KeySlinger and StarSlinger: Secure Key Exchange and Encrypted File Transfer on Smartphones

Jason Lee

CMU-CS-11-115
May 2011

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee
Adrian Perrig, Chair
Peter Steenkiste

Submitted in partial fulfillment of the requirements for the Degree of Master of Science

This research was supported in part by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389, MURI W 911 NF 0710287, and W911NF-09-1-0273 from the Army Research Office. The views and conclusions contained here are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, CMU, or the U.S. Government or any of its agencies.

Keywords: Security, Mobile, Android, iOS, Mobile Apps, Secure Key Exchange, Encryption, Authentication, Secure Communication, Secure File Transfer

Abstract

As the smartphone market continues to grow at a rapid pace, secure communication between these devices becomes an important issue. Smartphones are capable of being used in methods that differ vastly from the traditional desktop computing environment, making it possible to create new kinds of security protocols that take advantage of the mobility and other features provided by smartphones. This work presents KeySlinger and StarSlinger, apps for secure key exchange and authenticated/encrypted file transfer, respectively. KeySlinger is built on the SPATE protocol to exchange information between small (fewer than 8 people) or large groups of people in an authenticated manner. StarSlinger uses keys exchanged using KeySlinger to transfer authenticated and encrypted files between users.

Acknowledgements

There are several people I would like to thank for their help with this project.

I extend my deepest gratitude to my advisor Adrian Perrig for introducing me to research in computer security as well as his guidance and support throughout the time I have worked with him.

I would also like to thank Michael Farb, Jonathan McCune, and Ahren Studer of CyLab for their help in this project and in writing this document.

I thank Manish Burman for his role in the Android implementation of StarSlinger.

I thank Peter Steenkiste for agreeing to join my thesis committee and for his help throughout my M.S. program and in preparing this document.

Finally, I extend special thanks to Deborah Cavlovich for all her help with every miscellaneous question I have had throughout my M.S. program.

1. Introduction

The smartphone market is one of the fastest-growing consumer markets. RIM's Blackberry and Nokia's Symbian have been widely adopted in the business world for some time now. With the advent of Apple's iPhone and Google's Android operating system, the reach of smartphones has now extended well into the consumer market, with hundreds of thousands of devices being activated every day [3].

With this rapid growth comes a concern for security and privacy. Certain governments such as India demand access to all communication on Blackberry [4] and Symbian [5] devices. Malware and spyware that steal personal information on Android smartphones have recently made their way into headlines [6]. Apple's iPhone has also been shown to be vulnerable to attacks using malicious SMS messages [7], PDF files [8], and various other techniques. Although not specific to smartphones, SMS spoofing can also be an issue [9]. As smartphone adoption increases, such incidents will also certainly increase.

In light of these risks, this work introduces a method of secure data exchange between smartphone users comprising two parts: first an authenticated key exchange using KeySlinger [1], and secure file transfer using this exchanged key with StarSlinger.

KeySlinger is built on the SPATE [2] protocol, a protocol developed for authenticated data exchange among a small group of people consisting of seven or fewer members. A modified version of the protocol, known as Ho-Po Key [31], is used for exchanges in groups of larger size. Participants run the application, select the contact data they wish to share (phone number, e-mail address, public keys for third party applications, etc.), and begin the exchange. Data is exchanged through an untrusted server. At the end of the exchange, participants validate the data they received against that of other members by means of a hash-based word list. If the word list check succeeds, participants know that: (1) each participant contributed exactly one data element; (2) no one outside the group contributed data; and (3) the data distributed is exactly what each individual user intended [2]. Such an exchange allows users to share public keys for use in various applications such as StarSlinger or encrypted text messaging applications such as TextSecure [29].

StarSlinger is an authenticated, encrypted file transfer application that uses KeySlinger as its means of sharing public keys. Each user has an RSA key pair, the public key of which is exchanged along with other user-identifiable data using KeySlinger. Once two users have established public keys with each other, they can send authenticated and encrypted files to each other at any time. File transfer is done via a remote server. Once the sender uploads the file, the server notifies the receiver to download the file.

The rest of this paper is structured as follows. Section 2 discusses attacker models for both KeySlinger and StarSlinger. Section 3 reviews previous related work. Sections 4 and 5 discuss implementation details for KeySlinger and StarSlinger, respectively. Section 6 discusses how these applications address security issues described in earlier sections. Section 7 concludes the paper and offers plans for future work.

2. Problem Definition and Challenges

The users' first objective is to securely exchange public keys for one or more smartphone apps in a way that ensures that the data each user received is indeed the data sent by the intended sender or senders. This is not easy to accomplish in an ad-hoc setting. Finding a mutually trusted certificate authority to sign each user's public keys is infeasible in several ways, one of the more

significant reasons being that getting a valid certificate from a CA is very expensive in terms of both financial resources and time.

