some applications) is open-source software, whereas most Android devices ship with a substantial amount of proprietary software, such as Google Mobile Services, which includes applications such as Google Play Store, Google Search, and

Google Play Services—a software layer that provides APIs for the integration with Google-provided services, among others.

These applications must be licensed from Google by device makers, and can only be shipped on devices which meet its compatibility guidelines and other requirements. Although an update for Google Search app containing the relevant components was released through Google Play for all Android devices, the new home screen required an additional stub application to function, and was not provided in Android 4.4 updates for any other devices (which still used the existing home screen from Android version 4.3). The stub application was officially released on Play Store as Google Now Launcher in February 2014, initially for Nexus and Google Play Edition devices with Android version 4.4.

### Security and Privacy

Android applications run in a sandbox, an isolated area of the system that does not have access to the rest of the system's resources, unless access permissions are explicitly granted by the user when the application is installed. Before installing an application, Play Store displays all required permissions: a game may need to enable vibration or save data to an SD card. After reviewing these permissions, the user can choose to accept or refuse them, installing the application only if they accept. The "App Ops" privacy and application permissions control system, used for internal development and testing by Google, was introduced in Google's Android 4.3 release for the Nexus devices. Initially

hidden, the feature was discovered publicly; it allowed users to install a management application and approve or deny permission requests individually for each of the applications installed on a device.

## FEASIBILITY STUDY

The feasibility study is carried out to test whether the proposed system is worth being implemented. The proposed system will be selected if it is best enough in meeting the performance requirements.

The feasibility carried out mainly in three sections namely.

- Economic Feasibility
- Technical Feasibility
- Behavioral Feasibility

### Economic Feasibility

Economic analysis is the most frequently used method for evaluating effectiveness of the proposed system. More commonly known as cost benefit analysis. This procedure determines the benefits and saving that are expected from the system of the proposed system. The hardware in system department if sufficient for system development.

### Technical Feasibility

This study center around the system's department hardware, software and to what extend it can support the proposed system department is

having the required hardware and software there is no question of increasing the cost of implementing the proposed system. The criteria, the proposed system is technically feasible and the proposed system can be developed with the existing facility.

### Behavioral Feasibility

People are inherently resistant to change and need sufficient amount of training, which would result in lot of expenditure for the organization. The proposed system can generate reports with day-to-day information immediately at the user's request, instead of getting a report, which doesn't contain much detail.

### System Implementation

Implementation of software refers to the final installation of the package in its real environment, to the satisfaction of the intended users and the operation of the system. The people are not sure that the software is meant to make their job easier.

- The active user must be aware of the benefits of using the system
- Their confidence in the software built up
- Proper guidance is impaired to the user so that he is comfortable in using the application

Before going ahead and viewing the system, the user must know that for viewing the result, the server program should be running in the server. If the server object is not running on the server, the actual processes will not take place.

### **User Training**

To achieve the objectives and benefits expected from the proposed system it is essential for the people who will be involved to be confident of their role in the new system. As system becomes more complex, the need for education and training is more and more important. Education is complementary to training. It brings life to formal training by explaining the background to the resources for them. Education involves creating the right atmosphere and motivating user staff. Education information can make training more interesting and more understandable.

### **Training on the Application Software**

After providing the necessary basic training on the computer awareness, the users will have to be trained on the new application software. This will give the underlying philosophy of the use of the new system such as the screen flow, screen design, type of help on the screen, type of errors while entering the data, the corresponding validation check at each entry and the ways to correct the data entered. This training may be different across different user groups and across different levels of hierarchy.

### **Operational Documentation**

Once the implementation plan is decided, it is essential that the user of the system is made familiar and comfortable with the environment. A documentation providing the whole operation of the system is being developed. Useful tips and guidelines are given inside the application itself to the user. The

system is developed user friendly so that the user can work the system from the tips given in the application itself.

### **System Maintenance**

The maintenance phase of the software cycle is the time in which software performs useful work. After a system is successfully implemented, it should be maintained in a proper manner. System maintenance is an important aspect in the software development life cycle. The need for system maintenance is to make adaptable to the changes in the system environment. There may be social, technical and other environmental changes, which affect a system which is being implemented. Software product enhancements may involve providing new functional capabilities, improving user displays and mode of interaction, upgrading the performance characteristics of the system. So only through proper system maintenance procedures, the system can be adapted to cope up with these changes. Software maintenance is of course, far more than "finding mistakes".

### **Corrective Maintenance**

The first maintenance activity occurs because it is unreasonable to assume that software testing will uncover all latent errors in a large software system. During the use of any large program, errors will occur and be reported to the developer. The process that includes the diagnosis and correction of one or more errors is called Corrective Maintenance.

