LoKey: Leveraging the SMS Network in Decentralized, End-to-End Trust Establishment

Anthony J. Nicholson[1], Ian E. Smith[2], Jeff Hughes[3], and Brian D. Noble[1]

[1] University of Michigan
{tonynich, bnoble}@eecs.umich.edu
[2] Intel Research, Seattle
ian.e.smith@intel.com
[3] University of Washington
jeffdh@cs.washington.edu

Abstract. People increasingly depend on the digital world to communicate with one another, but such communication is rarely secure. Users typically have no common administrative control to provide mutual authentication, and sales of certified public keys to individuals have made few inroads. The only remaining mechanism is key exchange. Because they are not authenticated, users must verify the exchanged keys through some out-of-band mechanism. Unfortunately, users appear willing to accept any key at face value, leaving communication vulnerable. This paper describes *LoKey*, a system that leverages the Short Message Service (SMS) to verify keys on users' behalf. SMS messages are small, expensive, and slow, but they utilize a closed network, between devices—phones—that are nearly ubiquitous and authenticate with the network operator. Our evaluation shows LoKey can establish and verify a shared key in approximately 30 seconds, provided only that one correspondent knows the other's phone number. By verifying keys asynchronously, two example applications—an instant messaging client and a secure email service—can provide assurances of message privacy, integrity, and source authentication while requiring only that users know the phone number of their correspondent.

1 Introduction

People increasingly depend on the Internet for daily interactions with others. We send email instead of letters, send digital pictures rather than prints, and pay bills online rather than write and mail checks.

The financial sector of our digital lives has at least a modicum of protection and security. Businesses have certified public keys [1], and use SSL [2] to provide reasonable authentication of a service to its users. Of course, such services are still vulnerable to phishing [3], DNS spoofing [4], and users' apparent willingness to accept any certificate presented as valid, no matter how problematic [5].

Unfortunately, person-to-person communication remains largely vulnerable. Secure email has made few inroads, and many messaging systems provide no security model at all. There are several structural reasons for this. Family members,

K.P. Fishkin et al. (Eds.): PERVASIVE 2006, LNCS 3968, pp. 202–219, 2006.

friends, and colleagues often have no central point of administrative control, making mutual authentication based on third-party services, such as Kerberos [6], impossible. Furthermore, individuals have little conscious incentive to purchase their own certified public keys.

The only remaining model is key exchange. The essential weakness of this model is that exchanged keys are *unauthenticated*—the users have no idea if the key they have is the correct key, or if some attacker has replaced it with one of their own choosing.

To confirm the veracity of a key, users are expected to verify it using some out-of-band mechanism. For example, they can call one another on the phone, and compare their *key fingerprints*. Typically, such fingerprints are long sequences of digits. In practice, users rarely verify keys out of band and tend to accept whatever keys are presented to them [7], though there has been work on more user-friendly verification techniques [8, 9, 10].

The unique properties of pervasive and mobile computing devices exacerbate this problem. Users would clearly like the devices they carry with them to communicate securely with the ever-changing and expanding set of users and devices encountered in their everyday travels.

Rather than rely on users to manually verify potentially compromised keys, we have constructed a system, called *LoKey*, that exploits the Short Message Service [11] to verify keys on users' behalf. There are several advantages to SMS. It utilizes a closed network, making internal attacks more difficult. The end user devices—phones—are authenticated by network operators, nearly always connected, and rarely out of their users' possession. These facts together allow us to construct an out-of-band channel between two corresponding users' machines; if a key can be verified by such a channel, it can be used with high confidence.

LoKey removes the need for users to trust any party beyond the person they want to communicate with and their phone service provider—no certificate authorities, public key infrastructures, or third-party intermediaries. Instead, it leverages the security properties of a network of limited usefulness (the SMS network) to secure traffic on the insecure, but vastly more powerful, Internet.

However, there are also challenges in using SMS—messages must be small, cannot use the full symbol space, have very high latency, and are expensive to transmit. Furthermore, the SMS service must be used judiciously, else the phone's battery will be expended too quickly.

This paper describes LoKey's approach to meeting these challenges, while still establishing a valid key. First, we establish a secure association between each user's phone and their computing devices. Thereafter, a user can establish a secure relationship with another knowing only the correspondent's mobile phone number—something users are already accustomed to doing. The calling user initiates key establishment with the correspondent via SMS, the two exchange keys via the Internet, and confirm the veracity of those keys via SMS.

The established key can either be used directly as a symmetric key between two users, or used as a session key to provide an authenticated channel. Such channels can be used to distribute public keys, group keys, et cetera, in a reliable

and authenticated way. LoKey is a general framework for trust establishment that is agnostic to the specific key technologies in use.

In addition to describing our system, we also present two sample applications that use LoKey. The first is an instant messaging client, that establishes a secure session key when a new correspondent is added for the first time. The second is a plug-in for the Mozilla Thunderbird email client. This tool lets email users swap their public keys in an authenticated way, in order to send and receive encrypted email.

Two users can exchange and verify a key in just over 30 seconds; this time is dominated by SMS message latency. While this is too expensive for on-demand, synchronous communication, it is acceptable for asynchronous or one-time tasks, such as signing a piece of email before it is sent for delivery or adding a user to a list of correspondents for future use. Half of this cost is incurred to initiate key establishment via an SMS message, rather than over the IP Internet. This is an expensive feature, but an important one for mobile clients, as it prevents DNS-based attacks.

