Efficient Device Pairing using "Human-Comparable" Synchronized Audiovisual Patterns

Ramnath Prasad[1]* and Nitesh Saxena[2]

[1] Microsoft
raprasad@windows.microsoft.com
[2] Polytechnic University
nsaxena@poly.edu

Abstract. "Pairing" is referred to as the operation of achieving authenticated key agreement between two human-operated devices over a short- or medium-range wireless communication channel (such as Bluetooth, WiFi). The devices are ad hoc in nature, i.e., they can neither be assumed to have a prior context (such as pre-shared secrets) with each other nor do they share a common trusted on- or off-line authority. However, the devices can generally be connected using auxiliary physical channel(s) (such as audio, visual) that can be authenticated by the device user(s), and thus form the basis for pairing.
One of the main challenges of device pairing is the lack of good quality output interfaces (e.g., a speaker, display) as well as receivers (e.g., a microphone, camera) on both devices. In this paper, we present a new pairing scheme that is universally applicable to any pair of devices, supporting all possible pairing scenarios. Our scheme does not require devices to have good transmitters or any receivers, and is based upon the device user(s) comparing short and simple synchronized audiovisual patterns, such as in the form of "beeping" and "blinking".

1 Introduction

Short- or Medium-range wireless communication, based on technologies such as Bluetooth and WiFi, is becoming increasingly popular and promises to remain so in the future. This surge in popularity brings about various security risks. Wireless communication channel is easy to eavesdrop upon and to manipulate, and therefore a fundamental security objective is to secure this communication channel. In this paper, we will use the term "pairing" to refer to the operation of bootstrapping secure communication between two devices connected with a short-range wireless channel. The examples of pairing, from day-to-day life, include pairing of a WiFi laptop and an access point, a Bluetooth keyboard and a desktop. Pairing would be easy to achieve if there existed a global infrastructure enabling devices to share an on- or off-line trusted third party, a certification

* Work done while being a Master's student at Polytechnic University

authority, a PKI or any pre-configured secrets. However, such a global infrastructure is close to impossible to come by in practice, thereby making pairing an interesting and a challenging research problem.[3]

A recent research direction to pairing is to use an auxiliary physically authenticatable channel, called an out-of-band (OOB) channel, which is governed by humans, i.e., by the users operating the devices. Examples of OOB channels include audio, visual and tactile. Unlike the wireless channel, on the OOB channel, an adversary is assumed to be incapable of modifying messages, however, it can eavesdrop on, and possibly also delay, drop and replay them. A pairing scheme should therefore be secure against such an adversary.

The usability of a pairing scheme based on OOB channels is clearly of utmost importance. Since the OOB channels typically have low bandwidth, the shorter the data that a pairing scheme needs to transmit over these channels, the better the scheme becomes in terms of usability.

Various pairing protocols have been proposed so far. These protocols are generally based on the bidirectional automated device-to-device (d2d) OOB channels. Such d2d channels require both devices to have transmitters and the corresponding receivers. In settings, where d2d channel(s) do not exist (i.e., when at least one device does not have a receiver) and even otherwise, same protocols can be based upon device-to-human (d2h) and human-to-device (h2d) channel(s) instead. Depending upon the protocol, only two d2h channels might be sufficient, such as in case when the user has to perform a very simple operation (such as "comparison") of the data received over these channels. Clearly, the usability of d2h and h2d channel establishment is even more critical than that of a d2d channel.

The earlier pairing protocols required at least 160 to 80 bits of data to be transmitted over the OOB channels. The simplest protocol [1] involves devices exchanging their public keys over the wireless channel, and authenticating them by exchanging (at least 80-bits long) hashes of corresponding public keys over the OOB channels. The more recent, so-called SAS- (Short Authenticated Strings) based protocols, [5], [7], reduce the length of data to be transmitted over the OOB channels to only 15 bits or so.[4]

Based on the above protocols, a number of pairing schemes with various OOB channels have been proposed. These include schemes based on two bidirectional d2d infra-red channels [1]; two bidirectional d2d visual channels consisting of barcodes and photo cameras [6]; a unidirectional d2d visual channel consisting of blinking LED and video camera plus a unidirectional d2h channel consisting of a blinking LED and a unidirectional h2d channel [10]; two audio/visual d2h channels consisting of MadLib sentences and displayed text [4]. In addition, the SAS protocols trivially yield pairing schemes involving two bidirectional d2h and h2d channels – the user reads 15 bits of data displayed on one device and inputs it on the other, and vice versa. Most recently, [15] performed user studies of pairing

[3] The problem has been at the forefront of various recent standardization activities, see [14].

[4] For the SAS-based authentication and related prior work, refer to [16].

schemes based on user comparing the data transmitted over two independent d2h SAS channels.

The aforementioned schemes have varying degree of usability and are applicable to different device combinations. However, all the above schemes become inapplicable in pairing scenarios where,

1. both devices do not have good quality transmitters (such as displays, speakers, etc.), and
2. both devices do not have relevant receivers (such as cameras, microphones, etc.).

Notice that the pairing scenarios involving most commodity devices, such as access points, headsets, would fall into the above categories.

A very recent proposal, [11], focuses on pairing two devices with the help of "button presses" by the user. The scheme can be used to pair devices with minimal hardware interfaces (such as an LED and a button) using a SAS protocol. However, as we discuss in the next section and as indicated by the results of [11], the scheme is a bit slow.

In short, the previous pairing schemes are either not applicable or are slow in routinely performed pairing scenarios, such as pairing of a WiFi laptop/PDA/cell phone and an access point, a Bluetooth keyboard and a desktop/laptop/PDA.

Our Contributions. In this paper, we propose a new efficient scheme that is universally applicable to pair any two devices.[5] Such a universality of a pairing scheme with respect to devices, in our opinion, would improve both security as well as usability over time. Our scheme can use the existing SAS protocols and does not require devices to have good transmitters or any receivers, e.g., only a pair of LEDs are sufficient. The scheme involves users comparing very simple audiovisual patterns, such as "beeping" and "blinking", transmitted as simultaneous streams, forming two synchronized d2h channels.[6] We tested our scheme with the three combinations we call Beep-Beep, Blink-Blink and Beep-Blink. Our test results indicate that the Blink-Blink and Beep-Blink combinations perform very efficiently and robustly, with the former being preferable. However, we discard the Beep-Beep combination because it turns out to be quite inefficient and error-prone from our initial testing.

