Automated Device Pairing for Asymmetric Pairing Scenarios

Nitesh Saxena and Md. Borhan Uddin

Computer and Information Science
Polytechnic Institute of New York University
Brooklyn, NY 11201, USA
nsaxena@poly.edu, borhan@cis.poly.edu

Abstract. "Secure Device Pairing" is the process of bootstrapping secure communication between two human-operated devices over a short- or medium-range wireless channel (such as Bluetooth, WiFi). The devices in such a scenario can neither be assumed to have a prior context with each other nor do they share a common trusted 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.
Recently proposed pairing protocols are based upon bidirectional physical channels. However, various pairing scenarios are asymmetric in nature, i.e., only a unidirectional physical channel exists between two devices (such as between a cell phone and an access point). In this paper, we concentrate on pairing devices using a unidirectional physical channel and analyze recently proposed protocol on this topic [14]. Moreover, as an improvement to [14], we present an efficient implementation of a unidirectional physical channel based on multiple blinking LEDs as transmitter and a video camera as a receiver.

Key words: Distributed Protocols, Mobile/Ad-Hoc Systems, Security.

1 Introduction

Short-range wireless communication, based on technologies such as Bluetooth and WiFi, is becoming increasingly popular and promises to remain so in the future. With this surge in popularity, come 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, and so on. 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 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 real-world research problem. The problem has been at the forefront of various recent standardization activities, see [20].

A recent research direction to pairing is to use an auxiliary physically authenticatable channel i.e., physical channel, also 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 channels, etc. Unlike the wireless channel, on the OOB channel, an adversary is assumed to be incapable of modifying messages, however, it can eavesdrop on, 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 requires at least 80 to 160 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, [7] and [9], reduce the length of data to be transmitted over the OOB channels to only 15 bits or so. The concept of SAS-based authentication was first introduced by Vaudenay in [22].

Based on the above-mentioned protocols, a number of pairing schemes with various OOB channels have been proposed. We review these in the next section. In this paper, we concentrate on pairing devices using *unidirectional* OOB channels. The motivation for this is that in various pairing scenarios, bidirectional d2d channels do not exist because only one of the devices being paired has a receiver (such as while pairing wifi laptop and a cell phone). Since receivers are generally expensive, it is not feasible to add them onto commodity devices, such as access points, bluetooth headsets, etc. Moreover, even in scenarios, where bidirectional d2d or the equivalent bidirectional d2h-h2d channels do exist, it is always beneficial to use only one of them for efficiency and usability reasons.

With the above motivation, we take a closer look at our previously proposed protocol that can be used for pairing two devices using a "short" unidirectional OOB channel in one direction and a unidirectional "single-bit" OOB channel in the other direction [14]. Since a "single-bit" channel is easy and fast to implement, we ignore this bidirectionality and from here on, refer to the protocol of [14] as a protocol that can pair two devices using a unidirectional OOB channel. The protocol is reviewed in next section.

Our Contributions. In this paper, we make twofold contributions:

– First, we analyze the protocol of [14] (as it did not come with a security proof). We show that the protocol is insecure in a security model that allows an adversary

to delay/replay information transmitted over the OOB channels. In fact, we argue that in such a model, it is impossible to achieve pairing with only a unidirectional OOB channel. Next, we consider a weaker yet practical security model that does not allow an adversary to delay/replay messages over the OOB channel and prove that the protocol of [14] indeed remains secure in this model.

– Second, as an improvement to [14], we propose a new implementation of a OOB channel using LEDs as transmitter and video camera as receiver. Unlike the results of [14], the implementation of our channel is much more efficient and its bandwidth improves with the increase in the number of LEDs. Since most devices have multiple LEDs (and if not, they can be cheaply added on), our implementation is an efficient way to pair two devices (such as headset and camera phone, access point and camera phone), one of which has a video camera. Our implementation has other useful applications in Bluetooth/WiFi device discovery, sensor network key distribution and in general, in data transmission.

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 analyze the protocol of [14]. Finally, in Section 5, we discuss our implementation of a d2d channel using LEDs and video camera.

2 Related Work

There exists a significant amount of prior work on the general topic of pairing. In their seminal work, Stajano, et al. [19] 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. Moreover, the infrared channels can not be perceived by humans and thus are easy to attack.

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 [5] by Levienet et al., Random Arts visual hash [10] 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 [8]. 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. [6], 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. Note that, however, the scheme is not applicable to pairing scenarios where one of the devices does not have a display or a speaker.

As an improvement to SiB [8], we earlier proposed a new scheme based on visual OOB channel [14]. The is the scheme that we analyze and improve upon in this paper. We will review this scheme in the following section and show that it is not secure in a security model in which the adversary has delaying/replaying capability on the OOB channel.

Uzun et al. [21] 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. Some recent work has focused upon pairing devices which possess constrained interfaces. These include the BEDA scheme [17], which requires the users to transfer the SAS strings from one device to the other using "button presses;" the schemes [11], [12], which require the users to compare simple blinking or beeping patterns on two devices. Most recently, the approach of [11] was extended by making use of an auxiliary device, such as a smartphone [15].

In [18], 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.

To summarize, the prior schemes are applicable to different pairing scenarios and have varying degree of usability. In this paper, our focus is on automated pairing methods using unidirectional OOB channels.