2 Background: Why Key Establishment Is Hard

Key establishment is trivial when all users belong to one administrative domain. For example, all employees in a department trust their system administrator implicitly. This allows Kerberos-style authentication [6] between employees who have a common trusted entity (the sysadmin), but is obviously impractical for establishing trust across the global Internet.

The Diffie-Hellman key establishment protocol [12] lets two users with no previous relationship establish a key in a way that is secure against eavesdroppers. Alice and Bob each generate a random integer (a and b) and exchange ($g^a \bmod n$) and ($g^b \bmod n$), where g and n are public protocol parameters. The key is ($g^{ab} \bmod n$). Alice can calculate this, knowing a and having received ($g^b \bmod n$) from Bob, because $(g^b \bmod n)^a = g^{ab} \bmod n$. Bob can do likewise because $(g^b \bmod n)^a = g^{ab} \bmod n$. But an eavesdropper cannot calculate the key feasibly, because deriving a and b from $\{(g^a \bmod n), (g^b \bmod n), g, n\}$ is intractable [13].

Unfortunately, the protocol is vulnerable to active attacks. If Mal can remove and insert messages as shown in Fig. 1, she can force Alice and Bob to unwittingly establish keys with her instead of each other. As long as Mal tunnels all traffic between Alice and Bob, they cannot detect the attack. SSL [2] and the Station-to-Station protocol [14] both solve this problem by requiring users to have certified public keys. In this model, all users must trust a small number of certification authorities (CAs) [1], whose public keys are broadly distributed and well-known. Alice and Bob then sign all their key establishment messages to each other, and can detect if a message originated from someone else.

But how do users get these key certificates? This places an unreasonable burden on users to find a secure side-channel with the CA, such as physically visiting a key signing kiosk. There is little incentive for users to do this. For example, most merchants authenticate individuals by their credit cards. Many

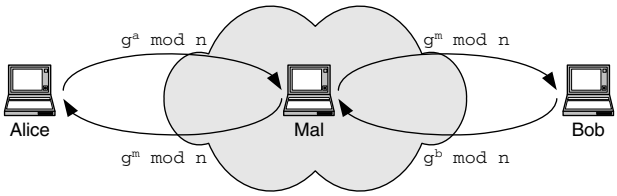

Fig. 1. Vulnerability of Diffie-Hellman to man-in-the-middle attacks

institutions also self-sign their certificates, acting as their own CA. This leads to a balkanization of the world into islands of trust. While there has been work toward bridging this gap [15], each trust domain must still somehow establish trust with each other or with a hierarchical set of CAs. SSL is also vulnerable to "DNS hijacking" attacks [16]. While users are informed the host key has changed when such an attack occurs, too often they are conditioned to just click "OK" on every security alert message [5].

At the other end of the spectrum lies the decentralized PGP "web-of-trust" model. Users sign the keys of others whom they trust, or keys that they receive over a secure side-channel. When Alice receives Bob's public key for the first time, she accepts it as valid if someone she already trusts has signed it, attesting to its integrity. In order to sign someone's key, one needs to receive or verify the key out-of-band to preclude man-in-the-middle (MiM) substitution attacks. In practice, users rarely verify keys out of band and tend to accept whatever keys are presented to them [7]. Unfortunately, these signature chains are only as strong as their weakest link. Furthermore, two users cannot communicate unless they have at least one trusted user in common.

The end result is the situation we have today, where each time we converse with a new correspondent, we are presented with a key fingerprint that we have no intention of verifying. While much work has focused on making key verification easier [7, 8, 9, 10], in general such techniques have not yet made the transition to practice. Meanwhile, most users just click "OK", accept any certificate or key which is presented to them, and go about their business [5, 7].

3 Design

The crucial point of the previous section is that Diffie-Hellman exchanges are sufficiently secure to establish pairwise trust between users, provided that the man-in-the-middle problem could be solved. This is possible if there was a trusted out-of-band channel Alice and Bob were willing to use to verify that their keys match. We argue that one such out-of-band channel already exists: the Short Message Service (SMS) network used to send text messages between mobile phones. This network has the following nice properties:

– It is a closed network. To complete a MiM attack, one must remove messages which are in transit. This requires access to phone company resources or the ability to masquerade as a network tower.

- Phone companies have a strong economic incentive to secure their network, to avoid customers defecting *en masse* to competitors.
- Users are universally identified by their phone number, a paradigm people already understand.
- SMS messaging is already standard on most phones, and all signs point toward increasing adoption [17].

We are not claiming attacks against the SMS network are not possible, but rather argue the bar is much higher than what is required for similar attacks against Internet traffic. A MiM attack against the SMS network would require coordinating radio eavesdropping with intrusion into at least one phone network.

Unfortunately, the SMS network has limitations. Each message holds at most 160 bytes. Delivery time is slow and variable, and typically has a per-message charge. Performing the entire key establishment over SMS would take dozens of messages, last prohibitively long and run up users' bills.

