Implementing an Attack on Bluetooth 2.1+ Secure Simple Pairing in Passkey Entry Mode

Johannes Barnickel, Jian Wang, Ulrike Meyer
IT Security Research Group
RWTH Aachen University
Germany

***Abstract*—Due to the serious security issues found in early Bluetooth revisions, Bluetooth revision 2.1 (and later) uses a new pairing process called Secure Simple Pairing (SSP). SSP allows two devices to establish a link key based on a Diffie-Hellman key agreement and supports four methods to authenticate the key agreement. One of these methods is called Passkey Entry method, which uses a PIN entered on one or both devices. The Passkey Entry method has been shown to leak this PIN to any attacker eavesdropping on the first part of the pairing process. If in addition, the attacker can prevent the pairing process to successfully complete and the user uses the same PIN twice (or a fixed PIN is used), the attacker can mount a man-in-the-middle attack on a new run of the pairing process.**

In this paper, we explore the practicality of this attack and show that it is should be taken very seriously. Lacking devices with a reasonably programmable Bluetooth stack to implement the attack upon, we created Bluetrial: our own implementation of the relevant Bluetooth parts using the GNU Radio platform on USRP and USRP2 devices.

I. Introduction

Bluetooth is a standard for personal area networks in the 2.4 GHz ISM band. Among other applications, it is used to connect mobile phones to headsets and hands-free car kits, to connect personal computers to keyboards, mice, and printers, and for data exchange between two mobile phones. To securely connect two devices, Bluetooth provides a pairing mechanism. Two devices are switched into a special mode by the user, and are then able to connect to each other and to establish a link key, which will be used to encrypt the traffic subsequently exchanged between them. For later connections, the same link key is reused such that the pairing process is used only when the devices connect for the first time.

Due to security issues in the pairing process of earlier Bluetooth versions, version 2.1 introduced a new pairing process called Secure Simple Pairing (SSP). SSP allows two devices to establish a so-called link key based on a Diffie-Hellman key agreement and supports four methods to authenticate the key agreement. These four methods target devices with different combinations of user interfaces such as key boards, displays, or special pairing buttons.

One of these method is the Passkey Entry method which uses a PIN (displayed on one and entered on the other device or entered on both devices) to authenticate the Diffie-Hellman key agreement. This Passkey Entry method has been shown to be vulnerable against a man-in-the-middle attack [1]. This attack allows an attacker that is able to record the legitimate pairing process and is able to prevent its successful completion (e.g., by jamming) to replace the link key established between the legitimate devices with link keys known to himself. As a consequence, the attacker can impersonate one device to the other and eavesdrop on and manipulate the traffic exchanged between these devices. The prerequisite for the success of this attack is that the user that pairs the two legitimate devices reuses the same PIN on the second try when his first try to pair the two devices has failed.

In this paper we explore the difficulty of implementing and mounting this attack in practice. Lacking devices with a reasonably programmable Bluetooth stack to implement the attack upon, we first present **Bluetrial**: our own standard compliant implementation of the Bluetooth Secure Simple Pairing process using the GNU Radio platform on USRP devices. We then present and evaluate our implementation of the man-in-the-middle attack on the Passkey Entry method on Bluetrial. We show that our attack has a success rate of 90% when using a data rate of 890 kbit/s. While this may still seem far from realistic Bluetooth devices which operate at 1000 kbit/s and apply frequency hopping every 625μs, it is only a matter of time until implementation issues will be overcome or better hardware becomes available. We conclude that it is high time to take this attack seriously and propose a number of solutions to prevent this attack in future revisions of Bluetooth devices, and in the Bluetooth specification itself.

II. Background

A. Brief History of Bluetooth

Revision 1.1, finalized in 2002, was the first major Bluetooth release. Bluetooth 1.1 was updated to 1.2 in 2005, introducing adaptive frequency-hopping spread spectrum to better cope with radio interference. All later Bluetooth revisions are still compatible to 1.2. Revision 2.0, introduced in 2004, adds the Enhanced Data Rate (EDR) extension for faster file transfers (increased from 1 Mbit/s to 3 Mbit/s) to the standard. All Bluetooth versions up till 2.0 use the same insecure pairing process for which various weaknesses have been published in the past [2], [3], [4]. With growing market penetration, implementations of these attacks on real-world devices appeared and were covered in the mainstream media. Thus, the 2007 revision 2.1 introduced the **Secure Simple Pairing** (SSP) mechanism, a new pairing mechanism designed to prevent the

Capabilities of Responding Device (B)	Capabilities of Initiating Device (A)		
	Display digits	Display and yes/no Keys	Keyboard Only
Display digits	Just Works	Just Works	Passkey Entry
Display, yes/no keys	Just Works	Numeric Comp.	Passkey Entry
Keyboard only	Passkey Entry	Passkey Entry	Passkey Entry

TABLE I
AVAILABILITY OF SSP METHODS ([5] VOL. 3 P. 207)

