LOCATION DATA SIGNING – PROTECTING THE INTEGRITY AND AUTHENTICITY OF POSITIONING SYSTEM DATA

Marcy E. Gordon, Sean J. Barbeau, Miguel A. Labrador

Center for Urban Transportation Research
and Department of Computer Science and Engineering
University of South Florida
4202 E. Fowler Ave., Tampa, FL 33620 USA
{megordon, barbeau}@cutr.usf.edu
{labrador}@cse.usf.edu

ABSTRACT

Ensuring the integrity and authenticity of location data is becoming increasingly important as businesses and governments use Global Navigation Satellite System data in many new domains such as pay-as-you-drive insurance, variable transportation taxes, logistics auditing, fleet tracking, and IntelliDrive applications. Location data signing is the technique of producing a digital signature for data obtained from a positioning system, such as the U.S. NAVSTAR Global Positioning System (GPS), in order to prove a mobile device was at a particular location at a given date and time. This paper describes the design and implementation of location data signing within a software system and investigates the overhead, trade-offs, and challenges encountered in a proof-of-concept mobile application for GPS-enabled cell phones. Results from experiments demonstrate that recent advances in mobile device technology now allow for computationally-intensive cryptographic algorithms to execute in a reasonable amount of time and with little noticeable overhead to the user. The feasiblity of location data signing on mobile devices therefore paves the way for secure uses of GPS data in the transportation domain.

Keywords: location data, integrity, digital signatures, Android, location-based services, mobile phones, GPS, GNSS

1. INTRODUCTION

As the use of location-based services (LBS) gathers popularity with mobile devices, the need arises to prove the validity of the data. Ensuring the integrity and authenticity of location data is becoming increasingly important in the field of transportation as Global Navigation Satellite System data are being used by business and government in many new domains such as pay-as-you-drive insurance, variable transportation taxes, IntelliDrive applications, maritime and military operations, logistics auditing, and fleet tracking. However, these applications that support critical business or government functions raise important legal questions: can location data, such as Global Positioning Systems (GPS) data, truly determine the historic or real-time location of an individual? In fact, GPS data is already being used in legal proceedings to decide whether an individual was at a particular place at a particular time (1). However, GPS data is subject to the same limitations of any other type of information: if not properly protected, GPS data can be altered by malicious users towards their own purpose. These limitations therefore required a solution that will ensure the integrity of the GPS data collected so that whatever is being tracked can reliably be shown to be at a particular location at a particular time.

To ensure the integrity of a physical document, one would simple sign the document with a handwritten signature. For digital documents and messages, the integrity is now proven by digital signatures, a technique involving hashing algorithms and public-key cryptography. Location data signing takes this idea one step further by integrating digital signatures into LBS in order to ensure the integrity of location data. "Location data" refers to information obtained from a positioning system, such as GPS, including latitude, longitude, altitude, speed, location or operating system time-stamp, and other information that uniquely identifies a device or user (e.g., Internaional Mobile Equipment Identify (IMEI) number, electronic serial number (ESN), phone number, user ID, etc). "Location data signing" is the technique of producing a digital signature to ensure the integrity and authenticity of the location data. It is critical that the process of location data signing guarantees that the data are reliable, verifiable, and unaltered. Location data signing must create a digital signature of the data as it is collected and provide a way to later ensure that current data stored is exactly the same as the original data generated by the positioning system on the mobile device.

This paper describes the design and implementation of location data signing within a software system and investigates the overhead, trade-offs, and challenges encountered in a proof-of-concept mobile application for GPS-enabled cell phones. TRAC-IT for the Google Android platform, an existing mobile application for GPS-enabled mobile phones used for travel behavior data collection and real-time personalized travel information, was utilzed as a host application for the location data signing module evaluated in this paper (2).

Although this research focuses on ensuring the integrity of location data collected from GPS-enabled mobile phones, the ideas and lessons learned from signing location data can be extended to ensuring the integrity of data extracted by many other sensing systems. For example, in a system that monitors air pollution, entities may try to fake data to avoid penalty fees for high pollution emissions. With an incentive to alter the data, it becomes important to implement a mechanism for data integrity before data leaves the device on which it is collected.

Because the implementation of location data signing evaluated in this paper will run on a mobile device, we first determine the most efficient algorithms and techniques for hashing and encrypting/decrypting (the two techniques used in digital signatures). We must also evaluate the overhead created from the data signing process, i.e., CPU load, battery life, communication load, and database storage size. Another challenge will be to determine the frequency of signing that would be useful to prove a user was at a particular place at a particular time, but without incurring unacceptable overhead on a mobile device.

The rest of this paper is orgranized as follows: Section 2 provides background on digital signatures and the TRAC-IT system. Section 3 describes the methodology to implement location data signing. Experimental results are presented in Section 4, and concluding remarks are included in Section 5.

2. BACKGROUND

Digital Signatures

Digital signatures are a mathematical method for showing the authenticity, integrity, and non-repudiation of a digital message, akin to signing a signature on a physical document. In a digital signature system, there are three phases: key generation, signing, and verification. Each

algorithm that implements the digital signature technique uses different methods and parameters for those three phases.

RSA (Rivest-Shamir-Adleman) is an algorithm used for encryption and digital signatures. It is based on the big integer factorization problem so its security lies in the complexity of solving the big integer factorization problem quickly (11). RSA has restricted use outside of US, so it may not be the best candidate to consider for including in our tracking system, but it is very popular and well-supported by most cryptography software.

