Don’t Bump, Shake on It: The Exploitation of a Popular Accelerometer-Based Smart Phone Exchange and Its Secure Replacement

Ahren Studer, Timothy Passaro, Lujo Bauer

February 10, 2011

CMU-CyLab-11-011

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Don't Bump, Shake on It: The Exploitation of a Popular Accelerometer-Based Smart Phone Exchange and Its Secure Replacement

Ahren Studer *Timothy Passaro* *Lujo Bauer*
Carnegie Mellon University

Abstract

As the capabilities of smartphones increase, users are beginning to rely on these mobile and ubiquitous platforms to perform more tasks. In addition to traditional computing tasks, people are beginning to use smartphones to interact with people they meet. Often this interaction begins with an exchange, e.g., of cryptographic keys. Hence, a number of protocols have been developed to facilitate this exchange. Unfortunately, those protocols that provide strong security guarantees often suffer from usability problems, and those that are easy to use may not provide the desired security guarantees.

In this work, we highlight the danger of relying on usable-but-perhaps-not-secure protocols by demonstrating an easy-to-carry-out man-in-the-middle attack against Bump, the most popular exchange protocol for smartphones. We then present Shake on It (Shot), a new exchange protocol that is both usable and provides strong security properties. In Shot, the phones use vibrators and accelerometers to exchange information in a fashion that demonstratively identifies to the users that the two phones in physical contact are communicating. The vibrated information allows the phones to authenticate subsequent messages, which are exchanged using a server. Our implementation of Shot on DROID smartphones demonstrates that Shot can provide a secure exchange with a similar level of execution time and user effort as Bump.

1 Introduction

As the functionality and computing power of smartphones increase, users are leveraging these devices to perform more of their computing tasks. In addition to traditional tasks such as email, gaming, banking, and maintaining a schedule, the mobility and ubiquity of smartphones allows people to use these devices to establish ad hoc associations with people they meet. For example, people may exchange phone numbers, email addresses, and social network identities or even use their phones to enable the exchange of funds via an online service (e.g., PayPal). During these exchanges, phones typically use the wireless channel to perform the majority of the communication. The wireless channel makes exchanges easier for users since they do not have to carry cables to connect the phones. However, users are unable to observe the endpoints of wireless communication and without a secure protocol a malicious party can insert themselves into the exchange as part of a man-in-the-middle (MitM) attack.

Exchange protocols for smartphones require varying levels of user involvement and provide different security guarantees. The widely deployed Bump [2] and Bluetooth pairing [11] protocols represent distant points in the spectrum of user involvement and security guarantees. Bump requires users to only perform a simple gesture, but, as we demonstrate, is vulnerable to MitM attacks under realistic conditions. Bluetooth pairing [11] is resistant to such attacks, but requires significant user involvement to correctly compare a checksum across phones.

In the first part of this paper, we highlight the danger of relying on usable-but-perhaps-not-secure protocols by demonstrating a MitM attack against Bump. Bump is the most popular exchange protocol for smartphones[1] and claims to provide an excellent user experience while ensuring security.[2] Due to the simplicity of the operation, Bump is used in a number of applications. For example, PayPal's mobile app uses Bump to exchange account information and send money to nearby friends

[1] Popularity is based on data from `http://techcrunch.com/2011/01/19/iphone-ipad-top-app-downloads/` and `http://www.androidapps.com/` which use download count to determine popularity. According to the Bump blog, over 25 million users have installed Bump.

[2] According to Bump's webpage, "With Bump *you* are in control of deciding with whom you share your information. You don't have to worry about anyone being able to get at your information unless you physically bump your phone with theirs." [2]

[20]. Bump's security is based on the Bump server's ability to determine what phones are physically interacting based on the time, location, and the force with which the two phones were physically bumped together. Bump provides a very nice user experience, but is vulnerable to attack. An attacker may be able to observe when and where users bump their phones together and estimate the force of the bump. Without any secret information to identify the pair of phones to the server, an attacker can submit similar information about a bump to the server. When presented with similar information, the server may transfer data between the wrong phones. We demonstrate how an attacker under realistic conditions can use this approach to launch a MitM attack against Bump. In addition to Bump, other works have also suggested using accelerometers or vibrators and accelerometers to facilitate an exchange information between phones [4,5,10,14,17,23]. However, our attack on Bump and prior work [9] demonstrate that a physically present attacker may be able to violate the secrecy of this channel, requiring a new approach to ensure security.

In the second part of this work, we demonstrate how phones can leverage accelerometer readings to assist in the secure exchange of information while maintaining the limited user involvement offered by Bump. We propose Shake on It (Shot), a protocol designed specifically for smartphones that requires little user interaction. The novel idea is to use phones' vibration function and accelerometers as an authentic, but not secret, human-observable communication channel. The phones leverage data exchanged on this channel to verify data exchanged over the wireless channel. Only when the two phones are in physical contact can they communicate via the vibration to accelerometer channel, providing demonstrative identification to the users of the devices exchanging information [1]. If a remote party (one not in physical contact) tries to inject information on the channel, users will notice the additional vibrations (a potential attack) and stop the exchange. Given attackers can eavesdrop on this channel [9], the phones use symmetric and asymmetric cryptography to authenticate exchanged information and detect active attacks against the wireless channel, without users having to compare any information. When two people run Shot, they only have to perform three simple tasks: select what data is to be exchanged, hold the phones together until the phones beep to indicate completion, and cancel the exchange if they feel vibrations from devices other than the two phones. We argue that Shot occupies a new point on the spectrum of exchange protocols, providing greater security than comparably convenient protocols, and greater convenience than comparably secure protocols.

We have implemented Shot on a commodity smartphone and evaluated the performance of the protocol. Our analysis and evaluation show that Shot provides a secure exchange while requiring the same level of user effort and execution time as the popular Bump protocol. When evaluated on the DROID smartphone, Shot was able to reliably complete an exchange in 15.8 seconds, on average.

In summary, this paper offers the following contributions: (1) demonstration of an attack on Bump under realistic conditions; (2) a description of the Shot exchange protocol for smartphones, which leverages an authentic, but not secret, vibrator-to-accelerometer channel; and (3) an implementation of the Shot protocol on a smartphone.

2 Problem Definition

The goal of this work is to provide two people (users A and B) who meet in person a user-friendly mechanism that allows the authentic exchange of information (A's information I_A and B's information I_B) using their phones (P_A and P_B). After the exchange is complete, P_A will have received I_B and P_B will have received I_A, or P_A and P_B will both detect with high probability that an error has occurred and discard the information.

A user-friendly solution is any exchange that only requires limited user involvement and can complete execution in a reasonable amount of time. The exchange should not require the users to type several bytes worth of information into either phone or compare a checksum (e.g., a string of hex digits [11, 13, 26], a series of words [8], or a graphical image [15, 21]). User studies have shown that a redesigned interface [25] or the comparison of words or images [12] reduce the number of errors, but still require non-negligible user involvement. The desired level of user involvement is a simple action, such as a physical gesture, that allows the phone to determine the other phone in the exchange. Solutions exist that require the user to take a photograph of the other phone [18], shake the two phones together [4, 10, 14, 17], or simply point the phones at each other [1], but each solution has drawbacks that negatively impact security or operation (see Section 9 for a discussion of related work). Given the wide acceptance of Bump, we consider the execution time of a Bump exchange as a reasonable amount of time. Depending on the platform, Bump takes between 9.4 and 37.8 seconds, from starting the application to receiving the exchanged data. Section 8 has more details on the execution time of Bump.