Term	Definition
A	Identifier of Device A
$capsA$	Capabilites of A
$PKax$	DH public value of A
$DHkey$	Newly established DH key between A and B
ra	PIN of A, 20 bit
ra_i	i-th bit of PIN of A
N_A	Nonce by A, 128 bit
Nai	Nonce by A for i-th round, 128 bit
Cai	Commitment of A in round i, 128 bit
$f1, f2, f3$	Cryptographic one-way functions
Ea	Authentication stage 2 value by A, 128 bit

TABLE II
OVERVIEW OF TERMS IN SECURE SIMPLE PAIRING

known attacks. In 2009, Revision 3.0 introduced an alternate physical 802.11 link for fast data exchange once the Bluetooth connection is established. The 2010 revision 4.0 added a complete alternate protocol stack called Bluetooth Low Energy (BLE) mode to reduce power consumption. Both revisions 3.0 and 4.0 continue to use SSP without changes.

Unfortunately, one of the four pairing methods introduced in SSP, the Passkey Entry method, has been shown to be vulnerable to a man-in-the-middle attack in 2008 [1]. However, to the best of our knowledge no implementation of this attack has so far appeared „in the wild“ and no efforts have been taken so far to modify the standard to prevent this attack.

B. Secure Simple Pairing

The SSP process establishes a shared secret key called link key between two pairing devices A and B. SSP starts with an initial Diffie-Hellman (DH) key exchange in which the devices A and B exchange their public Diffie-Hellman values $PKax$ and $PKbx$ in the clear.

In the following authentication stage 1, the initial DH exchange is authenticated using one of four supported methods called Just Works, Out of Band, Numeric Comparison, and Passkey Entry, respectively. In authentication stage 2, the two devices confirm that they indeed established the same DH-key. The link key is ultimately derived from the confirmed DH-key.

Which of the four different modes for authentication stage 1 is used depends on the input and output capabilities of the two pairing devices. Table I provides a overview on the recommended selection of the modes for different combinations of device capabilities. In the following, we briefly review first three modes and detail the Passkey Entry method, which is of primary interest in the context of this paper.

The **Just Works** mode does not require a display or keyboard, and is most commonly on headsets today. It does not provide any means to authenticate the initial DH-exchange and as such is naturally vulnerable to man-in-the-middle attacks.

The **Out Of Band** (OOB) mode uses another communication channel such as Near Field Communication (NFC) during the pairing process to exchange fingerprints of the exchanged public DH-values. To the best of our knowledge, OOB is not implemented in any current devices on the market.

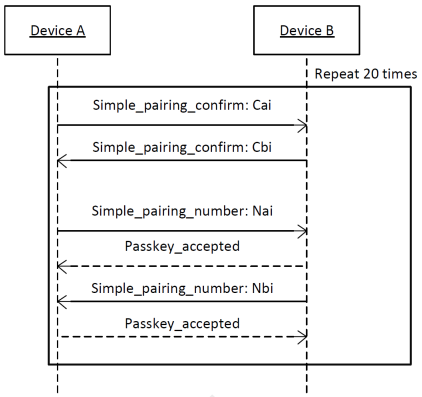


Fig. 1. SSP Authentication Stage 1 with Passkey Entry

The **Numeric Comparison** mode requires that both devices have a display and a yes/no button, but does not require a numeric keyboard on any of the two devices. After initiating the pairing process, both devices display a six-digit number, which are computed from the public DH-values and freshly chosen nonces with the help of a hash function. The user confirms that both numbers are the same by pressing a button on each of the devices and thereby authenticates the initial DH-exchange. An attacker cannot derive any information on the resulting link key from the displayed numbers. The Numeric Comparison method was proven to be secure in [6].

In **Passkey Entry** mode, either one device displays a numeric passkey and the user has to enter it on the other device's keyboard, or the user has to enter the same 6-digit PIN on both devices. The underlying protocol is the same in both cases, except that in the first case, the number is chosen by the device and not by the user.

The 6-digit passkey is converted into a string $ra1, \ldots, ra20$ of 20 bits. As shown in Figure 1, the two pairing devices commit to and verify the passkey bit by bit in 20 rounds. In each round, the device A (device B) computes a 128 bit commitment value $Cai(Cbi)$ as follows:

$$\text{Device A: } Cai = f1(PKax, PKbx, Nai, rai)$$
$$\text{Device B: } Cbi = f1(PKbx, PKax, Nbi, rbi)$$

Here, Nai and Nbi are 128 bit random numbers generated by the devices A and B, respectively, and exchanged in the clear after the commitments Cai and Cbi (see Figure 1). The idea behind this gradual comparison of the passkey bits is that each device commits on the next passkey bit before it opens the bit by revealing Nai to the other party. Once in possession of Nai (Nbi), B (A) can compute Cai (Cbi) using rai (rbi) and thus indirectly verify that $rai = rbi$. Only if all comparisons so far succeeded, the next passkey bit is revealed to the other device. In particular, none of the parties has to disclose the complete passkey to the other device at once.