3 Communication and Security Model, and Applicable Protocols

We first review the communication and adversarial model for the SAS protocols as described in [22]. 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, drop, delay, replay and modify messages. On the OOB channel, the adversary can eavesdrop, drop, delay, replay and re-order messages, however, it can not modify them. In other words, the OOB channel is assumed to be an authenticated channel. Note that if two parties run multiple (serial/parallel) sessions with each other, then the adversary has the capability to delay, replay and re-order messages on the OOB channels among these sessions. We call such a model a "DRR-OOB" model.

We believe that considering a DRR-OOB model might be an overkill for certain OOB channels and certain applications, and that it would be useful to consider a weaker

model where two parties never run parallel instances with each other and the adversary can eavesdrop and drop OOB messages, but it can not delay, replay and re-order them among multiple serial sessions between a pair of parties. We refer to such a model as an "nDRR-OOB" model. There might be various ways in which one can ensure such a model in practice. The easiest approach is to disallow a device to run more than one parallel session with a given party at a given time and whenever a new session is executed with the same party, have the device erase from its memory the old session with the same party.

The security notion for a pairing protocol is adopted from the model of authenticated key agreement due to Canneti and Krawczyk [2]. In the DRR-OOB model, we will consider an $(n, R, \bar{R})$-adversary $\mathcal{A}$ against the pairing protocol, which is allowed to launch only R sessions per player, and only $\bar{R}$ sessions between any *pair* of players. Note that in the DRR-OOB model, $\mathcal{A}$ is allowed to delay, replay and re-order OOB messages among multiple session between two parties, while in the nDRR-OOB model, $\mathcal{A}$ (which is effectively a $(n, R, 1)$-adversary) is not allowed to do so. In both these models, the security of the pairing protocol is modeled by an interaction between $\mathcal{A}$ and the challenger that operates the network of n players $P_1, ..., P_n$. In this game, the challenger has a *private input* of bit b. This 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.

Using the launch queries, $\mathcal{A}$ can trigger any of the n players P_i to start a session of the protocol with another player P_j. The challenger responds by initializing the state of the invoked session and sending back to $\mathcal{A}$ the message it generates. The adversary can also issue send queries for any previously initialized session on a message M as input, which triggers the challenger to deliver message M to that particular session and respond by following the protocol on its behalf. Moreover, on any of the launched sessions, $\mathcal{A}$ can also issue reveal query, which gives him the key output by that particular session, if this session computed a key, and a null value otherwise. Finally, on one of the sessions, $\mathcal{A}$ can issue a Test query. In response, if this session has not completed, the adversary gets a null value. Otherwise, if $b = 1$ then $\mathcal{A}$ gets the key output by the "tested" session, and if $b = 0$ then $\mathcal{A}$ gets a random BitString of the same length.

Eventually $\mathcal{A}$ outputs a bit $\hat{b}$. We say that an adversary has *advantage* ϵ in the attack, if the probability that $\hat{b} = b$ is at most $1/2 + \epsilon$. We say that the protocol is (T, ϵ)-**secure** if for all $\mathcal{A}$'s bounded by time T the above defined advantage of $\mathcal{A}$ is at most ϵ.

An example application of this model is during authentication for an ATM transaction, where there are only two parties, namely the ATM machine and a user, restricted to only three authentication attempts.

To date, two three-round pairing protocols based on short authenticated strings (SAS) have been proposed [9], [7]. These protocols all require bidirectional OOB channels and are proven $(T, nR\bar{R}2^{-k} + \epsilon)$-secure in the DRR-OOB security model. Of course, these protocols are secure in the weaker nDRR-OOB model as well. 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 3×10^{-4}, which gives us security equivalent to the security provided by 5-digit PIN-based ATM authentication [22].

4 Pairing with a Unidirectional OOB Channel

As we mentioned in the previous section, prior SAS protocols are proven secure in the DRR-OOB model, however they require bidirectional OOB channels. In this section, we focus upon pairing scenarios where bidirectional OOB channels do not exist.

We take a closer look at the protocol of [14], which requires a unidirectional OOB channel. We show the underlying protocol in Figure 1 (we base the protocol upon the SAS protocol of [9], although it can similarly work with other SAS protocols as well). The protocol works as follows[1]. Over the wireless channel, A and B follow the underlying SAS protocol. Then a unidirectional OOB channel is established by device A transmitting the SAS data. This is followed by device B comparing the received data with its own copy of the SAS data, and transmitting the resulting bit b of comparison over a OOB channel (say, displayed on its screen). Finally, the user reads the transmitted bit b and accordingly indicates the result to device A by transmitting the same bit b over an h2d input channel.

4.1 Protocol of [14] in the DRR-OOB Model

The protocol of [14] did not come with a security analysis [14]. Therefore, the first and a natural question is whether the protocol remains secure in a DRR-OOB security model. Unfortunately, the protocol turns out to be insecure in the DRR-OOB model. We show our attack next. In fact, we argue that it is hard to achieve pairing using only a unidirectional OOB channel in the DRR-OOB model.

The attack we describe next stems from the fact that only a single bit b, indicating a "success" or a "failure", is transmitted in the second step, i.e., over the d2h channel, and that this bit of information can be *delayed or replayed* (recall that our security model, described in Section 3, allows an adversary to do so!). Therefore, the attack is exploited as follows.