During an exchange, a malicious party M may attempt to inject its own information (I_M) or other data into the exchange such that P_A or P_B accept something other than I_B or I_A, respectively. A secure exchange allows P_A and P_B to exchange I_A and I_B or detect the insertion of any other information into the exchange with a high probability.

After discussing our assumptions, we give a detailed description of our attacker model.

2.1 Assumptions

In this work, we assume smartphones are equipped with the hardware, software, and connectivity needed to execute a Bump or Shot exchange.

Current smartphones are equipped with a vibrator and an accelerometer to provide silent notifications to the owner and to allow correct orientation of an image on a rotatable screen. Smartphones also allow the installation of generic software that can access the vibrator and accelerometer functionality. Smartphones also have Internet connectivity the majority of the time via the cellular network or a local WiFi access point.

2.2 Attacker Model

An attacker's goal during the smartphone exchange between phones P_A and P_B is to convince P_A or P_B to accept information other than I_A and I_B. We consider an attacker that may be in the same room as A and B, knows the value of I_A and I_B, and has bounded computational capabilities. We also assume any software on P_A and P_B is outside of the attacker's control.

We assume the attacker has control over the wireless channel between the two phones and is able to intercept, modify, delay, or inject messages. However, the attacker does not control human observable channels between the two phones (i.e., the visual or vibration channel). The attacker can accurately eavesdrop on these channels. However, users can detect the attacker's attempts to insert information on a human observable channel (i.e., insertion of a third phone into the visual channel or shaking the phones to inject data onto the vibration channel).

An attacker knows a priori what information the two users want to exchange. Given most users are exchanging contact information (e.g., phone numbers or online account IDs), this information can be found online.

It is infeasible for a computationally bounded attacker to break various properties of different cryptographic primitives of the appropriate strength. Hence, hash functions with sufficiently long outputs are one-way and second pre-image resistant (i.e., given a hash function $h()$ and a hash output $y = h(x)$, an attacker is unable to find x or another value x' such that $h(x') = y$). We also assume digital signatures are secure against selective forgery. This means that without knowledge of the private key, it is infeasible for the attacker to create a signature (σ) for an attacker selected message m such that the corresponding public key verifies the signature message pair (σ, m).

3 Bump Exchange Protocol

In this section, we begin by describing the Bump exchange. We then explain why the exchange is vulnerable and provide an analysis of our attack and its impact.

3.1 The Bump Protocol

The Bump exchange is meant to allow two phones (P_A and P_B) to exchange information. During an exchange, the users physically bump P_A and P_B together; each phone sends information about the bump and the information it wants to exchange, including a user-selected identifier (ID) and some additional data, to the bump server; the server uses the time, location, and force of the bumps to determine which phones want to exchange information and returns to each phone the other phone's information; and the user decides to save the information based on the ID in the information. Based on information provided on Bump Technology's webpage (`http://www.bu.mp`) and timing of packets sent during a Bump exchange, we were able to determine approximately how the Bump protocol works. We can divide the protocol into two main steps: Initialization and Exchange. Figure 1 provides a rough outline of the steps for a single phone involved in a Bump exchange.

Initialization involves each phone determining its location and connecting to the server. When Bump begins, the phone uses one of many techniques to determine its current location (Step 1). Popular smartphones can determine their location outdoors using GPS. When indoors, smartphones use nearby cell tower identifiers or WiFi access point information to estimate the current location. Even in the best case, these location estimations are within an accuracy of several meters. The phone then connects to the server via TLS [7] (Step 2). TLS provides secrecy and authenticity of communication with the server.[3] During the TLS handshake, the phone verifies it is communicating with the Bump server and not an attacker impersonating the server. Once established, the communication with the server is encrypted and authenticated to prevent an attacker from intercepting or modifying any information the phone exchanges with the server. Once initialization is complete, the phone knows approximately where it is and that it is connected to the correct Bump server.

After initialization is complete, the users bump phones together to start the actual exchange. During the physical bump, the phone's accelerometer measures the approxi-

[3] The Bump FAQ claims the new version of Bump does not use TLS, but is "encrypted end-to-end". However, analysis of intercepted traffic between an iPhone and the Bump server indicates that TLS is still in use. Even if TLS is "phased-out" the server stills needs to perform the matching task, which we attack to subvert Bump.

Initialization:

1.	P_A :	$loc \leftarrow findLoc()$	The phone determines its current location.
2.	$P_A \overset{Internet}{\longleftrightarrow} S$:	establish TLS	The phone establishes a TLS connection with the server.

Exchange:

3.	$A \overset{acc.}{\rightsquigarrow} P_A$:	a_{bump}, t	Bumping the phone induces an accelerometer reading a_{bump} at t.
4.	$P_A \overset{TLS}{\longrightarrow} S$:	a_{bump}, t, loc, I_A	The phone sends the acceleration, time, and location of the bump and the user's info to the server.
5.	S :	$info = match(a_{bump}, t, loc)$ $resp = info.ID$	The server checks its database of recent bump requests for similar accelerometer readings in a similar location, and returns the ID. $resp = \emptyset$ when no match is found. $resp =$ **"Bump Again"** when > 1 match is found.
6.	$S \overset{TLS}{\longrightarrow} P_A$:	$resp$	The server returns the ID of the match (or the error code).
7.	$P_A \overset{Screen}{\longrightarrow} A$:	If (($resp \neq \emptyset$) and ($resp \neq$ **"Bump Again"**)) **" Connect with** $resp$**? Yes/No"**	After a successful match, the phone asks the user if they want to save the information.

8. The phone then sends the user's selection to the server. If both phones return "yes", the server returns the other phone's data. If either phone returns "no", the server tells the other phone the exchange was cancelled.

Figure 1: Operations during a Bump exchange where user A uses phone P_A to exchange information I_A using the server S.

mate force of the bump (Step 3). Once a bump has occurred, the phone sends the acceleration associated with the bump, when the bump occurred, the phone's current location, and whatever information is to be exchanged to the server, including an ID and the data to be exchanged (Step 4). Using the received information, the server tries to match the bump with other recent bumps based on the force, time, and location (Step 5). Because of inaccuracies in the phone's location finding, clock, and accelerometer measurements, the server uses approximate matching and considers any values that are similar a valid match. If no other database entry has similar bump properties, the server responds indicating their was no match found (we use $\emptyset$ to indicate this). If more than one database entry is similar, the server asks the phone to bump again. The assumption is that if, for example four phones were bumped at approximately the same time and location with similar forces, a second set of bumps will yield information that makes the pairs distinct. If a single match exists, the server returns the identifier from the matching request (Step 6). The server tags the database entries associated with the match so that the same IDs are not returned to other requests (not shown in Fig. 1 for simplicity). The phone asks the user to confirm if she wants to exchange information with the ID the server returned (Step 7). It is important to note that the ID used during this confirmation is a value which an attacker can select to mimic a legitimate party in the exchange (i.e., the server will return the same ID when the exchange is correct and when under attack). After both users in the pair click "Yes", the server returns the data associated with the pair, which the phones then save. If either user clicks "No", the server never releases the full data (only the ID is always sent) and the exchange is cancelled.