LoKey leverages the strengths of both the Internet and the SMS network to establish a secret key between two users without requiring they start from any shared secret, use certified keys, or both trust any other entity—in an efficient and user-friendly way. After establishing a secret key using standard Diffie-Hellman key exchange, Alice and Bob each calculate a cryptographic hash of their key, and send this hash to their mobile phone. The phones swap these hashes via SMS messages, and then download the other party's hash to its user's computer. LoKey then checks if the hashes match—if so, Alice and Bob know with a high degree of confidence that their key is genuine.

Cryptographic hash functions are ideal for verifying keys because they map an arbitrary-length key to a small, fixed number of bytes [18]. Given this hash, it is infeasible to discover the key from whence it came, or to construct another key which will hash to the same value. We use SHA-256 [19], which outputs a 32 byte hash. The hash can be exposed to eavesdroppers because it gives them no advantage toward reconstructing the secret key.

Key verification may take several seconds, due to SMS delivery latency. Users may be unwilling to incur this overhead every time. LoKey can either cache secret keys or, preferably, leverage public key cryptography to make this a one-time cost. After establishing and verifying a secret key, LoKey swaps Alice and Bob's public keys under the cover of the secret key. They can then generate a session key using any number of protocols which rely on certified keys [14].

The user can secure the communication channel between her phone and her computer in several ways, since she controls both devices. We assume the phone and computer communicate via Bluetooth. Her phone remains in her pocket, and the entire process is user-transparent, apart from the one-time task of pairing her phone and computer [20]. To pair two devices, the user chooses a variable-length PIN and manually inputs it on both devices. The devices then negotiate a secret key, using the shared secret of the PIN to thwart MiM attacks. Recent work [21] demonstrated vulnerabilities in this pairing process, but only against PINs less than 8 digits long. Since the standard supports up to 128-bit PINs, this is not a blanket indictment of the pairing protocol but rather an implementation issue.

3.1 Threat Model

We assume the attacker Mal is an active attacker who can remove, insert, and modify messages in flight anywhere in the Internet. She may even completely control one or both users' access points to the Internet. With regard to the SMS network, we assume that Mal can eavesdrop on all text messages sent and received by both Alice and Bob, but that she cannot remove or modify SMS messages.

We also assume Mal cannot eavesdrop on the Bluetooth channel between the user's computer and phone. As we argue above, the pairing protocol is secure, given sufficiently-long PINs.

It must be noted that in current GSM technology, phones authenticate themselves to the network tower, but the converse is not true [22]. An attacker could therefore masquerade as a GSM tower, trick the user's phone into associating with it, and then act as the man-in-the-middle between the user and a legitimate phone company tower. The attacker could then modify key hashes as appropriate to conceal his presence. This requires specialized hardware, but is not beyond the capacity of organized crime, law enforcement, and national governments.

Emerging standards (specifically, 3GPP) will preclude this sort of attack. For the time being, users can leverage their mobility to re-confirm hashes from multiple locations via multiple network towers. Our previous work [23] establishes an insecure key between two users over the Internet, like LoKey, but then exchanges key hashes also over the Internet. In lieu of an out-of-band channel (such as the SMS network), this system repeatedly rebroadcasts key hashes over the different access points the two users encounter in the course of their ordinary, daily travels. Our initial results show such mobility ensures, with high probability, that multiple path-diverse routes between the two users will be generated, requiring an attacker to control an unreasonably large portion of the Internet in order to conceal his presence. We argue that adding such capabilities to LoKey would similarly thwart such "dummy tower" attacks. We have not implemented this in our prototype, however.

3.2 Protocol Design

Figures 2 and 3 illustrate our protocol for establishing a secret key between two users. It is a two-phase process. First, the two users perform standard Diffie-Hellman key establishment over the Internet. Second, the key is verified via the SMS network.

Consider an example scenario where a user Bob wants to establish a key with another user, Alice:

Insecure Key Establishment (Fig. 2):

1. Bob's laptop sends a key establishment request message to his phone, via Bluetooth. This request contains both Bob's IP address and Alice's mobile phone number.

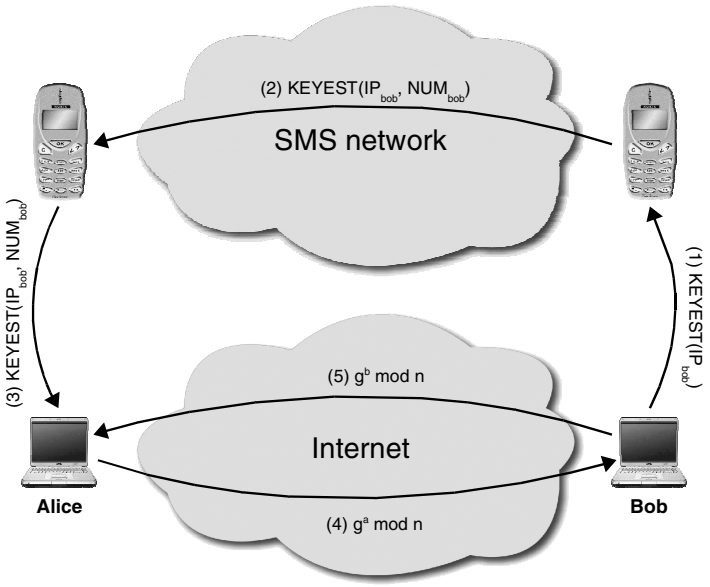

Fig. 2. Phase 1: Initiation and insecure key establishment

2. Bob's phone encapsulates the request in an SMS message payload, and sends it to Alice's phone via the SMS network.
3. Alice's phone receives the request, and forwards it to Alice's laptop.
4. Alice initiates Diffie-Hellman key establishment, by calculating a pseudorandom integer a and sending the integer $(g^a \bmod n)$ to Bob over the Internet. Bob generates his pseudorandom parameter b, and calculates the key: $K_B = (g^a \bmod n)^b = g^{ab} \bmod n$.
5. Bob sends Alice $(g^b \bmod n)$, and she also calculates the key: $K_A = (g^b \bmod n)^a = g^{ab} \bmod n$.