1. An instance of the above pairing protocol is run between two devices. The adversary does not insert any messages (specifically, its own public key(s), for which it knows the secret keys or its own secret shared key, based on the protocol) on the wireless channel. However, the adversary stalls the bit b (which indicates a "success") to be transmitted over the d2h channel. This forces the user to abort the protocol and re-run it.
2. During the second instance of the protocol, the adversary first inserts its own messages over the wireless channel and then delivers the previously stalled bit b over the d2h channel. Since the bit b indicates a "success", the user is fooled into accepting the protocol instance instead of aborting it.

A similar attack can be based upon replaying of the bit b, instead of its delaying, as follows. Over the first instance of the protocol, the adversary does nothing except for recording the bit b. The adversary hopes that another instance of the protocol is run

[1] A similar modification was suggested to the protocol where devices exchange their public keys over the wireless channel and exchange the (160-bits long) hash of the concatenation of the two public keys over the OOB channel [14].

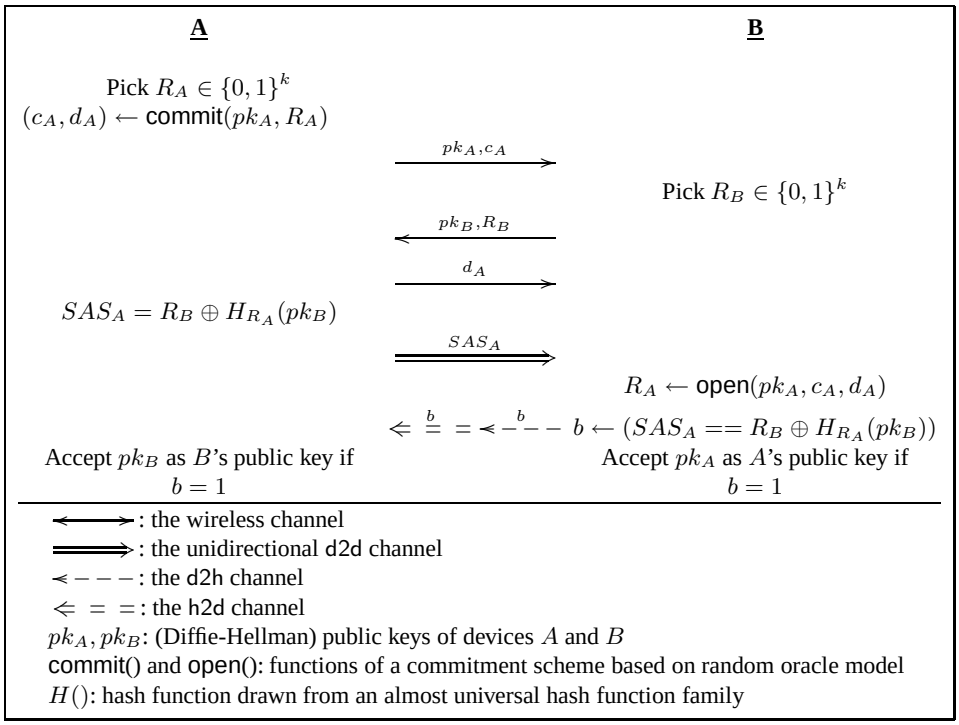


Fig. 1. The protocol of [14] based on the SAS protocol of [9]

and if so, it attacks the new instance by inserting its own messages over the wireless channel, and simply replaying the previously recorded bit b over the d2h channel.

A general implication of the above attack on the protocol of [14] is that it seems hard, if not impossible, to achieve pairing with a unidirectional d2d channel. In other words, it appears hard to establish mutual authentication with a unidirectional d2d authenticated channel. We know, from the original unidirectional message authentication SAS protocol of Vaudenay [22], that a device A can authenticate itself to device B if there exists a physical channel from A to B. The question is can this SAS channel also be used by B to authenticate to A. Suppose that B wants to authenticate a message m_B to A. B can simply send m_B to A over the wireless channel, which the adversary might modify to m'_B. Now, both B and A need to know if m_B was modified during the transmission or not, i.e., if $m_B = m'_B$ or not. It is easy for B to know this: A can authenticate to B m'_B using the unidirectional SAS from A to B, and B can simply verify if $m'_B = m_B$ or not. However, there appears to be no way for A to know if it received the same message m_B that B transmitted, except for B itself notifying A whether or not $m'_B = m_B$, which can be achieved by B transmitting the bit b indicating the result of match over a "single-bit" authenticated channel from B to A. However, this brings us back to the attack that we described on the protocol of [14], since the bit b can be delayed or replayed by an adversary.

4.2 Protocol of [14] in the nDRR-OOB Model

As shown in the previous section, the protocol of [14] is not secure in the DRR-OOB model. Now, we analyze the protocol in the nDRR-OOB security model. As pointed

out in Section 3, this is a more practical model for certain applications. Fortunately, the protocol can indeed be proven secure in the nDRR-OOB model. In fact, we show that using the modification as in the protocol of [14], any known SAS protocol P based on bidirectional OOB channels, which is secure in the DRR-OOB model (e.g., the protocol of [9]), can be converted into a pairing protocol Q based on a unidirectional channel in the nDRR-OOB model (e.g., the protocol of Figure 1).

