SafeSlinger: An Easy-to-use and Secure Approach for Human Trust Establishment

Michael Farb, Manish Burman, Gurtej Singh Chandok, Jon McCune, Adrian Perrig

December 22, 2011

CMU-CyLab-11-021

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

SafeSlinger: An Easy-to-use and Secure Approach for Human Trust Establishment

Michael Farb Manish Burman Gurtej Singh Chandok Jon McCune Adrian Perrig
CyLab, Carnegie Mellon University

ABSTRACT

Users regularly experience a crisis of confidence on the Internet. Is that email truly originating from the claimed individual? Is that Facebook invitation indeed from that person or is it a fake page set up by an impersonator? These doubts are usually resolved through a leap of faith, expressing the desperation of users.

To establish a secure basis for Internet communication, we propose SafeSlinger, a system leveraging the proliferation of smartphones to enable people to securely and privately exchange their public keys. Through the exchanged authentic public key, SafeSlinger establishes a secure channel offering secrecy and authenticity, which we use to support secure messaging and file exchange. Essentially, we support an abstraction to safely "sling" information from one device to another.[1] SafeSlinger also provides an API for importing applications' public keys into a user's contact information. By slinging entire contact entries to others, we support secure introductions, as the contact entry includes the SafeSlinger public keys as well as other public keys that were important. As a result, SafeSlinger provides an easy-to-use and understand approach for trust establishment among people.

1. INTRODUCTION

For many current Internet applications, users experience a crisis of confidence. Does the Facebook invitation we receive really originate from the claimed individual or was it created by an impostor? Is the email we received from the claimed individual or was it sent by a spammer?

Cryptography alone cannot address this problem. We have many useful protocols such as SSL or PGP for entities that already share authentic key material, but the root of the problem still remains: how do we obtain the authentic public key from the intended resource or individual? The global certification process for SSL is not without drawbacks and weaknesses [27, 34], and the usability challenges of decentralized mechanisms such as PGP are well-known [42].

The problem of human-oriented, trust establishment is fundamental; no amount of automation and "fail-safe" defaults can avoid the need for basic trust decisions to be made by humans (system administrators and ordinary users alike), since they ultimately assume the risks of digital communication, accessing remote sites, allowing remote access to their local resources, and employing other users' services.

Of course ordinary users can extensively rely on system administrators' help in making trust decisions. However, ordinary users inevitably face challenging decisions alone; most users at home, on travel, on vacation, or in small businesses do not benefit from skilled help. All this while the need and temptation to use new online services steadily increases.

The human-centric foundation of trust establishment makes this problem universally important; protocols and interfaces need to be designed for a diverse population of varying skills, interests, ages, and cultures. We postulate that usability is currently the major barrier for widespread adoption of cryptography. We observe a lack of well-designed security interfaces for current information and communication services. In addition to usability concerns, many systems do not provide the security that they advertise. Observing the lack of usable public key cryptography systems, Tim Berners-Lee has called upon security researchers and professionals to design a public key encryption system for the people [11].

The recent proliferation of smartphones offers a promising opportunity to address these challenges. Current smartphones are highly sophisticated, offering a general computing environment with a powerful processor, high-resolution display, several communication modalities (Bluetooth, WiFi, 4G), camera, and sensors.

Unfortunately, smartphone platforms suffer from many risks. Vulnerabilities exist in communication standards that enable eavesdropping or impersonation [31, 41]. Moreover, phone operators are disclosing information or introduce vulnerabilities through insecure or misconfigured systems [33].

We observe that individuals often have physical interactions with resources or other individuals before communicating digitally. Often, people communicate over the Internet or via SMS after having met in person. We propose to leverage the initial physical encounter to bootstrap trust, in essence converting physical trust into digital trust. We argue that this is a model that people can intuitively relate to.

Certainly, many people communicate over the Internet before physically meeting. To set up trust for these interactions, we propose a secure introduction mechanism for trust

[1]We thank Moxie Marlinspike for suggesting the "sling" metaphor.

establishment that is rooted in physical encounters with a common acquaintance. For example, if users could verify that the virtual person on a Facebook invitation page has physically met with one of their close friends, they can gain more trust that the invitation page was indeed issued by the correct individual.

In this paper, we describe SafeSlinger, a system for secure exchange of authentic information between two smartphones. In essence, SafeSlinger exchanges contact information, containing public keys in addition to standard contact list information such as name, picture, phone numbers, email addresses, etc. Thanks to the association between the individual holding the phone and the public key that is exchanged, users (with the help of the SafeSlinger App) can later associate digital communication with the previously met individual by verifying a digital signature. To make SafeSlinger usable, the cryptographic aspects are mostly hidden from the user. Achieving these goals while retaining the desired security properties is far more challenging than it initially sounds (Section 4 lists challenges). We have built-in several approaches to make SafeSlinger tolerant to user error, as we describe in this paper.

We envision SafeSlinger as a general approach to bootstrap secure digital communication. (1) First, we enable small groups (2–8 individuals) of physically co-located users to securely bootstrap trust by *slinging keys* between their devices (a one-time operation). SafeSlinger can also support remote setup, as long as users can authenticate the other individual (e.g., via live video conference or voice communication). (2) Second, we built-in secure phone-to-phone messaging and file transfer, both providing secrecy and authenticity. The user experience is nearly identical to that of traditional SMS and MMS messaging today. (3) Third, SafeSlinger enables *secure introductions* without physical meetings by allowing a common acquaintance to facilitate a mutual introduction enabled by SafeSlinger file transfer. (4) Fourth, we enable other applications to use the SafeSlinger API to add their public key to a contact entry. Now, when a user slings its updated contact list entry to another user, the applications' public key is automatically included, and the same application at the other end can extract the public key. This mechanism can enable applications such as secure email or secure SMS to solve the problem of securely exchanging the public key without a leap of faith.

This paper makes the following **contributions**. SafeSlinger is the first complete system that provides secure group credential exchange that is also privacy-preserving, such that no external party can learn any of the exchanged information. SafeSlinger is also the first group credential exchange system that can be used remotely over a telephone or video conferencing line. SafeSlinger is designed to be easy-to-use and defend against all attacks we are aware of. We have implemented SafeSlinger on Android and iOS, and have made it freely available. SafeSlinger includes mechanisms for secure messaging and file exchange, as well as secure introduction between two individuals.

2. PROBLEM DEFINITION, GOALS, ASSUMPTIONS AND ADVERSARY MODEL

In this section, we define the problem we set out to solve, discuss our goals, present assumptions that need to hold, and the adversary that we defend against.

2.1 Problem Definition

The basic bootstrapping primitive that we want to accomplish is to securely exchange information that is associated with the intended individuals participating in an exchange. The security properties that we seek are information secrecy (only the intended entities receive the information), and user-verifiable trustworthy association of data to an honest individual (user knows exactly the information that is associated with a specific honest individual).

Note that there are fundamental limits on the ability of an electronic protocol to protect one human against another human with the intent to deceive, hence the adjective "honest". There are some subtleties involved when the exchange includes more than two people. For example, if Fred, George, and Harry perform an exchange, and Harry *is* adversarial, we still wish for the information exchanged between Fred and George to have all of its security properties intact. Intuitively, we wish for the group exchange to produce the exact same security properties that would result from exhaustive pairwise exchanges between all members of the group.

Given such a secure exchange that includes a public key, all subsequent mechanisms for authentic, integrity-protected, and optionally secret communication can be implemented using well-known protocols, relying on the authentic binding of the public key to the correct individual.

2.2 Goals

Our core goal is to enable usability while retaining the security properties described in the problem definition. The lack of usability is likely to have been a core detriment to the limited adoption of PGP [42]. Therefore, we want to make SafeSlinger as easy as possible to use. Unfortunately, many cases present a tradeoff between security and usability; for example, requiring users to validate the equality of hash values during protocol execution is a tedious operation and users inevitably select "equal" without performing the comparison. In such cases, we need to make the system slightly less usable and demand more from the user to retain a high level of security.

Another goal is to support heterogeneous platforms, thus enabling interactions among smartphones of several manufacturers and running different operating systems. This turns out to be a major challenge, since even simple Bluetooth communication is not feasible across phones from different vendors, as an Apple iPhone, for example, only wishes to communicate via Bluetooth with another Apple product or a headset. We discuss these aspects in more detail in our implementation section.

Moreover, we want to support several users to simulta-

neously exchange their contact list information. Without this requirement, it would be tedious for a group of users to exchange their information via $N \cdot (N-1)/2$ pairwise exchanges.

2.3 Assumptions

Our protocols do need to assume certain user behaviors to execute successfully. First, we assume that users are computer literate and can follow basic instructions, such as "compare the words[2] presented on your screen with the words presented on other phones and select the ones that are equal". We also assume that users have a natural desire for security and that they do not want to deliberately disclose their private information. We also assume that users can authenticate (in the human sense, e.g., recognizing the other users' appearance, voice, etc.) the individuals that they perform information exchanges with, such that an adversary who impersonates another individual would be detected through personal identification.

We also assume that the smartphone hardware and software is free of vulnerabilities and malware, as securing these is out of scope for this work.

2.4 Adversary Model

We assume that some of the legitimate users that participate in the protocol may be malicious. (We call the other users honest.) We consider a Dolev-Yao style adversary that has complete control over all network messages. Furthermore, any Internet server we may use during protocol operations may be malicious. We also assume that adversaries may be physically present in the space where information exchanges happen.

We consider an adversary who wants to break the properties as described in the problem definition, i.e., violate secrecy and authenticity properties of the information exchange and subsequent communication. We consider denial-of-service attacks to be out of scope, as users can easily detect the lack of progress of the protocol and re-start it as needed.