The function $f1$ is a publicly known keyed hash function which outputs an 128-bit commitment value (see [5] vol. 2

p. 898) specified as

$$f1(U, V, X, Z) = \textit{HMAC-SHA}256_X(U|V|Z)/2^{128}.$$

After authentication stage 1, authentication stage 2 is executed to confirm the DH key and authenticate the nonces, the random numbers, and the pairing capabilities supported by the devices.[1] A sends Ea to B, B sends Eb to A:

Device A: $Ea = f3(\text{DHkey}, Na, Nb, rb, capsA, A, B)$
Device B: $Eb = f3(\text{DHkey}, Nb, Na, ra, capsB, B, A)$,

where the function $f3$ is a publicly known function specified in ([5] vol. 2 p. 900) as the 128 most significant (leftmost) bits of the output of HMAC-SHA-256 keyed with the DH-key.

In the final phase of the pairing process, both parties calculate the link key LK with function $f2$ which is again based on HMAC-SHA-256 keyed with the DH-key:

$$LK = f2(DHkey, Na, Nb, A, B)$$

While the Passkey Entry method was designed to be secure against man-in-the-middle attack, this does not hold if the user uses the same passkey again after his first try to pair two devices failed. In the next section we detail this attack.

III. The Attack in Theory

In the following, we describe the theoretical man-in-the-middle attack on the Bluetooth Passkey Entry method. For the attack to work, the attacker only needs to be able to control the radio channel between two Bluetooth devices, and needs to be in possession of a device capable of communicating on the Bluetooth channels. The attacker does not require access to the memory of the devices or manipulation of the devices in any form.

A. Outline of the Attack

The attack targets the Secure Simple Pairing protocol in Bluetooth Version 2.1 and higher when Passkey Entry is used in authentication stage 1. The attack works if the PIN is reused in a second try to pair two devices after the first try has failed. This is naturally the case if one of the devices uses a fixed PIN. But even if this is not the case and the user choses the PIN and enters them on both devices, it seems plausible to assume that the user will reuse the same PIN when his first try to pair the two devices has failed just a few minutes ago.

The attacker eavesdrops on a SSP process between two legitimate devices as shown in Fig. 2 and records all messages exchanged during the initial DH-exchange and authentication stage 1.

After recording *Na20*, the attacker interrupts the SSP session by jamming the wireless channel. Because of a design weakness in Passkey Entry, the attacker is now able to calculate the PIN that was used in the legitimate pairing process.

When the devices initiate a new pairing process and the PIN is reused once, the attacker can use it to impersonate

[1]The capabilities are included to prevent so called downgrading attacks, where an attacker manipulates the exchanged capabilities such that the pairing process uses a less secure method (e.g. Just Works) than supported by both devices.

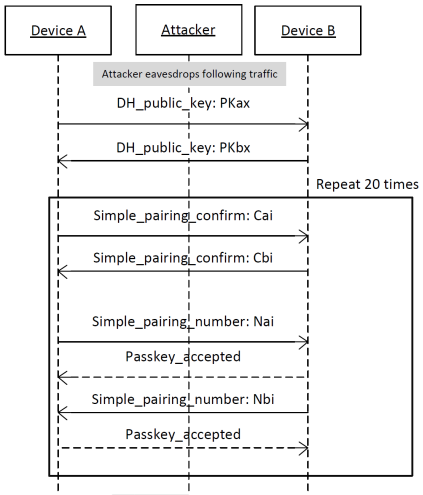


Fig. 2. Eavesdropping on Passkey Entry to obtain PIN

device A to device B and vice versa, thus acting as man-in-the-middle on the Diffie-Hellman key agreement, the attacker is able to negotiate link keys with each of the devices. The attacker is now able to forward the data sent between two Bluetooth devices such that they appear to function normally. The attacker can eavesdrop on the communication between the two devices, can manipulate the data exchanged, and insert new traffic. As the link key is reused on all future connections, the attacker is additionally able to impersonate one device to the other even in the absence of the second device.

In the following we detail the steps of this attack.

B. Obtaining the PIN

During the legitimate pairing process the attacker records the initial DH key exchange, i.e. *PKax* and *PKbx* and the Passkey Entry exchange, i.e. the twenty *Nai*/*Nbi* and *Cai*/*Cbi* values.

Recall that the commitments Cai and Cbi are calculated by the legitimate devices for every single bit rai of the PIN, as described in Section II-B:

Device A: $Cai = f1(PKax, PKbx, Nai, rai)$
Device B: $Cbi = f1(PKbx, PKax, Nbi, rbi)$

Where $f1$ is a keyed cryptographic hash function. All values in this equation are available to the attacker, except for the PIN bits rai and rbi. Therefore, the attacker can deduce rai and rbi by „brute forcing“ if they are 0 or 1 by calculating $f1$ on the known input and both potential values of the PIN bits. Thus, the attacker can easily determine the PIN as soon as he has received *Na20*.