The Blink-Blink and Beep-Blink combinations are quite efficient in pairing scenarios where both devices do not have good quality transmitters and only at most one device has a relevant receiver. The Blink-Blink combination can typically only be used for pairing two similar devices (such as a Bluetooth headset and a cell phone, two laptops, two cell phones) that are physically very close by and can be aligned properly with each other. The Beep-Blink combination, on

[5] Our proposal is also equally applicable to establish unidirectional authentication, such as between a printer and a laptop.

[6] We notice that in an independent result [9], the authors present a scheme similar to our "blinking" scheme, aimed at the detection of "evil twin" access points. The two schemes, however, differ significantly in their implementation and thus in terms of the underlying user experience. We discuss these differences in the next section.

the other hand, can be used for any two devices (generally, one of the devices being paired has an audio transmitter and the other has LEDs). Moreover, the Beep-Blink combination is applicable irrespective of the extreme proximity of devices. In other words, the Beep-Blink combination is also suitable for pairing, e.g., a wall-mounted access point with other devices.

We anticipate that, even in scenarios where devices have good transmitters and also have receivers, the Blink-Blink and Beep-Blink combinations would perform better than most prior solutions [6], [15]. Intuitively, this is due to the reason that the schemes in [15] require the users to perform two operations, reading and comparing, while our schemes only require comparing; whereas the scheme in [6] requires the users to handle specialized equipments, such as cameras, which they might not have used previously. Of course, these are mere expectations. Only a detailed comparative study of all these schemes (which is an item for our future work) can give a clear insight into their applicability among an average user population.

In addition to the single user pairing scenarios, our proposal is also equally and efficiently applicable to scenarios where two users pair their individual devices, such as their cell phones, laptops.

Organization. The rest of the paper is organized as follows. In Section 2, we review the prior pairing schemes. In Section 3, we describe the security model and summarize relevant protocols. In Section 4, we present our scheme, followed by the description of its design, implementation and performance in Section 5.

2 Related Work

There exists a significant amount of prior work on the general topic of pairing.

In their seminal work, Stajano, et al. [13] proposed to establish a shared secret between two devices using a link created through a physical contact (such as an electric cable). In many settings, however, establishing such a physical contact might not be possible, for example, the devices might not have common interfaces to do so or it might be too cumbersome to carry the cables along. Balfanz, et al. [1] extended this approach through the use of infrared as a d2d channel – the devices exchange their public keys over the wireless channel followed by exchanging (at least 80-bits long) hashes of their respective public keys over infrared. The main drawback of this scheme is that it is only applicable to devices equipped with infrared transceivers.

Another approach taken by a few research papers is to perform the key exchange over the wireless channel and authenticate it by requiring the users to manually and visually compare the established secret on both devices. Since manually comparing the established secret or its hash is cumbersome for the users, schemes were designed to make this visualization simpler. These include Snowflake mechanism [3] by Levienet et al., Random Arts visual hash [8] by Perrig et al. etc. These schemes, however, require high-resolution displays and are thus only applicable to a limited number of devices, such as laptops.

Based on the pairing protocol of Balfanz et al. [1], McCune et al. proposed the "Seeing-is-Believing" (SiB) scheme [6]. SiB involves establishing two unidirectional visual `d2d` channels – one device encodes the data into a two-dimensional barcode and the other device reads it using a photo camera. Since the scheme requires both devices to have cameras, it is only suitable for pairing devices such as camera phones.

Goodrich, et al. [4], proposed a pairing scheme based on "MadLib" sentences. This scheme also uses the protocol of Balfanz et al. The main idea is to establish a `d2h` channel by encoding the data into a MadLib sentence. Device A encodes the hash of its public key into a MadLib sentence and transmits this over a `d2h` channel (using a speaker or a display); device B encodes the hash of the (received) public key from device A into a MadLib sentence and transmit this over a `d2h` channel (using a speaker or a display); the user reads and compares the data transmitted over the two `d2h` channels, and vice versa. The scheme, as proposed in the paper, requires four `d2h` channels and the user needs to perform two comparisons. This is quite slow and tedious for the user. One can trivially improve the scheme by using a slightly modified protocol, the one where devices exchange the hash (of size at least 160-bits) of the concatenation of both public keys, after exchanging their public keys. The modified scheme would then require only one user comparison. Note that, however, the scheme is not applicable to pairing scenarios where one of the devices does not have a display or a speaker.

Note that the previously described schemes, with trivial modifications, can (and should) all be based upon one of the SAS protocols [5], [7]. Since the SAS protocols require only 15-bits of data to be transmitted over the OOB channel, such a migration will immensely improve the efficiency as well as the usability of these schemes.

Saxena et al. [10] proposed a new scheme based on visual OOB channel. The scheme uses one of the SAS protocols [5], and is aimed at pairing two devices A and B (such as a cell phone and an access point), only one of which (say B) has a relevant receiver (such as a camera). First, a unidirectional `d2d` channel is established by device A transmitting the SAS data, e.g., by using a blinking LED and device B receiving it using a video camera. This is followed by device B comparing the received data with its own copy of the SAS data, and transmitting the resulting bit of comparison over a `d2h` channel (say, displayed on its screen) . Finally, the user reads this bit transmitted and accordingly indicates the result to device A by transmitting a bit over an `h2d` input channel.