3. CRYPTOGRAPHIC BACKGROUND

We provide background on the cryptographic mechanisms used in this paper: multi-value commitments and group Diffie-Hellman key agreement.

3.1 Multi-Value Commitments

A cryptographic commitment protocol is used to lock an entity to a value V without disclosing V. Based on the commitment value, the decommitment can be validated and the protocol ensures that the correct value V is disclosed. A commitment protocol for value V can proceed as follows: $C = H(V,R)$, where C is the commitment value, H is a cryptographic hash function that is one-way and collision-free, and R is a random unpredictable nonce. Thanks to the properties of the hash function and the randomness of R, V cannot be inferred from the commitment C. To open the commitment, V and R are disclosed. The collision resistance property of H ensures that it is computationally infeasible to find another V or R that will result in the same commitment C. Note that if V is unpredictable (e.g., a freshly generated ephemeral public key), the additional nonce value R is not needed and we simply have $C = H(V)$.

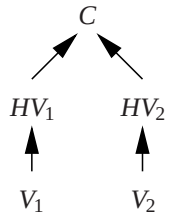


Figure 1: Multi-value commitment structure for authenticating and disclosing either V_1 or V_2.

In case we want to commit to either value V_1 or V_2 (deciding which to actually release at a future time) with a single commitment, we can employ a tree-like structure (Figure 1). $HV_1 = H(V_1)$, $HV_2 = H(V_2)$, and $C = H(HV_1 \,||\, HV_2)$, where $||$ indicates the concatenation operator. This structure enables decommitment of either V_1 or V_2 without disclosing the other. For example, to decommit V_1, we disclose V_1 and HV_2, and the case for V_2 is analogous. Note that this example is for the case when V_1 and V_2 are unpredictable to the adversary; additional use of nonces is required for well-known V_1 or V_2. This type of structure is similar to one-time signatures [29].

In our protocols, we further make use of hierarchical commitments, where the decommitment value is again a commitment value.

3.2 Group Diffie-Hellman Key Agreement

Group Diffie-Hellman (DH) key agreement is a generalization of the two-party DH key agreement [9], where multiple parties participate to establish a common group key. The Cliques protocol is an example for group DH key establishment [37]. We will make use of STR [36], a tree-based group DH protocol, which we briefly describe in this section.

In group DH protocols, each participant is placed at a leaf node of a binary tree, where each node of the tree has a private and a public key associated with it. The value of a leaf node is the private and public DH keys of the member at that node. The value of the parent node is derived through the DH operation on the values of the two child nodes, for example if the values of the child nodes are x for the private and $g^x \bmod p$ for the public key of the left child, and y for the private and $g^y \bmod p$ for the public key of the right child, the parent value is $g^{xy} \bmod p$ for the private and $g^{g^{xy}} \bmod p \bmod p$ for the public value. Figure 2 illustrates a group DH key agreement with three members, where each node in the tree lists the private and public DH

[2]Presented in the participants' common language, e.g., English.

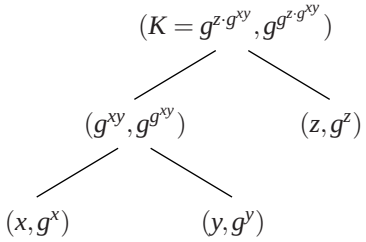


Figure 2: Group DH key agreement tree for 3 members (*mod p* operations omitted for clarity). The notation is (priv, pub), where the first element in the parenthesis lists the DH private key, and the second lists the DH public key. The group key is the private key K at the root.

keys. The private key that corresponds to the root node is the shared secret of all group members. In the STR [36] protocol, the tree shape is a maximally unbalanced tree (resembling a comb), and the TGDH [16] protocol uses a balanced tree. Research has shown that total protocol latencies are lower for STR in environments where the communication latency dominates over the computation of a modular exponentiation [15], which is the case in mobile environments.

4. ATTACKS AND CHALLENGES

Secure local exchange of information is a surprisingly intricate and challenging problem. Numerous attacks are possible, which we will illustrate with two simple protocols. We then discuss the attacks in more detail.

4.1 Strawman Approaches

We consider two simple protocols for local authenticated and secret exchange of information between users who are physically nearby each other and who want to simultaneously exchange their contact list information, which also contains their public key.

Password-based exchange.

In this protocol, a password shared among users is used to secure the information exchange. Several researchers proposed protocols of this flavor, for example Asokan and Ginzboorg [2], Valkonen, Asokan and Nyberg [39], and Abdalla et al. [1].

A simplified password-based protocol proceeds as follows: the users agree on a password P and enter it into their devices. From P, the devices derive a symmetric key with a cryptographic hash function H (i.e., $K = H(P)$), and K is used to encrypt and authenticate their communication. Each user broadcasts its information to the others as follows (where I_X represents the information of user X, and MAC represents a secure message authentication code, and $\{X\}_K$ stands for encryption of message X with key K):

$A \rightarrow *: \quad \langle \{I_A\}_K, \text{MAC}_K(\{I_A\}_K) \rangle$
$B \rightarrow *: \quad \langle \{I_B\}_K, \text{MAC}_K(\{I_B\}_K) \rangle$
$C \rightarrow *: \quad \langle \{I_C\}_K, \text{MAC}_K(\{I_C\}_K) \rangle$
$D \rightarrow *: \quad \langle \{I_D\}_K, \text{MAC}_K(\{I_D\}_K) \rangle$

Each user then uses their key K to authenticate and decrypt the received information. Unfortunately, this approach is vulnerable to several attacks. First, a malicious bystander could overhear or guess the password, and control communication such that the legitimate users' messages do not reach the receivers, while the attacker is able to insert information of his choosing. In the Dolev-Yao attacker model, where the attacker controls all communication, this is a simple attack. In practice, it is quite possible to mount such an attack. Especially in the case of wireless communication, the adversary can jam legitimate messages to prevent reception, and then re-insert messages that will be received by the receivers. Note that this weakness stems in part from the password-based key exchange protocol's requirement that the password be kept secret from all non-participants.

Second, a malicious participant (as opposed to a bystander) could easily mount this attack as well, since that user will know the password. Such a malicious user could control all the information that each member receives. Note the distinction between a malicious user's being able to misrepresent his own identity (which simplifies to a problem with human trust, and cannot be solved with a protocol to the best of our knowledge), and that user's being able to insert fraudulent information for honest protocol participants (which is a protocol weakness). This represents the protocol placing excessive trust in a single participant.

Both attacks are instances of Man-in-the-Middle (MitM) attacks, which are a common form of protocol attacks.

Group Diffie-Hellman with Key Comparison.

A promising approach to eliminate the requirement that human protocol participants communicate amongst themselves secretly is to leverage a group Diffie-Hellman (DH) key establishment protocol (Section 3.2). MitM attacks can be prevented by human verification of equality of the established key among the participants. A sample comparison-based protocol is by Valkonen, Asokan and Nyberg [39].

The idea in comparison-based DH key establishment is to detect MitM attacks by having the users visually compare a hash of the shared secret that results at the end of the DH protocol. To understand this defense, consider the following exchange between two users X and Y. X has the private DH key x, and Y has private key y. Following a successful key agreement, they would share the key $K_{xy} = g^{xy} \bmod p$. If the users compare a few bits of $H(K_{xy})$, where H is a cryptographic hash function, they try to ensure that they have the same key. Since H is a one-way function, this comparison does not reveal the secret key. A variety of approaches exist to compare these keys, ranging from hexadecimal representations of the hash to hash functions with pictures as output to simplify comparison [14]. If the comparison suggests that the keys are equal, the users inform the application to proceed.

In case of a MitM attack, where an adversary M impersonates X to Y and Y to X, M would have sent $g^{m1} \bmod p$ to X and $g^{m2} \bmod p$ to Y. Consequently, X computes the

secret key $g^{x \cdot m1} \bmod p$ and Y computes $g^{y \cdot m2} \bmod p$. Since the two keys are different, the hash values will also be different between X's and Y's device with high probability and the attack would be detected.

Unfortunately, numerous attacks are possible. First, most users prefer convenience over security, and they may simply select "OK, hash is equal" without performing the actual comparison. Since no attack occurs most of the time, the users would also lose interest to perform such a comparison. Another issue is the entropy of the values that are compared: to make this comparison more usable, only a subset of the full 256-bit hash of a SHA-256 hash function would be presented to the user, as the full-length comparison is too time intensive. Consequently, users may only compare about 20 bits, with the idea that an attack would be caught with probability of $1-1/2^{20}$, which means that only about one in a million attacks would go unnoticed and be successful (assuming users indeed perform the comparison). Unfortunately, another attack is possible on comparisons with limited entropy. The MitM adversary can compute $m1$ and $m2$ such that $[H(g^{x \cdot m1} \bmod p)]_{20} = [H(g^{y \cdot m2} \bmod p)]_{20}$, where $[.]_{20}$ indicates truncation down to the least significant 20 bits. Because the attacker controls both sides of the equation, finding a collision can exploit the birthday paradox and the expected number of operations to find the collision is $O(2^{10})$, requiring only around 1000 cryptographic operations, which is trivial to accomplish on current hardware (e.g., by outsourcing computations to a cloud).

Yet another attack on this protocol for groups of three or more participants is that a bystander could participate in the protocol – the invisible nature of electronic communication prevents humans from observing which devices are actually communicating. Consequently, the adversary could learn the shared secret, and of course all hash comparisons will be successful, as all members share the same group secret key! The problem is that the additional member is difficult to detect. A simple countermeasure against this attack is to require members to count and verify the number of legitimate group members. Unfortunately, as prior research shows, counting is unreliable when the number of group members is larger than 8, as people tend to make mistakes causing the protocol to fail [8]. Moreover, another attack is possible even if members could count correctly: the Group-in-the-Middle (GitM) attack [17].