DSA (Digital Signature Algorithm) is used only for signatures, not for encryption. It is based on the discrete logarithm problem (DLP) (11). DSA was proposed by NIST (National Institute of Standards and Technology) in 1991 and is patented but royalty-free worldwide.

ECDSA (Elliptic Curve Digital Signature Algorithm) is an improvement upon DSA and is based on the elliptic curve discrete logarithm problem (ECDLP) (11). Sub-exponential time algorithms are known to solve the factorization problem and DLP, but only exponential time algorithms are known to solve ECDLP (10). This makes ECDSA inherently more secure than RSA and DSA. Algorithms based on elliptic curve cartography (ECC) can provide the same security as RSA and DSA with a key size of 160-bits for the ECC key and a key size of 1024-bits for the RSA/DSA key (15). This means that ECC provides more security per bit than RSA, DSA, and other public-key cryptography methods (6). The patent holder, Certicom Corp., claims that this ability is what allows ECC-based techniques to be computed faster and use less memory, bandwidth, and power (6).

Table 1 demonstrates that smaller ECC key sizes have comparable security to larger RSA or DSA keys.

Decipher time (MIPS Years)	RSA/DSA Key Size (bits)	ECC Key Size (bits)	RSA Key Size : ECC Key Size
10^4	512	102	5:1
10^8	768	136	6:1
10^{11}	1024	160	7:1
10^{20}	2048	210	10:1
10^{78}	21000	600	35:1

Table 1: Relationship between key sizes and security (12)

Each digital signature algorithm usually requires that the message be hashed. The most widely used hashing algorithms are MD5, SHA-1 and SHA-2. MD5 has an output size of 128-bits. While still in popular use, MD5 has been broken and was then used to break SSL in 2008. SHA-1 outputs a hash of 160-bits. 2 collision attacks for SHA-1 were found in 2005 in 263 hashing operations, but it is still in common use. SHA-2 is a family of hashing algorithms that are more secure than SHA-1. SHA-256 and SHA-512 are members of the SHA-2 family and they output 256 and 512-bits respectively for the hash sizes. No attacks are currently known for SHA-256 and SHA-512.

Previous Work

Xuan et. al. compared RSA, DSA, and ECDSA performance on J2ME emulators (15),

specifically the Sun (now Oracle) Java Wireless Toolkit (WTK) version 2.5.2. For their implementation, they imported the BouncyCastle crypto package and used SHA-1 as the hashing algorithm for all tests. Their results are summarized in Table 2. Conclusions from this study recommend that a 1024-bit RSA scheme be used when the mobile device is required to verify the signature and a 160-bit ECDSA scheme when the mobile device needs to sign the data. Results for 2048-bit RSA/DSA keys and 210-bit ECC keys were not included as the running times were deemed impractical for mobile device usage.

Algorithm	Key Sizes (bits)	Key Generation (ms)	Signature Generation (ms)	Verification (ms)
RSA	768	609755	12103	1146
	1024	1262500	25755	1950
DSA	768	3307048	9751	19922
	1024	5869522	15171	33602
ECDSA	132	70702	70722	91811
	160	85149	91801	114453

Table 2: Experimental results on J2ME (12)

Jarusombat et. al. demonstrated in 2006 that mobile devices are not suitable for using traditional digital signature algorithms and proceed to introduce their geo-encryption system, which involves the use of GPS coordinates for encrypting and decrypting data (9). In this paper we demonstrate that mobile phones have now become powerful enough to execute traditional secure digital signature algorithms with a reasonable amount of overhead.

TRAC-IT

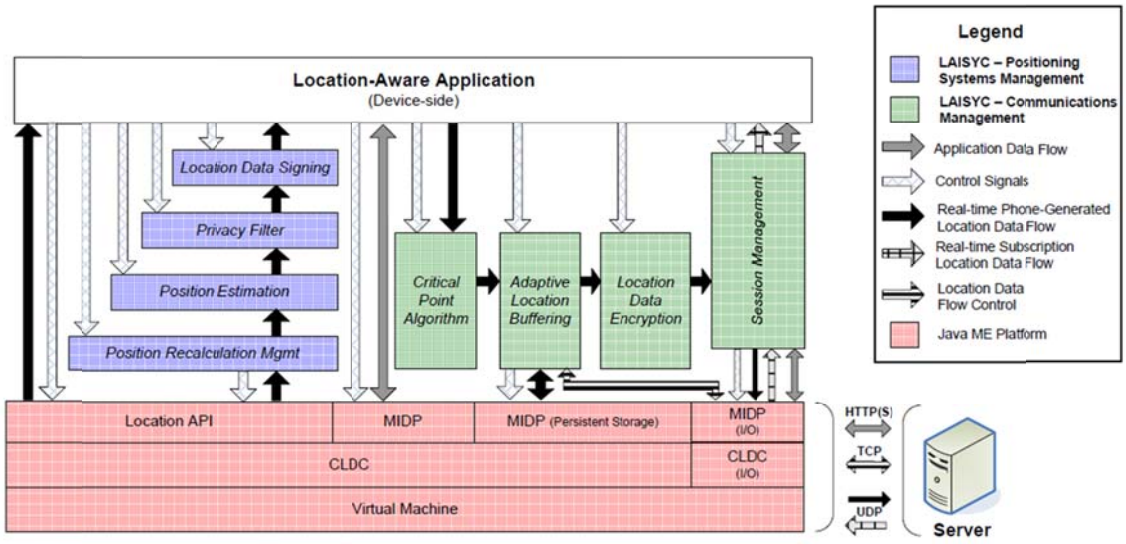


Figure 1: The LAISYC framework with a module for location data signing (2)