A very recent proposal, [11], focuses on pairing two devices with the help of "button presses" by the user. The scheme described in the paper is based upon a protocol that first performs an unauthenticated Diffie-Hellman key agreement and then authenticate the established key using a short password. Such a short password can be agreed upon between the two devices via three variants using button presses. The first variant involves the user simultaneously pressing buttons on both devices within certain intervals and each of these intervals are used to derive 3-bits of the password (and thus with 5 button presses, the user is able to inputs the same password on both devices). In the other two variants, one device picks up a short password, encodes each 3-bit block of the password into the delay between consecutive flashing of the device's screen or its vibration. As one device flashes or vibrates, the user presses the button on the other device thereby transmitting the password from one device to another. One drawback with the scheme, as described in [11], is that its security is based upon the secrecy of the agreed upon password. At least the button presses and the flashing of the screen can possibly be recorded by a video camera and therefore, the secrecy of the password is not guaranteed. The scheme, however, can easily be based upon a SAS protocol in a straight-forward manner and be used for pairing devices which do not have good transmitters or receivers. Assuming that both devices have an LED and a button each, we can have them transmit their SAS values by blinking of the LED (on one device) and pressing of button (on the other) and vice versa. Unfortunately, this would be quite slow – to transmit a 15-bit SAS value, it will take about a minute in each direction (see the results of the scheme called "D-To-B" in [11]; users can possibly not perform simultaneous "blink-press" faster than 3-4 seconds). One could apply the protocol variant of Saxena et al. [10] to avoid transmission of SAS in the other direction thereby reducing the execution time to close to a minute.

Uzun et al. [15] carry out a comparative usability study of simple pairing schemes. They consider pairing scenarios where devices are capable of displaying 4-digits of SAS data. In what they call the "Compare-and-Confirm" approach, the user simply reads and compares the SAS data displayed on both devices. The "Select-and-Confirm" approach, on the other hand, requires the user to select a 4-digit string (out of a number of strings) on one device that matches with the 4-digit string on the other device. The third approach, called "Copy-and-Confirm", requires the user to read the data from one device and input it onto the other. These schemes are undoubtedly simple, however, the results of [15] seem to indicate that Select-and-Confirm and Copy-and-Confirm are error prone.

In [12], authors consider the problem of pairing two devices which might not share any common wireless communication channel at the time of pairing, but do share only a common audio channel.

We notice that in an independent result [9], the authors present a scheme similar to the "blinking" scheme that we present in this paper. The scheme of [9] is aimed at the detection of "evil twin" access points in cafs, airport lounges, etc. The two schemes, however, differ significantly in their implementation and therefore in terms of user experience. Firstly, in the scheme of [9], the user controls the time period during which she compares each bit of the SAS data, by pressing and releasing a button on her device. Our scheme, on the other hand, is automatic in that this time period is a pre-determined experimental value. Secondly, in [9], the user's device needs to trigger the display of next bit on the other device by sending it a signal over the wireless channel. This requires k such signals for a k-bit long SAS and the user needs to verify whether or not these signals are delayed, dropped or injected. This is unlike our scheme, where only one synchronization signal is sent between the two devices.

3 Communication and Security Model, and Applicable Protocols

The pairing protocols are based upon the following communication and adversarial model [16]. The devices being paired are connected via two types of channels: (1) a short-range, high-bandwidth bidirectional wireless channel, and (2) auxiliary low-bandwidth physical OOB channel(s). Based on device types, the OOB channel(s) can be device-to-device (`d2d`), device-to-human (`d2h`) and/or human-to-device (`h2d`). An adversary attacking the pairing protocol is assumed to have full control on the wireless channel, namely, it can eavesdrop, delay, drop, replay and modify messages. On the OOB channel, the adversary can eavesdrop, delay, drop, replay and re-order messages, however, it can not modify them. In other words, the OOB channel is assumed to be an authenticated channel. The security notion for a pairing protocol in this setting is adopted from the model of authenticated key agreement due to Canneti and Krawczyk [2]. In this model, a multi-party setting is considered wherein a number of parties simultaneously run multiple/parallel instances of pairing protocols. In practice, however, it is reasonable to assume only two-parties running only a few serial/parallel instances of the pairing protocol. For example, during authentication for an ATM transaction, there are only two parties, namely the ATM machine and a user, restricted to only three authentication attempts. The security model does not consider denial-of-service (DoS) attacks. Note that on wireless channels, explicit attempts to prevent DoS attacks might not be useful because an adversary can simply launch an attack by jamming the wireless signal.

To date, two three-round pairing protocols based on short authenticated strings (SAS) have been proposed [7], [5]. For the sake of completeness, we depict the protocol of [7] in Figure 1.

In a communication setting involving two users restricted to running three instances of the protocol, these SAS protocols need to transmit only k $(= 15)$ bits of data over the OOB channels. As long as the cryptographic primitives used in the protocols are secure, an adversary attacking these protocols can not win with a probability significantly higher than 2^{-k} $(= 2^{-15})$. This gives us security equivalent to the security provided by 5-digit PIN-based ATM authentication.

Recall that the pairing scheme that we propose in this paper requires the users to "compare" the data transmitted over two `d2h` channels. Our scheme can be based on the existing SAS protocols. This is because in these protocols, the SAS messages are computed as a common function of the public keys and/or random nonces exchanged during the protocol, and therefore the authentication is based upon whether the two SAS messages match or not [7] [5]. The security of these SAS protocols is based upon different cryptographic assumptions. For example, [7] is based upon the random oracle model (ROM), while [5] is not. Moreover, these protocols have different computational requirements. Therefore, based upon the security requirements and underlying devices, our scheme can resort to either of the SAS protocols as desired.