In a GitM attack, a group member participating in the protocol is malicious and exploits the invisibility of wireless communication to split the group into several subgroups, adding additional virtual users to create the illusion to each member that they are in a group with the correct number of users. For instance, consider a group with 4 members A, B, C and D, who attempt to run the protocol. Consider that D is malicious, so he could create virtual identities X, Y, and Z, and since he controls the wireless signals and their reception through careful jamming and directional antennas, he can create the illusion of two groups: A, B, X, D, and Y, Z, C, D. Note that the legitimate members A, B, and C cannot detect this attack, since they believe that they are performing the protocol in a group with 4 users. Similar to the 2-party MitM collision attack discussed above, D can select the private keys for X, Y, Z, and D such that the two subgroups will end up with the same hash to compare, preventing the members A, B, C from detecting the attack even if they diligently check all the hashes.

We hope that these two strawman protocols demonstrate that designing a usable and secure information exchange protocol is a surprisingly intricate challenge. Next, we give a more thorough list of attacks and challenges.

4.2 List of Attacks and Challenges

We consider the following attacks:

- **Malicious bystander who participates in protocol:** a bystander can overhear conversation, and attack the protocol by controlling the local wireless communication (Dolev-Yao attacker model). The **Man-in-the-Middle (MitM) attack** is a specific instance of this attack.
- **Malicious group member:** an invited member of the group wants to violate protocol properties, such as mounting an **impersonation attack** by injecting incorrect information for another user, or performing a **Sybil attack** [10] by injecting multiple entries either for fictitious individuals or for individuals who are not present. A malicious group member can also perform a **Group-in-the-Middle (GitM) attack** [17], as described above.
- **Malicious server:** for protocols that rely on a back-end server, the server may be controlled by a malicious administrator or be compromised and thus execute malicious code.
- **Information leakage after protocol abort:** an adversary may be able to cause a protocol abort and trigger information leakage of private information about a participant.
- **Collision attack on low-entropy hash:** as described above, low-entropy hash values can be vulnerable to efficient birthday attacks if the correct precautions are not taken.

We consider the following challenges:

- **Exclusion of unintended participants:** legitimate users will expel an unwanted bystander who wants to participate in the protocol.
- **Correct member count:** users need to correctly count number of group members.
- **Identity validation:** users correctly validate the identity of information received from the exchange. More specifically, they map the identity information to the people who are physically present, and they would reject information about a person who is not physically present.
- **Impersonation detection:** users verify that no other user has injected information that impersonates them in the current exchange. For example, a malicious user

may also inject information about Alice, even though Alice is also participating in the exchange. The risk is that another user may discard the correct information and accept the wrong information.

- **Diligent hash comparison:** users correctly perform the hash comparison, even after executing the protocol numerous times without any attack.
- **Diligent error checking and aborting:** users will abort the protocol and restart the protocol when suspicious or error conditions are encountered.

In the next section, we describe our approach for designing SafeSlinger to prevent all attacks and overcome the challenges listed.

5. SAFESLINGER

In this section, we provide a detailed description of the SafeSlinger protocol. We first provide a high-level overview before we dive into the details.

5.1 Overview

The main purpose of SafeSlinger is to enable a set of users to exchange their contact information such that every non-malicious user receives the correct information about every other non-malicious user. Malicious users may collude and impersonate each other, for example, therefore we cannot provide any guarantees for those parties. Our main goal is provide high usability while preventing the attacks we describe in Section 4.

The first hurdle we need to overcome is to enable communication among the users' mobile devices. To capture most of the market share of current smartphones, we target Android, iPhone, and Windows Mobile devices. Unfortunately, these platforms do not offer native support for 802.11 ad-hoc mode or setup of a base station to enable other devices to connect to them. Bluetooth communication is also challenging because of the slow discovery phase and the unwillingness of iPhones to communicate to a non-Apple device except a headset. NFC is not yet widely deployed, and such communication does not scale to multiple devices. As a consequence, we use Internet-based communication, where all the mobile devices connect to a cloud server. This approach has the additional advantage that no device discovery is needed, as the devices can simply send packets to the server via IP.

We support groups of up to 50 users, however, we use two different protocols: the main SafeSlinger protocol which we describe in this paper for up to 7 users, and the Ho-Po Key protocol for groups of more than 7 users [30]. As prior work shows, users can reliably count the number of participants for groups of up to 8 users, but several people start to make errors for larger groups [8]. As the protocol fails if only a single person miscounts, we conservatively set the threshold at 7 users, up to which users can reliably count. Asking users to count the number of participants rules out several attacks, as we discuss in our Section 5.4.

After the mobile devices connect to the server, the server cannot know which devices belong to the same group. It is a challenging problem for the server to determine the grouping, especially if several concurrent exchanges are ongoing. Several approaches exist, which we discuss in Section 5.5. We propose the following approach, which does not leak any sensitive information to our untrusted server. The server assigns a unique ID to each mobile device, which it displays to its user. The devices ask the users to find and enter the lowest ID. The devices then send that ID back to the server, which can thus perform the grouping. Note that the actual grouping is not security sensitive, as an intruder will result in denial of service.

In a nutshell, the mobile devices send their information to the server, which redistributes it to the other devices. The users engage in a verification of all exchanged information to ensure that they all have received identical information from the server. This verification is done by the users who perform a comparison of short hash values displayed by the phones. Two problems that we discussed in Section 4 occur: (1) users simply click "OK" without performing the comparison, and (2) an attacker can compute a collision attack on the short hash value. We solve (1) by presenting 2 decoy hash values besides the correct value and asking users to verify which of the 3 hash values matches a value on other people's devices. This forces the user to perform the comparison, as a random guess will cause the protocol to fail 2/3 of the time. Moreover, this approach encourages users to select "no match" if they cannot find a match. We address problem (2) by using Short Authentication Strings (SAS) [6, 18, 19, 40, 43]. In SAS, all devices first commit to the values that are used in the hash comparison. Once all the commitments are distributed, the devices reveal the decommitments and the short hash comparison can proceed. This approach prevents the collision attack and in Zimmermann's words [43] converts the attack from a "safe attack" into a "daring attack." The collision attack is a safe attack, because the adversary knows that the attack will succeed with probability 1 as the collision has been found. However, with the commitment, the adversary cannot know ahead of time if the collision indeed will occur, and the attack only succeeds with a probability of 2^{-n}, where n denotes the bit length of the authentication string, thus resulting in a "daring attack."

A major challenge which we address in SafeSlinger is to prevent the server from learning any contact information. We accomplish this by leveraging a group DH protocol, which is described in Section 3.2. The group DH protocol establishes a shared secret key among all participants, which is used to encrypt the contact information. To prevent MitM attacks, the DH public key is included in the initial commitment, which the users validate through the hash comparison, authenticity providing of the DH public keys.

5.2 SafeSlinger Details

Figure 3 describes the SafeSlinger protocol in detail. In Step 1 (which we will abbreviate as S1 for short), the user

selects which data to share and enters the total number of protocol participants. In S2, the device computes the values needed for the group DH protocol by selecting the ℓ'-bit long DH private key n_i at random (in the current version, we use $\ell' = 512$). The device also randomly selects nonces to indicate "match" (Nonce match Nm_i) and "wrong" (Nonce wrong Nw_i). The device also encrypts the data to share with the Nonce match used as a symmetric encryption key (we use the AES block cipher with 128-bit keys). In the current version, the security parameter $\ell = 128$, and we use SHA-1 truncated to 128 bits as hash function H. Figure 4 depicts this multi-value commitment structure for user U_i. Finally, in S3 the device sends the commitment C_i to the server.

In the next phase, the server groups the users. First, the server sends a unique ID to the device (S4) which the device displays and prompts the user to find the lowest ID amongst all devices (S5). In S6, the user enters the lowest ID which in S7 the device sends to the server.

The server now knows which devices belong to the same group, and distributes ID and commitment pairs (ID_i, C_i) to all group members (S8). Once a device receives all commitments, in S9 it opens up the first level decommitment HN_i, G_i, E_i (Figure 4 illustrates this). If validation of all decommitments is correct (S10), devices compute a hash over all decommitments of C_i, i.e., over the triplets (HN_*, G_*, E_*), sorted by the value of the unique ID_i assigned to each device to ensure that all devices compute the hash over the same triplet ordering. Each device then computes a word phrase that represents the hash (S11) – we use the PGP word phrase which results in a representation that encodes 24 bits of the hash. The device also produces two decoy word phrases, which produces two interesting challenges: (1) if the words in the decoy word phrase match words in the actual word phrase, users may get confused, and (2) if words in different users' decoy word phrase match, users may select the wrong decoy phrase on their respective devices as a match. To avoid this, we make sure that the decoy phrases do not include any words from the actual word phrase, and we also make sure that all decoy word phrases are mutually exclusive among all devices, which is a challenge to implement. We address this by using the received commitments as a seed to a pseudo-random generator, and having each device draw words from the word phrase for their decoy phrases without replacement. Since the word phrase only contains 512 words in total, this limits the total number of users we can support. More intricate details on the word phrase selection are presented in Section 6.

If no phrase matches, the user selects "no match" and sends the "wrong" nonce Nw_i along with Hm'_i to enable verification to the server (S12). This case is also triggered if the user selects the wrong word phrase. This approach cryptographically authenticates the "no match" message from the commitment C_i and thus prevents injection of the wrong nonce by an adversary. In S13, users correctly selected the matching word phrase, and the device reveals the pair of values indicating success (Hm_i, Hw_i), which the server redistributes in S14. Each device can verify that all the users selected the correct word phrase (S15) and in S16 the devices proceed to construct the group DH tree as described in Section 3.2. The ordering in the tree is determined by the sorted order of the unique IDs ID_i. Since the STR group DH protocol we use is intricate we omit the details for enhanced readability, but we refer readers to the description in Section 3.2 and for more details to papers on STR [15, 36].