The LAISYC (Location-Aware Information Systems Client) framework is a modular software framework for intelligent, location-aware, real-time mobile applications (3). Figure 1 describes

the data flow though various modules for security, privacy, integrity, scalability, and energy intelligence of location-aware mobile applications. The LAISYC framework has been the foundation for many applications such as TRAC-IT, Travel Assistance Device real-time navigation software for public transportation, and TACLAN Tactical LBS system for real-time battlefield tracking and messaging between mobile devices and centralized dispatch station (5). TRAC-IT is a software system designed to track the location of GPS-enable mobile phones. TRAC-IT is used to study travel behavior and serve as personal travel coach that helps users reduce their "travel footprint" by showing their travel history and providing travel suggestions. TRAC-IT also provides real-time traffic alerts through personalized path prediction. This paper focuses on an implementation and evaluation of the Location Data Signing module of the LAISYC framework, used to specifically support the TRAC-IT mobile application.

3. METHODOLOGY

The implementation of a location data signing mechanism consists of three phases: signing the data, verifying the signatures, and evaluating design decisions while considering the effects on the system as a whole.

Digitally Signing the Location Data

The first phase consists of implementing code in the Android version of TRAC-IT that hashes the GPS data at certain intervals and then encrypts it using public/private key cryptography. The encrypted hash will be then sent to the server in a User Datagram Protocol (UDP) packet to be stored in a database.

This will require TRAC-IT's Android client to be modified for key pair generation, data signing, and transmitting the extra data and public key. TRAC-IT's web services will also need to be modified for receiving and storing public keys and the location data signatures.

Detailed steps for signing the data are as follows:

1. Compute unique hash of location data using an algorithm such as SHA-1 so that $h = SHA1(data_{phone})$
2. Encrypt the hash using private key so $e = E(h, priv_{key})$
3. Add encrypted hash (the signature) to the UDP payload and send to the server along with the location data that was used to generate the hash
4. Server stores the encrypted hash into a database field along with location data

Verifying the Integrity of the Location Data

This phase requires the creation of a procedure to verify the data based on the signature for that data. When someone desires to verify the integrity of the GPS data, the encrypted hash will be decrypted with a public key and another hash will be created of the currently stored GPS data. If the two hashes match, then the integrity of the data has been preserved.

The implementation of this phase will require the create of a new TRAC-IT desktop application for data verification.

The steps for data verification are as follows:

1. Create a hash of the location data in the database so $h_c = SHA1(data_{database})$

2. Decrypt the hash stored in DB with a public key so that $h_d = D(h_c, pub_{key})$
3. If $h_c = h_d$, then the location data is valid
4. Else the data is not valid

Design Decisions

We need to determine how often to hash and sign the data without slowing down the CPU, which of course is dependent on device's CPU speed (as well as Java virtual machine speed). The impact of extra computations on battery life needs to be investigated as well. The frequency of signing could occur at every location fix, at a fixed interval, or only at critical points. The Critical Point (CP) algorithm reduces data set by 66% (dependent on travel pattern) and is included in a module in the LAISYC framework (3)(4). The goal of the CP algorithm is to reduce the amount of data sent to the server while preserving the information describing a user's path in order to preserve battery life. Using this approach, only the important fixes required to reconstruct a user's path are signed. The disadvantage here is that we won't be able to prove the validity of non-CP fixes.

Communication overhead is a concern as an increased UDP payload will decrease the amount of UDP packets successfully received by the server due to network issues and wireless interference that may increase the probability of packet corruption as the packet size increases. Location data currently being sent via UDP in TRAC-IT. If assurance of packet receipt is required, we may consider sending signatures via TCP. Public keys, which are vital to the verification phase, will be sent via TCP.

The TRAC-IT database needs to be modified to store the location data signature (per fix or for some other interval) as well as the public key for each user or phone. LBS databases can grow quickly so it is important to observe the impact of location data signing on the database. Investigation showed that for RSA-512, the signature is 46 or 47 bytes, the public key is 243 to 245 bytes, and the private key is 201 to 202 bytes.

Private key security is another topic to consider. If we opt for one time key pair generation, we need to keep the private key stored on the mobile device and only allow the TRAC-IT application to extract the key. The Android platform assigns each application their own user and a protected directory for file storage that can only be read by that application: /data/data/<package-name>/. Of course, a root user is able to read files in this directory very easily so this is not a very secure approach. An alternative may be to use a Java KeyStore (JKS), which can optionally be password protected. For even more security, we could consider storing this password on TRAC-IT's web server and sending it via HTTPS after the TRAC-IT user has been authenticated.

An important design decision is the issue of key pair regeneration. Private keys may become compromised, which will require the regeneration of the private-public key pair. The keys could be regenerated every session of the application or after a certain time interval. There is also the possibility to allow the user or administrator of the system to manually request the regeneration. Key generation typically takes longer than signature generation and so we want to generate keys as infrequently as possible. If the system allows for regeneration of key pairs, then there will also be a need to track a history of past public keys in the database in order to verify signatures that used old keys.