Theorem 1 *If any known SAS protocol P is $(T, nR2^{-k}+\epsilon)$-secure against a $(n, R, 1)$-adversary in the DRR-OOB model, then the pairing protocol Q is $(T + \delta, nR2^{-k} + \epsilon)$-secure against a $(n, R, 1)$-adversary in the nDRR-OOB model, where δ denotes a small polynomial amount of time.*

Proof. We prove the above theorem by contrapositive. In other words, we show that if there exists a $(n, R, 1)$-adversary $\mathcal{A}$ in the nDRR-OOB model that can win against the protocol Q with a probability significantly better than $nR2^{-k}$ and in time T, then we can construct a $(n, R, 1)$-adversary $\mathcal{B}$ in the DRR-OOB model that can win against the protocol P with a probability significantly better than $nR2^{-k}$ and in approximately the same time T.

The idea of the construction of the adversary $\mathcal{B}$ is very simple. Basically, $\mathcal{B}$ receives the queries from $\mathcal{A}$, submits them to its challenger and forwards the responses received back to $\mathcal{A}$, thereby perfectly simulating the role of the challenger to $\mathcal{B}$.

$\mathcal{A}$ starts off by issuing the launch queries, which $\mathcal{B}$ submits to its challenger and responds back to $\mathcal{A}$ with the messages delivered by the challenger. $\mathcal{B}$ does the same when $\mathcal{A}$ issues the send queries for all the protocol messages on the wireless channel and the message transmitted over the OOB channel in one direction (say SAS_i from P_i to P_j) (since the two protocols are alike in terms of the messages exchanged over the wireless channel and message transmitted over the OOB channel in one direction). However, when $\mathcal{B}$ receives the message transmitted over the OOB channel in the other direction (say SAS_j from P_j to P_i) from the challenger, then $\mathcal{B}$ computes $b = (SAS_i == SAS_j)$ and sends b to $\mathcal{A}$.

When $\mathcal{A}$ issues the reveal queries and finally the test query, $\mathcal{B}$ simply submits these queries to the challenger and replies back with the responses it receives from the challenger.

If $\mathcal{A}$ succeeds in correctly distinguishing the key output by the "tested" session, so does $\mathcal{B}$; since the two protocols are exactly alike in terms of the messages exchanged over the wireless as well as the OOB channel in one direction, and also both essentially have the same winning condition, since $(b = 1) \Rightarrow (SAS_i = SAS_j)$. Both $\mathcal{A}$ and $\mathcal{B}$ win with the same probability and have only a small δ time difference that is needed by $\mathcal{B}$ in computing the boolean expression $b = (SAS_i == SAS_j)$.

5 Automated d2d Channel using LEDs and Video Camera

In this section, we discuss our implementation of a d2d channel in which the transmitter is equipped with LEDs and the receiver is equipped with a video camera. Our channel implementation is quite efficient and its bandwidth, unlike the results of [14], improves with the increase in the number of LEDs. The implementation can also be used on regular displays by simulating the LEDs on them. In the pairing application, we use this channel to transmit 15-bits of SAS data.

5.1 Encoding using LEDs

In our encoding, we need two types of LEDs: a "sync" LED for synchronization at the beginning and end of SAS data transmission, and one or more "data" LEDs for transmitting the SAS data. The sync LED is different in color from the data LEDs – in our setup we keep a red LED as the sync LED and green LEDs as the data LEDs. LEDs are placed horizontally and vertically on the transmitter display. The sync LED can be placed at any vertical or horizontal position. The bit locations of the data LEDs increases from left-to-right and top-to-bottom. So, the top left data LED shows the first bit of the SAS data. There should be some gap between the two LEDs which needs to be at least half of the width of the LED itself.

The sync LED is used for indicating the beginning and end of the SAS data transmission in order to detect any synchronization delays, adversarial or otherwise, between the two devices. The sync LED is kept in "ON" state only at the beginning and end of data transmission and in "OFF" state otherwise.

The data LEDs are used for SAS data transmission by indicating different bits ('0'/'1') for different states (OFF/ON) of LEDs; in our setup, we used the ON state of a data LED as a bit '1' and the OFF state as bit '0'. Each transmitter needs to have one or more data LEDs and the more in number of data LEDs are, the speedier the SAS data transmission becomes. The transmitter can send the number of bits equal to the number of data LEDs, i.e., one bit per LED at a time. If N is the number of Data LEDs, the transmitter can display N bits of SAS data at a time. For transmitting 15-bits SAS data it requires $\lceil \frac{15}{N} \rceil$ frames. The state of the sync and data LEDs is kept unchanged for a certain time period so that a stable state can be easily captured from the video stream of the receiver video camera. Each stable state captured from the video stream is termed as a "BitFrame". For our setup, the time duration a BitFrame is kept unchanged (henceforth referred to as the "hold time") is set to an experimentally determined value of 300 ms. After every 300 ms, next N bits of the SAS data are shown in the next frame. This process continues until all bits of SAS data are transmitted. If the last frame does not have N number of SAS bits to show, the first few LEDs show the data bits and the remaining are kept OFF.

For discovering the LEDs' location, color, dimension at the receiver side, we need two extra frames – an "All-ON" frame having all LEDs in ON state and an "All-OFF" frame having all LEDs in OFF state. Before transmitting the frames containing the SAS data, the All-ON and All-OFF frames are first displayed. These two frames are displayed within the same hold time of 300 ms. In addition to All-ON and All-OFF frames, we need another frame, to detect synchronization delays, having the sync LED in ON state and the data LEDs in OFF state. This frame is displayed at the end, after the completion of SAS data transmission. Therefore, overall we need a total of three extra frames. Thus, the total number of frames to be transmitted is $\lceil \frac{15}{N} \rceil + 3$, which yields a total transmission time of $(\lceil \frac{15}{N} \rceil + 3) \times 300$ ms, where N is the number of data LEDs.