3.2 Vulnerabilities in Bump and Their Exploitation

Bump is insecure due to three aspects: inaccuracies in the measurement of the physical aspects of the bump, the ability of a sender to control the ID used during confirmation, and server flexibility in accepting delayed bump requests. Given these three items, an attacker is able to successfully launch a MitM attack.

Sensor Inaccuracies. Given limited sensor accuracy, the phone is unable to know its exact location or how hard the phone was bumped. Given inaccuracies in different phones' clocks, the server is unable to know exactly when a bump occurred. Without accurate information, the server must use approximate matching, which can associate bumps that occurred several meters apart, with different forces, and at slightly different times. We were able to confirm that the server will match bumps that differed in each of these three aspects. Unless every user is willing to pay for more accurate sensors for their phones, the server will be forced to use approximate matching and maintain this vulnerability.

Without a way to decrypt TLS traffic, we were unable to determine the range of acceleration, location, or time values the server would consider as similar. However,

during testing, we were able to bump 2 phones across the room from each other ($\sim$10 meters apart) with different forces (e.g., slam one against a table and tap the other) at different times, and still have the server match the two phones. This vulnerability allows an attacker to block P_B's bump request and have the server match an attacker's request with P_A. However, a full MitM attack is not achieved because both the confirmation question ("Communicate with [ID]?") and the failure of P_B's request allow the users to detect the attack.

Spoofable ID. Circumventing the confirmation question such that a MitM attack goes undetected in Bump is simple. A participant controls the ID sent to the other phone during an exchange. As such, the attacker can use the same ID as the victim it is impersonating. This is still a valid attack because the underlying data is different (e.g., the email or public key the users accept are the attacker's). Bump could require users to compare all of the exchanged data or a hash of the data to remove this vulnerability. However, that approach would negatively impact usability, resulting in a protocol similar to the Bluetooth exchange. With spoofable IDs, all that remains is ensuring that both P_A and P_B are able to complete exchanges for the MitM to go undetected.

Acceptance of Delayed Requests. Different networks can cause different delays, such that the request from P_A arrives before the request from P_B, which arrives long after when P_B claims to have bumped. Given this potential delay, if no matching database entry exists, the server will wait some period of time before responding to P_A's request, rather than responding you were the only one to bump. The server also considers P_B's request with the older bump time valid because the delay may be network related. However, to improve response times, the server will associate a phone with a similar bump, rather than waiting until the correct request arrives. In short, Bump accepts stale requests, but will try to find a match as soon as possible to provide a better user experience.

This acceptance of delayed requests and preference for fast responses means an attacker can inject her own information into a legitimate exchange and allow both victims requests to complete. All the attacker has to do is delay P_B's request until the server associates P_A's request with an attacker submitted request. After the server associates P_A with the attacker, the attacker can forward P_B's delayed request and a second attacker generated request that allows P_B's exchange to complete successfully.

The Attack. An attacker with the ability to submit similar bumps to the server, spoof legitimate users' IDs, and allow both legitimate requests to complete can successfully complete the following MitM attack against Bump. Figure 2 sketches the attack. After A and B bump phones, the attacker delays P_B's request containing I_B and sends its own request impersonating P_B with information I_{M_B} and similar bump values. The server associates the attacker's request with P_A's request since no other similar entries exist in the database. Next, the attacker forwards P_B's delayed request and sends a request impersonating P_A with information I_{M_A} and the same bump values. In response, the server associates those two requests. Delay Δ, the duration of time by which the attacker delays the server's receipt of P_B's request, is an important parameter in the attack. If Δ is too short, the server will correctly associate the legitimate users' bumps or claim all of the bumps were too similar and ask the phones to bump again. If Δ is too large (> 3 seconds), the server generates an error and returns $\emptyset$ in response to the delayed request.

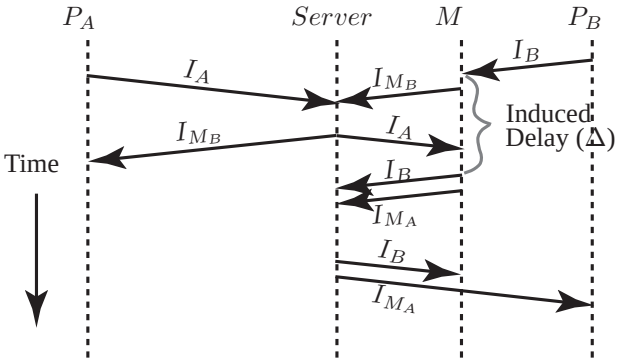

Figure 2: Timing of packet exchanges during the attack on Bump (accelerometer and location information omitted for clarity).

In the next subsection, we describe how such an attack can happen in practice and analyze the probability of a successful attack based on different values of Δ.

3.3 Attack Implementation and Analysis

To successfully perform a man-in-the-middle attack against Bump an attacker must be able to observe the bump and delay packets. A maliciously controlled access point (AP) provides control of packets. In our attack setup, we use a MacBook Pro with OS X 10.5 and Dummynet [3] as an AP. The laptop is connected to the Internet via the ethernet port and can forward packets from the WiFi network to the Internet. In a real attack, the attacker could trick users into associating with the attacker's AP by selecting a common SSID (e.g., "linksys"). By default, a phone will use a WiFi network with a known SSID, rather than the cellular network.

To evaluate the attack, we used real phones with three different people to bump the two sets of phones. We used 4 different iPhone[4] 3G smartphones running iOS version

[4]We used iPhones for our attack because it supports a more recent

4.2.1 and Bump 2.4.0 (1 each for P_A and P_B and 1 for each of the attacker's roles, P_{M_A} and P_{M_B}). Two humans bumped phones P_A and P_B together while another human played the role of the attacker and tried to bump phones P_{M_A} and P_{M_B} together at roughly the same time and with the same force. This is meant to simulate an attacker observing victims across the room using bump to exchange phone numbers at a bar or exchange money to reimburse one user for the other user's share of a tab. Dummynet was configured to delay P_B's and P_{M_A}'s requests by Δ as part of the attack (see Figure 2).

To evaluate Bump, we varied the induced delay (Δ) from 0 to 3 seconds in increments of 0.5 seconds and ran 10 exchanges for each setting. During an exchange, a phone can experience one of three potential outcomes:

SE Successful Exchange: The server returns a potential match and asks if the user wants to communicate with that ID.

BA Bump Again: The server finds multiple similar entries and asks the phone to bump again.

OO Only One: The server is unable to find a similar entry in the database. During analysis, we found that this is also returned if the induced delay is large.

Based on the individual phones' outcomes, we classify the result of the attack as one of the following:

- **Successful Exchange:** P_A receives B's information (**SE**) and P_B receives A's information (**SE**).
- **Successful Attack:** P_A receives the impersonation of B (I_{M_B}) and P_B receives the impersonation of A (I_{M_A}). Both phones have outcome **SE**, but receive attacker's information.
- **Detectable Attack:** P_A receives the impersonation of B (I_{M_B}), but P_B receives bump again or only one. **SE** for A and **BA** or **OO** for B.
- **Other:** P_A and P_B receive Bump Again or Only One. Neither phone experiences outcome **SE**.