KeySlinger looks to avoid this problem by providing a method of establishing ad-hoc trust. This is a difficult task in that there is no PKI or mutually trusted third party to rely on when a group of people are meeting for the first time, which is the use case for KeySlinger. KeySlinger takes advantage of the fact that physically collocated people can verify each others' identities. This bootstraps the trust required for the secure key exchange.

In addition to bootstrapping trust, another challenge is group formation in an ad-hoc setting. The original SPATE implementation accomplished this by using Bluetooth as the connection mechanism. Another option is to use one of the phones as an access point to connect to. However, neither of these options was viable for KeySlinger because of the restrictions iOS places on what the developer can do with the device. The Nokia phones which SPATE was originally implemented on allowed access to device layer of information of the Bluetooth radio, which made it possible to bypass the Bluetooth device discovery phase. Because this is not possible on iOS devices, using Bluetooth would have meant going through device discovery, which in testing took anywhere between 10 seconds and several minutes. Such an impact on performance was unacceptable. Access point functionality on the phones was also not an option until the most recent version of iOS introduced WiFi tethering. Even with this new functionality, the device is limited to 5 connections which makes this option not usable for KeySlinger. For these reasons it was decided that KeySlinger would use a Web connection to a remote server to perform exchanges.

This leaves open the question of how the server will determine which users belong to the same group. For this the users need to provide some common value. One possibility for such a value that was discussed but discarded was the phones' location data. Because KeySlinger requires users to be physically collocated, users in a group would have very close location data, given some degree of error depending on the accuracy of each device's sensors. The problem with using location data, however, is that people in general are uncomfortable with disclosing their location information. A good example of this was the recent "iPhone Tracker" fiasco experienced by Apple. It was discovered that iPhones on iOS 4 had been accumulating the phone's location data for the past year. Apple released a statement to answer questions about the incident, stating that the data collected was not actually the phone's location but rather a database of cell towers and WiFi hotspots [30]. Nonetheless, this data could be used to place the user within a certain radius and iPhone users were very unhappy to learn of this. In light of such reluctance to disclose location data, it was discarded as an option for group formation methods.

Once the keys are established, the users should be able to securely send files to each other with a guarantee of authenticity, integrity, and secrecy. There are several aspects to consider when developing such a scheme, including schemes for encryption and signatures and the actual file transfer mechanism.

For both apps, considerations must be made for the fact that the apps run on mobile devices with limited resources and potentially limited monthly data usage. The user interface should be simple and intuitive with minimal user interaction to reduce user errors. There are also restrictions on what the developer can do with public SDKs. For example, iOS does not allow device layer manipulation of components such as the Bluetooth radio, which was an important part of the original SPATE implementation [2]. Due to this limitation, both KeySlinger and StarSlinger were designed to use a remote server as an intermediary instead of having the group members' devices communicate peer-to-peer via Bluetooth or an ad-hoc WiFi network.

2.1 Threat Model for Attacks Against KeySlinger

This paper assumes an active adversary who can eavesdrop, intercept, and manipulate all communications. The attacker's goal is to manipulate the data being exchanged among the group without being detected. Details of attacks against the exchange protocol itself are described in [2]. They include deleting, modifying, or inserting data.

In addition to the scheme described in [2], KeySlinger adds a new entity - the untrusted server. The server adds several attack possibilities. Because all user data resides on the server during an exchange, this data can potentially be exposed to an outside eavesdropper located anywhere with an internet connection. If the server itself is malicious or compromised, it can attempt to manipulate the data. These possibilities will be further explored in Section 6.

2.2 Threat Model for Attacks Against StarSlinger

An adversary attempting an attack on StarSlinger is assumed to have the same capabilities as in 2.1. Analyses on StarSlinger are based on an assumption that RSA public keys were exchanged in a secure fashion, in this case using KeySlinger.

It is assumed that signature forgery or file decryption is infeasible without possession of the corresponding RSA private key. Because files in transfer must reside on the server until the receiver downloads them, attackers can potentially gain access to them during this time. As long as the recipient's private key is not compromised the attacker cannot actually access the contents of the file because it is encrypted with a session key which itself is encrypted with the recipient's RSA public key. Spoofing attacks are also infeasible because it was assumed that RSA signature forgery is infeasible. Potential attacks then are limited to denial of service or spamming. These issues are addressed in Section 6.

3. Related Work

3.1 Works Related to SPATE and KeySlinger

There have been several attempts to achieve secure data exchange between a pair of devices. These include schemes using common passwords [12], [13]; visual string comparison [12], [13], [14]; string comparison via human audio representation [15]; visual comparison of graphics that encode data [16], [17]; shaking devices to create shared entropy pools [18], [19], [20]; using common properties of the wireless channel [21]; and location-limited channels [22], [23], [24].

Closely related to the SPATE exchange is GAnGS [25]. Both attempt to distribute authentic information within a group of physically collocated users. However, GAnGS is designed only for the exchange of public keys and requires the installation of the private key on the user's device. In addition, SPATE is more efficient in that users are required to perform fewer total interactions in the absence of infrastructure [2].