Data Selection & Counting

1. $U_i \xrightarrow{UI} M_i$: D_i (the data to be exchanged)
   $U_i \xrightarrow{UI} M_i$: $\tilde{N}$ (number of people in the group)

Commitment, Group DH Key Setup

2. M_i : $Nm_i \xleftarrow{R} \{0,1\}^\ell$ ("match" nonce)
   $Hm_i = H(Nm_i)$, $Hm'_i = H(Hm_i)$
   $Nw_i \xleftarrow{R} \{0,1\}^\ell$, $Hw_i = H(Nw_i)$ ("wrong" nonce)
   $HN_i = H(Hm'_i || Hw_i)$ (multi-value commitment)
   $n_i \xleftarrow{R} \{0,1\}^{\ell'}$, $G_i = g^{n_i} \bmod p$ (group DH key)
   $E_i = \{D_i\}_{Nm_i}$ (encryption of data)
   $C_i = H(HN_i \,||\, G_i \,||\, E_i)$ (commitment)
3. $M_i \rightarrow S$: C_i

Server Unique ID Assignment, User Grouping

4. $S \rightarrow M_i$: ID_i (unique ID per user)
5. U_i : find lowest unique ID $\rightarrow ID_L$
6. $U_i \xrightarrow{UI} M_i$: ID_L (enter lowest ID)
7. $M_i \rightarrow S$: ID_L

Collection and Distribution of Initial Decommitment

8. $S \rightarrow M_i$: ID_j, C_j $(j \neq i)$ (other users' ID and commitment)
9. $M_i \rightarrow S$: HN_i, G_i, E_i
   $S \rightarrow M_i$: HN_j, G_j, E_j $(j \neq i)$ (other users' decommitment)
10. M_i : $C_j \stackrel{?}{=} H(HN_j || G_j || E_j)$ $(j \neq i)$ (verify)

Wordlist-based Comparison of Integrity of Commitments

11. WordPhrase($H([\{(HN_*, G_*, E_*)\}]_{24})$) (displayed on screen)
    $U_i \xrightarrow{UI} M_i$: Select Matching Word Phrase
12. M_i : if "no match" then $M_i \rightarrow S : Hm'_i, Nw_i$, "abort"
13. M_i : if "match" & correct phrase selected, then
    $M_i \rightarrow S : Hm_i, Hw_i$
14. $S \rightarrow M_i$: Hm_j, Hw_j $(j \neq i)$
15. M_i : $HN_j \stackrel{?}{=} H(H(Hm_j) || Hw_j)$ $(j \neq i)$ (verify)
    Abort if any verification failed

Group DH Key Establishment

16. M_i : Computation of group DH tree (Section 3.2)
    K = Private key of root node (see Section 3.2)

Distribution and Verification of Data Decryption Key

17. $M_i \rightarrow S$: $\{Nm_i\}_K$
    $S \rightarrow M_i$: $\{Nm_j\}_K$ $(j \neq i)$
18. M_i : Decryption of Nm_j $(j \neq i)$
    $Hm_j \stackrel{?}{=} H(Nm_j)$ $(j \neq i)$ (verify)

Decryption of Data and User Acquisition

19. M_i : Decryption of E_j with Nm_j $(j \neq i) \rightarrow D_j$
    Save user data D_j

Figure 3: Steps for user U_i ($i \in 1 \ldots N$) to exchange data D_i with the other $N-1$ users via their mobile devices. $U_i \xrightarrow{UI} M_i$ indicates input by user U_i into mobile device M_i. $M_i \rightarrow S$ represents wireless communication from mobile device M_i to server S. $\{X\}_K$ represents encryption of X with symmetric key K.

Once the secret group key K is established, the devices then proceed to send their final match nonce Nm_i (which also serves as the decryption key) to the group, encrypted with K (S17). In S18, each device decrypts and verifies the correctness of Nm_i, and finally uses Nm_i to decrypt the data D_i in S19.

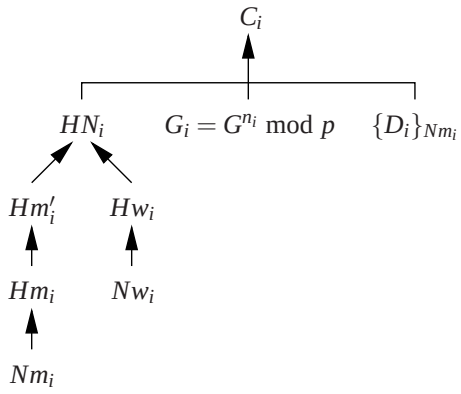


Figure 4: Multi-value commitment structure for user U_i.

5.3 User Experience

Although the protocol appears quite complex to achieve all the security properties we set out for, the user experience is actually quite simple as SafeSlinger performs all the cryptographic operations and checks without the users' involvement. The user experiences the following steps: (1) select the data items to be shared, (2) count and select the number of users, (3) find and enter the lowest ID displayed by the devices, (4) compare the word phrases and select the one that matches and click "match", or click on "no match", (5) select which users' data to import into ones contact list. We optimized the implementation such that a complete run only requires on the order of 10 seconds.

5.4 Security Analysis

We will use the list of attacks and challenges from Section 4.2 to guide this security analysis, however, we consider additional attacks that are specific to the operations of the SafeSlinger protocol. We consider attacks on the SafeSlinger protocol for groups of fewer than 8 members that we describe in this paper, for the security analysis of the protocol with more than 8 members we refer to the Ho-Po Key paper [30].

We first consider attacks by malicious outsiders (who are not legitimate group members). Such adversaries can contact the server, ask for a unique ID, and attempt to join arbitrary groups by sending the server a plausible group ID to join. Since users specify the group size, this attack will be detected as the server will send too many commitments to the participants.

A more sophisticated version of that attack is where a local adversary jams communication of one of the local devices, and attempt to join the group that way. In that case, the local user who is being suppressed by the adversary needs to inform the other group members to abort the protocol, since she is not receiving commitments. If the user is receiving other users' commitments, then the hash comparison will fail with high probability $(1 - 2^{24})$ and at least one other user who is part of the group needs to be informed not to press "match". Preventing this attack requires some amount of user diligence, essentially the suppressed user has to inform at least one honest group member to select "no match" in the word phrase comparison. Thus, the users need to ensure that they perform the hash comparison with *all other* users, fortunately, as long as at least one user who was not suppressed is diligent, then these attacks will be detected.

A malicious legitimate participant of the group could attempt several attacks. First, attempts to infiltrate additional virtual members into the group, for example through a Sybil [10] attack, would fail because the number of virtual members would be larger than the count of physical members which users enter at the beginning of the protocol. A Group-in-the-Middle (GitM) attack as described in Section 4 is prevented if users diligently perform the hash comparison step, as members who end up in different groups will have different hashes to compare with high probability. The adversary could also send wrong contact information for himself, for example attempting to impersonate another person who is currently in the group. SafeSlinger defends against this attack by enabling users to verify at the end of the protocol which contact phrase entries they import into their address book, and if a suspicious entry exists the user can attribute that to the adversary. If the adversary impersonates a user who is present, that user can detect that someone else injected an entry for herself and inform the others.

A malicious server could try to split the group up into different subsets of users, and fake another set of users (GitM) to ensure that each user encounters the correct number of users, even though some are virtual users created by the server. Again, diligent checking of the hash with all other members in the group will detect this attack. The server also does not learn any information about the users, as it cannot participate in the group DH protocol to discover the established group key K. The only way to obtain K is to participate as a user and inject a commitment, which would be detected by the device as the number of members is larger than the user entered.

The multi-commitment with several stages of decommitment (as depicted in Figure 4), ensures that no information is revealed unless all members M_i reveal their "match" nonce's hash HNm_i because the devices would not reveal the decryption key Nm_i. Hence, if any group member detects an anomaly before the hash comparison, all devices will abort the protocol. The multi-commitment also prevents the collision attack on the low-entropy hash that is used for the comparison, by computing the hash over the ordered triplets (HN_*, G_*, E_*). Since the commitment $C_i = H(HN_i \mid\mid G_i \mid\mid E_i)$ locks in the value of the triplet, an adversary cannot change its triplet or predict any other triplet before the hash is pre-determined through all users' choices. Hence, the attacker only has a chance of 1 in 2^{24} to create a collision, which is sufficiently small for our purpose.

5.5 Discussion

In this section, we discuss the rationale behind design decisions we made.

With respect to the server's grouping approach, we considered numerous alternatives. The BUMP application [5] performs the grouping by having two users "bump" their phones together, and by measuring location, time, and acceleration the server can pair up the two phones. Although this approach is fun for the users, we did not use if for numerous reasons: (1) BUMP reveals the user location to the server, which is an invasion of privacy, (2) the approach is not secure as a malicious bystander can simultaneously simulate the bump and often be paired with one of the two devices and steal the user's contact information [38], (3) does not scale to more than 2 users, (4) cannot be performed remotely over the telephone, (5) acquiring the location can be unreliable and often has a delay of 10 seconds or more, (6) the protocol is often unreliable and the server cannot pair the devices.

Another alternative for grouping we considered is to use ambient noise, but this may also reveal privacy-sensitive sound to the server, and may also be unreliable in many circumstances. We finally settled on the unique *ID* assignment by the server and having users find and enter the lowest *ID*, which is fast and reliable, and can be fun as users compete to see who is "the winner". Based on the *ID*s, all the server needs is to end up with a connected graph, where each device represents a node and an edge is formed by having one device send the *ID* of another device to the server. We considered the approach of having users simply enter an *ID* of any other user, but this may be confusing in case multiple users are present, it can also lead to a non-connected graph in case there are more than 3 users. By having users enter the lowest *ID*, the resulting graph forms a star topology, which is connected.