Another issue to be mindful of is verification of public keys. If public keys are stored in the database, either per user/device or per session, we will need to assume the database can be

trusted. That is, it should be assumed that the database administrators can be trusted or that safegaurds will be in place to prevent data from being tampered with while in storage. Possible safegaurds can include revoking privilleges, triggers to prevent updates, or database vaults. If we cannot trust the database, it would be possible for a malicious database administrator to alter location data, generate a new key pair, resign the location data, and then update then new signatures and public key, effectively defeating our data signing scheme.

4. RESULTS

Experimentation and Investigation

The Android API, just like Oracle's Java API, has two main packages for dealing with security protocols: java.security and javax.crypto. The Javadocs for the Android API specify that three standards are supported in the java.security.spec package:

- PKCS#1 RSA encryption standard
- FIPS-186 DSA signature standard
- PKCS#8 private key information standard

As for support for ECC-related algorithms, the documentation states "[t]he parameters for the Elliptic Curve (EC) encryption algorithm are only specified as input parameters to the relevant EC-generator" (8). To the authors knowledge, ECDSA is currently unavailable on the Android platform, perhaps due to intellectual property constraints. It may be possible to manually port the ECDSA implementation found in other versions of the BouncyCastle package. Further investigation found that the intellectual property for ECC-based security was licensed in 2004 by the patent holder, Certicom Corp., to the National Security Agency (NSA) for government use and Research in Motion (RIM) for their Blackberry devices (6). In 2009, Certicom became a subsidiary of RIM.

Due to restricted use of some algorithms, not all algorithms can be used on all Android devices. For example, RSA can only be used in the US. If the phone is designed to be used in and out of the US, the manufacturer may (hypothetically) opt out of RSA support. In order to determine which cryptographic methods are supported by the test device, a T-mobile G1, a small separate Android application was written.

Security providers are Java packages that contain algorithms and objects designed to implement security protocols such as ciphers, key agreements, message digests, and signatures. In order to determine which security providers were available on the G1 device, a piece of code was written to list all the security providers and their descriptions. The code determined that the Android 1.6 platform has four security providers (with their descriptions) by default:

- **DRLCertFactory** - ASN.1, DER, PkiPath, PKCS7
- **Crypto** - HARMONY (SHA1 digest; SecureRandom; SHA1withDSA signature)
- **HarmonyJSSE** - Harmony JSSE Provider
- **BC** - BouncyCastle Security Provider v1.34

Then functionality was added to the code to determine which signature algorithms were supported by each of the four security providers. The results are shown in Table 3.

Table 3: Listing of signature algorithms supported by different security providers in Android 1.6

Security Provider	Signature Algorithms Supported
DRLCertFactory	None
Crypto	SHA1withDSA
	SHA1withDSA ImplementedIn
HarmonyJSSE	None
BC	MD5withRSA/ISO9796-2
	RSASSA-PSS
	SHA256withRSA/PSS
	SHA1WithRSAEncryption
	DSA
	MD4WithRSAEncryption
	SHA384withRSA/PSS
	NONEWITHDSA
	SHA224withRSA/PSS
	SHA224WithRSAEncryption
	SHA1withRSA/ISO9796-2
	SHA512WithRSAEncryption
	MD5WithRSAEncryption
	SHA512withRSA/PSS
	SHA256WithRSAEncryption
	SHA384WithRSAEncryption
	SHA1withRSA/PSS

After reviewing the results of Xuan et. al., where it was shown that a J2ME emulator may take up to 90 minutes to generate a key pair, code was written to test out the DSA algorithm on Android in order to determine how feasible it would to contantly run the algorithm (15). Using a 512-bit key size and the message "abc", the results are shown in Table 4 for two emulator images and a physical device.

All of the results described below were generated on a Google Android G1 phone running Android 1.6. Part of the runtimes may include time spent by the garbage collector or other OS processes. Future experiments may want to use the tracing feature of the Android SDK in order to provide a more accurate assessment of the runtimes. For our purposes, the runtimes are sufficient as they reflect general performance on the device and provide a basic insight to the computational intensities of the algorithms tested. A screen-shot with test results on the actual device is depicted in Figure 2. In all experiments, the message signed was a simple string ``abc''. Future work may want to investigate if different setups perform differently on varying message sizes. For each signature generated, a verification was also performed to ensure the correct signature generation. The runtime results of the verification steps are not provided here as our system will provide the verification module on the server-side, so mobile device performance for verification is not of concern in our case.

	Emulator	Emulator	G1
Runtime	Android 1.6	Android 2.1	Android 1.6
Key Generation (ms)	5669	7929	1079
Signing (ms)	42	26	15
Verification (ms)	63	33	11

Table 4: Initial runtime results with DSA and a key size of 512-bits

th various

Our first experiment tests the runtimes of six signature algorithms from the BouncyCastle (BC) provider and one algorithm from the Crypto provider, in particular SHA-1 with DSA, DSA, MD5 with RSA, SHA-1 with RSA, SHA-256 with RSA, and SHA-512 with RSA all from the BC provider and SHA-1 with DSA from the Crypto provider. The results for three key sizes (512, 1024, and 2048-bits) are shown for each algorithm in Figure 3. The runtimes are shown in milliseconds on a logarithmic scale. It should be noted that the test was run once and the algorithms were run in succession (this implies that the garbage collector may have been particularly busy clearing space for large key generations). The algorithm passed into the key generator is independent of the version of the implementation of the algorithm. This means that the key generation process for MD5 with RSA is the same as the key generation process for SHA-1 with RSA. This can be seen in the results as the first three algorithms based on DSA have similar runtimes. The same goes for the last four algorithms that are based on RSA.