PGP uses key signing parties to extend the web of trust. This involves attendees physically verifying each others' identities using passports or driver's licenses and verifying their key fingerprints. The process is much more cumbersome than SPATE/KeySlinger.

3.2 Works Related to StarSlinger

As of this writing there are no public solutions for iOS or Android that provide secure file

transfer. There are some file transfer apps such as Bump [26] and Hoccer [28] but they do not provide the authenticity and secrecy of StarSlinger. Bump provides secrecy via HTTPS but does not necessarily guarantee authenticity (at one point it did not even use HTTPS [27]). The only verification method involved in Bump is the other user's name. As such, it remains quite vulnerable to man-in-the-middle attacks.

4. KeySlinger Design and Implementation

KeySlinger is currently available on the iOS App Store and Android Market. Both versions have virtually identical user interfaces. This section describes the implementation and high level details of KeySlinger. Details of the underlying exchange protocol are described in [2].

4.1 SPATE and Ho-Po Key

Although the SPATE protocol itself is not in the scope of this paper, it is necessary to discuss it as it is the underlying exchange protocol used in KeySlinger.

The SPATE protocol allows small groups of people (2~8 users) to perform authenticated key exchange in the absence of a mutually trusted entity. Each device begins by generating a protocol commitment based on a randomly generated nonce. This protocol commitment is then hashed with the user data to produce the data commitment. The two commitments are exchanged along with the user data during the exchange. Once all data items are received, each device first verifies the data commitment against the data it received to ensure integrity of the exchange. A hash is then computed over all data commitments and data. The first 24 bits of this hash are used to produce a T-Flag which users use to verify that the data received by each user is consistent throughout the group. If users indicate matching T-Flags, the devices disclose their nonces which are then used to verify the protocol commitments. This step ensures that only users within the group contributed data items.

For large group exchanges, a modified version of the SPATE protocol called Ho-Po Key is used [31]. The first difference is just before the T-Flag verification phase. Each user is presented with a position number and asked to form a ring by standing between users with neighboring position numbers. This step ensures that all users in the group contributed exactly one data item and data was not inserted by an outsider. The second difference is in the verification phase itself. In Ho-Po Key users conduct two comparisons. For each comparison, each user compares flags with the user to his left, right, or both, depending on his position in the ring.

4.2 Design Considerations

There were several API restrictions that forced design choices in KeySlinger. One of the most important challenges in designing a mechanism for interaction with third party apps was providing a way for multiple apps to share data. Modern smartphone operating systems, including both iOS and Android, use a "sandbox" approach to app data. This means that no app can access the data of any other app. There are very few exceptions to this rule. Apple has provided an option for apps to connect themselves to certain file types. For example, if a user attempts to view a PDF document in the Mobile Safari browser, he is given the option to open the document using any app that handles PDF files, such as Apple's iBooks. Android apps are capable of reading from and writing to the phone's external storage area. While this potentially

offers more options for the Android implementation, it was deemed better to have a consistent approach throughout various platforms.

The address book is one of the exceptions to the sandbox rule in that it can be read and written by all apps. This is also likely to be true for most current or future smartphone platforms as many apps require access to user contacts. It also provides a useful standardized interface to format data to exchange because contact information can be exported to V-Cards using existing APIs. For these reasons it was chosen as the medium to store third party app key data to be processed by KeySlinger. Third party key data is written as a customized instant message field because the V-Card format allows customization of this field. The service provider field serves to identify the app the key is for and the username field stores the public key data. A sample V-Card field looks like Listing 1 below.

```
IMPP;TextSecure-
IdentityKey:QVFLNEtEZXhmam5lRGNsc2dXRkFIdThhL1ZJRFRlY1FOMHdZVHFiNnUrUH
dBUT09
```

Listing 1. Sample V-Card field

4.3 Interaction With Third Party Apps

```
ks://[calling application name]?[record ID used by calling application]?[URL to launch
calling application]
```

Listing 2. Custom URL scheme for calling KeySlinger on iOS

```
Intent intent = new Intent();
intent.putExtra(EXTRA_CONTACT_ID, contactId);
intent.putExtra(EXTRA_CONTACT_LOOKUP_KEY, contactLookupKey);
if (keyName != null) {
        intent.putExtra(EXTRA_KEY_NAME, keyName);
        if (requestCode > 0)
                intent.setAction(USER_KEY_RETURN);
        else
                intent.setAction(USER_KEY);
} else {
        if (requestCode > 0)
                intent.setAction(USER_ONLY_RETURN);
        else
                intent.setAction(USER_ONLY);
}
if (requestCode > 0)
        startActivityForResult(intent, requestCode);
else
        startActivity(intent);
```

Listing 3. Sample code to call KeySlinger on Android

Third party apps can use KeySlinger to exchange their own key data. The process is similar on both platforms. The app first writes all necessary data into a custom IM field of a contact entry representing the owner of the device, then launches KeySlinger with an argument

to indicate which contact entry to use. In iOS this is accomplished using custom URL schemes and on Android using Intents. These methods are outlined in Listings 2 and 3. Once KeySlinger completes the exchange, the third party app can read in new users' data written as either new contact entries or additional fields in an existing entry.