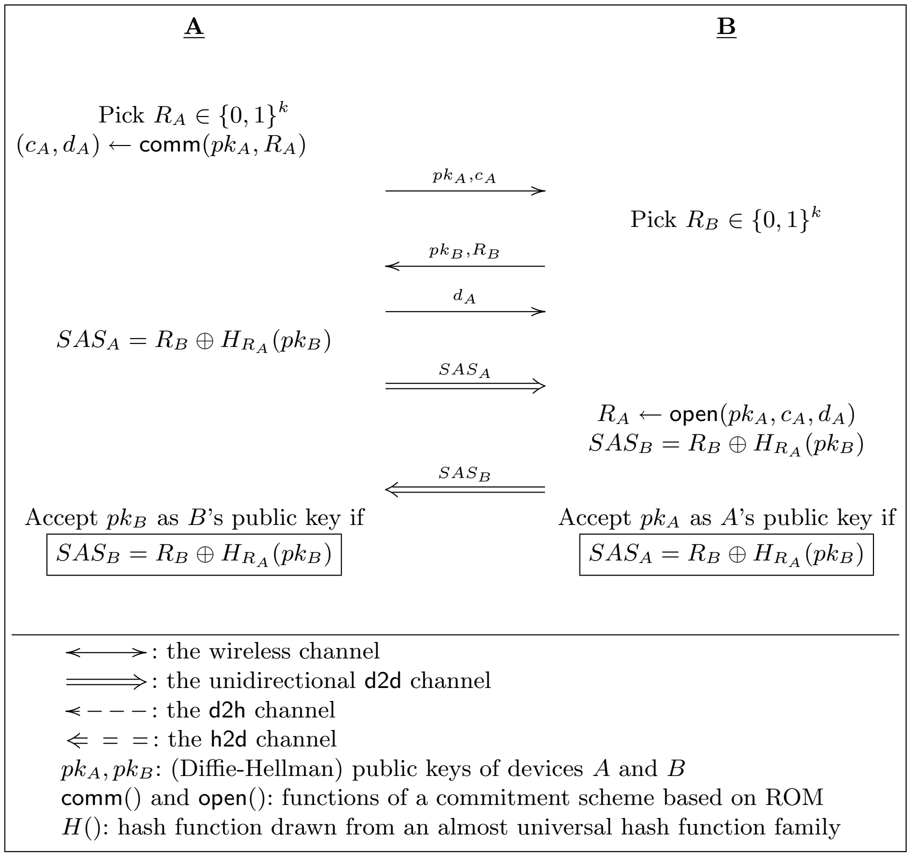
A
B
Pick $R_A \in \{0,1\}^k$
$(c_A, d_A) \leftarrow \mathsf{comm}(pk_A, R_A)$
pk_A, c_A
Pick $R_B \in \{0,1\}^k$
pk_B, R_B
d_A
$SAS_A = R_B \oplus H_{R_A}(pk_B)$
SAS_A
$R_A \leftarrow \mathsf{open}(pk_A, c_A, d_A)$
$SAS_B = R_B \oplus H_{R_A}(pk_B)$
SAS_B
Accept pk_B as B's public key if
$\boxed{SAS_B = R_B \oplus H_{R_A}(pk_B)}$
Accept pk_A as A's public key if
$\boxed{SAS_A = R_B \oplus H_{R_A}(pk_B)}$
$\longleftrightarrow$: the wireless channel
$\Longrightarrow$: the unidirectional d2d channel
$\dashleftarrow$: the d2h channel
$\Leftarrow = =$: the h2d channel
pk_A, pk_B: (Diffie-Hellman) public keys of devices A and B
comm() and open(): functions of a commitment scheme based on ROM
$H()$: hash function drawn from an almost universal hash function family

Fig. 1. The SAS protocol of [7]

4 “Human-Comparable” Audiovisual Patterns

Our primary goal is to support pairing scenarios in which both devices do not have good transmitters (e.g., displays) and only at most one device has a receiver (e.g., cameras). Recall that these scenarios include pairing of a laptop and an access point, a keyboard and a desktop, a cell phone and an access point, etc. Our idea is to make use of the existing SAS protocols and implement two synchronized d2h channels to transmit the SAS strings in the form of streams that the user can compare. The synchronization can be achieved by one device signalling the other over the wireless channel. Since this synchronization signal can possibly be tampered with by the adversary, each d2h channel also consists of an “END” marker indicating the end of the respective SAS string and the two END markers also need to be compared by the user. Such d2h channels can be implemented using the following audiovisual combinations.

1. **“Beep-Beep”.** This combination requires both devices to have audio transmitters (such as basic speakers or “beepers”). The two devices encode their respective SAS strings using a sound denoted by “S1” – a ‘1’ bit corresponds to the sound S1 and a ‘0’ bit to a “silence” for a certain period. The two devices also encode the END markers using a *distinct* sound denoted by “S2”. The user listens to both devices and determines if the two play two sounds S1 and S2 in synchronization with each other or not. In other words, if S1 on one device is not played with S1 on other device or if S2 on one device is not played with S2 on the other, the user indicates a “failure”.

2. **“Blink-Blink”.** This combination requires both devices to have visual transmitters, the simplest of which are LEDs. In cases where devices have good displays, one could use the whole or a part of the screen as a transmitter. The two devices encode their respective SAS strings into blinking of a green LED – a ‘1’ bit corresponds to a “blink” period and a ‘0’ bit to an “off” period. The two devices also encode the END markers using the glowing of a red LED. The user looks at the two devices and determines if the green LEDs on two devices blink in synchronization with each other and if the red LEDs glow together. In other words, if the green LED on one device does not blink with the green LED on other device or if the red LED on one device does not glow with the red LED on the other, the user indicates a “failure”.

3. **“Beep-Blink”.** This combination requires one device to have an audio transmitter and the other to have a visual transmitter. One device A encode its SAS string using sound S1 and the END marker using sound S2, and the other device B encodes its SAS string into blinking of a green LED and the END marker by glowing a red LED. The user listens to device A while looking at device B and determines if S1 is played in synchronization with the blinking of the green LED and if S2 is played with the glowing of the red LED. In other words, if S1 on device A is not played with the green blinking LED on device B or if S2 on device A is not played with the glowing of the red LED on device B, the user indicates a “failure”.

The security of our schemes is equivalent to the security of the underlying SAS protocol under the assumption that the user does not commit any errors. This brings us to the design, implementation and usability evaluation of our schemes.

5 Experimentation and Testing

We describe the design and implementation of our pairing schemes based on the Beep-Beep, Blink-Blink and Beep-Blink combinations, and their experimental usability study.

5.1 Design and Implementation

Our goal was to design the Beep-Beep, Blink-Blink and Beep-Blink combinations in a manner that can be used on most devices and that the users find simple and easy to perform. To this end, we chose to realize the "beeping" with simple sounds, a "system beep" and a "buzz", which can be easily distinguished even amidst some noise, and the "blinking" with LEDs.