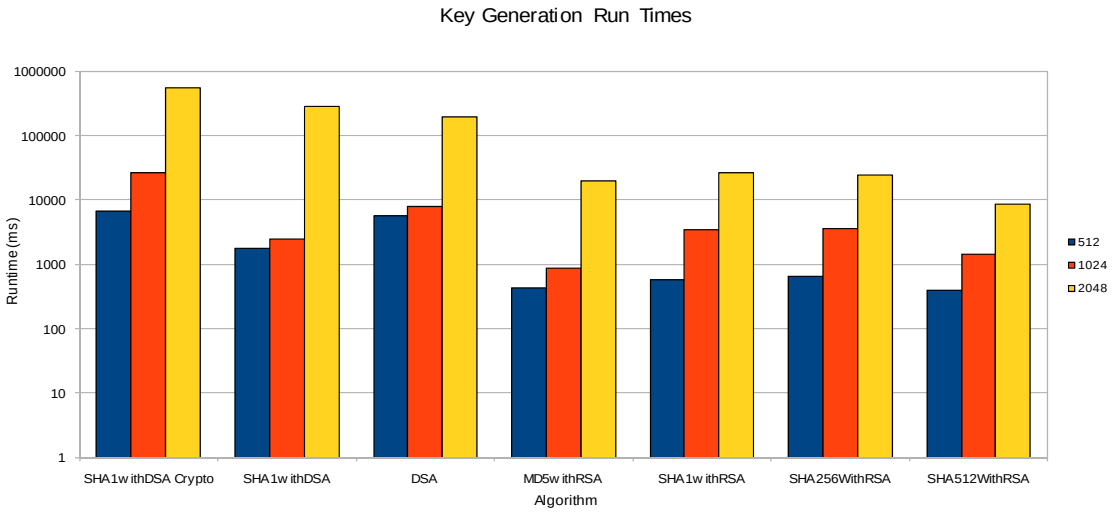


Figure 3: Algorithm runtime comparison for key generation

With the same setup, the results for signature generation runtimes are shown in Figure 3. The results show that signing the data is much quicker than generating a key pair. Note that the scale for the runtimes is linear. There is no result for SHA512WithRSA with a 512 bit key as that algorithm requires a key of minimum size 1024-bits. Strangely there was no result for SHA-1 with DSA from the Crypto provider with a key size of 2048. No result was printed to the phone or log and no exception or error message was received. Since this test was run once, further investigation would be required if the result continued to be omitted.

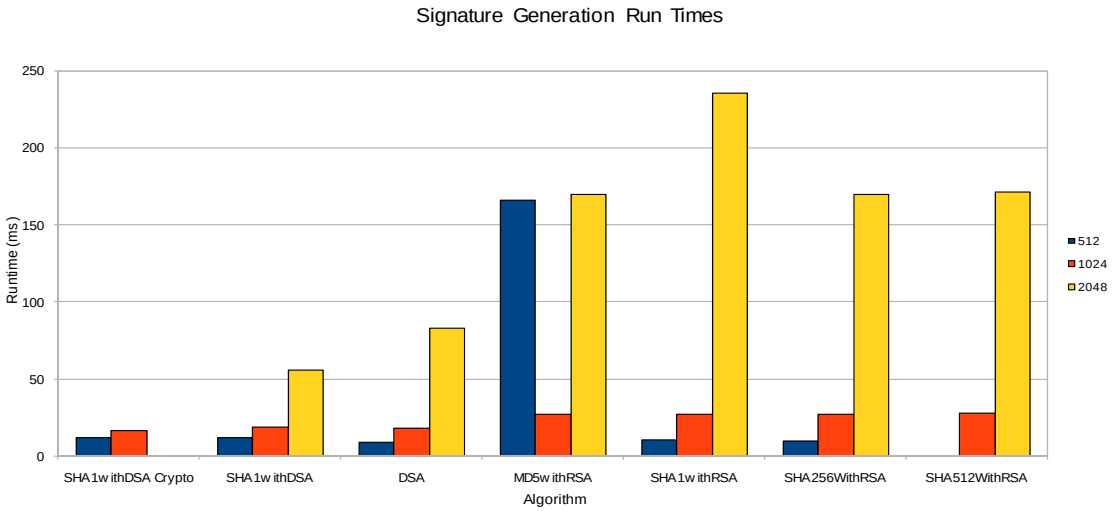
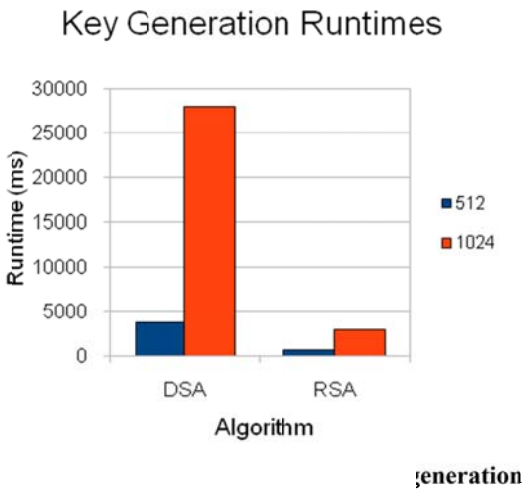