Our results are summarized in Figure 3 and show that a MitM attack against Bump is feasible. We found the optimal induced delay (Δ) to be around 2 seconds. With that delay, 70% of the attacks succeeded. With less delay, the server detects multiple similar requests and returns

and reliable version of Bump. iPhones are able to run Bump version 2.4, while phones with Android are limited to version 1.3.2. If Bump Technologies secured the application between major revisions, attacking version 1 would be futile. The Android version of Bump was also much less stable during our testing. Over the course of 20 exchanges without attacker interference, the iPhone version was able to exchange data 19 times. However, the Android version was only able to exchange data 7 out of 20 times.

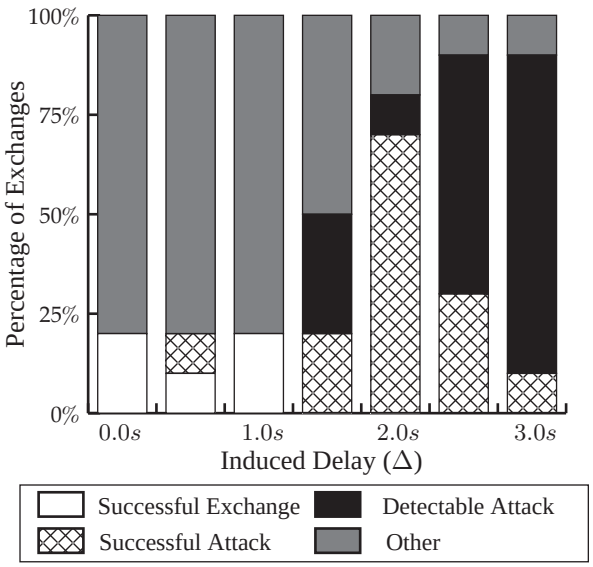

Figure 3: Attack Results on Bump with Varying Values of Δ

"Bump Again". With more delay, the server detects the long delay for P_B's request and instead of associating the request with I_{M_A}, returns an "Only One" response. For delays longer than 3 seconds, the attack is detectable; the server consistently pairs I_A and I_{M_B}, but returns "Only One" in response to I_B and I_{M_A}.

3.4 Attack Impact

A MitM attack against Bump has varying levels of impact depending on the application using Bump, including user annoyance, interception and/or impersonation during personal communication, and theft of money.

Some attacks merely result in user annoyance. For example, smartphone games such as BumpFour (a version of Connect Four) leverage Bump to establish a connection between opponents. In the event of a MitM attack, a malicious party can insert themselves between the two phones and play against the victims.

When Bump is used to exchange contact information, a MitM attack threatens the security of any communication that relies on the exchanged information. The standalone Bump application, which we attacked in Section 3.3, exchanges contact information and creates a new address book entry with the received data. If the victims send text messages, phone calls, emails, or other online communication using the information received during a MitM attack, the attacker, rather than the intended user, will receive the information. In addition, the attacker can impersonate one victim if the recipient relies on the information received from Bump (e.g., an email

address) to identify the source.

An attacker can exploit Bump to steal money. The PayPal smartphone application allows users to transfer money and use Bump to identify the transfer endpoints. However, a MitM attack against PayPal does not allow the attacker to empty a user's account. When using Bump to transfer money, the party sending the money selects the amount (or at least confirms the amount) to send. This limits the attacker to stealing only the amount one user meant to send. In addition, PayPal uses an account's email addresses as the ID. Given most users will simply trust the Bump exchange if it does not fail, an attacker could insert their own email address and steal the money. However, diligent users may notice the wrong email address and cancel the exchange. An attacker can create a similar looking email using "typejacking" to effectively evade detection by all but the most diligent of users [6]. It is important to note that the onus of detection in such a scenario is on the users, and Bump itself provides no protection on its own.

Exploitation of Bump provides an attacker with a range of capabilities. While some attacks only allow the attacker to annoy legitimate users, other instances of the attack enable interception of personal communication or money.

In this section, we have analyzed the Bump exchange, demonstrated how an attacker can successfully launch a man-in-the-middle attack against the protocol, and discussed the implications of the attack.

4 Exchange Guidelines

This section discusses some of the lessons learned and problems encountered in Bump. Based on these guidelines and challenges, we developed the Shake on It (Shot) exchange, described in Section 5.

Server-based Communication: Smartphones are connected to the Internet the majority of the time via the cellular network or WiFi. By using a server to exchange information, Bump circumvents some challenges associated with trying to establish local communication. For example, iPhones only allow Bluetooth connections to other iPhones or computers.[5] Ad hoc communication over WiFi is another option, but requires users of Android-based phones to subvert the OS to enable phone-to-phone WiFi broadcast.[6] A server allows smartphones from different vendors to communicate quickly and easily, without requiring the owner to modify the operating system.

One drawback to using a server to communicate is that the server needs some information to know which phones are trying to exchange data. With wireless communication, the phones can assume any broadcast data came from the other phone. Given all phones using Shot share the same server, the server needs a way to differentiate each pair of phones. As such, before the phones communicate via the server, the phones must agree on a value we call the "pair identifier" which allows the server to route traffic from one phone to the other.

Accelerometers as an Authentic Channel: A number of works use a bump, shake, or gesture to intuitively indicate which phones are to exchange data [2, 4, 10, 14, 17]. The accelerometer reading provides a way to convert the physical interaction into a label which identifies potential endpoints to a server and/or to derive a secret used to detect a MitM attack. This human observable communication also provides the users demonstrative identification of the endpoints of the exchange.

Unfortunately, a physically present attacker can observe the movements and leverage real-time motion tracking [16] and high speed cameras to quantify the accelerations during the bump, shake, or gesture. Once the accelerations are known, the information the phones share lacks the secrecy needed to secure communication. Rather than assuming secrecy, we need a protocol that can provide security and usability with only authenticity.

Attack Detection on the Phones: Given attackers can observe any information received by the accelerometer, the phones are unable to present any information to the server that allows the server to isolate the correct endpoints in an exchange. Without such a mechanism, the phones may receive the wrong information, but are unable to detect the error. Increasing user involvement by comparing the received data would solve the problem, but reduces usability. Instead, we want a protocol that allows the phones to detect reception of the wrong information from the server while limiting the user involvement to something as simple as putting the phones together.

Based on these observations, we need a protocol that begins by establishing a pair identifier. The phones can use the pair identifier to exchange information using the server and use authentic information exchanged via accelerometers to verify the correct phones and data were involved in the exchange, without requiring secrecy or involving users in the verification.

5 The Shake on It (Shot) Exchange

In this section, we describe Shot, a secure exchange protocol for smartphones that provides a user experience similar to Bump. We leverage the smartphone's vibration function to transmit an authentic phone-selected message

[5] `http://discussions.apple.com/thread.jspa?threadID=1460770`

[6] `http://code.google.com/p/android/issues/detail?id=82`

from one phone to the other phone's accelerometer. This allows the phones to bootstrap communicate via an untrusted server and verify received data without involving the user. We begin with a discussion to explain why we made certain design decisions and provide an overview of the protocol to describe the high level goals of Shot. Section 5.2 contains a detailed explanation of Shot. We conclude this section with a a security analysis of Shot.

The version of Shot presented here only ensures the integrity of the exchanged information. If secrecy is desired, the two parties can use Shot to authentically exchange public keys, and use those keys to establish a shared symmetric key for encryption of information.