A pairing scheme, in its entirety, consists of three phases: (1) the device discovery phase, wherein the devices exchange their identifiers over the wireless channel, prior to communicating, (2) the pairing protocol execution phase, wherein the devices execute the desired pairing protocol over the wireless channel, and (3) the authentication phase, where the devices, using the OOB channels, authenticate the messages exchanged during the previous phase. For the sake of our experimentation, we skipped the first two phases and concentrated on the third phase, because our main goal was to test the feasibility of the way we intended to implement the OOB channels, i.e., using the Beep-Beep, Blink-Blink and Beep-Blink combinations. As mentioned previously, our pairing scheme can be built on top of the SAS protocols [7], [5].

Let us assume that we want to pair two devices A and B. Given that A and B already performed the device discovery and protocol execution phases over the wireless channel, our job is now reduced to A and B encoding the 15-bits of their respective SAS data SAS_A and SAS_B into beeping or blinking, and transmitting it in a synchronized fashion for the user to compare. This encoding should enable the user to easily identify both the good cases, i.e., when $SAS_A = SAS_B$, and also the bad ones, i.e., when $SAS_A \neq SAS_B$.

To achieve synchronization, we simply have one device sending a synchronization signal S to the other device over the wireless channel. The devices encode and start transmitting their respective SAS data right after sending and receiving the bit S. Note that this synchronization signal can possibly be modified, delayed or dropped (either maliciously or otherwise), possibly fooling the users into accepting non-matching SAS strings (for example, strings $SAS_A =$ "010010" and $SAS_B =$ "100100", will appear to be equal to the user if the synchronization signal is delayed by a bit). To counter this, the end of each SAS string is indicated by an END marker, which can be easily distinguished from the beeping and blinking of the SAS strings and compared by the users enabling them to detect any synchronization errors.

Encoding for "beeping". A '1' bit in the SAS string is signalled using a "system beep", whereas a '0' bit is signalled using a "silence" for a certain period of time. The END marker is signalled using a distinctively sounding "buzz". Every bit signal is followed by a brief "sleep interval". Note that the "human-comparison" is integral to our scheme and it is important that users are able to identify two distinct bit signals. The sleep interval is inserted for this purpose. The time required to compare two SAS strings is inversely proportional to the duration of the sleep interval – the shorter the sleep interval, the faster the comparison, and vice versa. Based on the Beep-Beep, Blink-Blink and Beep-Blink combinations, an optimal range for the sleep interval needs to be determined through experiments. This range needs to be optimal with respect to the comfort level of a typical user and with respect to the time taken for comparison of encoded data. Figure 2 illustrates the encoding process using a sleep interval of 500 msec.

Sleep Interval	$\leftarrow 500ms \rightarrow$	$\leftarrow 500ms \rightarrow$	$\leftarrow 500ms \rightarrow$	$\leftarrow 500ms \rightarrow$	$\leftarrow 500ms \rightarrow$
Bits	1	1	0	0	END
Bit Signal	beep	beep	silence	silence	buzz

Sleep interval	$\leftarrow 500ms \rightarrow$	$\leftarrow 500ms \rightarrow$	$\leftarrow 500ms \rightarrow$	$\leftarrow 500ms \rightarrow$	$\leftarrow 500ms \rightarrow$
Bits	1	1	0	0	END
Bit Signal	green blink	green blink	off	off	red blink

Fig. 2. Encoding for "beeping" and "blinking" using a sleep interval of 500 msec

Encoding for "blinking". We use two LEDs, one green and one red, connected to the data pin of the parallel port of a desktop. On receiving a '1' bit in the SAS string, at the data pin, a voltage is applied at that particular data pin and the green LED "glows" on receiving this voltage. The voltage stays on unless it is explicitly cleared. Through our experiments, we found out that the "blinking" is more suitable than the "glowing". So to modify glowing to blinking, in our implementation, when the bit a '1', a voltage is applied to the pin for a fixed time interval a, followed by another fixed time interval b where voltage to the data pin is cut off. As in the encoding using system beep, we call the time interval $a + b$ as the sleep interval. For example, on a '1' bit, the LED glows for 80 msec and it stays off for the next 420 msec; on a '0' bit, the LED stays off for the entire duration of 500 msec. See Figure 2 for the encoding process using a sleep interval of 500 msec. Again, the optimal values for a, b and thus for the sleep interval will be determined through experiments. The END marker is similarly implemented using a red LED.

Implementation. For our experiments, we used a Dell machine 1.2Ghz running windows and a Laptop Compaq AMD 64 bit Turion 1.7GHz processor machine also running windows. In our experiments, the Dell machine simulates a device which only has a transmitter in the form of two LEDs, one green and one red. To enable blinking feature on Dell machine, we connect external LEDs to the data pins of the parallel port. We use C programming for our implementation and make use of the Visual Studio environment. We use the winsock libraries to establish sockets for communication over a wireless 802.11*b* channel between the Dell machine and the Compaq machine configured in the ad hoc mode. In our implementation, we configure a fixed port to listen for incoming device pairing requests. The Compaq machine is a device that initiates the pairing with the Dell machine. The Compaq machine has inbuilt speakers, that are used to play out the sounds "beep" and "buzz".

5.2 Usability Testing

Once the implementation for our schemes was complete, we started doing the most critical part, i.e., the user testing. We tested our scheme with **21 subjects**. Being at a University campus, our subjects were young, enthusiastic group open to new ideas. They belonged to a wide array of background. All of them were given a brief overview of our pairing schemes and they were also made aware of the possible scenarios where one could make use of pairing. Each user was briefed about the experimental setup comprising of the two devices, and was explained what they are supposed to do while testing the Beep-Beep, Blink-Blink and Beep-Blink combinations.

Our primary goal for the user testing was to figure out if the users are easily able to identify the good cases, i.e., when the two signals match, and more importantly, the bad ones, i.e., when the two signals mismatch. In other words, we wanted to know how often do the users commit, if at all, the *safe errors* (i.e. false positives, or identifying a match as a mismatch), and the *fatal errors* (i.e. false negatives, or identifying a mismatch as a match), following the terminology of [15].