Figure 4: Algorithm runtime comparison for signature generation

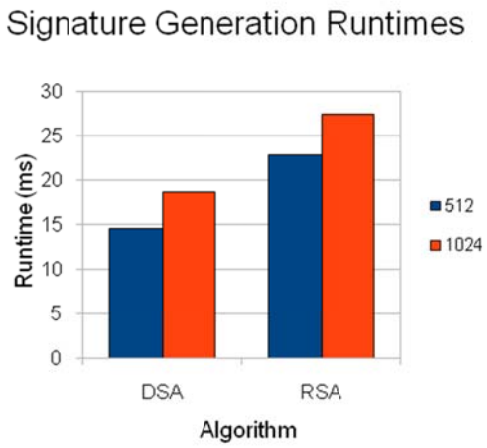
The results of the first test seem to hint that RSA is faster at generating key pairs and is slightly slower for signing the data when compared to DSA. It is also clear that key generation with a key length of 2048-bits can take up to 15 minutes, which is clearly too long to be suitable for real-time applications. Of course, if this key length was desired, key generation could be done server-side and the private key could be distributed to the mobile device (which is generally not a secure technique due to possible attacks during the key's transmission).

These observations lead us to the second experiment, which tests runtimes of the same set of algorithms with key sizes of 512 and 1024-bits. Five trials were run and averaged. The averages for each algorithm implementation was then averaged into two general categories, RSA and DSA, for ease of comparison. Figure 5 shows the results for key generation runtimes and Figure 6 shows the results for signature generation runtimes. We can observe from the graphs that DSA key generation takes around four seconds for a 512-bit key and almost 30 seconds for a 1024-bit key. In contrast, RSA takes roughly 0.5 seconds and three seconds for 512 and 1024-bit keys respectively. Signature generation seems to be faster for both key sizes with the DSA algorithm but a statistical test would need to be performed in order to determine if DSA is significantly faster at signing data.



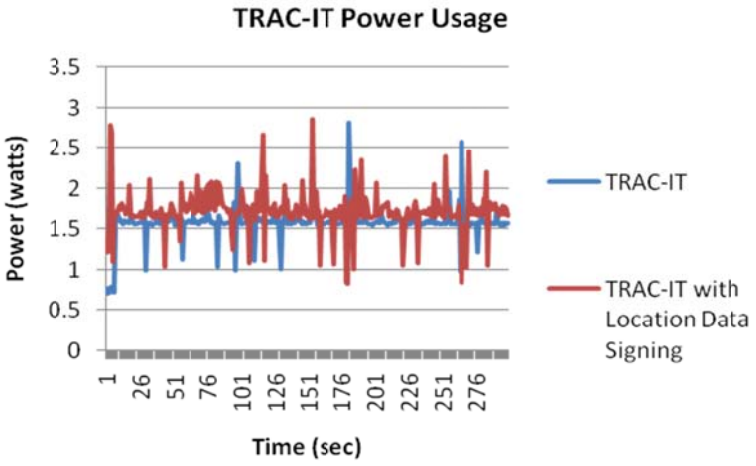
eneration

Overhead Analysis



With added security comes added overhead. This phase seeks to determine the effects of digitally signing location data in terms of overhead battery life, communication load, and database storage.

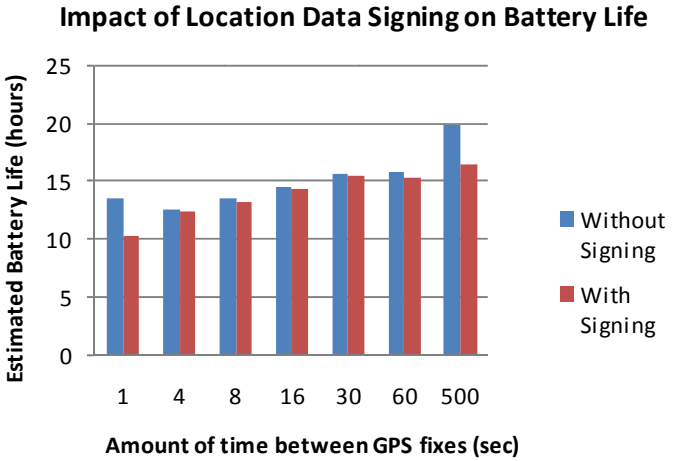
In order to determine the effects of location data signing on the phone's battery life, a Motorola Droid X phone was hooked up to an Agilent E3631A power supply to measure the power consumed by the device while running TRAC-IT with and without location data signing. The tests were run with two versions of TRAC-IT (one with the location data signing module active) and the results can be seen in Figure 7. The tests were run with Wi-Fi, bluetooth, screen timeout, and aGPS disabled with the phones remaining stationary in the same physical positions while the power measurements were taken at one second intervals for 300 samples. Regular TRAC-IT averaged a power consumption of 1.57 watts while TRAC-IT with location data signing consumed an average of 1.71 watts. It appears that adding location data signing does require that the phone use more power (which is intuitive since more calculations are being performed), but it should be noted that the results cannot be fully relied upon since the Android operating system allows many processes to share the CPU and it is possible that a background process may have been the cause for the variance in power usage.



Further experimentation also used the same equipment to analyze the effects of location data signing on battery life. These tests varied the amount of time between GPS samples while measuring power consumption both with location data signing active and with it disabled. To estimate battery life, Peukert's Law was used as $C_P = I^k t$ where C_P is the capacity, I is the discharge current, k is the Peukert constant, and t is the discharge time in hours.