C. Using the PIN

To profit from knowledge of the PIN, the attacker has to prevent that the legitimate pairing process succeeds. This can be done by jamming the communication channel after *Na20* was sent. Also, instead of jamming the connection, the attacker could manipulate *Na20* or *Nb20* on the air, so that the verification by B or A will fail. The attacker then has to initiate a new connection to the honest devices. As shown in

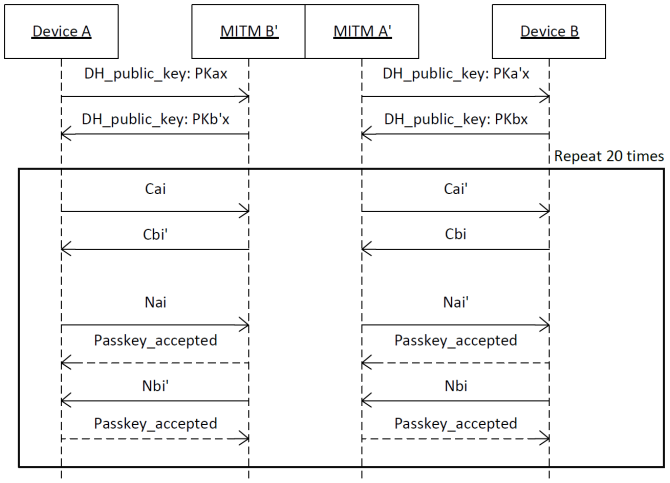


Fig. 3. Man-in-the-Middle Attack on Passkey Entry reusing the PIN

Fig. 3, the attacker can now impersonate A to B (noted A') and B to A (noted B') using his own Diffie-Hellman values and reusing the PIN. As a consequence the attacker will share a link key with device A and (another) link key with device B. Note that the devices will accept new DH values from the other devices as the Bluetooth standard explicitly states that „a device may, at any time, choose to discard its public-private key pair and generate a new one“ ([5] vol. 2, p. 889).

When the attack was successful, the attacker will forward the data from A to B and back. The attacker is then able to eavesdrop on and manipulate the data sent between the two Bluetooth devices, although they appear to function normally.

D. Countermeasures

When users are able to enter the PIN on both devices, the users can avoid this attack by not using the same PIN twice.

1) A Stopgap Solution: The Bluetooth standard explicitly demands freshly chosen nonces each time the protocol is repeated in the Numeric Comparison method ([5] vol. 2 p. 891), but fails to demand a fresh PIN each time the Passkey Entry method is run. In fact, the standard makes no requests about the PIN other that it has six digits („from 000000 to 999999“) ([5] vol. 1 p. 60) and that it may be generated by a pseudo-random number generator ([5] vol. 2 p. 857). There is no minimum length, and no guidelines how to chose the PIN.

We suggest that devices that display a PIN that the user has to enter on the other devices do not use a fixed PIN and always choose the PIN using a good random number generator. User generated PINs should be verified by devices to be at least 20 bit with 1 as the most significant bit, and devices should not accept the same PIN twice in Passkey Entry mode. The Bluetooth standard could easily be updated to include these guidelines without breaking compatibility to existing devices, unless they use a fixed PIN.

2) Fixing Authentication Stage 1 Passkey Entry: The DH key exchange is run directly before the vulnerable authentication stage 1. It results in an unauthentic key $DHkey$ shared between the parties which know the corresponding private DH parameters. So far, the $DHkey$ is only used in authentication stage 2 and the following link key calculation, but not in authentication stage 1 with Passkey Entry. We suggest that this $DHkey$ should be used in authentication phase 1 when Passkey Entry is used. Transmitting Nai/Nbi only encrypted with $DHkey$ would prevent the attack discussed in this paper. A passive observer could not decrypt Nai/Nbi, and thus could no longer determine the PIN in a passive attack.

Adding this change to a new Bluetooth specification would remove the reused-PIN vulnerability for new Bluetooth devices connecting to each other. However, guessing a fixed PIN would still be possible with an average of ten attempts, as described in [1]. While this is obviously much more suspicious for the user than two attempts, this is still a threat for automatically repairing devices. Therefore, the suggestions from Section III-D1 should still be followed.

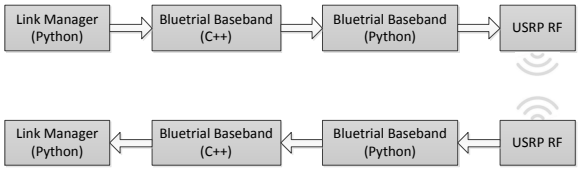


Fig. 4. Bluetrial Dataflow

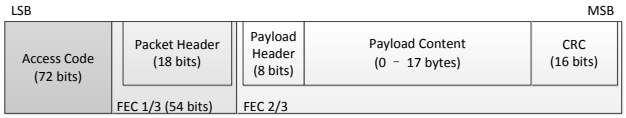


Fig. 5. DM1 Packet Format

E. Passkey Entry with Fixed PIN vs. Just Works

The security provided by Passkey Entry when used with a fixed PIN as the same as that of Just Works: Active attackers will always succeed, and the devices cannot prevent or detect the attack. None of these attacks has been implemented.