The Set-up and Test Cases. The tests for the Beep-Beep, Blink-Blink and Beep-Blink combinations were carried out in an office room of our University, where volunteering subjects helped us perform 15 test cases that we prepared for each combination. 11 of these test cases were designed to test for the match or mismatch in SAS strings and 4 were designed to test for the synchronization delays.

Our tests comprised of 15 different pairs of 21-bit long strings running as different instances of our written programs. 11 of these pairs were intented to test for match or mismatch of the SAS data (with some padding in the beginning) assuming no synchronization delays (i.e., the END markers were always synchronized). The remaining 4 pairs of strings were intended to test for the synchronization delays (i.e., the END markers were not synchronized). In order to determine the fatal errors, we grouped the strings based upon the following types of mismatch. A "single bit" mismatch denotes a single bit mismatch in the SAS string; a "multiple bit" mismatch denotes a multiple bit mismatch in the SAS string; a "single bit-END marker" mismatch denotes a single bit mismatch in the SAS string in conjunction with a mismatch in the END marker (an example being strings "111111" and "111110", which appear to be matching in presence of a single bit delay if the user misses the mismatch in the first bit and if no END marker is in place. Identifying a mismatch in the first few bits is a common mistake committed by users as we will see next from our initial tests.); and a "multiple bit-END marker" mismatch denotes a multiple bit mismatch in the SAS string in conjunction with a mismatch in the END marker (an example being strings "010010" and "100100", which appear to be matching in presence of a single bit delay). The order in which these test cases were administered to the users was randomized to prevent users from learning as they proceeded.

The testing for the Beep-Beep combination was done on two Compaq laptop machines. For the Beep-Blink combination the testing was done with one Compaq laptop machine and one Dell desktop machine, simulating, e.g, an access point, with two LEDs connected to a data pin of its parallel port. To test the Blink-Blink case, on the other hand, we simply connected four LEDs (two green and two red) to four data pins of the parallel port of the Dell desktop machine. This simulated two close by devices with LEDs. See Figures 4 and 3 for some pictures from our experimental test set-up.

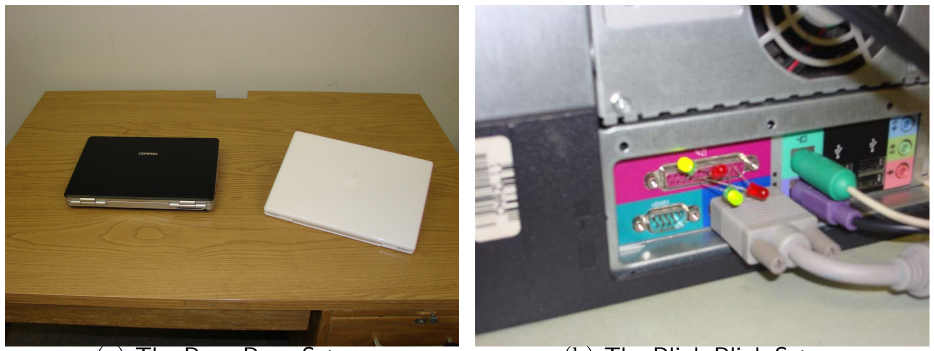
(a) The Beep-Beep Set-up (b) The Blink-Blink Set-up

Fig. 3. First two scenarios for the experimental set-up

Users had to start the process by clicking a button on the Compaq laptop machine, which transmits to the Dell machine the synchronization bit S. Right after, both machines start signalling their respective SAS string by blinking/beeping according to the bit pattern, based on the Beep-Beep, Blink-Blink and Beep-Blink combination being tested.

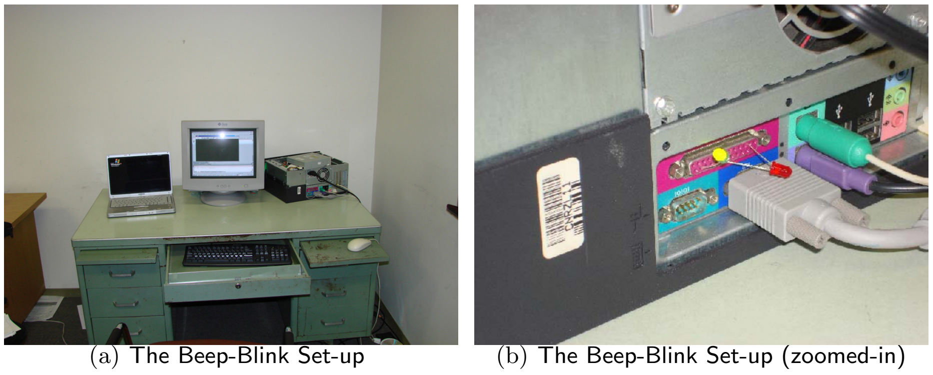
(a) The Beep-Blink Set-up (b) The Beep-Blink Set-up (zoomed-in)

Fig. 4. The third scenario for the experimental set-up

Some Initial Tests. From some initial tests that we ran, we found out the following. The subjects commit fatal errors when there is a single bit mismatch between the two SAS strings and that the error rate increases when we reduce the sleep interval to anywhere between $200 - 300$ msec. Moreover, the subjects commit fatal errors when the 1st bit in the two SAS strings differ.

Immediately analyzing these errors, we concluded that our subjects did not have any learning phase during the signaling. Another important factor was that a subject had to click start one device and almost immediately shift his/her focus onto both devices simultaneously. If the user is delayed in doing so, then he/she misses the first bit and is thus prone to committing the fatal error. To counter this occurrence of fatal error, we prepend a learning phase of five bits added as an "inner pad" to each SAS string. All the padding bits were 1's at both devices. For obvious reasons, the use of all 1's as padding bits was preferred to all 0's or a combination of 0's and 1's. This gave the subjects a learning time before each test. It turns out that including this learning phase of using the inner pad improved the accuracy of our schemes to a great extent. However, it should be noted that subjects were not told about the padding until the last test was carried out by the subject. By including a inner pad to SAS string, we also reduce the bit mismatch to occur somewhere mid string or at the end. Our results show that users were able to clearly identify bit mismatch for these strings with a very high degree of accuracy.