5.2 Decoding using a Video Camera

The two devices being paired first execute the protocol as in Figure 1 over the wireless channel. When the receiver device is done with SAS data computation, it turns on its video camera, asks the user of the device to adjust its camera setting, focus on the LED-based display of the transmitting device and press "OK" button when done.

The user does the adjustment as needed and presses the OK button. After this, the receiver sends the "ready" signal to the transmitter and requests the transmitter to send the acknowledgement over the wireless channel when it is done with computing its SAS value and ready to start transmitting over the unidirectional channel. The transmitter acknowledges the receiver when it is ready for transmitting the SAS data and starts transmitting over the unidirectional channel. In this setting of unidirectional channel, the receiver must have higher reception rate than transmitter's transmission rate. So, the video camera must have higher frame rate than frame rate of the transmitters displayer. If frames are not carefully captured from the video stream, there is a chance of obtaining the counterfeit frames which contain the transition state of LEDs. Such frames may contain some LEDs of one state and some LEDs of next state.

Resolving the Timing Issue of Frame Capturing from Video Stream. Assuming that the transmission delay of acknowledgement from the transmitter to receiver is negligible (5-6 ms) compared to the "hold time" (of 300 ms) between two successive frames at the transmitter, the receiver captures the first frame from the video stream after a time equal to the half of the hold time (i.e., 150 ms) after receiving the acknowledgement. The receiver video camera also has a delay (about 30-40 ms, as most common cameras have a rate of 30-40 frames per second) of capturing the frame from video stream. So, the first frame is captured after the half of the hold time, after getting the acknowledgement from the receiver. The timestamps of capturing rest of the frames is pre-calculated by adding the hold time (300 ms) for each frames with capturing timestamp of the first frame. The captured frames are processed after the completion of capturing of all transmitted frames. During capturing, the captured frames are saved from the video stream buffer location of frames to another location in main memory for later processing. There is some initial delay in capturing of first frame and it is adjusted by capturing the frame in the middle of hold time, however, there is no delay per frame for capturing the rest of the frames. Thus, the frame capturing from the video stream works successfully in real time. Note that our scheme does not require global clock synchronization for the transmitter and receiver. Figure 2 depicts the synchronization of transmission and reception of data. In this figure each small rectangle on the receiving window denotes a video frame of video stream and brown arrow marked with "Video Stream Capturing" denotes the propagation of transmitted signal to streamed frame in video stream, which makes sense that there is some propagation delay of an input transition from transmitter's side to receiver's video stream.

Detection of LEDs and Retrieval of SAS data from Video Frames. The frames are processed after the completion of capturing of all transmitted frames. The captured frames are processed by direct access to the memory address location of pixels. Direct addresses of pixels are calculated by knowing the pointer of memory address of first pixel of the frame and calculating other pixels address using the stride, width and length of frames.

Our LEDs location and dimension detection algorithm is a simple but fast, robust and efficient one - unlike any existing object/face detection algorithms [13, 16, 23]. It detects the position and dimension of LEDs deterministically. It is able to detect any shape/geometry of LEDs unlike [16, 23] and doesn't require any prior training unlike

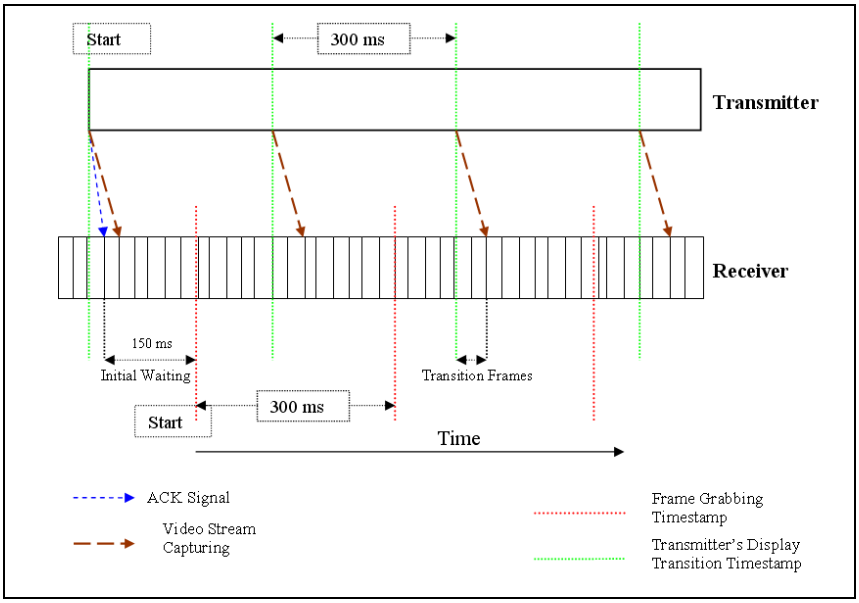


Fig. 2. Synchronization of Transmission and Reception of Data

[13, 16]. It uses the color threshold adjustment technique like [24] to detect the LEDs position and dimension.