Results of this experimentation are shown in Figure 8. As expected, the battery life is longer when location data signing was disabled, although by a slim margin at some intervals. It should be noted that the largest energy overheads when using location data signing occur when it is used very frequently (i.e., one second intervals), and when it is used very infrequently (i.e., 500 seconds). From this data, it appears that the CPU is constantly active at one second intervals when a heavy signing load creates a large amount of processing, which causes additional impact on battery versus when not signing the data. At occasional use (intervals from four second to 60 second intervals) the CPU can absorb the additional overhead of signing within other processing loads without a large impact on battery life. However, at 500 seconds, it appears that without location data signing the device hardware (e.g., GPS, CPU) is able to enter a low-level power state. When adding the additional processing for digital signatures, it appears that the phone is prevented from entering this low-level power state and the CPU is active longer, thus resulting in reduced battery life. Future experimentation should measure additional intervals, and on additional phones, to see if this trend repeats.

Before adding the location data signature to the UDP packet, the UDP packet size was 69 bytes. After adding in the signature, the size of the packet grew to 115 to 117 bytes. Although this means that the packet is now 66% larger, only 0.17% of the available packet size is filled (since for IPv4 UDP packets can hold 65,507 bytes of data). Since the packet size is still very small, it is unlikely to affect the likelihood that a packet may be lost in



d without cies

transmission, but further study is recommended to verify this theory through future experiments.

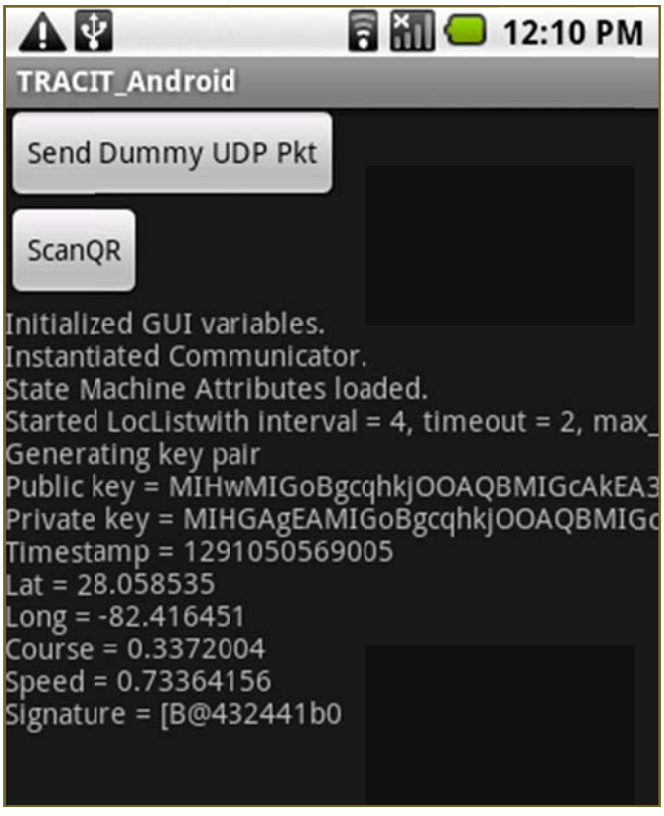
Two new fields were added to the database: signature in the table for trip data and public key in the table for session data. Each field was created as a varbinary(256) in order to allow for future experimentation that might vary the key and signature sizes. This resulted in a 150% increase in row size for the tip data table and a 74% increase in row size for the session table.

Implementation

The implementation for TRAC-IT with location data signing uses a key generation per session, a key size 512-bits, and the RSA algorithm with SHA-1. For a prototype, there is no need to store the key on the device. This will also save database space. Figure 9 shows the modified Android TRAC-IT client with added print statements showing the generation of public and private keys and the signature created from the location data.

There are currently four separate projects needed to run the TRAC-IT system for Android. The projects and brief descriptions are as follows:

- **wsTRACIT** - the web service that interfaces between the phones and the database
- **TRACIT_Android** - the TRAC-IT application installed on Android
- **CUTR_Android_Library** - a library of location-based classes for Android usage
- **CUTR_Shared_Library** - a library of location-based classes for general Java usage



The main contribution of this project was the creation of the package **edu.usf.cutr.microedition.util.LocationDataSigner**. The main methods of this class are initialize(), which generates or retrieves the stored key pair, and signLocationData(String userName, Point location, long time), which takes in the user's name, location, and time-stamp and returns a signature.

A location data verfication tool was also created as a seperate project. A sample run (edited for brevity) can be seen in

Figure 10. The tool consists of a Java class that performs the verification and a controlling class that handles input and output to the commandline. The verification class can easily be moved to another project if one wants to perform the verification on the TRAC-IT website or as part of the webservice. To perform a verification of a session, the user only needs to

provide the tracking session ID. The tool will then get the username, public key, location data, and the signatures that correspond to the selected tracking session.

```
Please enter a TRAC-IT Session ID: 6626
Public Key: 081F03081A806072A8648CE38040130819C024100E3A4B16CC...
Message: megordon@cutr.usf.edu28.124498-82.4688641291907408700
Tracking ID 1931436: true
Message: megordon@cutr.usf.edu28.1244-82.468821291907410700
Tracking ID 1931437: true
Message: megordon@cutr.usf.edu28.124405-82.468821291907411700
Tracking ID 1931438: true
...
Session validated: true
```