Note: if there was a man-in-the-middle attack, then $K_A \neq K_B$. Otherwise, they are the same. The next phase of the protocol determines which case it is.

Key Verification via SMS (Fig. 3):

6. Both Alice and Bob calculate the SHA-256 cryptographic hash of their key. They then send the hash, and the other party's mobile phone number, to their phone via the secured Bluetooth link.
7. Alice's phone sends `SHA-256(`K_A`)` to Bob's phone in an SMS message. Bob's phone likewise sends `SHA-256(`K_B`)` to Alice's phone.
8. Once each phone receives the other's text message, it downloads the hash and the sender's phone number to its paired computer. Both Alice and Bob check if `SHA-256(`K_A`)` = `SHA-256(`K_B`)`. If not, then the key establishment failed and LoKey discards the key. If they match, Alice and Bob know with high confidence that $K_A = K_B$.

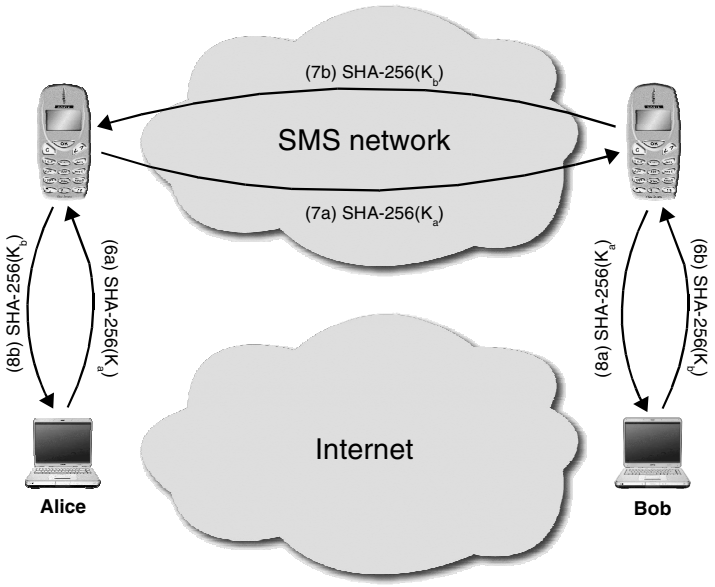

Fig. 3. Phase 2: Key verification via SMS

A lightweight version of the protocol omits steps 1-3 in cases where Bob already knows Alice's IP address. In that case, he can initiate Diffie-Hellman establishment directly. This saves the overhead of one SMS message delivery latency (step 2 of the above protocol).

3.3 Discussion

Usability and Privacy. To establish a key with another user, all we need to know is their mobile phone number. This is attractive since users are already accustomed to identifying people by their phone numbers. While remembering a phone number is more difficult than remembering an email address (numbers rather than names and words) we argue it is not unreasonable. Phone numbers are also easy to communicate out-of-band since they are short strings of digits, and people are already accustomed to doing so.

LoKey fails if someone else has the user's phone, because we will establish a key with the attacker and his laptop, rather than the user we intended. We argue that the window of opportunity between when a phone is lost and its owner cancels service or recovers it will be short—at most, on the order of one day. Since users typically pay per usage, there is a strong incentive for them to quickly stop unauthorized use.

LoKey raises some privacy concerns, however. In the above example, Alice may not want to disclose her IP address to Bob. Automatically initiating Diffie-Hellman key establishment in response to Bob's SMS message does just that. While an IP address doesn't provide GPS-level information, it can reveal a user's presence on a certain university campus, or at least in a certain city. We resolve

this tension between privacy and usability by providing the user with the phone number of the person requesting key establishment. Since users opinions on privacy vary [24], we empower users to decide their own privacy policies. LoKey users can activate a privacy option, so that when the user's phone receives a key establishment request, it displays the phone number of the requester on the mobile phone. The user then allows or denies the request via her mobile phone keypad. A whitelist of pre-approved users prevents common requests from annoying the user.

For even more privacy, users could use an anonymous routing system to hide their IP address from others. For example, a *tor* (`http://tor.eff.org/`) uses onion routing to redirect packets through a set of overlay peers, each of whom only know the identity of the source immediately preceding it in the sequence. Once packets arrive at their destination, only the identity of the most recent node in the overlay can be ascertained.