4.4 Initial Phase of Exchange

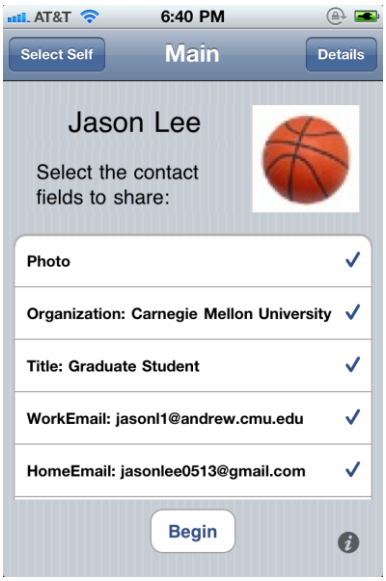

Figure 1. Initial screen of KeySlinger

The initial screen directs the user to select an entry from his address book that contains his own information and select which fields to share during the exchange (Figure 1). Supported fields include phone numbers, portrait images, e-mail addresses, physical addresses, and public key values for third party applications in the form of custom instant message service fields.

Figure 2. Group size selection screen

Once the contact entry and field selection is complete, the user indicates how many members are in the group (Figure 2). For small groups (7 or fewer), the user specifies the exact number. Larger groups are identified only as "8 or more people." The threshold for group size was set to 8 people because it has been shown that people begin to make counting errors when the groups get larger [29].

After the user indicates the group size the device requests an ID from the server. This is a random number generated by the server and is unique to every user. To avoid user errors, these numbers are kept as small as possible. Groups are formed by each user reporting the lowest user ID in the group to the server. Technically any common value can be used. Earlier approaches involved the group members designating a leader and entering his ID number. However, user testing showed many groups had trouble deciding on who would be the leader. The minimum ID approach was chosen because it was deemed the easiest way to arrive at a common value. Based on this information and the intended size of the group, the server groups users together. At this point the exchange is ready to start.

4.5 Data Exchange

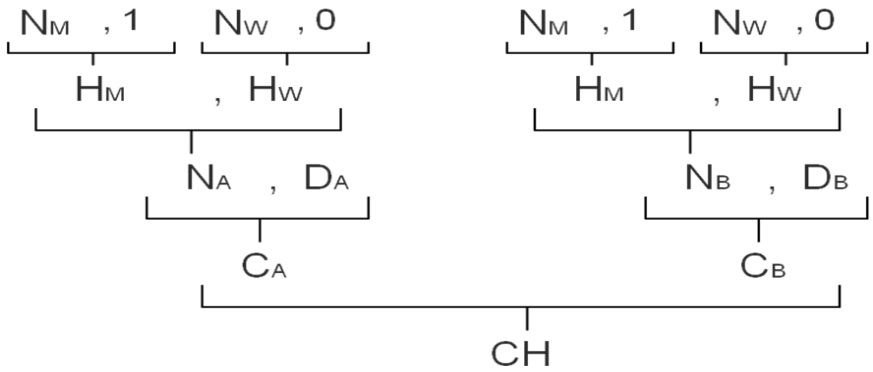

Figure 3. Outline of KeySlinger of commitment generation process. Left side represents process for user A, right side for user B. The hash of all data (CH) is later used to generate the word lists.

Each device generates two nonces (N_m, N_w). These are then hashed with known values that correspond to the success and fail cases, represented by the 1 and 0 in Figure 3. In KeySlinger, the string literals "match" and "wrong" are used for compatibility with earlier implementations of SPATE. The resulting hashes (H_M, H_W) are then hashed together to produce the protocol commitment (N_A, N_B). The protocol commitment is hashed with the data to be exchanged to produce the data commitment (C_A, C_B). All devices send their commitments and data to the server and the server distributes them to other devices in the group. Once all items are received, each device checks the data commitments against the data. In the case of large groups, each device then receives a position number and the users are asked to form a ring by standing next to members with neighboring positions. This step ensures no additional data was injected into the group. In small groups, the device simply checks the number of members specified at the beginning against the number of items it received. The ring procedure is used for large groups to accommodate the fact that users have trouble counting accurately when 8 or more people are present.

Figure 4. Word list screen

If the commitment check succeeds, each device generates a triplet of words based on the hash of all the data it received (including its own). Each of the first three bytes of the hash corresponds to a word from the PGP word list [10]. In addition each device generates two random triplets. Users confirm the exchange was successful by selecting the triplet that matches the other members of the group (Figure 4). The lack of a matching triplet indicates data was manipulated and the exchange should be aborted. The word list scheme was chosen over the original T-Flag [2] scheme to reduce errors from careless users. The T-Flag allowed users to simply indicate a success without actually verifying the data. With the word lists, users must compare values with each other to find the matching triplet. Another advantage of the word lists is that the words can be read aloud, making it easier to compare values than trying to look at every user's screen.