5.1 Shot Overview

Based on the guidelines in Section 4, we want a protocol that takes advantage of smartphones' accelerometers for demonstrative identification and authentic communication. For Shot, we take this one step further and leverage the smartphone's vibration function to send a phone selected message to the other phone's accelerometer. With this capability, the naive approach would be to have users hold the phones P_A and P_B together. Once together, P_A would vibrate I_A to P_B and P_B would vibrate I_B to P_A. Unfortunately, the vibration to accelerometer channel is too slow for this to be user-friendly (see Section 7 for more details on the bitrate).

Several works explain how two parties can exchange data over an insecure medium (e.g., our server) and use an authentic channel (e.g., human comparison or vibration) to verify the exchange, using a checksum derived from the exchanged data [8, 11, 13, 21, 26]. However, these protocols require the phones to exchange data before they can calculate the checksum. If using a server to communicate, the phones need a pair identifier to communicate via the server, before the phones use the vibration channel to exchange the checksum. A protocol could use two long vibrations (one at the beginning to exchange a pair identifier and another to exchange the checksum), but vibrating all of that information would be slow. Users could copy a pair identifier from one phone to the other, but that is cumbersome.

Instead Shot starts by using the vibration channel to send a message that performs two functions: 1) acts as a pair identifier and 2) acts as a pre-authenticator to verify the authenticity of data exchanged from one phone to the other. Much like Talking to Strangers [1] and Seeing-Is-Believing (SiB) [18], Shot begins with the exchange of a pre-authenticator over the authentic channel, which allows verification of subsequent exchanges over the insecure channel. Unlike these protocols, Shot provides demonstrative identification and secure exchange with a single pre-authenticator. After P_A vibrates the pair identifier, the phones exchange data using the server and confirm the exchange in the following fashion:

1. P_B uses the pre-authenticator/pair id to verify a public key from the server belongs to the phone pressed against P_B.
2. P_A digitally signs I_A and a copy of the other phone's information, which P_A received from the server.
3. P_B verifies the signature from the server is a valid signature over the other phones information and its own information (I_B). If the signature is correct, P_B vibrates back a positive response to indicate that the other phone signed a copy of the information P_B received and sent.
4. P_A uses the vibrated response to determine if it signed its own information and the vibrating phone's information or if an attack occurred.

Shot does use two vibrated messages. However, one vibration in each direction is needed to allow each phone to verify the information was exchanged with the phone physically pressed against itself. If only one vibrated message was used, either more human interaction is needed, or there is no way for the first phone to verify the other phone received its information. The second vibration is a yes/no response that requires 1 bit of data and can be quickly transmitted in order to maintain a short execution time.

5.2 Shot Exchange

Shot provides a user friendly way to exchange information between smartphones by leveraging the authentic nature of the vibration to accelerometer channel, communication via a server, and the phones' ample computational capabilities. To minimize computation and communication over the vibration channel each phone has a different role. For the remainder of this work, rather than P_A and P_B, we call one phone the Endorser (P_E) and the other phone the Verifier (P_V). After users select what data to exchange and agree on or are assigned each phone's role, Shot consists of 4 phases to securely exchange data between two smartphones: 1) exchange of the pair identifier, 2) exchange via the server, 3) signing of the data, and 4) confirmation of the data. Figure 4 contains the steps associated with the Shot exchange.

① **Exchange of the Pair Identifier** During the initial phase, the phones exchange a pair identifier which bootstraps communication through the server. In Shot, this identifier also is a pre-authenticator that allows P_V to verify P_E's public key ($K^+_{P_E}$) later in the protocol. When the protocol begins, P_E calculates a shortened hash of its

① **Exchange of the Pair Identifier:**
1. P_E : $h = Hash(K^+_{P_E})$
2. $P_E \overset{acc.}{\rightsquigarrow} P_V$: h

② **Exchange Via the Server**
3. $P_E \rightarrow S$: $h, K^+_{P_E}, I_E$
4. $P_V \rightarrow S$: h, I_V
5. $S \rightarrow P_V$: $h\prime, K^+_{P_E}\prime, I_E\prime, I_V\prime$
6. $S \rightarrow P_E$: $h\prime, K^+_{P_E}\prime, I_E\prime, I_V\prime$

③ **Signing of the Data:**
7. P_E : $\sigma = Sign(K^-_{P_E}, I_E||I_V\prime)$
8. $P_E \rightarrow S$: h, σ
9. $S \rightarrow P_V$: $h\prime, \sigma\prime$

④ **Confirmation of the Data:**
10. P_V : if($h == Hash(K^+_{P_E}\prime)$ **and**
11. $Verify(\sigma\prime, K^+_{P_E}\prime, I_E\prime||I_V))$
$result$ = "yes"
save($I_E\prime$)
else $result$ = "no"
12. $P_V \overset{acc.}{\rightsquigarrow} P_E$: $result$
13. P_E : if($result$ == "yes")
save($I_V\prime$)
14. P_s, P_V : Sound Tone

Figure 4: Shot exchange between P_E and P_V utilizing the server S. ($X\prime$ is used to indicate a potentially modified value of X that has been transfered over the attacker control wireless medium.)

public key (Step 1). We truncate a hash output to 80 bits to balance security and time needed to transmit the value over the vibration channel. After the users physically place their phones together, P_E vibrates this truncated hash (Step 2). This vibration demonstrates to P_V that the other phone in the exchange has a public key that hashes to this value. Given that this hash must be unique to P_E for security, we can also use this hash as the pair identifier to help establish communication through the server.

② **Exchange via the Server** Once the phones know how to identify the pair to the server, the two phones use the server to exchange the Endorser's public key and any other information to be exchanged (P_E's info I_E and P_V's info I_V).

During each data transmission (Steps 3 & 4) and data retrieval (Steps 5 & 6), the phone begins the connection by sending the pair identifier (h). The server uses h to know how to record and retrieve the information associated with a pair of phones. If at any point a phone receives the wrong information from the server (e.g., P_V finds that $I_V \neq I_V\prime$ or $h\prime \neq h$), the phone assumes an attack has occurred and aborts the protocol.

③ **Signing of the Data** Once I_E and I_V have been exchanged, the phones start the process of verifying the data. P_E uses its private key ($K^-_{P_E}$) to sign the data it believes was exchanged so P_V can verify if the server or an attacker modified the information. Specifically, P_E signs the concatenation of its own information and the potential copy of the Verifier's information (Step 7). P_E then uses the server to send the signature to P_V (Steps 8 & 9). Once this phase is complete, neither phone has verified any information. P_V has a signature that potentially represents what information the other phone sent and received.

④ **Confirmation of the Data** The final phase has three checks to detect if the exchange was successful:

- P_V verifies the public key it received from the server belongs to the phone P_V is pressed against.
- P_V verifies that the owner of the authenticated public key received I_V and sent the information received in Step 5.
- P_E verifies that the phone it is pressed against received a valid signature. A valid signature confirms that the data P_E signed in Step 6 matches what was sent and received by both phones during the exchange.