A point of frequent confusion is that people believe that the members who perform the exchange are the only group that can communicate. This is not the case, however. The SafeSlinger exchange protocol is only used for acquiring other users' information in a secure fashion. Since the contact information includes a public key, only that public key is used to establish subsequent secure communication. In particular, all cryptographic values created during the exchange (as phraseed in Figure 3 are erased right after the exchange – the sole purpose of their brief existence was to protect a single exchange. Subsequent secure group communication can be easily established by using the exchanged public keys in conjunction with a group DH protocol, in which case it does not matter whether the public keys of other group members were acquired in the same or in several separate exchange sessions. Thus, there is no relationship of the group that was used to exchange the information and subsequent membership composition of secure group communication.

6. IMPLEMENTATION

The implementation of SafeSlinger runs on a central server and multiple smartphone platforms. The client application is written for Android 2.1 in Java and Apple iOS 3.0 in Objective-C. The server application is written for the Google App Engine platform in Python. We tested our key exchange over cellular and Wi-Fi networks and on several smartphone devices: Motorola Droid 855, Google Nexus S, Samsung Galaxy, Samsung Galaxy SII, Apple iPod Touch, and Apple iPhone 4.

In this section we provide an overview of design decisions intended to optimize the adoptability and usability of our key exchange and a detailed walk-through of the steps in a key exchange protocol execution. Usability aspects drove many of our design considerations to make the experience convenient, efficient, and comfortable to use.

Figure 5 depicts the steps for a contact list exchange. In 5(a) we can see how a user can select the entries of the contact list to share. 5(b) shows the dialog where the number of users are selected, the next entry on the bottom that is not visible selects "8 or more users", which would trigger the Ho-Po Key protocol. In 5(c), the user sees her unique ID assigned by the server and enters the lowest ID of any user in the group. The word phrase selection screen is shown in 5(d), and if all operations were successful, in 5(e) the user can select which contact list entries to import into which of their address book accounts.

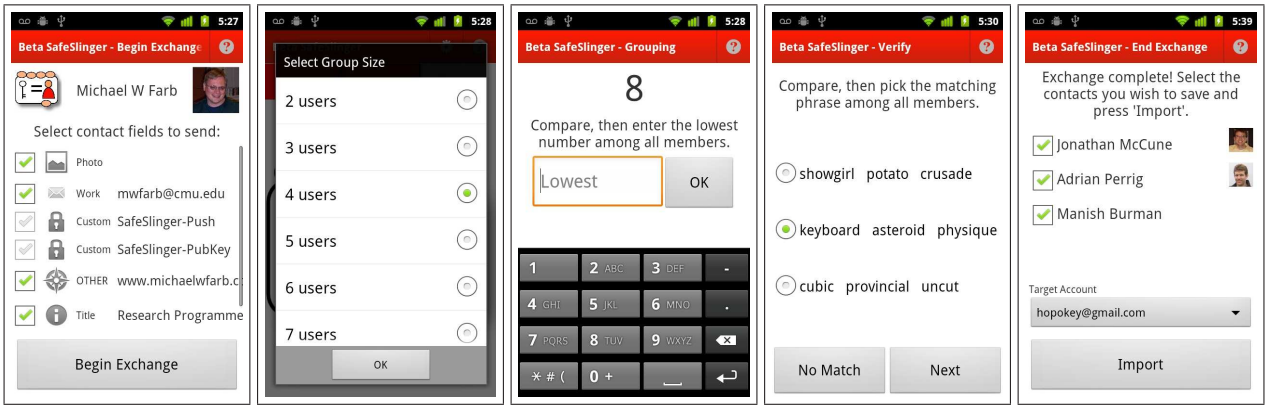
6.1 Address Book Key Management

As our implementation attempts to share contact data and keys with others, we rely heavily on use of the mobile operating systems' contact list. At the beginning of our exchange, each user must identify which contact entry they wish to use as their identity, or create one. Since we want to provide a place to store public key data for the user, it would be beneficial to keep it in a recognizable field that the smartphone's address book synchronization service would backup for us offline. We extend the ability for the contact list to store the name and value of a new instant messaging (IM) provider. Some contact synchronization services may require ASCII-only values in this IM field, so we Base-64 encode our public key under a custom label that the third-party application defines. Further discussion of this API can be found in Section 7.1.

6.2 vCard Construction

Before we can send our contact information we need to format it in a standard manner so that will be recognizable across platforms. We format our contact data in vCard 3.0,[3] and the key material specifically uses the IETF vCard Extensions for Instant Messaging[4] proposal. We use this IMPP field type with a naming pattern so that keys are arbitrary data to the key exchange which do not require special handling relative to other contact fields. This will encourage developers to adopt the exchange of key material or other data made available by SafeSlinger without creating special support for their application. The SafeSlinger Messaging application (Section 7.2) uses the labels "SafeSlinger-PubKey" and "SafeSlinger-Push" for its public key and push token

[3]http:// tools.ietf.org/html/rfc2426
[4]http://tools.ietf.org/html/rfc4770



(a) Data selection (b) Number of users (c) Lowest ID (d) Word phrase comparison (e) Contact import

Figure 5: Secure contact information exchange sequence.

which can be seen in Figure 5(a). Although we have modified the vCard construction to share these public keys in the IMPP field which translate to fields that can be synchronized, no smartphone OS currently supports the import or export of the existing KEY field supported in vCard 3.0.

6.3 Contact Data Confidentiality

The contact data we share in the exchange is encrypted using AES in CBC mode. The actual encryption key K_D and initialization vector IV_D are derived from the 160-bit "match" nonce Nm_i: $K_D = [\text{HMAC-SHA-1}_{Nm_i}(1)]_{128}$, $IV_D = [\text{HMAC-SHA-1}_{Nm_i}(2)]_{128}$. The $[x]_y$ notation indicates that the value x is truncated to the y least significant bits. Contact data integrity is achieved through verification of the commitment C_i, hence, no additional Message Authentication Code (MAC) is needed.

As we discuss in Section 5, the "match" nonce needs to be distributed encrypted by K (the private key of the root node of the group DH key tree). The encryption is using AES in ECB mode, with the AES key derived from the group DH key K as follows: $K_{ECB} = [\text{HMAC-SHA-1}_K(1)]_{128}$. Since Nm_i is unique, the use of ECB mode suffices. Since we can validate Nm_i based on the committed value Hm_i, no additional MAC value is needed to ensure integrity and authenticity for that encryption.

6.4 Fast, Consistent, Wireless Communication

We chose an Internet-based communication protocol over several other approaches. Bluetooth provides proximity and avoids use of a server, but can be slow to discover devices, may require barcode scans to improve discovery, and may be difficult for all devices to play equal roles in the exchange if one device acts as a host to the others. In addition, Bluetooth pairing may be restricted to certain devices depending on how the smartphone OS implementation. Ad-hoc Wi-Fi broadcast messages may allow each device to play an equal role, but suffer from lack of widespread availability on the most popular smartphone platforms. While proximity-based communication protocols may assist in the grouping process, they still cannot defend against an adversary with a strong amplifier and a high-gain antenna who could still participate in the protocol from a distance. Internet-level messages (Wi-Fi, GSM, EVDO) between smartphones and server provide an equal role for each device, and provides fast message exchange. Moreover, Internet-based communication enables remote operation of the information exchange, assuming the users can communicate over a channel that enables them to authenticate each other, such as voice over telephone or video conferencing (pure text over SMS or MMS could be easily spoofed in a MitM attack).

6.5 Server Data Management

Our server, written in Python for Google App Engine, consists of a simple database table to hold the message data submitted by each member of the group at any given time. The lifetime of this data is 10 minutes, since we have implemented a script which removes all entries older than that window of time.

In return for a device submitting the initial commitment C_i, the server assigns a simple grouping ID ID_i, and returns it to the user. This grouping ID is chosen by the server such that it will be:

- Pseudo-random, to reduced predictability.
- Low-entropy, to reduce user error and effort for choosing and entering the lowest number (Figure 5(c)).

These properties are achieved by querying the table for remaining sets of available grouping IDs first in the exclusive range 1–9, removing collisions with numbers currently in use, and using a random function to pick and assign from what remains. Should all 9 numbers in the lowest set be in use, the query is repeated for the range 10–99. Similarly, if those numbers are in use, we continue to increase the range to 100–999 and so on.

6.6 Server Supported Messages

Our SafeSlinger information exchange protocol depends

fundamentally on the ability for each client to validate all data received from other members without trusting the server or the network communication. Nevertheless, to reduce the effect of potential network-based attacks, we set up an SSL connection between the client and server.

The communication happens through a pull model, where the client sends HTTPS POST calls to the server to request transmission of as much information possible at each step.

Each client will accept information from up to $N-1$ other users, where N is the number of group members the user selected. The server stores each piece of client information for up to 10 minutes before an automatic cleanup routine removes it from the server database.

Our server supports the following messages:

1. M_i sends commitment C_i, server responds with grouping ID ID_i.
2. M_i sends ID_x of each C_x it has, server responds with any C_x client does not yet have.
3. M_i sends ID_x of each (HN_x, G_x, E_x) triplet it has, server responds with any (HN_y, G_y, E_y) triplet the client lacks.
4. M_i sends ID_x of each (Hm'_x, Hw_x) tuple it has, server responds with any (Hm'_y, Hw_y) tuple the client still needs.
5. In groups with more than 2 users, send the public group DH tree value V_i of an intermediate node of the group DH tree. (Section 3.2 describes the group DH tree.)
6. In groups with more than 2 users, request the public group DH tree value V_i of an intermediate node of the group DH tree.
7. M_i sends ID_x of each $\{Nm_x\}_K$ it has, server responds with any $\{Nm_y\}_K$ the client still needs.