IV. Bluetrial: Bluetooth on USRP

While the attack described in the last section is well known since 2008, to the best of our knowledge, the attack has not never been implemented, such that a proof-of-concept is still missing. In this paper we close this gap. However, lacking an open source implementation of SSP and a truly programmable Bluetooth dongle, we implemented SSP in the GNU Radio software framework using the Universal Software Radio Peripheral (URSP) as hardware.

We use both the original USRP devices and newer USRP2 devices. USRP2 devices come with better analog-to-digital and digital-to-analog converters and a faster FPGA. In addition, USRP2 devices have a Gigabit interface, while USRP only offered a slower USB 2.0 interface. Despite the clearly superior specification of the USRP2, we used both types of devices as the USRP has been available longer and is generally considered to be less buggy. All tests were performed on both types of devices, so we also created a comparison of the USRP and USRP2 in the process. For clarity, the USRP will be called „USRP1“ in the following.

A. Testbed

Our „Bluetrial“ implementation is built on GNU Radio version 3.4. We did not use the latest UHD (USRP hardware driver) in Bluetrial, as it was first released when our project was already in development. The implementation involves two USRP1 and two USRP2 devices, and each one uses a single RFX2400 daughter board that is designed for the 2.4 GHz ISM frequency band used by Bluetooth.

B. Implementation

Our implementation covers the Bluetooth specification [5] parts required to execute the attack described in Section III. In particular, our implementation comprises all steps of the pairing process, and the link manager, and physical layer.

Note that due to the limitations of the USRP1 and USRP2 hardware used in the process, our Bluetrial devices work at a lower data rate and use a slower frequency hopping scheme than Bluetooth itself. Therefore, our Bluetrial devices are currently incompatible to off-the-shelf Bluetooth devices.

1) Bluetrial Baseband Layer: Bluetooth uses Gaussian Frequency Shift Keying to modulate and demodulate, which is not supported natively by GNU Radio. We used Gaussian Minimum Shift Keying instead, like the gr-bluetooth project [7], which showed that it works withreal Bluetooth packets.

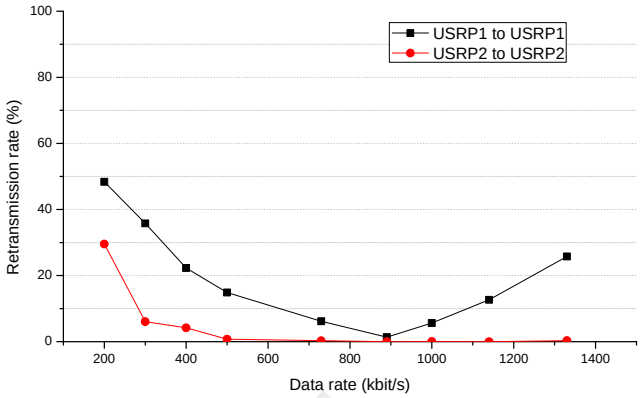
The DM1 packet shown in Fig. 5 is a commonly used data type in the Bluetooth link manager protocol, and we used it for all packets in Bluetrial. Its payload has a length between 1 and 18 bytes, which includes the 1 byte payload header. At the end of the payload, a 16 bit CRC is added. The whole DM1 packet payload is encoded with 2/3 Forward Error Correction, which is a (15,10) shortened Hamming code capable of correcting single bit errors and detecting double bit errors. The packet is sent together with 54-bit packet header.

Transmission errors are handled by the Link Manager Protocol (LMP) through the automatic retransmission request (ARQ) scheme. Lost and broken packets are sent repetitively until the sender receives an acknowledgment.

2) Secure Simple Pairing in Bluetrial: The implementation of SSP Passkey Entry is done according to a state machine for all the phases discussed in Section II-B, which are carried out on top of the Bluetrial baseband. In each phase, both LM devices alternatively send packets, e.g., the initiating LM starts the SSP by sending the IO request packets, then the other device responses to this request. Then, the LM waits for the initiating LM to send the P192 public key.

V. Evaluation of Bluetrial

In this section we evaluate our implementation and explore the limitations of the USRP hardware. We start by evaluating the transmission quality for different transmission rates of unidirectional communication between two USRPs. We then move on to the transmission quality of bidirectional communication for different transmission rates and different frequency hopping rates. Finally, we evaluate the retransmission rate for two USRP devices executing the SSP. Finally, in Section VI we evaluate the success rate of our attack implementation on Bluetooth Passkey Entry described in Section III to recover the PIN on our Bluetrial implementation.



kbit/s	200	300	400	500	730	900	1000	1140	1330
USRP1	48.4%	35.8%	22.3%	14.9%	6.2%	1.4%	5.6%	12.7%	25.8%
USRP2	29.6%	6.1%	4.2%	0.8%	0.3%	0.0%	0.1%	0.0%	0.3%

Fig. 6. Retransmission Rate, no Interleaved Transmission, Frequency Hopping disabled