The maximal differences of RGB values, $max(dR, dG, dB)$ (denoted as μ), of each pixel of All-OFF and All-ON frames are measured and kept in memory, and using a threshold value for μ, BitStrings are built for each row of pixels. For example, if the μ exceeds a certain threshold, the corresponding bit in string becomes '1', otherwise it becomes '0'.

Each BitString is matched against a regular expression for consecutive 1s. For each matching, its center is calculated and its safeness and centeredness as an LED center is checked by matching against the already explored LEDs and exploring only the adjacent pixels of this center in the frame. If its safeness and centeredness is proved, it is taken as an LED and put in explored list of LEDs. After checking for each match of regular expression for each bit strings, counts of explored LEDs is matched against the original count of the LEDs. This process continues up to a number of times by adjusting the threshold value of μ and constructing the new BitStrings until the count of LEDs matches. See Figure 3 for an example of detection of LEDs from the BitString.

After successful discovery of LEDs, the length, width, average RGB values of ON and OFF states of LED area for each LEDs are stored in memory for detecting the On-Off state of LEDs in subsequent BitFrames.

From the successfully discovered LEDs, red colored the sync LED is detected on the basis of its color. On the basis of the location of the sync LED, rest of the LEDs are clustered according to a threshold value of proximity among the LEDs. After successful detection of LEDs, the data LEDs are sorted according to the left-to-right and top-to-bottom ordering of coordinates for a maximum value of tolerance limit in deviation of coordinates. Tolerance limit is set by measuring the average width and length of discovered LEDs. Now for bit frames containing SAS data, average RGB values of the

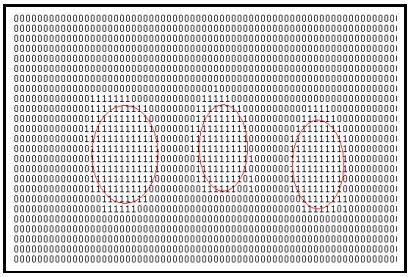


Fig. 3. Detected LEDs from BitString

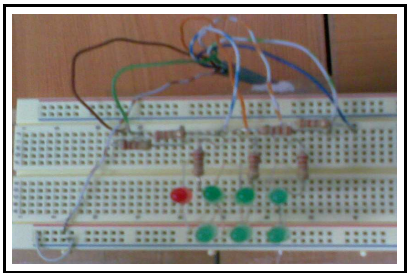


Fig. 4. Second Setting: breadboard with LEDs

area of only the discovered LEDs are explored and matched against the ON and OFF state of RGB values of the LEDs with a tolerance limit. If the average RGB values of the area matches with the ON state of the LED, the corresponding bit is detected as '1' and if these match with the OFF state, it is detected as '0'. In this manner, the whole SAS string is retrieved by exploring the discovered LEDs for each bit frames. If there arises any ambiguity, i.e., the average RGB values match with both the condition of ON and OFF state or neither of them, tolerance limit is adjusted and decoding is repeated up to a threshold number of times.

The last frame is examined to determine whether the sync LED is in the ON state and that all data LEDs are in the OFF state. If not, there is an indication of a sync bit failure due to synchronization delays.

If the extracted SAS matches with the computed SAS on the receiver and the frames pass the synchronization test, the receiver and transmitter are successfully paired. Otherwise, they fail due to mismatch of SAS or delay in synchronization. For a successful pairing, the LEDs are marked with a rectangle of green color around them and for a failed case, the LEDs are crossed with red color. Observing the graphical result on screen of the receiver, the user either accepts or aborts the pairing on the transmitter's device.

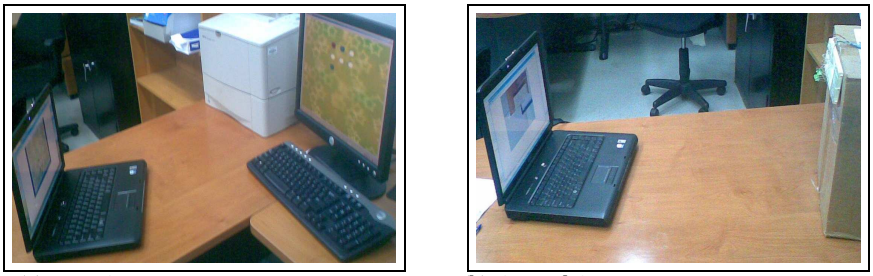
5.3 Experimental Setup

We implemented and tested our channel in two different settings. One showed the BitFrames on the monitor of a desktop PC and the other showed the BitFrames on real implementation of the scheme on breadboard using 7 LEDs (1 sync and 6 data LEDs), the breadboard being interfaced to a desktop PC using DB-25 parallel printer port. In the first setting, bitmap images of actual ON and OFF states of real LEDs are used. The pictures from the two settings are shown in Figures 5(a), 4 and 5(b).