### **Adaptive Maintenance**

The second activity that contributes to a definition of maintenance occurs because of the rapid change that is encountered in every aspect of computing. Therefore Adaptive maintenance termed as an activity that modifies software to properly interfere with a changing environment is both necessary and commonplace.

### Perceptive Maintenance

The third activity that may be applied to a definition of maintenance occurs when a software package is successful. As the software is used, recommendations for new capabilities, modifications to existing functions, and general enhancement are received from users. To satisfy requests in this category, Perceptive maintenance is performed. This activity accounts for the majority of all efforts expended on software maintenance.

### Preventive Maintenance

The fourth maintenance activity occurs when software is changed to improve future maintainability or reliability, or to provide a better basis for future enhancements. Often called preventive maintenance, this activity is characterized by reverse engineering and re-engineering techniques

### 9. Conclusion

We have presented STAMP, which aims at providing security and privacy assurance for mobile users' proofs for their past location. STAMP relies on mobile devices in vicinity to

mutually generate location proofs or uses wireless APs to generate location proofs. Integrity and non-transferability of location proofs and location privacy of users are the main design goals of STAMP.

### 10. References

- [1] S. Saroiu and A. Wolman, "Enabling new mobile applications with location proofs," in Proc. ACM HotMobile, 2009, Art.no.3.
- [2] W. Luo and U. Hengartner, "VeriPlace: A privacy-aware location proof architecture," in Proc. ACM GIS, 2010, pp. 23–32.
- [3] Z. Zhu and G. Cao, "Towards privacy-preserving and colluding-resistance in location proof updating system," IEEE Trans. Mobile Comput. vol. 12, no. 1, pp. 51–64, Jan. 2011.
- [4] N. Sastry, U. Shankar, and D. Wagner, "Secure verification of location claims," in Proc. ACM WiSe, 2003, pp. 1–10.
- [5] R. Hasan and R. Burns, "Where have you been? Secure location proof for mobile devices," CoRR 2011.
- [6] B. Davis, H. Chen, and M. Franklin, "Privacy preserving alibi systems," in Proc. ACM ASIACCS, 2012, pp. 34–35.

- [7] I. Krontiris, F. Freiling, and T. Dimitriou, "Location privacy in urban Sensing networks: Research challenges and directions," *IEEE Wireless Commun.*, vol. 17, no. 5, pp. 30–35, Oct. 2010.
- [8] Y. Desmedt, "Major security problem with the 'unforgeable' (feige)-fiat-shamir proofs of identity and how to overcome them," in *Proc. SecuriCom*, 1988, pp. 15–17.
- [9] 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*. New York, NY, USA: Springer, 2005.
- [10] X. Wang et al., "STAMP: Ad hoc spatial-temporal provenance assurance for mobile users," in *Proc. IEEE ICNP*, 2013, pp. 1–10.
- [12] A. Pfitzmann and M. Köhnert, "Anonymity, unobservability, and pseudonymity—a proposal for terminology," in *Designing Privacy Enhancing Technologies*. New York, NY, USA: Springer, 2001.
- [13] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Wormhole attacks in wireless networks," *IEEE J. Areas Commun.*, vol. 24, no. 2, pp. 370–380, Feb. 2006.
- [14] S. Halevi and S. Micali, "Practical and provably-secure commitment schemes from collision-free hashing," in *Proc. CRYPTO*, 1996, pp. 201–215.
- [15] I. Damgård, "Commitment schemes and zero-knowledge protocols," in *Proc. Lectures Data Security*, 1999, pp. 63–86.
- [16] I. Haitner and O. Reingold, "Statistically-hiding commitment from any one-way function," in *Proc. ACM Symp. Theory Comput.*, 2007, pp. 1–10.
- [17] D. Singelee and B. Preneel, "Location verification using secure distance bounding protocols," in *Proc. IEEE MASS*, 2005.
- [18] J. Reid, J. Nieto, T. Tang, and B. Senadji, "Detecting relay attacks with the timing-based protocols," in *Proc. ACM ASIACCS*, 2007, pp. 204–213.
- [19] C. Kim, G. Avoine, F. Koeune, F. Standaert, and O. Pereira, "The Swiss-knife RFID distance bounding protocol," in *Proc. ICISC*, 2009, pp. 98–115.
- [20] H. Han et al., "Senspeed: Sensing driving conditions to estimate vehicles speed in urban environments," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 727–735.

- |   |   |
|---|---|
| <p>[21] I. Afyouni, C. Ray, and C. Claramunt, "Spatial models for context-aware indoor navigation systems: A survey," <i>J. Spatial Inf. Sci.</i>, no. 4, pp. 85–123, 2014.</p> | <p>[22] N. Roy, H. Wang, and R. R. Choudhury, "I am a smartphone and I can tell my user's walking direction," in <i>Proc. ACM MobiSys</i>, 2014, pp. 329–342.</p> |
|---|---|