P_V begins by verifying the hash of the received public key matches the hash it received over the vibration channel (Step 10). This verifies that the public key P_V received belongs to the phone physically pressed against P_V. Provided P_V has the other phone's public key, P_V verifies the signature from the server (Step 11). If the public key and signature are correct, P_V knows the data it received came from the phone it is pressed against and that that phone received I_V. We could have the users look at the result on P_V and press "success" or "failure" on P_E [22], but this is work for the user. A more user-friendly approach is for P_V to use the vibration channel to send a short confirmation to P_E (Step 12). This vibration lets P_E know that the phone it is pressed against received a valid signature (verified using P_E's public key) over the exchanged data. As a final step, the phones sound a tone to indicate that the protocol has completed and users can stop holding the phones together.

In the next section we describe why Shot is secure provided some properties of the underlying cryptographic primitives and authenticity of the vibration channel.

6 Security Analysis of Shot

For Shot to securely exchange information the following four properties are necessary:

- Only P_E can send P_V a pre-authenticator—a copy of the hash of P_E's public key ($K^+_{P_E}$).
- P_V can detect if it received a copy of $K^+_{P_E}$ or the wrong public key.

- P_E is the only entity that can generate a signature which $K^+_{P_E}$ verifies.
- Only P_V can indicate to P_E if the signature it received verifies the information P_V sent and received.

We discuss the properties needed from different cryptographic primitives to fulfill the second and third properties before discussing the authenticity of the vibration channel which is needed to fulfill the first and last properties.

6.1 Cryptographic Primitives

Assuming the inability of an attacker to inject information onto the vibration channel (see next subsection), a hash function that is second preimage resistant allows P_V to detect if it received a copy of P_E's public key. Given an authentic copy of P_E's public key, a signing algorithm that is secure against selective forgery allows P_V to verify P_E received P_V's information and sent the information P_V received.

If an attacker wants P_V to accept a different public key, the attacker has to find a different public key ($K^+_{P_M}$) such that the truncated hashes are the same (i.e., $Hash(K^+_{P_E}) == Hash(K^+_{P_M})$).[7] However, if the hash function is second-preimage resistant, it is infeasible for an attacker to find such a public key, even if the hash is truncated to 80-bits [22].

Without a way to convince P_V to accept a different public key, an attacker needs to produce a signature over incorrect exchange information which $K^+_{P_E}$ verifies. If the authentic public key were to verify an attacker generated signature for the message $X||I_V$, P_V would believe the other phone signed that message, indicating P_E sent X instead of I_E. However, if the signature scheme used is secure against selective forgery, it is infeasible for an attacker to produce such a signature. Without a signature which verifies these values, P_V will reject the signature and thwart the attack.

Provided the hash function is second pre-image resistant and the signature scheme is secure against selective forgery, the phones will detect any type of active attack against data exchanged over the wireless channel during the Shot exchange. Once the attack is detected, the phones will discard the information. This fail safe operation does mean an attacker can launch a Denial-of-Service attack against Shot. However, DoS attackers are outside of the scope of this work since the attacker could also jam the wireless channel to prevent any wireless communication.

[7] We assume the attacker does not know P_E's keys so $K^+_{P_E} \neq K^+_{P_M}$.

6.2 Vibration as an Authentic Channel

For Shot to be secure, only P_E and P_V are able to send information on the vibration to accelerometer channel. If an attacker were to send information on the channel, a user can detect the vibrations and abort the exchange. Without a way for attackers to send messages on the channel, only P_E is able to send a pre-authenticator of its public key and only P_V is able to send confirmation of a valid signature.

Users can detect when other parties are trying to send information on the vibration to accelerometer channel. The two users have in mind what phones are supposed to exchange information, and will detect if some other device is vibrating against the phones during the exchange. Holding two phones together is an intuitive way to indicate which phones should exchange information and should experience a low rate of operator error. Instead, we have to worry about situations where the attacker tries to remotely induce vibrations. However, without a way to focus those vibrations precisely on the phones (some kind of audio version of a laser), the user(s) holding the phones will notice the additional vibrations and stop the exchange. For example, consider an attacker which produces a loud tone at a low frequency in an attempt to vibrate the two phones remotely. The tone needs to be quite loud to induce vibrations which are comparable to the vibrations from another phone in direct physical contact.[8] Even if the frequency of the tone is below the human audible range, a tone with this much energy is perceptible (e.g., the users' clothes and skin will vibrate). As such, users can detect the additional vibrations and abort the exchange.

Without a way to break the security properties provided by the underlying cryptographic primitives, an attacker must find a way to inject a message on the vibration to accelerometer channel to successfully subvert a Shot exchange. However, the users can detect attempts to remotely induce vibrations and thwart such attacks.

7 Shot Implementation

In this section, we describe our implementation of Shot for the Motorola DROID smartphone and how we used a java program on a desktop as the server to simplify non-vibrator to accelerometer communication.

7.1 Shot for Android

Our implementation of Shot was written for and tested on Motorola DROID smartphones with Android version 2.2.

[8] Audio engineers suggest using accelerometers as "contact microphones" to reduce pickup from sources that are not in direct physical contact [19].

However, the system can be ported to any mobile phone with a vibration function and accelerometer. If installed on other phones effective bit rates for the vibrator to accelerometer may change with different hardware based on access to vibration functions[9] or accelerometers. In this section, we describe how we achieved communication between phones and between phones and the server, what library and parameters we used to perform cryptographic operations, and how users run the protocol.

Communication between phones using the vibrator and the accelerometer uses the `android.os.Vibrator` class to vibrate a phone, the `android.hardware.SensorManager` to access the phone's accelerometer, and a `android.hardware.SensorEventListener` to know when the accelerometer has new data. With the DROID, we used a simple on/off keying and tested varying bit lengths from 60ms per bit to 100ms per bit to test the reliability of the channel (see Section 8.2 for comparison of reliability versus bit rates). For example with 100ms/bit, the phone vibrates for 100 ms to transmit a 1. Given the possibility of errors, we use Reed Solomon to allow the Verifier to recover from errors during reception of the pair identifier. We use Reed-Solomon encoding with 8 bit symbols and include 4 error correction symbols. This allows the Verifier to recover from errors in 2 independent bytes. When sending the confirmation, the Verifier transmits 2 bytes of 1s to indicate "yes" or sends 0xC003 (2 ones, 12 zeros, 2 ones) to indicate "no". Given the redundancy in the message, the Verifier simply transmits the two bytes without any error-correction. The Endorser only considers the confirmation a "yes" if the confirmation contains a series of several consecutive 1s.

Communication with the server uses TCP sockets. At the beginning of the protocol the phone connects to the server and maintains a single TCP connection over the course of the entire protocol.

All of the cryptographic operations use the Bouncy Castle[10] Java package. We use SHA-1 to create the preauthenticator and to verify the public key. 1024-bit RSA signatures are used to sign and verify the exchanged information. One could generate a new RSA key pair for each exchange. Instead, we generate a key pair during the first execution and save the key pair for future executions to keep subsequent execution times shorter and more consistent.

To run the protocol, all the users have to do is hold the phones together back-to-back with one phone facing up and press a button to start the exchange. This orientation provides good transmission between the phones' vibrators and accelerometers and allows the phones to automatically assign Endorser or Verifier roles. Currently, the phone with the screen facing down becomes the Verifier. Once the exchange is started, the users simply hold the phones together until the phones beep to indicate completion. Once the confirmation is complete, the phones play a lighter tone to indicate success and a harsher sound to indicate a failed exchange. These tones are selected such that they do not induce a false "yes" or "no" on the vibration channel during the final confirmation.