Network Address Translation (NAT). Network address translators (NATs) multiplex one public IP address across a number of private IP addresses. This is often used by wireless routers to share one DSL or cable modem, causing problems in establishing point-to-point IP connections. To solve this, we leverage the well-known technique of "hole-punching" [25, 26]. Each host maintains a connection with a well-known rendezvous server (RS), which determines the host's globally-visible IP address and port pair. When requesting key establishment via SMS, a host which is behind a NAT sends both its local IP address (behind the NAT) and the globally visible IP address and port. When Alice then establishes a TCP connection to Bob (to initiate Diffie-Hellman key establishment) on his global IP and port, the existing outbound TCP connections to the RS are broken, and Alice and Bob are connected directly. Full details can be found in the literature [25, 26].

Clearly, the channel between the RS and the user must be authenticated or Mal can trick users into connecting to the wrong user. An active attacker could pose as a rendezvous server and connect Alice to a third-party, Charlie, rather than the principal (Bob) with whom she intended to exchange keys. Charlie can then perform a man-in-the-middle attack by tunneling traffic between Alice and Bob, as described above in Sect. 2. One solution is to use LoKey to establish a secret key between Alice and the RS, then use this shared secret to authenticate all subsequent traffic between Alice and the rendezvous server.

Multiple User Devices. Users have multiple computing devices. If all of these are connected to the Internet then IP discovery is not a trivial one-to-one mapping. When your phone receives a key establishment request from a friend, and you have both your PDA and your laptop with you, which device should handle the key establishment? Should your 3G phone handle the entire process itself?

In our current implementation, we elide this issue by having users choose which device to associate with when they start the LoKey process on their mobile phone. Since a user owns and controls all her devices, she can establish

a shared secret which allows them all to communicate confidentially. Ongoing work is focused on extending LoKey to forward keys established and verified by one of the user's devices to all of the others. For example, if Alice's PDA was her only device in contact with her mobile phone when Bob tried to establish a secret key, it would handle the key establishment and then transmit the key to her laptop, desktop, et cetera, the next time they were reachable via the network.

4 Implementation

We developed a working prototype of LoKey, consisting of two main components: a *service daemon*, running on the user's computer, and an *SMS bridge*, running on the user's phone. The computer and phone communicate via Bluetooth.

4.1 Service Daemon (SD)

The SD is a user-level service on the user's computer, listening in the background for incoming data on one of three connections: (1) a well-known, local TCP port number to which user applications connect to request services from the SD, (2) an externally-visible socket on another well-known LoKey port, to which SDs running on other users' computers connect, and (3) the computer's Bluetooth stack creates a pseudo-device which behaves like a normal RS-232 serial port. The SD listens on that port for incoming data from the user's mobile phone.

We developed the SD in C++, using the OpenSSL crypto library for Diffie-Hellman establishment to leverage its optimized implementation. The SD has been ported to both Windows XP and Linux.

4.2 SMS Bridge

We implemented the SMS bridge process as a Python script, running on a Nokia 6600 mobile phone. The 6600 runs the Symbian operating system with the Nokia Python runtime library. When the script starts, it presents the user with a list of Bluetooth devices in the area. The user chooses her computer from the list, and pairs the phone with the computer if she has not done so previously. The script now retreats into the background—the user can make calls and use all other phone features, even while key verification messages are passing back and forth.

The SMS bridge consists of two threads of execution. An upstream thread listens on the Bluetooth serial port for data from the user's computer. These messages consist of a phone number and a payload. The upstream thread parses requests from the computer and sends the payload to the specified number as an SMS message. A downstream thread waits for a text message to arrive in the phone inbox. If it is a LoKey message rather than a text generated by a human, it removes the message from the inbox and sends the payload and the sender's phone number to the user's computer. LoKey messages are identified by a special control code to prevent users from accidentally deleting them while manually sending SMS messages.

4.3 Application Services

Regardless of how useful LoKey may be, if application programmers cannot easily use its services, it will be abandoned. This is why we pushed the complexity of interacting with mobile phones and other users down into the service daemon. Applications merely request one of the following two services by sending a request over the local LoKey socket and waiting for a response. We implemented this interface as a socket rather than using IPC or named pipes to both maximize portability and increase ease of use, since the Berkeley socket interface is a common, simple abstraction that most programmers already understand.

Secret Session Key Establishment

```
key = est_and_verify_key( remote_phone )
```

An application sends the SD a message containing the phone number of the user with which they want to establish a secret key. The SD then performs key establishment as described in Sect. 3.2. Once the key has been established but before it has been verified, the SD returns the key to the calling application. Once the key has subsequently been verified or disproved, the SD returns an appropriate success code. If the hashes matched, the application can now establish a secure connection to its peer in confidence, using any symmetric cryptographic cipher of its choosing.

Authenticated Public Key Exchange

```
remote_PK = auth_pk_swap( remote_phone )
```

Since establishing and verifying a secret key can take tens of seconds, users will want to make this a one-time cost. Caching the key accomplishes this, but at the cost of exposing the pairwise key if one user's computer is stolen or compromised. A better solution is to use the secret key LoKey provides as a one-time session key, and leverage public key cryptography to swap both users' public keys in a completely authenticated fashion. They can subsequently establish a session key entirely over the Internet via any number of well-known methods [14] that require certified keys (since both public keys are completely trusted). Our prototype uses Gnu Privacy Guard (GPG), a PGP-style system, to manage public keys locally.