6.7 Word Phrase Verification

Each user must compare their separate calculations of the hash of all data exchanged in the protocol as a PGP word phrase based 3-word phrase and we aim to discourage any careless comparison of this hash.

We use the standard PGP approach for converting a 24-bit hash value into 3 words. PGP uses two word lists, an "even" and "odd" list with 256 words each. In SafeSlinger, the word phrase is constructed from the first 24 bits of the 160 bit SHA-1 hash, as indicated in Step 11 of Figure 3. Based on the standard PGP approach, the first 8 bits select a word in the "even" list, the second 8 bits select a word in the "odd" list, and the final 8 bits select another word from the "even" list.

In addition to the common phrase displayed on each device, we construct 2 more decoy phrases for each device. In this way, users are forced to compare phrases with at least one other user and choose which phrase matches among them. However, to prevent attacks, all users need to validate that they all have the same word phrase. The word phrase provided in this exchange will enable out-of-band verification in proximity where users may view each other's screen, or over the phone / teleconference, where the users read their word phrase out loud. In some contexts, it may be considered impolite in common company to look at or take a picture of another user's phone, and thus this protocol provides screen privacy and sharing of phrases in comfortable conversation. The audio sharing of the verification phrase also allows users to perform the exchange remotely while speaking on the telephone, since they will recognize each other's voices and can be assured of the others physical presence in real-time remotely.

6.8 Word Phrase Collision Avoidance

If a word in our decoy word phrases is the same as in the actual word phrase, users may get confused and select the decoy word phrase as the match. Although unlikely, the words in a decoy phrase may match the words in a decoy phrase on another device, causing the user to select the decoy phrase which results in an error detected by the local device.

We want to avoid true randomness in the decoy phrases so that careless users will not chose the wrong phrase if the actual hash phrase and either of the decoy phrases contain the same word in the same position.

We thus chose our decoy phrases deterministically such that each decoy word will be unique across all decoy phrases displayed in the group. After computing the actual word phrase, we mark all words as used. Using the original 160-bit verification hash, we repeatedly hash it with SHA-1, using the bits produced as follows. We assign all decoy words to each decoy word list by user id ID_x in descending order, producing 24 bits for ID_a decoy phrase 1, then ID_a decoy phrase 2, then ID_b decoy phrase 1, then ID_b decoy phrase 2, until each client has produced their own 2 decoy phrases and can display them with the actual hash phrase. During the selection process, if a selected words is already marked as consumed, it will be skipped and the next available word in the list will be used. Newly selected words are also marked as consumed. Duplicate words (i.e., the first and third words may be the same since they are picked from the same "even" list) within the same phrase are permitted.

This method assures that all decoy phrases contain words that will not collide with each other or the actual word phrase for all users N. Since we use the words from the "even" list twice as fast as the "odd" list, we can at most support 63 users, as $((4*64)+2) > 256$.

7. APPLICATIONS

We have fully implemented the SafeSlinger Key Exchange as an API library in Android 2.1 and as a utility application in Apple iOS 3.0. Any third-party application for either platform may compile into or execute the key exchange including its GUI. Additionally, we have leveraged our key exchange to create a text messaging and a file exchange application for Android 2.1. We will discuss the potential of these applications here, and describe an application for providing secure introduction. The Android beta version of our key exchange and messaging implementation can be found on the Android Market. The iOS version of our key exchange will be available from the iTunes App Store soon (it is cur-

rently available as KeySlinger, but we will soon update it to SafeSlinger), and the full SafeSlinger iOS application with full encrypted messaging and file exchange support will be available in early 2012.

7.1 Secure Key Exchange API

The purpose of the secure key exchange API is to enable third-party applications to leverage SafeSlinger to exchange public keys with other users in an authenticated fashion. For example, a secure email application can export its public key to SafeSlinger and use it as a transport to safely sling the key to other users. On the other user's platform, the secure email application will pull out the public key and use it to authenticate received or encrypt to-be-sent email.

Our secure key exchange API may be launched from a third-party application. Cryptographic operations are computed using the operating system-provided libraries for Android and iOS, with the addition of open source OpenSSL[5] libraries for Apple iOS. Figure 6 shows the information flow between multiple devices during execution of the key exchange, outlined as follows.

1. Third-party application generates a public/private key pair.
2. Third-party application inserts its public key in the device's contact list.
3. Third-party executes the SafeSlinger key exchange API providing the location of its contact in the contact list and name of the public key to exchange as parameters.
4. During the SafeSlinger key exchange protocol (Section 5), multiple messages are exchanged between devices via our server, and validated independently by each device.
5. SafeSlinger key exchange saves the new public key and contact data received in the device's contact list.
6. Third-party applications may now make use of newly imported public keys from the contact list.

7.2 Messaging Implementation

We have implemented SafeSlinger text messaging and file exchange for the Android 2.1 OS and leverage the secure information exchange to share public keys that are used in turn to encrypt text messages and file data. When SafeSlinger first starts, it generates a RSA 2048-bit key pair. The application then obtains a Google Android C2DM Push token[6] for addressing the device. During a SafeSlinger information exchange, the public keys and push tokens of all group members are exchanged and imported into the address book.

The actual messages and files that are sent are encrypted and authenticated in the OpenPGP[7] message format, to provide compatibility with other PGP applications. The Android implementation uses the open-source Java Bouncy Castle[8] library for formatting OpenPGP messages. Our iOS implementation uses the the open source C OpenPGP:SDK[9] library for formatting OpenPGP messages.

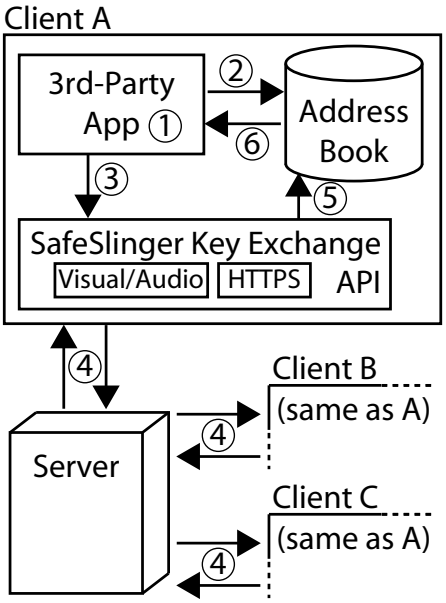


Figure 6: SafeSlinger secure key exchange API interaction.

The sender constructs an encrypted text message using an OpenPGP message format by encrypting the plaintext text message with the recipient's public key, and signed using the sender's private key. The sender uses the recipient's push token as an address to send the encrypted text message to. The recipient will decrypt the OpenPGP message using the recipient's private key and verify the OpenPGP signature using the sender's public key.

Sample screenshots from our messaging application are depicted in Figure 7. In 7(a) we can see the login screen requiring a password entry. 7(b) shows the message composition and the current selected image that is ready to be sent. Finally, 7(c) shows the received message screen along with a decrypted message that indicates a received file that is ready to be downloaded and viewed.

7.2.1 Push Message Notification

We make use of push notifications on smartphone operating systems to deliver message data, to avoid text messaging charges for our users, and to conserve battery energy. Push notifications are a better solution to any low-power device which must make constant contact to a server for data updates. Rather than requiring each application developer to implement a background task to keep a separate connection to their server to poll for updates, push notifications are an OS-provided unified update service in which the OS main-

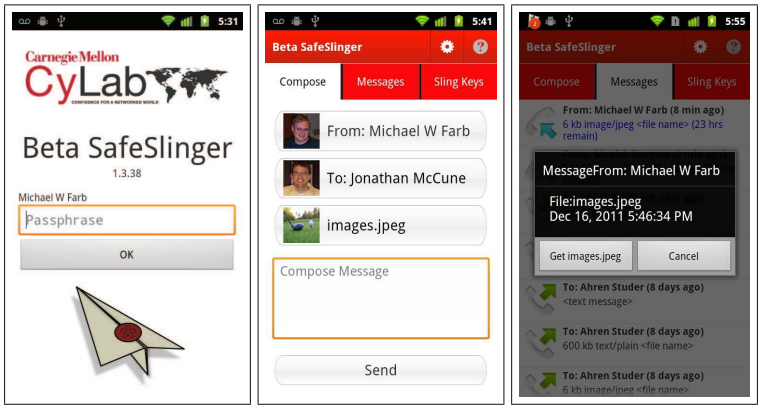
[5]http://www.openssl.org

[6]C2DM Push was first released for Android 2.2 OS. Unfortunately, Android 2.1 devices can only send but not receive messages, however, they represent about 10% of current Android devices.

[7]http://tools.ietf.org/html/rfc4880

[8]http://www.bouncycastle.org

[9]http://openpgp.nominet.org.uk



(a) Password login (b) Message composition (c) Message list with sample message

Figure 7: Secure Messaging Screenshots.

tains one connection to its own notification service. Then, each developer can register their application with the OS push service, and the OS only needs to maintain one connection to check for updates across several applications which the user may have installed, preserving battery energy.

Our Android implementation uses the Android Cloud to Device Messaging[10] framework (C2DM) for push notifications, and our iOS implementation uses the Apple Push Notification Service (APNS).[11] Since APNS is not available to use directly from Google App Engine, we use an intermediary service, Urban Airship (UA)[12] to bridge this gap.

7.2.2 *Messaging Server Construction*

We must work within the size limitations of these push messages. C2DM push messages cannot exceed 1024 bytes, while APNS messages my not exceed 256 bytes. This may not allow a OpenPGP formatted text message to fit, so we create a 160 bit random nonce to act as a message UUID or retrieval token for each message the user sends.