If every user indicates a successful exchange, each device discloses its "success nonce (N_M)" and "failure hash (H_W)." If a user indicates a failed exchange by selecting "No Lists Match" or selects the wrong word triplet, the "success hash (H_W)" and "failure nonce (N_M)" are disclosed. The presence of "failure nonces" indicates one or more users indicated something wrong with the received data. Even if a user indicates a success, if his device receives a "failure nonce" it will abort the exchange. This protects against users simply indicating a match without actually verifying the word lists. The nonces and hashes are used to validate the protocol commitments. If this check passes, the exchange is finally deemed successful. A more detailed description of the exchange protocol can be found in [2].

At the end of a successful exchange, the user selects which of the received contact entries to save into his address book. New people are added as new entries and new information for previously known people is merged with existing contact entries.

5. StarSlinger Design and Implementation

StarSlinger is currently under development for both iOS and Android platforms. The current goal is to get an initial release before the end of May.

5.1 Design Considerations

One thing to consider for StarSlinger was how the transfer process would work. Several options were considered. One scheme that was considered establishing a direct connection between two devices, but this was deemed infeasible because it required a synchronization process between the users.

After it was decided that a remote server would act as an intermediary, the next problem was finding a way to notify a recipient when a file was ready to download. SMS was the initial choice, but this was later abandoned in favor of the various platform dependent push notification systems. While SMS offers the advantage that it is universal (Apple's and Google's push services require knowledge of the recipient's platform and may or may not be available depending on the OS version), it would have incurred extra cost for the users. There was also the problem that iOS apps cannot intercept SMS messages. Using SMS would have required the messages to be formatted into a custom URL scheme and the user would have needed to open the message and tap the URL, adding an extra step to the process.

Another issue was where to save received files and store files to be sent. This location should provide read/write access to the StarSlinger application and also offer an easy option to transfer to a user's computer. On Android the external storage area provides both of these properties. Since version 3.2, iOS has offered a feature called iTunes Document Sharing whereby an application can make its Documents directory visible to iTunes so that the user can transfer easily between the phone and computer. If other platforms are later supported, any location that offers these properties can be used for file storage.

5.2 Initiating KeySlinger Exchange

From the application's main screen, StarSlinger offers an option to run KeySlinger with the StarSlinger contact selected as the entry to share. This allows users wishing to perform a StarSlinger transfer to easily establish keys first if necessary.