Applications send a phone number to the SD, which first establishes and verifies a secret key. Alice and Bob then swap copies of their public keys, encrypted by the secret key. The keys are ASCII-armored for transport, which adds a standard PGP header to the key. This is critical for verification, since an attacker cannot just send an arbitrary message of the correct length to a user, and trick her into importing a bogus public key. If the data sent was not encrypted by the secret LoKey key it will decrypt to gibberish, without the correct PGP header formats. Thus, users only accept keys which originate from each other.

5 Example Applications

Along with our LoKey prototype, we developed two examples to illustrate how applications can leverage LoKey to enhance user security and usability.

5.1 Instant Messaging: Jabber

Jabber is an open-standard Internet chat protocol, also known as XMPP. We used the xmpppy open-source Python library to write a Jabber client. LoKey-Jabber is an ordinary IM client, with one exception: each time the user adds a new contact to her "buddy list", LoKeyJabber establishes and verifies a key with that buddy. We do this immediately rather than on demand, because it may take several seconds due to SMS latency.

When Alice first imports Bob into her buddy list, she specifies both his chat handle and his mobile phone number. His name first shows in the buddy list as red, indicating Alice has no key with him. LoKeyJabber immediately starts the key establishment process. Meanwhile, Alice and Bob are free to communicate without a key, if they wish. Once the key has been established over the Internet, but before it has been confirmed via SMS, the SD returns the key to LoKeyJabber. Bob's name now turns yellow, because LoKeyJabber has a key with him which may or may not be trustworthy. Our implementation uses AES (Rijndael) [27] symmetric encryption, with 128-bit keys, to secure chat messages. Meanwhile, the SD verifies key integrity via hash exchange. This may take on the order of 20 seconds or more. During this time, Alice and Bob can communicate *provisionally* using the unverified key, if what they need to say is not particularly confidential. Once the key has been confirmed, each party's SD returns the result to their LoKeyJabber. Bob's name turns green and Alice knows she can communicate with him in full confidence. Confirmed keys are cached, so key establishment is a one-time cost.

5.2 Email Client: Mozilla Thunderbird

One arena in which public key cryptography has made some inroads is email. Basic encryption/decryption is standard in many email clients. As we have observed, the problem is distributing everyone's public key over the insecure, unauthenticated Internet. EnigMail, a third-party plug-in to the Mozilla Thunderbird email client, is a graphical front-end to PGP. When users compose messages, they choose a user's public key from a provided list, and EnigMail encrypts the message with that key. Likewise, EnigMail automatically decrypts received messages with the user's private PGP key.

We extended EnigMail to add an option to swap public keys with a user, given their phone number. Assume Alice wants to swap keys with Bob. First, Alice's Thunderbird requests authenticated public key exchange with Bob. If the SD returns success, then Bob's public key is now on Alice's PGP keyring, and Alice's public key is on Bob's keyring. If Alice now composes an email to Bob, she will see his public key on the list of possible recipients, since EnigMail also uses the PGP keyring.

6 Evaluation

In evaluating our implementation of LoKey, we sought to answer three questions:

1. What is the user-perceptible time overhead imposed by using LoKey for authentication?
2. What are the reasons for this overhead? Are we limited by SMS network latency, or some artifact of our design?
3. Since these are mobile, battery-powered devices, is LoKey's power consumption acceptable?

Our test setup consisted of two x86 laptops running Windows XP. Each had a 866 MHz CPU and 256 MB of RAM. The laptops were both connected to the same campus 802.11 wireless network. We created a GMail email address, PGP public key, and Jabber chat handle for two imaginary users (Alice LoKey and Bob LoKey). We paired a Nokia 6600 mobile phone with each laptop and ran the experiments inside an office to simulate the GSM network conditions users would commonly experience. Our GSM signal strength was good, typically at the high end of the scale. Note that SMS delivery latency will vary on different networks.

6.1 Application-Level Metrics

All numbers in this section are from the initiator—the user who started the chat or the public key exchange—because the time delay is longest for that user.

To test the overhead a user would see in using the LoKeyJabber chat client, we added Bob to Alice's buddy list 20 times. This triggered chat key establishment and verification. Table 1 shows the delay in seconds, averaged across all 20 runs. LoKey requires just over 30 seconds to establish and verify a key.

Similarly, we used our email client plug-in to swap Alice and Bob's public keys 20 times. As Table 2 shows, the time required for key establishment is

Table 1. LoKeyJabber chat key establishment. Values in seconds.

	Key establishment
mean	36.05
median	36.41
stdev	6.17

Table 2. Public-key exchange plug-in for Mozilla Thunderbird. Values in seconds.

	Total	Key establishment	Public key exchange
mean	34.17	29.61	4.30
median	34.45	30.06	4.07
stdev	2.01	1.90	0.54

comparable to that shown in the chat client test. An additional 4 seconds is required, on average, to securely exchange public keys under the cover of the new secret key. This overhead is primarily the result of communication overhead between LoKey and the user-level Gnu Privacy Guard client, incurred when importing a new user's key into a local keyring.