Figure 10: Sample run of the location data verification tool

5. CONCLUSIONS

This paper presents the concept of location data signing, used to protect the integrity and authenticity of location data, and a succesful implementation of location data signing within a mobile application for the Google Android platform on GPS-enabled cell phones. It was shown that a key size of 2048-bits is too large to be quickly calculated on the T-Mobile G1 mobile phone. A key size of 1024-bits seems to have quick runtimes for key generation and signing with the RSA algorithm. Until further studies examine the effects of CPU load and battery life in detail, the system will use 512-bit key size for prototype testing. As devices become faster and include more memory, a 2048-bit key may become feasible. Future analysis of digital signature algorithms, particularly for use in the LAISYC framework, should include testing how data size affects signature generation runtimes and implementing the ECDSA algorithm for use on Android (if licensing permits). Experimentation should also further explore the impact of location data signing overhead on battery life at different GPS sampling intervals. The feasiblity of location data signing on mobile devices, as demonstrated in this paper, therefore paves the way for secure uses of GPS data in the transportation domain.

ACKNOWLEDGMENTS

The authors would like to thank Philip Winters, Nevine Georggi, and Isaac Taylor for their contributions to the TRAC-IT project as well as all the other authors and code contributors of TRAC-IT, including Marcel Muñoz Figueroa for his work on estimating battery life, shown in Figure 8. The development of TRAC-IT and portions of the LAISYC framework have been supported in part by the Florida Department of Transportation, and the United States Department of Transportation through the National Center for Transit Research under grant numbers BD-549-24 “Testing the Impact of Personalized Feedback on Household Travel Behavior (TRAC-IT Phase 2)”, BD-549-35 “Smart Phone Application to Influence Travel Behavior (TRAC-IT Phase 3)”, and BDK-85 TWO 977-14 “Dynamic Travel Information Personalized and Delivered to Your Cell Phone.”

REFERENCES

(1) Mitch Stacy. “Small GPS devices help prosecutors win convictions,” *Associated Press*. August 28th, 2008. © 2008 Associated Press.

(2) Barbeau, S. and Labrador, M. and Georggi, N. and Winters, P. and Perez, R. “TRAC-IT – A Software Architecture Supporting Simultaneous Travel Behavior Data

Collection & Real-time Location-Based Services for GPS-Enabled Mobile Phones". *Proceedings of the National Academy of Sciences' Transportation Research Board 88th Annual Meeting*, 2008.

(3) Barbeau, S. Perez, R. Labrador, M. , Perez, A. , Winters, P. and Georggi, N. "LAISYC - A Location-Aware Framework to Support Intelligent Real-time Applications for GPS-Enabled Mobile Phones". *Pervasive Computing, IEEE*, PP(99):1 -1, 2010.

(4) Barbeau, S., Labrador, M., Winters, P., and Georggi, N. L., Aguilar, D., and Perez R., "Dynamic Management of Real-Time Location Data on GPS-enabled Mobile Phones", *Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM)*, Valencia, Spain, Sept. 2008.

(5) Barbeau, S. Labrador, Georggi, Winters, Perez. "The Travel Assistance Device: Utilizing GPS-enabled Mobile Phones to Aid Transit Riders with Special Needs", *Institution of Engineering and Technology (IET) Intelligent Transportation Systems*, 2010, Vol. 4, Iss. 1, pp. 12–23. doi: 10.1049/iet-its.2009.0028. © The Institution of Engineering and Technology 2010.

(6) Certicom. "Certicom Licenses Intellectual Property to Research In Motion (RIM)". 2004. Available at http://www.certicom.com/index.php/2004-press-releases/36-2004-press-releases/299-rim-expands-use-of-elliptic-curve-cryptography-ecc-in-blackberry-products

(7) Farley, L.G. "The Adam Walsh Act: The Scarlet Letter of the Twenty-First Century". *Washburn LJ*, 47:471, 2008.

(8) Google. "Android API Documentation". 2010. Available at http://developer.android.com/reference/packages.html

(9) Jarusombat, Santi and Kittitornkun, Surin. "Digital Signature on Mobile Devices based on Location", *Proc. Int. Symp. Communications and Information Technologies ISCIT '06*, pages 866--870, 2006.

(10) Murat Fiskiran, A. and Lee, R.B. "Workload characterization of elliptic curve cryptography and other network security algorithms for constrained environments". *Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on*, pages 127 - 137, 2002.

(11) Potlapally, N.R. and Ravi, S. and Raghunathan, A. and Jha, N.K. "A study of the energy consumption characteristics of cryptographic algorithms and security protocols", *Mobile Computing, IEEE Transactions on*, 5(2):128 - 143, 2006.

(12) Potlapally, N.R. and Ravi, S. and Raghunathan, A. and Jha, N.K. "Analyzing the energy consumption of security protocols", *Low Power Electronics and Design, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on*, pages 30 - 35, 2003.

(13) Chengming Qi. "A Zero-Knowledge Proof of Digital Signature Scheme Based on the Elliptic Curve Cryptosystem", *Intelligent Information Technology Application, 2009. IITA 2009. Third International Symposium on*, pages 612 -615, 2009.

(14) Soewito, B. and Vespa, L. and Ning Weng. "Characterizing Power and Resource Consumption of Encryption/Decryption in Portable Devices", *Region 5 Conference, 2008 IEEE*, pages 1 -6, 2008.

(15) Zuguang Xuan and Zhenjun Du and Rong Chen. "Comparison Research on Digital Signature Algorithms in Mobile Web Services", *Management and Service Science, 2009. MASS '09. International Conference on*, pages 1 - 4, 2009.