5.3 Sending Files

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfStarSlingerKeys>
<StarSlingerKey OS="Android" RegistrationDate="20110401"
PushToken="1215145FKJK1215145FKJK1215145FKJK1215145FKJK"
PublicKey="5234HJ234L5KJH245LK1;1;L45;2345;JKL7JKL45GH8KHLG95LKHG9L5KJ67"></StarSlinge
rKey>
<StarSlingerKey OS="iOS" RegistrationDate="20110415"
PushToken="1234124-76734562-13241234-12341234"
PublicKey="NERGHFJK145GHF4678GHF5VC79MNCH24HJ5D2JKF253CVM2NV45,345.3467.B46.NB7
B48"></StarSlingerKey>
</ArrayOfStarSlingerKeys>
```

Listing 4. Sample StarSlinger public key data

StarSlinger first requires users to exchange RSA public keys and other data to identify each user. It does not require KeySlinger to be the medium of this exchange, as long as it is authenticated and the data is written into the address book in the proper format. The additional data other than the RSA keys includes a field to indicate the user's device platform (iOS, pre-2.2 Android, post-2.2 Android) and the push token of the device. This token is a platform-specific value that identifies an application on a device. The Apple Push Notification service uses 32-byte

tokens while Google's Cloud to Device Messaging service uses 119-byte tokens. All of this data is formatted into an XML document as shown in Listing 1. This facilitates processing other users' data using existing XML parsing APIs and leaves room for further extension if other fields or support for additional platforms are later added. When saving this information into the user's address book, the entire data blob is Base64-encoded into a single string to prevent the user from accidentally corrupting the data.

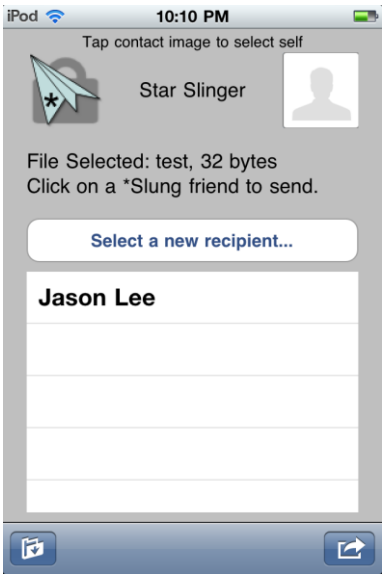

Figure 5. StarSlinger main screen. Lower left icon brings up file selection, lower right launches KeySlinger.

Similarly to KeySlinger, StarSlinger also directs the user to select his own entry from the phone's address book (Figure 5). This is to indicate the sender to the receiver. In addition to this information, the user selects the recipient from a list of known contacts with StarSlinger keys established and the file to send. The file is then encrypted with a randomly generated session key which is itself encrypted using the recipient's public key, signed with the sender's private key, and uploaded to the server. Upon completion of the upload, the server divides the file into chunks and computes a Merkle hash tree on the chunks. The server then pushes a notification message to the recipient's device. For Android devices with version 2.2 or later, Google's C2DM (Cloud to Device Messaging) service is used. For iOS devices with version 3.0 or later, APNs (Apple Push Notification service) is used. Devices that do not support these mechanisms will use SMS messages (sent by the sender's device) as notifications.

This model of file transfer provides flexibility for both the sender and receiver. By using the remote server as an intermediary, the users need not be physically collocated or even coordinate their schedules. The sender can upload a file whenever he needs to and the receiver can download the file at his own convenience. The use of push notifications also allows StarSlinger to be used for secure messaging by sending messages as files.

5.4 Receiving Files

When a user receives a notification, he activates the application which uses a file identifier, a random string, included in the notification message to retrieve the file from the server. The receiver's device first downloads the root of the Merkle hash tree, the chunk that contains the sender information, and the hashes required to verify it. This sender information

determines whether or not to download the rest of the file. This is for use in the blacklist scheme to avoid spamming, discussed later in Section 6.

Once the file is completely downloaded the device first confirms the integrity and authenticity of the sender by verifying the signature on the file. If this check succeeds, the device can then decrypt the contents to present to the user. Unclaimed files are deleted after 24 hours and a notification is sent to the sender in case he wishes to retry.

5.5 Authentication and Encryption Scheme

StarSlinger is designed to be compatible with the OpenPGP Message Format [11]. For each file, a random session key (AES 128) is generated. The session key is encrypted along with the sender's identity using the receiver's RSA public key. This information is prepended to the file data which is encrypted using the session key. A digital signature using the sender's private key is also appended to the file data.

Making StarSlinger compatible with PGP has the advantage that the scheme is already well established and also provides a wide potential initial user base. PGP users will have much more incentive to install StarSlinger if they know it can handle their PGP messages and files. Initially a fixed Diffie-Hellman key establishment scheme was considered. This was mainly for the fact that Diffie-Hellman does not require the additional overhead of sending session keys with each file. However, this would have required two separate keys to be exchanged, one for encryption and another for signing. Another disadvantage was that every file transfer between a given pair of users would use the same key.

6. Addressing Security Challenges

6.1 KeySlinger Server Vulnerabilities

Section 2.1 introduced some security concerns for the KeySlinger server – eavesdropping by an outside attacker and data manipulation by the server. To prevent eavesdropping by an attacker sniffing packets, all communication with the server is done via SSL. The exact low-level details such as key exchange and encryption algorithms involved in setting up the SSL connection depend on the platform of the user's mobile device. Attacking user data by polling the server for a group's data is prevented by making the group ID a nonce that is infeasible to brute force. This is possible because users never actually have to enter the group ID (it is never even presented to the users), eliminating the concern for user error. Without the proper group ID the server will not provide any user data.

Another possibility for eavesdropping is that an outsider successfully joins a group. Because the exchange proceeds automatically after the user reports the lowest user ID, users may not realize this problem until the exchange is actually complete or someone's device times out, indicating they failed to join the group. To address this issue, the next release will include an additional layer of encryption using the "success nonce" as the encryption key. This value will be withheld from users until *every* user has indicated a matching word triplet and thus a successful exchange. This way even if an outsider joins the group, they will not be able to access the data because when it is time for the legitimate users to validate their exchanged data, they will notice that one or more members have not been involved in the exchange and indicate a protocol failure. As a result, the attacker will not be able to gain access to the proper decryption keys.