6.2 Infrastructure-Level Delays

We instrumented the Service Daemon and collected internal profiling information during all of the 40 test runs described above. Tables 3 and 4 show the breakdown of time spent in each phase of the key exchange and public key exchange protocols. As expected, SMS delivery delays comprise the overwhelming majority of overhead. One does see a small communication delay in key establishment for exchanging the Diffie-Hellman key material over the Internet. Even across a true WAN this delay is unlikely to be more than a few seconds. As discussed above, importing the exchanged public keys into the local GPG keyring incurs overhead of several seconds. All these local delays are still dwarfed by SMS network delays, however.

Table 3. Secret key establishment. Values in seconds.

	Total	Request key exchange	Diffie-Hellman exchange	Verify hashes
mean	35.84	17.83	0.14	17.87
median	36.33	18.02	0.11	18.53
stdev	6.22	2.55	0.48	3.99

Table 4. Secure public key exchange. Values in seconds.

	Total	Request key exchange	Diffie-Hellman exchange	Verify hashes	Public key exchange
mean	33.27	14.04	0.14	14.31	3.44
median	33.52	13.96	0.08	14.52	3.50
stdev	1.91	0.77	0.29	1.80	0.53

6.3 Power

Since both sending SMS messages and communicating with the user's computer via Bluetooth can be power-intensive for mobile phones, we sought to quantify the effect LoKey would have on battery life. To examine power drained during active operation, we started from fully charged batteries and performed secret key establishment from Alice to Bob 100 times. After the 100 runs, neither phone had dropped off the highest battery setting, meaning that at least 87.5% of the

battery remained. We did not run until the batteries drained because these are live, expensive SMS messages.

We believe that because the standby battery life of the phone is at most 10 days, users are unlikely to establish brand-new relationships with hundreds of users before recharging the phone at least once. Users' phone plans also typically budget only several hundred messages per month.

We also considered standby power consumption, because LoKey requires that the Bluetooth interface on the user's phone is always enabled, expending power while waiting for a connection. Bluetooth radios draw on the order of 0.3 mA while quiescent [28]. The Nokia 6600 battery is rated at 850 mAh (milli-amp hours) when fully charged. According to the 6600 user manual, the phone should last 150-240 hours in standby mode. Thus, the standby mode current draw of the phone must be in the range of $(850\ mAh)/(150\ h)$ to $(850\ mAh)/(240\ h)$, or 5.7 mA - 3.5 mA. We therefore expect LoKey will reduce standby time by at most 10%, by one day (from 10 to 9 days) in the worst case.

7 Related Work

Thompson describes a system whereby banks push secret PINs to users via text messages [29]. While the risk is small, one cannot consider the SMS network secure from eavesdropping. That is one reason we exchange cryptographic hashes. Claessens [30] uses SMS messages as a sort of receipt for online transactions. Both require that users read the message and manually perform some action, while LoKey is automatic.

Maher [31] first suggested using a short hash of a long key to ease key verification. This approach has been adopted by many others [32, 33], but all still require a good deal of user intervention—one must either connect the devices physically, read and verify hashes on two different screens, or manually input a code on several devices. Users seem resistant to all of these tasks.

These limitations have sparked work on helping users more easily verify fingerprints [8, 9, 7]. Unfortunately, few or none of these techniques have yet made the transition into everyday use. Perrig and Song [10] generate images from fingerprints, exploiting the fact that humans recall images much better than strings of letters and numbers. Similarly, Madhavapeddy et al. [34] suggested the voice channel of mobile phones could be used to verify keys through user to user communication. While we could modify LoKey to verify hashes in any of these ways, this would require user intervention, where our solution is automatic.

Stajano and Anderson introduced the "Duckling" [35]—a small device which performs actions on behalf of its "mother" device. This is similar to the role users' mobile phones play in LoKey. They are bound to the user's computer, and are trusted completely because the user controls both. Other work in the ad-hoc networking space has shown how capitalizing on such side channels when they arise can enhance security [36, 37].

8 Conclusion

As users are more mobile, they interact with a varied set of people and devices. Since the Internet is insecure and unauthenticated, we need to use data encryption to ensure confidentiality of our communications. Current methods for establishing end-to-end trust put too much burden on users, and demand trust in either a centralized authority or strangers on key signature trails.

We introduced LoKey, a decentralized, automatic system for generating end-to-end trust between users. LoKey uses standard Diffie-Hellman key establishment, and defeats the man-in-the-middle by leveraging the mobile phones that users already carry. By exchanging key hashes over the SMS network, LoKey detects after the fact if a MiM attack occurred. We provide users with a simple API for establishing secret keys and exchanging public keys in an authenticated fashion.

We developed two proof-of-concept applications to showcase our implementation. Evaluation of our prototype shows one-time delays of approximately 30 seconds to establish a secure communication channel with a remote user. By either caching the generated secret key or leveraging public key cryptography, this cost can be eliminated for future communications between the two users.

Acknowledgements

We would like to thank our shepherd Nigel Davies, and the anonymous reviewers, for their insightful comments and feedback that greatly improved the quality of our paper. We also gratefully acknowledge the helpful feedback of James Mickens and Sam Shah.