[9] iPhone only allows reduced access to the vibrate function if distributing an application through the App Store.

[10] `http://www.bouncycastle.org/`

7.2 Java-Based Server

Instead of using local communication (Bluetooth or ad hoc WiFi), Shot uses a server on the Internet to transfer the majority of data between phones. Our server is a Java program running on a desktop machine that listens for incoming connections and uses a database to store and lookup information based on the received hash value.

Currently, the server does not verify if any of the information is correct (e.g., hash a potential Endorser_Key to verify it matches Pair_ID). Instead the server acts as a means to forward information between the two phones. In the end, the phones perform the final verification.

8 Evaluation

In this section, we evaluate the reliability of the vibration-to-accelerometer channel used in Shot, and compare the performance of Bump on both iPhone and DROID to Shot on DROID. We also present some microbenchmarks to help explain differences in the execution times.

8.1 Microbenchmarks

Table 1 shows the average time required to perform network and cryptographic operations on either an iPhone 3G with iOS 4.2.1 or a DROID with Android 2.2. Each value represents the average from 20 executions of the operation. We measured the time needed to connect to the Bump server over WiFi and send a kilobyte of information using TCP without TLS, and the time needed to sign a random value and verify the signature with 1024-bit RSA keys. The phones exhibit similar network capabilities and require roughly 90 ms to establish a connection and send the data. However, the DROID appears to have greater processing power. Signing is almost five times faster on the DROID than on the iPhone (41.9 ms versus 201.5 ms). Verifying is also faster on the DROID (7 ms versus 8.4 ms).

Platform	Connect & Send 1kB	1024 RSA Sign	1024 RSA Verify
iPhone	90.0 ms	201.5 ms	8.4 ms
DROID	91.2 ms	41.9 ms	7.0 ms

Table 1: Network and cryptographic performance for the iPhone and DROID.

Millisecond per bit	60	80	100
Rate of successful transmissions	0.53	0.67	0.94

Table 2: Impact of data rate on transmission reliability.

8.2 Reliability of the Vibration Channel

To measure the reliability of the vibration channel, we attempted to exchange the 112-bit message consisting of a pre-authenticator and error-correcting code payload while varying the time needed to communicate one bit from the vibrating phone to the other phone's accelerometer from 60 to 100 ms and measured how often the receiving phone could successfully decode the message. The results of this test are presented in Table 2. With 60 ms/bit encoding, the receiver was able to successfully decode the message 53% of the time. However, decoding was successful all but one time with 100 ms/bit. The limiting factor for using vibration and accelerometers for communication appears to be the scheduling on the DROID. There is a function call to turn on the vibrator for a set period of time, but multitasking prevents the system from promptly turning on and off the vibrator for values smaller than 75ms.

8.3 Complete Protocol Execution Times

We measured the execution time of each protocol over the course of 10 successful exchanges. During our evaluation, all of the wireless communication was sent over the WiFi network, as opposed to the cellular network. For Bump, we measured execution time from when the application was started until the phone received the other phone's information. For Shot, we measured the execution time form when the application was started and the tone at the end of Shot. Figure 5 summarizes the results of our evaluation.

The performance of Bump is highly platform dependent, requiring an average of 21 seconds on the DROID and almost half of that (10.4 seconds) on the iPhone. The DROID appears to have faster processing and similar network performance (see Section 8.1), but appears to take longer to determine its location. Our experiments were performed indoors and the iPhone appears to quickly switch from trying to use GPS to using WiFi to estimate its location before connection to the Bump server (total execution time between 9.4 to 13.5 seconds). However, the DROID version of Bump appeared to spend a variable amount of time trying to use GPS before switching to using WiFi (total execution between 14.2 to 37.8 seconds).

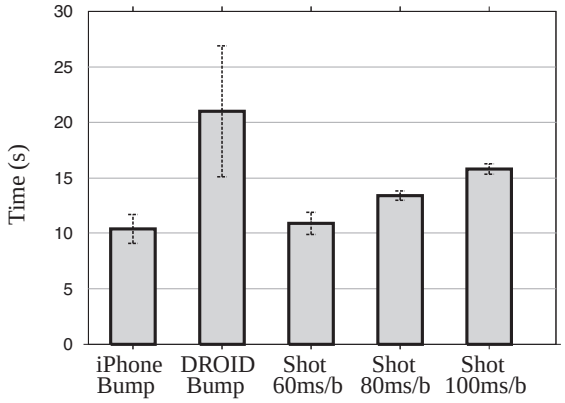

Figure 5: Average execution time for Bump and Shot (error bar = one standard deviation)

With fixed size pre-authenticators and confirmation vibration, Shot provides different execution times depending on the encoding used. When compared to Bump on the iPhone, Shot provides similar execution times when using the the less reliable 60ms/b encoding (9.9 to 12.8 seconds). However, the more reliable version of Shot with 100ms/b encoding (15.1 to 16.9 seconds) is slower than the version of Bump for the iPhone. Compared to the DROID version of Bump, Shot provides faster execution times. The performance of Bump on DROID may improve with the next release for DROID (DROID uses v1.3.2 while iPhone uses v2.4). Note that our implementation of Shot was not optimized for efficiency, and the purpose of this evaluation was simply to confirm that Shot could be executed roughly as quickly as Bump.

9 Previous Work

Several works have examined the problem of how to exchange information or establish a shared secret between two devices to secure the exchange of information. We discuss different categories of previous work according to the mechanism used to provide security and discuss the user-involvement and security of each.

Human Assisted Comparison A number of prior works require the user to perform a comparison after the exchange to detect an attack [8, 11, 13, 21, 26]. Given an attacker is unable to change the output on the screens or the value the user enters into a device, this technique can successfully detect MitM attacks. Many works focus on comparing a string of hexadecimal digits [11, 13, 26].

However, Uzun et al. [25] found usability issues with string comparisons (i.e., users failed to detect attacks or indicated the strings were different when they were the same). Other works have proposed encoding the comparison value into a sentence [8] or image [21] to improve usability. However, these schemes still require users to perform a comparison and are vulnerable to attacks when users click "Accept" without comparing the strings, sentences, or images. A protocol could leverage the vibration to accelerometer channel to have devices perform the comparison after exchanging data and reduce user involvement. However, if the devices use a server to communicate, user involvement is needed to indicate to the server how to route traffic (i.e., establish a pair identifier), before the verification is performed.

Exchange of Secrets A number of works assume the attacker is unable to observe the users talking in the room (e.g., sharing a password [11]), motion of the phones [4, 10, 14, 17], communication over the vibration channel [5,23], or communication over an electrical connection between the devices [24]. Our threat model includes a more powerful attacker who may be able to violate the secrecy of all but Stajano and Anderson's protocol [24]. Nearby parties can eavesdrop on spoken communication to recover a password. Our attack on Bump demonstrated how attackers can observe and imitate simple movements. Mayrhofer and Gellersen showed that it is possible to tune exchange algorithms to be resistant to this kind of attack, but at significant cost to the usability of the system: their system experienced a 10% failure rate of legitimate exchanges, and 16% of participants were unable to ever successfully complete the protocols [17]. Beyond sacrificing usability to provide security against a simple attacker, these protocols may be vulnerable to more sophisticated attacks, such as those that use real-time motion tracking [16] to reconstruct movements than an unaided human could not. When the phones' vibration function is used, user involvement is limited since users only have to hold the phones together, but it is still possible for an attacker to eavesdrop on the vibration channel to acquire the secret [9]. Stajano and Anderson use a wire or other electrical connection between the two devices [24]. In that scenario, the only way an attack will succeed is if the attacker can control communication on the wire without the users noticing. The drawback to this approach is that users have to carry around cables in order to perform the exchange.