To evaluate the quality, we use the transmission error rate. The automatic retransmission request (ARQ) in Bluetooth allows us to calculate the transmission error rate, because broken and lost packets will be resent until they are received correctly. We count the number of packets actually sent s, and compare it to the number of different packets intended to be transmitted t. In the following, we use:

$$\text{retransmission rate} = \frac{s-t}{t}$$

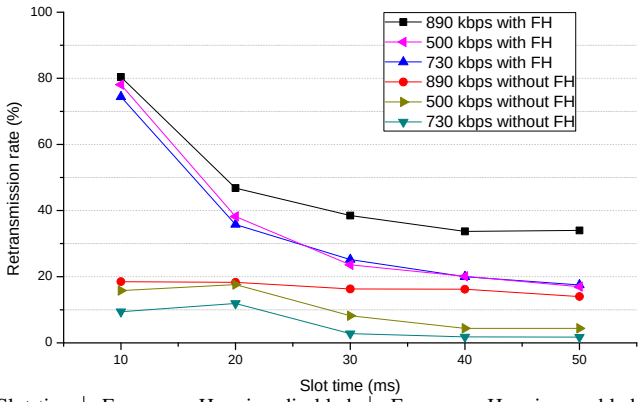
Even with a very high retransmission rate the data will eventually be transfered correctly. Dropped packets will always be counted like broken packets.

All parts described in the following were tested successfully according to Bluetooth test vectors to verify their correctness ([5] vol. 2 p. 915ff).

A. Unidirectional Communication

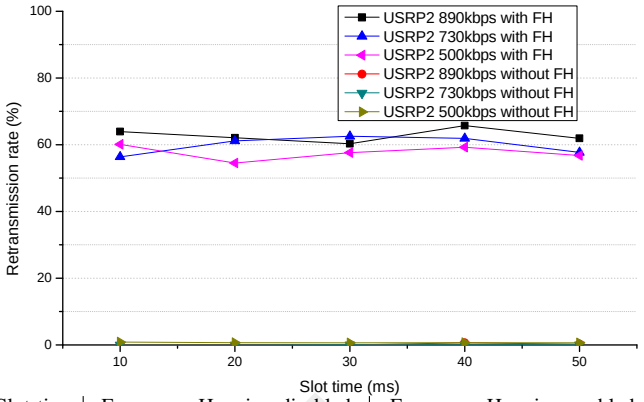
We start with testing the reliability of communication between two USRPs using Bluetrial. The Bluetrial kernel LMP generates DM1 packets, and sends them to the baseband physical layer. Finally, the packet is sent over the air through USRP and received by the other USRP (see Fig. 5). The DM1 packet type will be also used in SSP later. The packets sent to the other USRP device are considered to be correct if there is no error, or the forward error correction (FEC) is able to correct the error in the packets, so that the CRC verifies correctly. For each data rate, 20 rounds of result are averaged, with 250 packets sent each round.

The results are given in Fig. 6, which show that the USRP1 best supports a transfer speed around 900 kbit/s with a 1.4% retransmission rate. However, the retransmission rate grows above 15% below 500 kbit/s and above 1140 kbit/s. The URSP2 clearly shows better results, and the retransmission rate is below 1% at 500 kbit/s and above, which is a promising result. Using these results, the following tests focus on the



Slot time	Frequency Hopping disabled			Frequency Hopping enabled		
(ms)	500 kbps	730 kbps	890 kbps	500 kbps	730 kbps	890 kbps
10	15.8%	9.4%	18.5%	78.1%	74.4%	80.4%
20	17.6%	11.9%	18.3%	38.2%	35.8%	46.8%
30	8.2%	2.8%	16.3%	23.6%	25.2%	38.5%
40	4.4%	1.8%	16.2%	20.1%	20.0%	33.7%
50	4.4%	1.7%	14.0%	17.0%	17.5%	34.0%

Fig. 7. Retransmission Rate, Interleaved Transmission, two USRP1



Slot time	Frequency Hopping disabled			Frequency Hopping enabled		
(ms)	500 kbps	730 kbps	890 kbps	500 kbps	730 kbps	890 kbps
10	0.9%	0.0%	0.0%	60.1%	56.3%	63.9%
20	0.7%	0.0%	0.0%	54.5%	61.2%	62.1%
30	0.7%	0.1%	0.0%	57.6%	62.5%	60.3%
40	0.7%	0.0%	0.7%	59.3%	61.9%	65.7%
50	0.0%	0.5%	0.2%	56.8%	57.7%	61.9%

Fig. 8. Retransmission Rate, Interleaved Transmission, two USRP2

middle range of the data rate, which is from 400 to 900 kbit/s.

The real Bluetooth protocol uses both transmitting and receiving on both devices, so the previous result is only a lower bound on the retransmission rate of the communication between two USRPs using Bluetrial.

B. Bidirectional Communication and Frequency Hopping

In theses tests, both devices alternate in transmitting data, using a fixed time slot. The tests are executed with frequency hopping disabled at first, and then enabled. When frequency hopping is enabled, the devices hop every other time slot, just as in adaptive frequency hopping in Bluetooth. All result are computed on the average of 20 times runs, with 200 packets sent from both sides during each run. The result are shown in Fig. 7 and Fig. 8.