References

1. CCITT, Draft Recommendation X.509: The Directory-Authentication Framework. Consultation Committee, International Telecommunications Union, Geneva (1989)
2. Freier, A., Karlton, P., Kocher, P.: Secure Socket Layer 3.0. Internet Draft (1996)
3. Warner, B.: Billions of "phishing" scam emails sent monthly. Reuters News Service (2004)
4. Bellovin, S.M.: Using the Domain Name System for system break-ins. In: Proceedings of the 5th USENIX Security Symposium. (1995)
5. Xia, H., Brustoloni, J.C.: Hardening web browsers against man-in-the-middle and eavesdropping attacks. In: Proceedings of the 14th International World Wide Web Conference (WWW '05). (2005)
6. Neuman, B., Ts'o, T.: Kerberos: An authentication service for computer networks. IEEE Communications Magazine **32** (1994) 33–38
7. Whitten, A., Tygar, J.D.: Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In: Proceedings of the 8th USENIX Security Symposium. (1999)
8. Dohrmann, S., Ellison, C.: Public-key Support for Collaborative Groups. In: Proceedings of the First Annual PKI Research Workshop. (2002)

9. Garfinkel, S., Margrave, D., Schiller, J., Nordlander, E., Miller, R.: How to make secure email easier to use. In: Proceedings of the Conference on Human Factors in Computing Systems (CHI). (2005)
10. Perrig, A., Song, D.: Hash Visualization: A New Technique to Improve Real-World Security. In: Proceedings of the International Workshop on Cryptographic Techniques and E-Commerce (CryptEC). (1999)
11. Peersman, C., Cvetkovic, S.: The global system for mobile communications: Short Message Service. IEEE Personal Communications **7** (2000) 15–23
12. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory **6** (1976) 644–654
13. Maurer, U.: Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In: Proceedings of the 14th Annual International Cryptology Conference (CRYPTO '94). (1994)
14. Diffie, W., Oorschot, P., Wiener, M.: Authentication and Authenticated Key Exchanges. Designs, Codes, and Cryptography **2** (1992) 107–125
15. Kaminsky, M., Savvides, G., Mazieres, D., Kaashoek, M.: Decentralized User Authentication in a Global File System. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles. (2003)
16. Burkholder, P.: SSL Man-in-the-middle Attacks. The SANS Institute (2002)
17. Xu, H., Teo, H., Wang, H.: Foundations of SMS Commerce Success: Lessions from SMS Messaging and Co-opetition. In: Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS). (2003)
18. Naor, M., Yung, M.: Universal one-way hash functions and their crytographic applications. In: Proceedings of the 21st ACM Symposium on the Theory of Computing (STOC '89). (1989)
19. National Institute of Standards and Technology (NIST): Secure Hash Standard (SHS). National Technical Information Service (2002)
20. Bluetooth SIG: Specification of the Bluetooth System, http://www.bluetooth.org/spec/ (2005)
21. Shaked, Y., Wool, A.: Cracking the Bluetooth PIN. In: Proceedings of the Third International Conference on Mobile Systems, Applications, and Services (MobiSys '05). (2005)
22. Anderson, R.: Security Engineering. Wiley (2001)
23. Nicholson, A.J., Han, J., Watson, D., Noble, B.D.: Exploiting Mobility for Key Establishment. In: Proceedings of the Seventh IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '06). (2006)
24. Smith, I., Consolvo, S., Abowd, G.: Social Disclosure of Place: From Location Technology to Communication Practice. In: Proceedings of the Third International Conference on Pervasive Computing. (2005)
25. Biggadike, A., Ferullo, D., Wilson, G., Perrig, A.: NATBLASTER: Establishing TCP Connections Between Hosts Behind NATs. In: Proceedings of the SIGCOMM Asia Workshop. (2005)
26. Ford, B., Srisuresh, P., Kegel, D.: Peer-to-Peer Communication Across Network Address Translators. In: Proceedings of the USENIX Annual Technical Conference. (2005)
27. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. NIST (2000)
28. Fischer, K.: Bluetooth Wireless Technology. In: Proceedings of the IEEE EMC Wireless Workshop. (2000)
29. Thompson, K.: A Security Review of the ASB Bank Netcode Authentication System (2004) http://www.crypt.gen.nz/papers/asb_netcode.html.

30. Claessens, J., Preneel, B., Vandewalle, J.: Combining World Wide Web and Wireless Security. In: Proceedings of IFIP Network Security. (2001)
31. Maher, D.: Secure communication method and apparatus. U.S. Patent Number 5,450,493 (1995)
32. Gehrmann, C., Mitchell, C., Nyberg, K.: Manual Authentication for Wireless Devices. RSA Cryptobytes **7** (2004)
33. Hoepman, J.H.: The Ephemeral Pairing Problem. In: Proceedings of the 8th International Conference on Financial Cryptography. (2004)
34. Madhavapeddy, A., Sharp, R., Scott, D., Tse, A.: Audio Networking: The Forgotten Wireless Technology. IEEE Pervasive Computing **4** (2005)
35. Stajano, F., Anderson, R.: The Resurrecting Duckling. In: Proceedings of the 7th International Workshop on Security Protocols. (1999)
36. Balfanz, D., Smetters, D., Stewart, P., Wong, H.C.: Talking to Strangers: Authentication in Ad-Hoc Wireless Networks. In: Proceedings of the Network and Distributed System Security Symposium (NDSS '02), San Diego, California, USA (2002)
37. Capkun, S., Hubaux, J.P., Buttyan, L.: Mobility Helps Security in Ad Hoc Networks. In: Proceedings of the Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '03), Annapolis, Maryland, USA (2003)