Any manipulation attempts by the server will be detected by the SPATE protocol itself.

6.2 StarSlinger Server Vulnerabilities

The StarSlinger server uses a random string (currently 20 bytes, possibly as short as 6 bytes in the final version if deemed sufficient) as the file identifier for each file. An attacker will only gain access to a file if they guess this identifier or compromise the server. While this probability is very low (2^{-48} even if only 6 bytes are used), the attacker has very few options even if he does succeed. The attacker will not have access to the contents of the file without the recipient's private key.

Another possible attack is spamming whereby an attacker uploads numerous junk files to the server. As the current server resides on Google App Engine, it has some level of defense against Denial of Service built in. Therefore taking the server down completely is not feasible without a very sophisticated DoS or DDoS attack. As for junk files accumulating on the server, these will get deleted automatically after 24 hours and they are expensive in terms of the attacker's bandwidth as well, making this an unattractive attack to begin with. The other aspect of this Denial of Service attack, if performed, concerns the user. If an attacker targets a specific user with junk files, the victim can be forced to download unwanted files. However, this can be managed with a combination of whitelist and blacklist approaches. Only people in a user's address book with whom the user has performed a key exchange will be able to send files. Transfer requests from outsiders will be ignored. As for a valid sender attempting a spam attack, a blacklist can be constructed to block certain senders in a user's address book. These measures are currently under development. Both will be accomplished using the initial chunk of the Merkle hash tree of the encrypted file data, which contains sender information.

6.3 Key Revocation and Update

If it becomes necessary for a user to revoke or update his public keys, this will be accomplished as a simple StarSlinger exchange. A message (in the form of a file) instructing receivers to remove or update a specified user's keys for a certain app will be signed using the user's StarSlinger private key and sent to all relevant parties. Upon receiving this message, the specified keys for the sender will be removed or updated. In the case that a revocation message and update message for the same key arrive close together in time, the revocation message will take priority. This situation can occur if an attacker gains access to a user's private key to send an update message and the user sends a revocation message for the same key. While neither choice can be correct for all situations, the worst case outcome of a revocation message is that users will need to perform another key exchange, while accepting a malicious update message can lead to accepting spoofed data.

7 Performance

KeySlinger performance depends on the number of users and the strength of the wireless connections on the participating devices. The following results were obtained with one tester operating 2~7 devices.

Number of Devices	Completion Time (sec)
2	21
3	24
4	35
5	40
6	42
7	45

Table 1. KeySlinger performance testing

The completion time measures the time from the first user pressing the "Begin Exchange" button to the last user confirming the word list. These results do not necessarily reflect real world situations because one person had to operate all devices. While it is reasonable to assume larger groups will take longer, the differences in completion times will likely be much shorter as the majority of the additional time in testing of larger groups was spent moving between devices, something that is not necessary if multiple users are operating their respective devices in parallel.

Performance of StarSlinger is difficult to measure because there are so many variables: the time it takes to encrypt and upload a file, the delay in the arrival of the notification message on the receiver's device, the time it takes for the receiver to act on the notification, and the time it takes to download and decrypt the file. Encryption/upload and decryption/download of a 500KB file each took approximately 3 seconds. The delay in delivery of the push notification message depends on the state of Apple's (or Google's) servers.

8. Future Work

While these applications currently exist separately as stand-alone applications, they will eventually be molded into a single application under the name StarSlinger because it was deemed inconvenient for the user to have to download two separate applications when their functions are so closely intertwined. While KeySlinger performs the initial exchange based on physical collocation, StarSlinger is later used for key updates or revocation. Thus it makes more sense for there to be a single application with multiple interfaces that handle specific situations. Another problem with having two separate apps is that users can get confused as to which app performs which function. The functionality of KeySlinger will be used to establish StarSlinger RSA public keys between people who have met for the first time. Once these keys are in place, third party app keys can be exchanged as files sent using StarSlinger, which provides the necessary authenticity and integrity. The interaction between third party apps and StarSlinger will remain transparent: an app will use an API call to launch StarSlinger to perform a key exchange. StarSlinger will then determine whether it has the necessary RSA keys with the other party or parties. If not, it will notify the user that a physical exchange (the current KeySlinger app) is required and proceed with this. If the necessary public keys are already established, the third party app keys will be exchanged as encrypted and signed files of a known format. The receiving device will process these to save the relevant contact information into its address book.