The results using the USRP1 show that a smaller time slot causes higher retransmission rate, especially when frequency hopping is enabled. Frequency hopping increases the retransmission rate, and even more so with high data rates. The results with USRP2 show a very low retransmission rate below 1% without frequency hopping, but a higher retransmission rate than the USRP1 with frequency hopping. Contrary to the USRP1, the USRP2 is not affected much by the different data rates and time slot lengths tested, both with and without frequency hopping.

Bluetrial could be compatible to Bluetooth (without EDR) in terms of data rate, which is 1 Mbit/s. However, there is a large gap between Bluetrial and Bluetooth with regard to the target time slot length of 625 μs.

C. Secure Simple Pairing

In this test, two USRP devices execute a SSP session using Passkey Entry. The test runs 10 times each with different data rates and different frequency hopping time slot lengths. The tests without frequency hopping use fixed time slots of 50 ms, and the ones with frequency hopping use fixed time slots of 20 ms and 50 ms, which are chosen according to the earlier test results. The result is shown in Fig. 9 and Fig. 10.

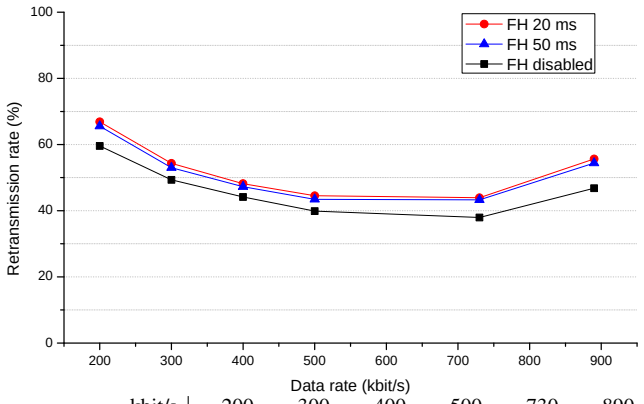
With the USRP1, the middle data range is again the most reliable, and frequency hopping increases the retransmission rate, although only by ca. 5% and not as much as in previous tests. Again, USRP2 shows better results than the USRP1 without frequency hopping with a retransmission rate below 3% for the higher data rates, and very high retransmission rates with frequency hopping greater that 63%.

VI. SSP Attack on Bluetrial

The attack against the SSP consists of two phases. The first phase is the eavesdropping on a SSP Passkey Entry pairing between two honest parties. The attacker collects all the packets and computes the PIN, which is called the passive attack phase. In the second phase, the attack becomes active by executing pairings with the two legitimate devices as a man-in-the-middle. Since the later MITM attack phase is exactly the same as two SSP pairing process at the same time, maybe with added directional jamming or desynchronization, this is not tested here. The only thing the attacker needs to make sure is that both pairing processes finish at the same time, so that the legitimate user does not notice the attack.

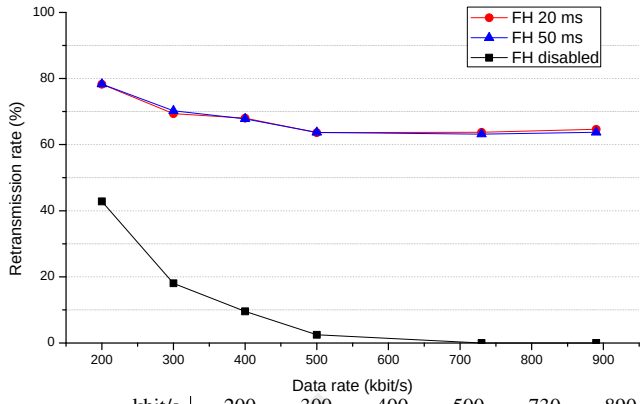
Eavesdropping on Passkey Entry is tested with a time slot length of 50 ms between two USRP1s executing a legitimate run of SSP Passkey Entry, and one USRP2 eavesdropping on the pairing process, which also computes the shared PIN between the legitimate devices.

The critical packets include the first two Diffie-Hellman key packets from both devices containing $PKax$ and $PKbx$, and the confirm and nonce values from either master device or slave device (Cai, Cbi, Nai, Nbi) during all 20 rounds. In total, 84 packets are sent. For each Passkey Entry round, it is sufficient to capture one pair of (Cai, Nai) or (Cbi, Nbi), since they both allow recovery of the i-th PIN bit. After the packet capture, the shared PIN is calculated as described



kbit/s	200	300	400	500	730	890
no Frequency Hop.	59.6%	49.3%	44.1%	39.9%	37.9%	46.8%
50 ms slot length	65.6%	53.0%	47.3%	43.4%	43.3%	54.4%
20 ms slot length	66.9%	54.3%	48.2%	44.5%	43.9%	55.6%

Fig. 9. Retransmission Rate in Secure Simple Pairing, two USRP1



kbit/s	200	300	400	500	730	890
no Frequency Hop.	42.8%	18.1%	9.6%	2.5%	0.0%	0.0%
50 ms slot length	78.3%	70.2%	67.8%	63.7%	63.2%	63.7%
20 ms slot length	78.3%	69.4%	68.0%	63.6%	63.7%	64.6%

Fig. 10. Retransmission Rate in Secure Simple Pairing, two USRP2

in Section III. We tested different data rates both with and without frequency hopping 20 times each.