When a user wants to send a message, we pass to our server: a message retrieval token; the push token and notification type (C2DM or UA/APNS) of the recipient; an OpenPGP message containing text; identity, and file preview data; and optionally another OpenPGP message (up to 2 GB) containing the file itself. The two OpenPGP messages are stored in the server datastore for 24 hours before automatic deletion, and we construct and send a push message to the notification service type specified containing just the message retrieval token. When the recipient's OS push service collects the push message, it will launch SafeSlinger to read the message, and then SafeSlinger will notify the user that a message is available for download.

7.2.3 *Secure Text Messages*

The text message itself, along with user name or verification, timestamp, and preview data of any attachments are formatted as an OpenPGP message. After the recipient receives the push message, they can download the text message, decrypt and verify it.

7.2.4 *Secure File Transfer*

Similar to our Secure Messaging application, we have implemented SafeSlinger file transfer for the Android 2.1 OS and leverage the key exchange to share public keys and encrypt files stored on our smartphones such as images and music, or other large binary data. Our implementation allows any file just under 2GB in size to be transmitted as an OpenPGP message. The file formatted as a separate OpenPGP message described prior, and downloaded with the same retrieval token as the text message token.

We make use of the Intents in Android OS to advertise to other applications that SafeSlinger Messaging is available for sending media files to other people. In this way, users viewing a photo in their photo gallery application can select "share" from the menu and receive a list of applications which includes SafeSlinger to transmit the photo to another user. We have enabled images, video, audio, vCards, text, and other application files to use SafeSlinger for transmission to other users.

7.3 Secure Introduction

Based on the secure file transfer mechanism we have implemented, we can support secure introductions, where a common friend of two users sends contact data that includes public keys to each other. More concretely, consider Alice with two friends: Bob and Carol. Alice has performed a SafeSlinger exchange with both Bob and Carol and has thus received an authentic SafeSlinger public keys for both Bob and Carol, vice-versa, both Bob and Carol have Alice's authentic SafeSlinger public key. In a secure introduction, Alice first encodes Bob's contact information (which includes Bob's SafeSlinger public key and Push token) as a custom

[10] http://code.google.com/android/c2dm
[11] http://support.apple.com/kb/HT3576
[12] http://urbanairship.com/docs

vCard and uses an OpenPGP message format to protect secrecy and authenticity. Alice then sends this message via a SafeSlinger transfer to Bob. Hence, Bob can validate that the information indeed originates from Alice, whom he trusts not to send bogus information. Analogously, Carols trusts Bob's information received from Alice. Now that Bob and Carol have each other's public keys and push tokens, they can use SafeSlinger to securely communicate.

8. EVALUATION

We evaluated SafeSlinger on 2 to 5 devices. Our population includes 2 Motorola Droid devices under Android 2.2.3 OS and 3 Google Nexus S devices under Android 2.3.6 OS. Our devices were using a mixture of cellular messages through 3G, and Wi-Fi messages through Carnegie Mellon's campus wireless access points.

Our measurements include 5 phases of the SafeSlinger protocol between the sending the user's initial commitment (S4) and receiving the final decryption key (S17) as detailed in Figure 3. We include 3 phases in which client and server communicate: *ID Assignment, Data Verification, and Match Verification*, and 2 phases where users must select matching values: *Low ID Selection and Phrase Selection* (Figure 8).

- *ID Assignment* includes communication time to submit the user's initial commitment and for the server to assign and return an unique grouping ID (S4–S5).
- *Low ID Selection* includes the user's time to compare and enter the lowest grouping ID (S5–S8).
- *Data Verification* includes communication time between client and server to collect all other user's initial commitments under the chosen grouping ID, and to distribute and verify the decommitments (S8–S10).
- *Phrase Selection* includes the user's time to compare and select the 3-word phrase that matches with other user's (S10–S14).
- *Match Verification* includes communication time between client and server to distribute and verify all "match" or "no match" multi-value commitments, calculate the shared secret key, distribute intermediate group DH tree nodes (in case of groups with more than 2 users), and to and to distribute and verify the encrypted match nonce (S14–S17).

Except for the calculation of the shared secret key, these measurements do not include the time to generate nonces or symmetric keys, or to encrypt or decrypt the user's vCard data, which is negligible compared to the other overheads.

Our results (Table 1) indicate that the total run time for 2 users is a comfortable average of 17 seconds to complete the exchange. The total time is close to 35 and 40 seconds for 4 and 5 devices, respectively.

The increased time for larger numbers of participants is mainly due to the additional bandwidth required for the additional information, as well as the additional processing, especially for the STR group Diffie-Hellman protocol requiring expensive modular exponentiations. As we can see from the table, the execution time is efficient and usable.

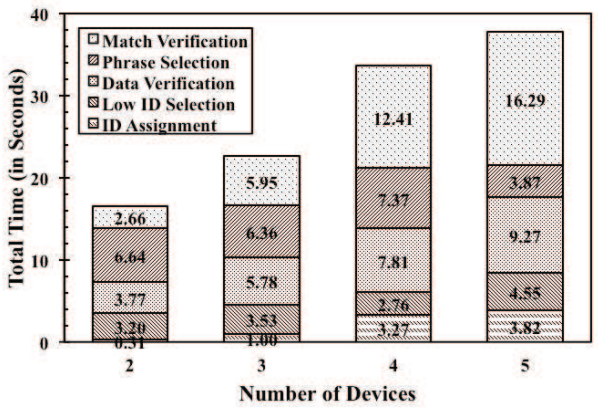


Figure 8: Time consumed during a SafeSlinger exchange.

9. RELATED WORK

Closely related research by Asokan-Ginzboorg [2], Abdalla et al. [1], Valkonen et al. [39], and Laur and Pasini [20] provide techniques for secure local establishment of a shared secret key without relying on PKIs or any prior trusted information. Unfortunately, a secret shared among nodes cannot be used to provide integrity and authenticity for exchanged messages, because any malicious group member with the key could have created that message. Therefore, a shared secret is insufficient for secure exchange of authentic information. In particular, GitM attacks are possible, exploiting the absence of message authenticity.

The most closely related works provide secure group-based exchange of contact information: GAnGS [8], SPATE [22, 23], Ho-Po Key [30], and Nithyanand et al. [32].

The GAnGS protocol [8] was designed to scale trust establishment to a larger number of users – it supports large groups of 25 or more users. Unfortunately, GAnGS requires users to perform many operations and is thus cumbersome to use. SPATE [22, 23] was designed for contact exchange in smaller groups (8 or fewer people). SafeSlinger offers numerous improvements over these prior systems: (1) GAnGS and SPATE required the acquisition of a 2D barcode displayed by another phone, which can require a significant amount of time especially in difficult lighting conditions such as bright sunshine; (2) GAnGS and SPATE use a visual hash function for users to perform comparison of the hash values – such a visual hash cannot support remote execution and enables users to simply click "match" without actually performing the comparison; (3) GAnGS and SPATE enable bystanders to learn everyone's contact information, potentially disclosing sensitive private information. We believe that the improvements that SafeSlinger offers are significant enough that it is now ready for wide-spread adoption. In particular,

Table 1: Average execution time per number of devices (*unit: second*), averaged over 5 runs with standard deviation in parenthesis

# of Devices	ID Assignment	Low ID Selection	Data Verification	Phrase Selection	Match Verification	Total
2	0.31 (0.24)	3.20 (0.99)	3.77 (0.18)	6.64 (1.66)	2.66 (0.23)	16.58 (2.39)
3	1.00 (0.61)	3.53 (1.17)	5.78 (1.07)	6.36 (1.66)	5.95 (1.69)	22.62 (3.84)
4	3.27 (1.62)	2.76 (0.90)	7.81 (1.05)	7.37 (3.73)	12.41 (2.89)	33.62 (6.01)
5	3.82 (0.64)	4.55 (1.06)	9.27 (0.82)	3.87 (0.38)	16.29 (1.61)	37.84 (1.08)

the privacy protection achieved with the group DH protocol is critical for people who want to protect their privacy. Ho-Po Key [30] key establishment is useful for large groups, requiring people to form a physical ring in space to avoid counting while detecting Sybil and other attacks. SafeSlinger uses Ho-Po Keys when users indicate a large group size.

Nithyanand et al. recently studied the usability of secure group association protocols [32]. Their results with real user studies concludes that the ideal group credential exchange protocol does not use a leader (i.e., is peer-based), requires users to count and input the number of participants, and requires users to verify Short Authentication Strings (SAS). Hence, their study confirms the approaches we have selected for SafeSlinger.

PGP key-signing parties [4] enable users to obtain each other's public key, but they are cumbersome for participants. SafeSlinger can be viewed as a more modern and usable approach using smartphones to securely exchange public keys.

Many researchers have studied device pairing or key setup between two devices [3, 5–7, 12, 13, 21, 24–26, 28, 35, 38]. These systems, however, do not easily generalize to multiple parties, as they would encounter the issues we describe in Section 4.

10. LIMITATIONS AND RESEARCH CHALLENGES

The main limitations of the current SafeSlinger application are: (1) its reliance on users to diligently perform the world-list comparisons, thereby ensuring that *all users* have the same hash value, and (2) its requirement that users check the received contact list entries before finally importing them into their address book. Although SafeSlinger is more usable and secure than all prior work that we are aware of, it is an open research problem to determine whether the reliance on the user can be further reduced without requiring additional trust assumptions on other users. Another research challenge is a formal verification of the protocol that includes user actions, which may help in exploring optimizations that limit reliance on the user.