Optimal Values for the sleep interval. Recall that we need to determine the values for the sleep interval, which are optimal with respect to the overall execution time as well to the user comfort level. Through our experiments, we determined the following values for the first 20-bits of the strings being compared, that we will use in our test cases. For the Beep-Beep combination, $300 - 500$ msec range was optimal. The Blink-Blink combination fared well with $300 - 800$ msec sleep interval, with 300 msec corresponding to 80 msec of a "glow" and 220 msec of an "off"; 500 msec corresponding to 150 msec of a "glow" and 350 msec of an "off"; 800 msec corresponding to 300 msec of a "glow" and 500 msec of an "off". The sleep interval for the Beep-Blink combination was in the range 300 – 500 msec, with blinking of 300 msec corresponding to 80 msec of a "glow" and 220 msec of an "off"; 400 msec corresponding to 80 msec of a "glow" and 320 msec of an "off"; 500 msec corresponding to 80 msec of a "glow" and 420 msec of an "off". In each case, the sleep interval for the "blinking" END marker was set to 800 msec, corresponding to 300 msec of a "glow" and 500 msec of an "off'. This higher value was chosen for the users to be able to easily identify any mismatch in the END markers.

The Test Results and their Interpretation. The results[7] of our user testing for the Beep-Beep, Blink-Blink and Beep-Blink combinations are depicted in Tables 1, 2 and 3, respectively.

Safe Errors			Fatal Errors			
Sleep Interval	Error Rate	Execution Time	Type of Mismatch	Sleep Interval	Error Rate	Execution Time
500 ms	6/21	10.8sec	single bit	500 ms	16/21	10.8sec
500 ms	5/21	10.8sec	multiple bit	500 ms	11/21	10.8sec

Table 1. Responses of 21 users when tested for the Beep-Beep combination

Safe Errors			Fatal Errors			
Sleep Interval	Error Rate	Execution Time	Type of Mismatch	Sleep Interval	Error Rate	Execution Time
300 ms	0/21	6.8 sec	single bit	300 ms	3/21	6.8 sec
500 ms	0/21	10.8 sec	single bit	500 ms	1/21	10.8 sec
800 ms	**0/21**	**16.8 sec**	single bit	**800 ms**	**0/21**	**16.8 sec**
800 ms	**0/21**	**16.8 sec**	single bit-END marker	**800 ms**	**0/21**	**16.8 sec**
300 ms	0/21	6.8 sec	multiple bit	300 ms	0/21	6.8 sec
500 ms	0/21	10.8 sec	multiple bit	500 ms	0/21	10.8 sec
800 ms	**0/21**	**16.8 sec**	multiple bit	**800 ms**	**0/21**	**16.8 sec**
800 ms	**0/21**	**16.8 sec**	multiple bit-END marker	**800 ms**	**0/21**	**16.8 sec**

Table 2. Responses of 21 users when tested for the Blink-Blink combination

We tested the Beep-Beep combination for bit mismatch in SAS strings only (and not in END marker mismatch). The results indicate quite high (around 50-75%) fatal error rates, as well as high safe error rate (around 30%). This is

[7] No user reaction timings are taken into account.

Safe Errors			Fatal Errors			
Sleep Interval	Error Rate	Execution Time	Type of Mismatch	Sleep Interval	Error Rate	Execution Time
300 ms	0/21	6.8 sec	single bit	300 ms	2/21	6.8 sec
400 ms	0/21	8.8 sec	single bit	400 ms	1/21	8.8 sec
500 ms	**0/21**	**10.8 sec**	single bit	**500 ms**	**0/21**	**10.8 sec**
500 ms	**0/21**	**10.8 sec**	single bit-END marker	**500 ms**	**0/21**	**10.8 sec**
300 ms	0/21	6.8 sec	multiple bit	300 ms	0/21	6.8 sec
400 ms	0/21	8.8 sec	multiple bit	400 ms	0/21	8.8 sec
500 ms	**0/21**	**10.8 sec**	multiple bit	**500 ms**	**0/21**	**10.8 sec**
500 ms	**0/21**	**10.8 sec**	multiple bit-END marker	**500 ms**	**0/21**	**10.8 sec**

Table 3. Responses of 21 users when tested for the Beep-Blink combination (given one matching learning instance)

due to the fact that in order to identify matching as well as mismatching strings, i.e, to determine whether the devices beep together or not, the user needs to be aware of the pitch at which the two devices beep and also of the exact orientation of the devices. Gauging the pitches of the devices is not easy for the users and it requires them to concentrate heavily. Moreover, increasing the sleep interval duration did not improve the error rates (this is why we are only showing the results for 500 msec sleep interval). Clearly, as the results indicate, a mismatch in a single bit is even harder for the users to identify than mismatch in multiple bits. Based on these poor results, we decided not to pursue any further tests (i.e., for the END marker mismatch) for the Beep-Beep combination.

For the Blink-Blink combination, the results are quite promising. There are no safe errors, irrespective of the sleep interval duration. The users do commit some fatal errors (around 3-15%) in cases of single bit mismatch in the SAS strings when the sleep intervals are 500 msec and 300 msec. However, with a sleep interval of 800 msec, there are absolutely no errors at all – the users are easily able to detect correctly a match or a mismatch in the SAS strings as well a match or a mismatch in the END markers. The whole process took 16.8sec to complete. Since all four LEDs were attached to the same parallel port in our testing, the users were able to focus upon two of them simultaneously and as the sleep interval duration was increased, they felt more comfortable and were able to accurately identify the mismatch.