Once concern about using the phone's address book to store user data for StarSlinger is that the data can be edited in many ways, not always initiated by the user. Both iOS and Android have options to wirelessly sync address books to cloud-based services such as Google Contacts. This can occur in the background without the user being aware. This can be problematic because

during the sync process third party app keys, which are written as custom IM "services" and therefore might not be recognized by the contact sync service in question, can be corrupted (testing showed that sometimes syncing to Google Contacts overwrote custom IM provider names to AIM or Google Talk) or deleted altogether. This problem remains in the case of a manual sync to a user's computer. Data residing on a user's phone does not get corrupted in this case, but if the user were to lose their phone and later attempt to restore their contacts on a new phone from the information backed up on the computer, the custom IM fields may be ignored. Even without syncing with other sources the phone's address book is quite volatile because any app can gain read/write access to it. Third party key data can be altered or deleted, either by accident or by a malicious app. To address this problem, the next release of KeySlinger (or StarSlinger if integration is completed by then) will include a "history" feature which provides an overview and hash of the data exchanged during past exchanges. By comparing current data against these records, the user can verify a contact's public key information before deciding to use it.

9. Conclusion

This paper introduced two smartphone applications, KeySlinger and StarSlinger. These address the issues of secure key exchange in the absence of a trusted PKI and secure file transfer between mobile devices, respectively. The verification techniques used in KeySlinger can detect attempts at data manipulation and user privacy is protected with SSL and encryption using a value that is not disclosed until the very end of a successful exchange. StarSlinger uses the OpenPGP message format to guarantee authenticity, integrity, and secrecy of data that is transferred between two users. In the future these will be integrated into a single application under the name StarSlinger to avoid the inconvenience of multiple downloads and user confusion.

10. References

[1] A. Perrig, J. M. McCune, A. Studer, G. Mezzour, M. Farb, J. Lee, "KeySlinger," http://www.cylab.cmu.edu/keyslinger/.

[2] Y. Lin, A. Studer, Y. Chen, H. Hsiao, L. Kuo, J. Lee, J. M. McCune, K. Wang, M. Krohn, P. Lin, A. Perrig, H. Sun, and B. Yang, "SPATE: Small-group PKI-less Authenticated Trust Establishment," *IEEE Transactions on Mobile Computing*, Volume 9, Issue 12, 2010.

[3] "Android hits 300,000 daily activation milestone", http://venturebeat.com/2010/12/09/android-activations-300k/.

[4] "India still wants BlackBerry access but ban unlikely," http://news.cnet.com/8301-1009_3-20030189-83.html.

[5] "Nokia Joins BlackBerry In India User Monitoring Game," http://www.phonesreview.co.uk/2011/04/18/nokia-joins-blackberry-in-india-user-monitoring-game/.

[6] "DroidDream Becomes Android Market Nightmare," http://www.pcworld.com/businesscenter/article/221247/droiddream_becomes_android_market_nightmare.html.

[7] "How To Hijack 'Every iPhone In The World'," http://www.forbes.com/2009/07/28/hackers-iphone-apple-technology-security-hackers.html.

[8] "Apple "Investigating" JailbreakMe PDF Exploit," http://www.cultofmac.com/apple-investigating-jailbreakme-pdf-exploit/53700.

[9] "SMS Spoofing," http://www.smsspoofing.com/.

[10] "PGP Word List," http://en.wikipedia.org/wiki/PGP_word_list.

[11] "OpenPGP Message Format," http://www.ietf.org/rfc/rfc2440.txt.

[12] Linksky, J. et al, "Simple Pairing Whitepaper, revision v10r00," http://www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf , August 2006.

[13] V. Lortz, D. Roberts, B. Erdmann, F. Dawidowsky, K. Hayes, J. C. Yee, and T. Ishidoshiro, "Wi-Fi Simple Config Specification, version 1.0a," February 2006, now known as Wi-Fi Protected Setup.

[14] S. Vaudenay, "Secure communications over insecure channels based on short authenticated strings," in *Advances in Cryptology (Crypto)*, 2005, pp. 309–326.

[15] M. T. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Uzun, "Loud and clear: Human-verifiable authentication based on audio," in *International Conference on Distributed Computing (ICDCS)*, 2006, p. 10.

[16] C. Ellison and S. Dohrmann, "Public-key support for group collaboration," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 4, pp. 547–565, 2003.

[17] A. Perrig and D. Song, "Hash visualization: A new technique to improve real-world security," in *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99)*, M. Blum and C. H. Lee, Eds., Jul. 1999, pp. 131–138.

[18] C. Castelluccia and P. Mutaf, "Shake them up! a movement-based pairing protocol for cpu-constrained devices," in *Proceedings of ACM/Usenix MobiSys*, 2005.

[19] J. Lester, B. Hannaford, and B. Gaetano, "Are you with me? - using accelerometers to determine if two devices are carried by the same person," in *Proceedings of Pervasive*, 2004.

[20] L. E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl, and H.-W. Gellersen, "Smart-its friends: A technique for users to easily establish connections between smart

artefacts," in *Proceedings of Ubicomp*, 2001.
[21] M. Cagalj, S. Capkun, and J.-P. Hubaux, "Key agreement in peer to-peer wireless networks," *IEEE (Special Issue on Cryptography)*, vol. 94, pp. 467–478, 2006.
[22] D. Balfanz, D. Smetters, P. Stewart, and H. Wong, "Talking to strangers: Authentication in ad-hoc wireless networks," in *Proceedings of the 9th Annual Network and Distributed System Security Symposium (NDSS)*, 2002.
[23] NFC Forum, "NFC Forum: Specifications," http://www.nfc-forum.org/specs/.
[24] ——, "BEDA: Button-enabled device association," in *International Workshop on Security for Spontaneous Interaction (IWSSI)*, 2007.
[25] C.-H. O. Chen, C.-W. Chen, C. Kuo, Y.-H. Lai, J. M. McCune, A. Studer, A. Perrig, B.-Y. Yang, and T.-C. Wu, "GAnGS: Gather Authenticate 'n Group Securely," in *Proceedings of the ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, Sep. 2008.
[26] "Bump," http://bu.mp/.
[27] "Popular data exchange app "Bump" suffers security lapse," http://www.scmagazineus.com/popular-data-exchange-app-bump-suffers-security-lapse/article/190426/.
[28] "Hoccer," http://hoccer.com/.
[29] "TextSecure," http://www.whispersys.com/.
[30] "Apple Q&A on Location Data," http://www.apple.com/pr/library/2011/04/27location_qa.html.
[31] G. Mezzour, A. Studer, M. Farb, J. Lee, J. McCune, H. Hsiao, and A. Perrig, "Ho-Po Key: Leveraging Physical Constraints on Human Motion to Authentically Exchange Information in a Group," CMU-CyLab-11-004.