Observable Channel Between Devices Much like Shot, other works have examined the use of a channel between two devices that allow users to infer which parties are communicating. Prior works considered using IR [1] or light in the visual spectrum [18, 22] as authentic human-observable channels. If the hardware exists, these techniques can be used on smartphones. However, IR is not available on Blackberry, iPhone, or Android phones, which account for over 75% of all smartphones.[11] Seeing-is-Believing (SiB) [18] and Saxena's follow-up work [22] use the phone's camera to photograph barcodes or film a blinking light. However, SiB is a directional exchange, so users have to execute the protocol twice.Saxena's protocol allows a complete exchange with a single execution of the protocol. However, in addition to filming on P_A, the user has to confirm the exchange by pressing a button on P_B. Shot allows phones to exchange authentic information in both directions, without involving the users to reposition the phones or press a button, if both phones have vibrators and accelerometers.

10 Conclusion

The exchange of information between smartphones allows users to play games, share contact information, and even transfer money. If a malicious party is able to perform a Man-in-the-Middle (MitM) attack during an exchange, she can interfere with games to annoy users, intercept communication, and even steal money.

In this paper, we presented a MitM attack against Bump, the most popular smartphone exchange protocol. We analyzed our attack under real-world settings and found that in some realistic scenarios a malicious party can launch a MitM attack against Bump and succeed 70% of the time.

We also presented Shot, a new secure and simple-to-use smartphone exchange. Shot uses the phones' accelerometers and vibrators to establish a human-observable channel between two phones that are held together. Since users can feel the vibrations, this channel provides demonstrative identification of the devices participating in the exchange, and allows users to detect if an attack is occurring (e.g., a remote party is trying to vibrate the phones). Shot leverages asymmetric cryptography to bootstrap authentic communication over the higher-bandwidth wireless channel.

We implemented Shot and compared its execution time and user experience to Bump. Our evaluation found that, Shot is comparably quick to Bump, while providing greater security and placing similar demands on the user. As such, Shot represents a new point on the spectrum of exchange protocols, providing improved security or reduced user involvement when compared to existing solutions.

[11] `http://blog.nielsen.com/nielsenwire/online_mobile/mobile-snapshot-smartphones-now-28-of-u-s-cellphone-market/`

References

[1] BALFANZ, D., SMETTERS, D., STEWART, P., AND WONG, H. C. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of the Network and Distributed System Security Conference (NDSS)* (2002).

[2] BUMP TECHNOLOGIES. Bump. `http://bu.mp`.

[3] CARBONE, M., AND RIZZO, L. Dummynet revisited. *SIGCOMM Computer Communications Review 40* (April 2010).

[4] CASTELLUCCIA, C., AND MUTAF, P. Shake them up! A movement-based pairing protocol for cpu-constrained devices. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)* (2005).

[5] D. HALPERIN ET AL. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Proceedings of the IEEE Symposium on Security and Privacy* (2008).

[6] DHAMIJA, R., TYGAR, D., AND HEARST, M. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)* (2006).

[7] DIERKS, T., AND ALLEN, C. The TLS protocol version 1.0. Internet Request for Comment RFC 2246, Internet Engineering Task Force, Jan. 1999. Proposed Standard.

[8] GOODRICH, M. T., SIRIVIANOS, M., SOLIS, J., TSUDIK, G., AND UZUN, E. Loud and clear: Human-verifiable authentication based on audio. In *International Conference on Distributed Computing (ICDCS)* (2006), p. 10.

[9] HALEVI, T., AND SAXENA, N. On pairing constrained wireless devices based on secrecy of auxiliary channels: The case of acoustic eavesdropping. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (2010).

[10] HOLMQUIST, L. E., MATTERN, F., SCHIELE, B., ALAHUHTA, P., BEIGL, M., AND GELLERSEN, H.-W. Smart-its friends: A technique for users to easily establish connections between smart artefacts. In *Proceedings of Ubicomp* (2001).

[11] J. LINKSKY ET AL. Simple Pairing Whitepaper, revision v10r00. `http://www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf`, August 2006.

[12] KUMAR, A., SAXENA, N., TSUDIK, G., AND UZUN, E. Caveat emptor: A comparative study of secure device pairing methods. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom)* (2009).

[13] LAUR, S., AND NYBERG, K. Efficient mutual data authentication using manually authenticated strings. In *Cryptology and Network Security (CANS)* (2006), pp. 90–107.

[14] LESTER, J., HANNAFORD, B., AND GAETANO, B. Are you with me? - using accelerometers to determine if two devices are carried by the same person. In *Proceedings of Pervasive* (2004).

[15] LIN, Y.-H., STUDER, A., HSIAO, H.-C., MCCUNE, J., WANG, K.-H., KROHN, M., LIN, P.-L., PERRIG, A., SUN, H.-M., AND YANG, B.-Y. SPATE: Small-group PKI-less Authenticated Trust Establishment. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)* (2009).

[16] LIPTON, A., FUJIYOSHI, H., AND PATIL, R. Moving target classification and tracking from real-time video. In *Proceedings of the IEEE Workshop on Applications of Computer Vision* (1998).

[17] MAYRHOFER, R., AND GELLERSEN, H. Shake well before use: Intuitive and secure pairing of mobile devices. *IEEE Transactions on Mobile Computing 8*, 6, 792–806. Submitted 2008-06-15, accepted 2009-02-10.

[18] MCCUNE, J. M., PERRIG, A., AND REITER, M. K. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *Proceedings of the IEEE Symposium on Security and Privacy* (2005).

[19] OREILLY, R., KHENKIN, A., AND HARNEY, K. Sonic nirvana: Using mems accelerometers as acoustic pickups in musical instruments. *Analog Dialogue 43* (Feb. 2009).

[20] PAYPAL. Paypal mobile payments. `https://personal.paypal.com/us/cgi-bin/?&cmd=_render-content&content_ID=marketing_us/mobile_payments`.

[21] PERRIG, A., AND SONG, D. Hash visualization: A new technique to improve real-world security. In *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC)* (1999).

[22] SAXENA, N., EKBERG, J.-E., KOSTIAINEN, K., AND ASOKAN, N. Secure device pairing based on a visual channel. In *Proceedings of the IEEE Symposium on Security and Privacy* (2006).

[23] SAXENA, N., AND WATT, J. Authentication technologies for the blind or visually impaired. In *Proceedings of the USENIX Workshop on Hot Topics in Security (HotSec)* (2009).

[24] STAJANO, F., AND ANDERSON, R. J. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Protocols Workshop* (1999).

[25] UZUN, E., KARVONEN, K., AND ASOKAN, N. Usability analysis of secure pairing methods. In *Proceedings of Usable Security (USEC)* (2007).

[26] VAUDENAY, S. Secure communications over insecure channels based on short authenticated strings. In *Advances in Cryptology (Crypto)* (2005).