From some of our initial tests for the Beep-Blink combination, we found out that the users commit a great number of fatal errors. Except a few users, all of them could not detect correctly the very first mismatching instance consisting of a single bit mismatch in the SAS string in conjunction with a single bit delay. However, we also observed that when the users were given the very first instance as a matching instance, they could very accurately identify any other mismatching instances. This implies that a matching instance at the beginning acts as a learning instance for the users that trains them to correctly detect any errors later on. Our tests show that given one matching instance of learning, the only errors were the single bit fatal errors (around 3-9%) with the sleep interval of 400 msec and 300 msec. The sleep interval of 500ms yielded absolutely no errors and an execution time of only 10.8sec. Our results are intuitive – it is somewhat hard for the users to compare two blinking LEDs with two sounds right away, however, given one learning instance, the users get attuned to the process and correctly perform the comparison.

User Feedback. After the users finished their tests, we asked them about their order of preference among the three combinations. Out of the 21 users, 19 users preferred the Blink-Blink combination over the Beep-Blink combination. Clearly, the Beep-Beep combination was everyone's last choice.

6 Discussion and Conclusion

With the results in hand, we discuss the applicability of our proposal. We decide to discard the Beep-Beep combination, and choose the Beep-Blink combination with a sleep interval of 500 msec and an execution time of 10.8sec (given one learning instance) and the Blink-Blink combination with a sleep interval of 800 msec and an execution time of 16.8sec.

The Blink-Blink and Beep-Blink combinations are quite efficient solutions for the pairing scenarios where both devices do not have good quality transmitters and only at most one device has a relevant receiver. The Blink-Blink combination can typically be used for pairing two similar devices (such as a Bluetooth headset and a cell phone, two laptops, two cell phones) that are physically very close by and can be aligned properly with each other. The Beep-Blink combination, on the other hand, can be used for any two devices (generally, one of the devices being paired has an audio transmitter and the other has LEDs). Moreover, the Beep-Blink combination is applicable irrespective of the extreme proximity of devices. In other words, the Beep-Blink combination is also suitable for pairing, e.g., a wall-mounted access point with other devices. Of course, there is a downside to the Beep-Blink combination in that it requires the users be trained with one matching instance beforehand. However, such a training can easily be administered to the user on one of user's own devices, for example, in the form of a simulation of the Beep-Blink combination on user's cell phone. Notice that the user is anyway generally explained various steps involved in setting up its devices, using brochures or CDs.

Overall, we rate Blink-Blink over Beep-Blink because it does not require any learning and was preferred by most users in our tests. Furthermore, in noisy environments, Blink-Blink is anyway a better choice than Beep-Blink.

We hope that, even in scenarios where devices have good transmitters and also have receivers, the Blink-Blink and Beep-Blink combinations would perform better than most prior solutions [6], [15]. This is due to the reason that the schemes in [15] require the users to perform two operations, reading and comparing, while our schemes only require comparing; whereas the scheme in [6] requires the users to handle specialized equipments, such as cameras, which they might not have used previously. We expect that our schemes and the scheme of

[4] (modified using one of the SAS protocols and with one device displaying a MadLib sentence while the other "speaking" it out) might be comparable. Of course, these are mere expectations. Only a detailed comparative study of all these schemes (which is an item for our future work) can give a clear insight into their applicability among an average user population. In our future work, we would also like to compare the Blink-Blink and Beep-Blink combinations with the schemes of [11] and [9]. Recall that, similar to Blink-Blink and Beep-Blink, these two schemes also require only minimal interfaces on the devices.

In conclusion, we believe that our Blink-Blink and Beep-Blink schemes can be adopted in practice. This is so not only because of their efficiency and robustness as indicated by our test results and their universal applicability, but also because of another subtle, yet obvious, reason. Notice that "beeping" and "blinking", naturally and universally so, often serve the purpose of alerting humans in day to day life, such as when used in car indicators, in fire alarms, on ambulances, fire brigades and police vehicles, and so on. The use of beeping and blinking, as proposed, is therefore a natural approach to realize a critical security operation, such as pairing.

Acknowledgements

We would like to thank Gene Tsudik for discussions on the subject of the paper and N. Asokan for his comments on the previous versions. We would also like to thank the anonymous reviewers for their helpful comments.

References

1. D. Balfanz, D. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Network and Distributed System Security Symposium (NDSS)*, 2002.
2. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2001.
3. I. Goldberg. Visual Key Fingerprint Code, 1996. Available at http://www.cs.berkeley.edu/iang/visprint.c.
4. M. T. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Uzun. Loud and clear: Human-verifiable authentication based on audio. In *International Conference on Distributed Computing Systems (ICDCS)*, 2006.
5. S. Laur, N. Asokan, and K. Nyberg. Efficient mutual data authentication based on short authenticated strings. IACR Cryptology ePrint Archive: Report 2005/424, 2005.
6. J. M. McCune, A. Perrig, and M. K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *IEEE Symposium on Security and Privacy (S&P)*, 2005.
7. S. Pasini and S. Vaudenay. Sas-based authenticated key agreement. In *Theory and Practice of Public-Key Cryptography (PKC)*, 2006.
8. A. Perrig and D. Song. Hash visualization: a new technique to improve real-world security. In *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99)*, 1999.

9. V. Roth, W. Polak, E. Rieffel, and T. Turner. Simple and effective defenses against evil twin access points. In *ACM Conference on Wireless Network Security (WiSec), short paper*, 2008.
10. N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel. In *IEEE Symposium on Security and Privacy (S&P), short paper*, 2006.
11. C. Soriente, G. Tsudik, and E. Uzun. Beda: Button-enabled device association. In *International Workshop on Security for Spontaneous Interactio (IWSSI)*, 2007.
12. C. Soriente, G. Tsudik, and E. Uzun. Hapadep: Human asisted pure audio device pairing. IACR Cryptology ePrint Archive: Report 2007/093, 2007.
13. F. Stajano and R. J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Protocols Workshop*, 1999.
14. J. Suomalainen, J. Valkonen, and N. Asokan. Security associations in personal networks: A comparative analysis. In *European Workshop on Security and Privacy in Ad hoc and Sensor Networks (ESAS)*, 2007.
15. E. Uzun, K. Karvonen, and N. Asokan. Usability analysis of secure pairing methods. In *Usable Security (USEC)*, 2007.
16. S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *International Cryptology Conference (CRYPTO)*, 2005.