11. CONCLUSION

To realize the vision of secure online communication, we need to overcome several human challenges: some users are ambivalent about security or privacy, most users lack security expertise, and many users prefer convenience over security and may not want to expend much effort for security.

To counteract these challenges, we designed SafeSlinger as an easy-to-use application that offers many benefits to drive usage. Per Metcalfe's law, the utility of a system grows with the square of the number of users. Our goal is thus to provide immediate utility to enable epidemic growth.

We achieve immediate utility through the robust exchange of contact list information between different smartphone platforms, which does not require any location information or leak private information outside the participating phones. SafeSlinger also provides simple and secure messaging and file transfer that is immediately usable. Because the messages are encrypted and require a password to access, many teens may find this appealing to protect their messages from peers and parents.

To seed epidemic growth, SafeSlinger supports secure remote exchange of contact list information and secure introductions, where a common friend can securely sling respective contact list information between two users, setting up a secure channel between them.

Through free multi-platform applications available on smartphone markets,[13] open documentation, and open-source code, we anticipate wide adoption of SafeSlinger. Assuming wide adoption, we hope to provide usable and secure communication for the masses, and a security platform that will enable numerous security services and applications.

12. ACKNOWLEDGMENTS

Numerous people helped with this project. We especially would like to thank Emmanuel Owusu for his help with an early version of this paper. We would also like to thank Virgil Gligor for his helpful suggestions.

This research was supported by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and W911NF-09-1-0273, from the Army Research Office, and by support from NSF under award CCF-0424422, and CNS-1050224, and by gifts from Google. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, CMU, Google, NSF or the U.S. Government or any of its agencies.

[13]The current version is available under SafeSlinger on the Android market. Our iPhone version will be released in January also under the name SafeSlinger. We currently have a version of KeySlinger on the iPhone market that supports the basic exchange, but not yet the messaging and file transfer.

13. REFERENCES

[1] Michel Abdalla, Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Password-based group key exchange in a constant number of rounds. In *Public Key Cryptography (PKC)*, pages 427–442, 2006.

[2] N. Asokan and Philip Ginzboorg. Key-agreement in ad-hoc networks. *Computer Communications*, 23(17):1627–1637, November 2000.

[3] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of the 9th Annual Network and Distributed System Security Symposium (NDSS)*, 2002.

[4] V. Alex Brennen. The Keysigning Party HOWTO. `http://cryptnet.net/fdp/crypto/keysigning_party/en/keysigning_party.html`, Jan. 2008.

[5] Bump technologies. `http://bu.mp/`.

[6] Mario Cagalj, Srdjan Capkun, and Jean-Pierre Hubaux. Key agreement in peer-to-peer wireless networks. *IEEE (Special Issue on Cryptography)*, 94:467–478, 2006.

[7] Claude Castelluccia and Pars Mutaf. Shake Them Up! A movement-based pairing protocol for CPU-constrained devices. In *Proceedings of ACM/Usenix MobiSys*, 2005.

[8] Chia-Hsin Owen Chen, Chung-Wei Chen, Cynthia Kuo, Yan-Hao Lai, Jonathan M. McCune, Ahren Studer, Adrian Perrig, Bo-Yin Yang, and Tzong-Chen Wu. GAnGS: Gather authenticate 'n group securely. In *Proceedings of the ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, September 2008.

[9] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[10] John R. Douceur. The Sybil attack. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, March 2002.

[11] Tom Espiner. Berners-lee: We Need PGP for the People. ZDNet. `http://www.zdnet.co.uk/news/security-management/2011/10/17/berners-lee-we-need-pgp-for-the-people-40094198/`, October 2011.

[12] Michael T. Goodrich, Michael Sirivianos, John Solis, Gene Tsudik, and Ersin Uzun. Loud and clear: Human-verifiable authentication based on audio. In *International Conference on Distributed Computing (ICDCS)*, page 10, 2006.

[13] Lars Erik Holmquist, Friedemann Mattern, Bernt Schiele, Petteri Alahuhta, Michael Beigl, and Hans-W. Gellersen. Smart-its friends: A technique for users to easily establish connections between smart artefacts. In *Proceedings of Ubicomp*, 2001.

[14] Hsu-Chun Hsiao, Yue-Hsun Lin, Ahren Studer, Cassandra Studer, King-Hang Wang, Adrian Perrig, Hung-Min Sun, Bo-Yin Yang, and Hiroaki Kikuchi. A study of user-friendly hash comparison schemes. In *Proceedings of Annual Computer Security Applications Conference (ACSAC)*, December 2009.

[15] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Group key agreement efficient in communication. *IEEE Transactions on Computers*, 53(7):905–921, July 2004.

[16] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Tree-based group key agreement. *ACM Transactions on Information Systems Security (ACM TISSEC)*, 7(1):60–96, May 2004.

[17] Cynthia Kuo, Ahren Studer, and Adrian Perrig. Mind your manners: Socially appropriate wireless key establishment for groups. *Proceedings of First ACM Conference on Wireless Network Security (WiSec)*, March 2008.

[18] Sven Laur, N. Asokan, and Kaisa Nyberg. Efficient mutual data authentication using manually authenticated strings. Report 2005/424, Cryptology ePrint Archive, November 2005.

[19] Sven Laur and Kaisa Nyberg. Efficient mutual data authentication using manually authenticated strings. In *Cryptology and Network Security (CANS)*, pages 90–107, 2006.

[20] Sven Laur and Sylvain Pasini. Sas-based group authentication and key agreement protocols. In *Proceedings of Public Key Cryptography (PKC)*, 2008.

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

[22] Yue-Hsun Lin, Ahren Studer, Yao-Hsin Chen, Hsu-Chun Hsiao, Li-Hsiang Kuo, Jason Lee, Jonathan M. McCune, King-Hang Wang, Maxwell Krohn, Phen-Lan Lin, Adrian Perrig, Hung-Min Sun, and Bo-Yin Yang. Spate: Small-group pki-less authenticated trust establishment. *IEEE Transactions on Mobile Computing (TMC)*, 9(12), December 2010.

[23] Yue-Hsun Lin, Ahren Studer, Hsu-Chun Hsiao, Jonathan M. McCune, King-Hang Wang, Maxwell Krohn, Phen-Lan Lin, Adrian Perrig, Hung-Min Sun, and Bo-Yin Yang. SPATE: Small-group PKI-less authenticated trust establishment. In *Proceedings of the 7th Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*, June 2009.

[24] 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`, Aug. 2006.

[25] Vic Lortz, David Roberts, Bozena Erdmann, Frank Dawidowsky, Kevin Hayes, James C. Yee, and Takashi Ishidoshiro. Wi-Fi Simple Config Specification, version 1.0a. Now known as Wi-Fi Protected Setup, Feb. 2006.

[26] Justin Manweiler, Ryan Scudellari, and Landon P. Cox. SMILE: Encounter-based trust for mobile social services. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, November 2009.

[27] Moxie Marlinspike. Breaking SSL with null characters. In *Presented at Black Hat*, 2009.

[28] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. Seeing-Is-Believing: Using Camera Phones for Human-Verifiable Authentication. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2005.

[29] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology (Crypto)*, pages 369–378, 1987.

[30] Ghita Mezzour, Ahren Studer, Michael Farb, Jason Lee, Jonathan McCune, Hsu-Chun Hsiao, and Adrian Perrig. Ho-Po Key: Leveraging physical constraints on human motion to authentically exchange information in a group. Technical Report CMU-CyLab-11-004, CyLab, Carnegie Mellon University, December 2010.

[31] Dave Neal. Defcon hackers get hacked over 4G. `http://www.theinquirer.net/inquirer/news/2100989/defcon-hackers-hacked-4g`, August 2011.

[32] Rishab Nithyanand, Nitesh Saxena, Gene Tsudik, and Ersin Uzun. Groupthink: Usability of secure group association for wireless devices. In *Proceedings of Ubicomp*, September 2010.

[33] Associated Press. Carrier IQ executive says FBI approached firm about its tracking tool, but denies relationship. `http://www.washingtonpost.com/business/technology/2011/12/15/gIQAHNffwO_story.html`, December 2011.

[34] Christopher Soghoian and Sid Stamm. Certified lies: Detecting and defeating government interception attacks

against SSL. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, July 2010.

[35] Frank Stajano and Ross J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Protocols Workshop*, pages 172–194, 1999.

[36] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A Secure Audio Teleconference System. In *Advances in Cryptology (Crypto)*, volume 403 of *Lecture Notes in Computer Science*, pages 520–528. International Association for Cryptologic Research, Springer-Verlag, 1988.

[37] M. Steiner, G. Tsudik, and M. Waidner. Key Agreement in Dynamic Peer Groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, Aug. 2000.

[38] Ahren Studer, Timothy Passaro, and Lujo Bauer. Don't bump, shake on it: The exploitation of a popular accelerometer-based smart phone exchange and its secure replacement. In *Proceedings of ACSAC*, 2011.

[39] Jukka Valkonen, N. Asokan, and Kaisa Nyberg. Ad hoc security associations for groups. In *Security and Privacy in Ad-Hoc and Sensor Networks (ESAS)*, pages 150–164, 2006.

[40] Serge Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *Advances in Cryptology (Crypto)*, pages 309–326, 2005.

[41] David Wagner, Bruce Schneier, and John Kelsey. Cryptanalysis of the cellular message encryption algorithm. In *Advances in Cryptology (Crypto)*, 1997.

[42] Alma Whitten and J.D. Tygar. Why Johnny can't encrypt. In *USENIX Security*, August 1999.

[43] Philip R. Zimmermann. Pgpfone, pretty good privacy phone, owners manual, version 1.0 beta 7. `ftp://ftp.pgpi.org/pub/pgp/pgpfone/manual/pgpfone10b7.pdf`, July 1996.