In both the schemes receiver is the Dell Vostro 1500 (Intel Core 2 Duo 1.6 GHz with 2 GB RAM) Laptop having Integrated Webcam and wireless channel is Wireless LAN (54 Mbps) of our university. The integrated webcam on the laptop has the capability of capturing 30 frames/second and it can take frames of fixed dimension of $640X480$ pixels. We wrote the simulator in Microsoft VC# to implement the last two OOB message exchange as in the Figure of 1, assuming that the devices have already exchanged the first three messages over the wireless channel and each has computed its SAS value. The implementation processes the frames, extracts the SAS data from video stream and shows the result on screen. Actuation to real-timing is maintained

by `Environment.TickCount` variable of MS VC#. The webcam is interfaced on port 1024 of the computer. Message passing is used to communicate with the webcam. We used "avicap32.dll" for capturing the video. The webcam can be replaced with any good IP Camera for better resolution of frames without any modification in the existing simulator. The camera is set in NON-STOP video capturing mode and frames are taken setting the camera in preview mode. Camera controller is added in the simulator to adjust the focus, tilt and pan of camera. For both the schemes transmitter is Dell Desktop (Intel Xenon Processor 1.8 GHz with 1 GB RAM). Communication with the transmitter is done by implementing the Client-Server communication model between the transmitter and receiver.

The simulator is used to generate huge number of test cases for different numbers and orientations of LEDs. We tested our system for different brightness of frames by setting different level of brightness of the monitor. In the first setting, frames are prestored in transmitter and are displayed when the receiver sent the request of transmitting the frames. It is used to test our pairing scheme on this channel easily and rigorously.

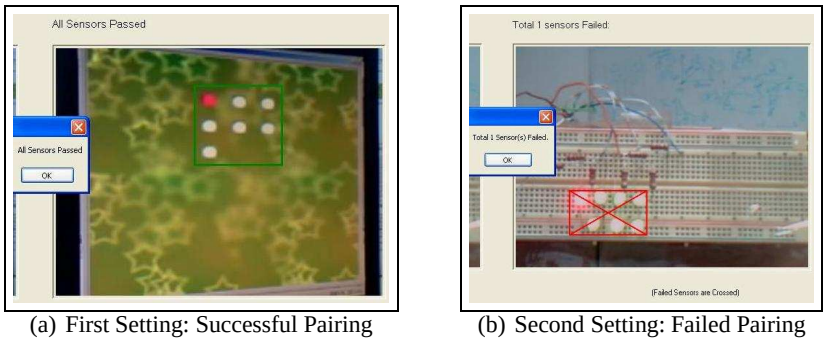


(a) First Setting: transmitter is monitor (b) Second Setting: transmitter is LEDs on breadboard

Fig. 5. Two Settings: receiver is laptop camera for both the settings

5.4 Experiment Results

A couple of snapshots of the results of execution of our scheme in both settings are shown in Figures 6(a) and 6(b).



(a) First Setting: Successful Pairing (b) Second Setting: Failed Pairing

Fig. 6. Partial Screenshots of Results of the Two Settings

From the result of the first setting, we found that our implementation works for different number and orientation of LEDs. LEDs need not be in any fixed location on the screen and they can be on any random location while maintaining the left-to-right and top-to-bottom ordering among themselves. We tested lot of test cases for the different number and orientation of LEDs and all of them passed without any error. We also executed some test cases by changing SAS data on test cases, replacing the valid frames with invalid frames and replacing the sync frame with other frames. All these test cases were successfully determined as failing cases. By changing the brightness of the monitor, we found that our scheme works for any brightness of the monitor. Our first scheme works fine if the distance between the transmitter or receiver is less than 1.5–2 meters. If the distance is greater than 2 meter, the LEDs become to tiny to detect for the camera. In this case LED count doesn't match and the simulator searches up to a threshold number of times by adjusting the color threshold value. If it doesn't find any match of count of LEDs, it shows the failure message. By using a camera with a better resolution (which is currently 640X480) the distance of the transmitter and receiver can be increased.

Our implementation in the second setting also works fine. The Light from illuminating LEDs scatters around the LEDs. For testing the maximum light scattering effect we did not separate the LEDs by any type of separator or did not keep them inside the small holes to negate the lighting effect among the LEDs. All the test cases (for successful as well as failing pairing) passed without any problem. Our implementation of color threshold adjustment strategy during discovery of LEDs also works fine in this setting. This setting is also tested with varying distances between the transmitter and receiver. It works fine up to 2.5 meters of distance, longer than in the first setting. Lights from LEDs directly falls on camera and thus ON-OFF state detection becomes easy as there is more change in lighting effect in this setting.

Based on the results obtained, we summarize the following salient characteristics of our implementation in both settings.

Transmission Time. Using N data LEDs and one sync LED, the transmission requires about $(\lceil \frac{15}{N} \rceil + 3) \times 300$ ms for 15-bit of SAS data. Extraction of SAS data from captured frames requires less than 1 second. Therefore, for a typical display which has 2 data LEDs and 1 sync LED, our scheme requires less than 4.15 s to complete the whole process. If a device has 5 data LEDs and 1 sync LED, it will require less than 2.8 s. Of course, since we need at least three extra frames, the transmission can not take place quicker than 900 ms, no matter how many LEDs we have. Moreover, we require a device to have at least two LEDs, one of which has a unique color.

Distance. Our scheme works well, if the distance between the sender and receiver is less than 1.5-2.5 meters. Real implementation of channel with the LEDs on breadboard in second setting shows that our channel is efficient for about 2-2.5 meter distance between the receiver and transmitter. This represents a promising improvement over the existing d2d channels which can work for very little distance between the transmitter and receiver.

Brightness and Intensity of Light. Our channel is robust to varying brightness and intensity of light. It compares the states of LEDs ON and OFF state on current default settings of brightness and intensity of light on devices we tested on. From first two, All-OFF and ALL-ON frames, it learns the environment.

5.5 Other Applications of Our Implementation

Our channel implementation using LEDs and video camera and our underlying algorithms for real-time capturing of the frames from video stream, processing the frames efficiently and extracting data from video frames, can be used in a number of applications. We briefly discuss some applications as follows.

Device Discovery. A device needs to, before starting to communicate with another wireless device, first determine its address. Currently, in Bluetooth and WiFi, such a device discovery is performed over the wireless channels and has serious implications in terms of efficiency as well as usability – the device senses for devices in the neighborhood and dumps a list of devices (which could be long) and asks the user to select the device it wants to connect to. Using our d2d channel, one can discover a device over the physical channel itself – the user goes near the device it wants to connect its device to and presses a button on it to start transmitting its address in the form blinking LEDs and reads the address using the camera. Using just two data LEDs, this would only take about 9 seconds to discover a 48-bit long Bluetooth device address, and ease the burden on the user.

Secure Key Distribution in Sensor Networks. Before, deploying a sensor network, the nodes need to be provided with keys that they can use to secure communicate among themselves. Due to the lack of a trusted infrastructure, such a key distribution needs to be performed on-site by the administrator of the network. Moreover, due to lack of hardware interfaces (such as USB interfaces) on sensor nodes and for usability reasons, the key distribution must be performed wirelessly. Using the existing, so called key pre-distribution schemes, e.g., [4], the key distribution can be achieved, if one could establish a secure channel or a key between each sensor node and the base station. We are currently in the process of extending our implementation of the d2d channel and the protocol of Figure 1 to simultaneously pair each sensor node with the base station. Most existing commercial sensor nodes generally have three LEDs and the base station PC can be easily connected with an IP camera. We claim that our approach would be much more efficient, scalable and user-friendly than a recently proposed scheme [3].

General Data Transmission. Our implementation can also be used in applications other than security, for data transmission. Using N data LEDs (or an equivalent sized display) and a hold time of 300 ms, we are able to achieve a bandwidth of $3.33N$ bps. With such a bandwidth, one could efficiently send out information such as advertisements, calendars, low-resolution images, etc.

6 Conclusion

In this paper, we focused upon pairing two devices using unidirectional OOB channels. We analyzed the protocol of [14] and proved its security in a reasonable security model. We also devised an efficient implementation of an OOB channel using LEDs as transmitter and video camera as a receiver. With a display consisting of just two LEDs, our implementations takes less than 6.5 seconds. With increase in the number of LEDs, the bandwidth of our channel gets better. For example, with six LEDs, we need less than 2.8 seconds. Our implementation has other useful applications in Bluetooth/WiFi device discovery, sensor network key distribution and in general for data transmission.

Acknowledgments. We would like to thank N. Asokan for his comments on an earlier version of this paper.

References

1. D. Balfanz, D. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *NDSS*, 2002.
2. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, 2001.
3. K. Cynthia, M. Luk, R. Negi, and A. Perrig. Message-in-a-bottle: User-friendly and secure key deployment for sensor nodes. In *ACM SenSys*, 2007.
4. W. Du, J. Deng, Y. S. Han, and P. K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *ACM CCS*, 2003.
5. I. Goldberg. Visual Key Fingerprint Code, 1996. Available at `http://www.cs.berkeley.edu/iang/visprint.c`.
6. M. T. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Uzun. Loud and Clear: Human-Verifiable Authentication Based on Audio. In *ICDCS*, 2006.
7. S. Laur, N. Asokan, and K. Nyberg. Efficient mutual data authentication based on short authenticated strings. IACR Cryptology ePrint Archive: Report 2005/424, 2005.
8. 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*, 2005.
9. S. Pasini and S. Vaudenay. SAS-Based Authenticated Key Agreement. In *PKC*, 2006.
10. A. Perrig and D. Song. Hash visualization: a new technique to improve real-world security. In *CrypTEC*, 1999.
11. R. Prasad and N. Saxena. Efficient device pairing using human-comparable synchronized audiovisual patterns. In *Applied Cryptography and Network Security (ACNS)*, 2008.
12. 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.
13. H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. In *Pattern Analysis and Machine Intelligence(PAMI)*, 1998.
14. 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 (ISP'06), short paper*, 2006.
15. N. Saxena, M. B. Uddin, and J. Voris. Universal device pairing using an auxiliary device. In *Symposium On Usable Privacy and Security (SOUPS)*, 2008.
16. H. Schneiderman and T. Kanade. A statistical method for 3d object detection applied to faces and cars. In *TRINITY*, 2003.
17. C. Soriente, G. Tsudik, and E. Uzun. BEDA: Button-Enabled Device Association. In *International Workshop on Security for Spontaneous Interaction (IWSSI)*, 2007.
18. C. Soriente, G. Tsudik, and E. Uzun. Hapadep: Human asisted pure audio device pairing. Cryptology ePrint Archive, Report 2007/093, 2007.
19. F. Stajano and R. J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Protocols Workshop*, 1999.
20. J. Suomalainen, J. Valkonen, and N. Asokan. Security associations in personal networks: A comparative analysis. In *ESAS*, 2007.
21. E. Uzun, K. Karvonen, and N. Asokan. Usability analysis of secure pairing methods. In *USEC*, 2007.
22. S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *CRYPTO*, 2005.
23. P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition*, 2001.
24. J. S. Weszka. A survey of threshold selection techniques. *Computer Graphics and Image Processing*, 7:259–265, 1978.