The test worked, and the result are shown in Fig. 11. The eavesdropper captures >94% of the critical packets and succeeds in obtaining the PIN in at least 75% of the SSP pairings for speeds >200 kbit/s even with frequency hopping.

The test result shows a high packet capture rate and a high success rate for the Passkey recovery attack. A reason for this is the better performance of the USRP2 compared to the USRP1 without frequency hopping, which means that the USRP2 profits from retransmissions between the USRP1s. The attacker might not always receive all packets correctly the first time they are sent, and he cannot request retransmissions himself. With frequency hopping, the URSP1 performed better than the URSP2 (compare Fig. 9 and Fig. 10), and thus, the attacker's success rate is reduced, but still very high.

VII. Lessons Learned

A. Attack Evaluation

We have shown that the attack on Passkey Entry is not only theoretical, but works with reasonable reliability: 90% success rate at 890 kbit/s with 50 ms frequency hopping. While it cannot yet be applied to Bluetooth devices (1000 kbit/s, 625μs), it is only a matter of time until implementation issues will be overcome or better hardware becomes available, with Ubertooth [8] being an interesting candidate.

B. Practical Hurdles

Here is a short description of challenges we encountered during the development of Bluetrial:

Interdisciplinary Nature: The project requires a wide range of knowledge from physics, math, software development, and cryptography, and a lot of patience when developing software using beta software.

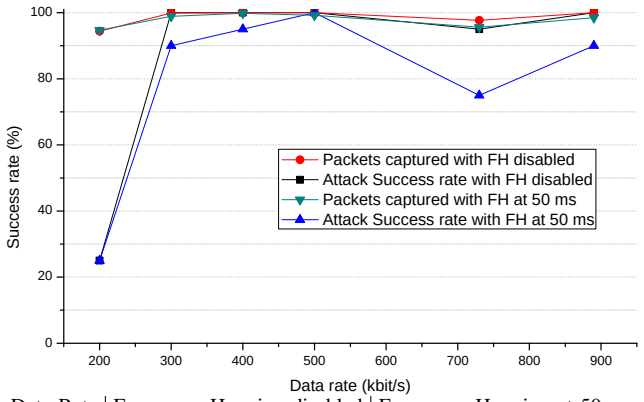
The documentation of **GNU Radio** is not as good as one would hope, and some functions are not implemented at all, e.g., antenna selection, which caused us to have poor results during earlier measurements.

The **USRP** platform is not a finished product and has its own difficulties. The internal clock is not very precise. While the results show the URSP2's improvement over the USRP1 especially regarding the URSP2's indifference to higher data rates, our results when using frequency hopping with the USRP2 are significantly worse than with the USRP1. This could be due to a bug in our code, in the library, or a hardware issue in design or just in our devices. The **FPGA** core in the USRP is especially difficult to use, with its unique programming paradigms centered around a hardware description language.

Vendors of consumer Bluetooth devices rarely advertise the Bluetooth version implemented and the security mechanisms supported. This made it hard to find Bluetooth 2.1+ enabled devices supporting Passkey Entry in the first place.

Frequency Hopping was introduced as a mechanism to make Bluetooth devices resistant to interference. However, frequency hopping is sometimes misunderstood as a security mechanism, which it is not. While it is a practical obstacle to manipulation of Bluetooth communication, the hopping pattern itself is public information. In fact, discoverable Bluetooth devices will send their internal clock value when queried, and together with the address of the device the internal clock value gives the hopping pattern.

Nevertheless the **retuning time** on the USRP platform is an obstacle in making our implementation compatible with real-world Bluetooth devices. A workaround to frequency hopping could be **jamming the 2.4 GHz band** so that only a low number of neighboring Bluetooth channels remains usable. After a short time, the Bluetooth devices will only use the remaining few channels. The URSP2 has a receive bandwidth of 25 MHz and is thus able to monitor 25 of the 79 Bluetooth channels at once. With regards to interface bottleneck, leaving even fewer channels usable seems advisable. So far, we have used the USRP2 as a jamming device, jamming 25 out the 79 Bluetooth channels at once. By alternating the jamming center frequency between channels 12, 27, and 42 every 100 ms, we were able to prevent Bluetooth devices from using any of



Data Rate	Frequency Hopping disabled		Frequency Hopping at 50 ms	
(kbit/s)	Success Rate	Capture Rate	Success Rate	Capture Rate
200	25.0%	94.4%	25.0%	94.8%
300	100.0%	99.9%	90.0%	98.9%
400	100.0%	100.0%	95.0%	99.8%
500	100.0%	100.0%	100.0%	99.2%
730	95.0%	97.7%	75.0%	95.6%
890	100.0%	100.0%	90.0%	98.5%

Fig. 11. Success Rate of a USRP2 Attacking two USRP1

the channels 0-55, which we could verify using the Frontline FTS4BT [9]. However, using a dedicated configurable jamming device would be much easier, have much lower cost, and block all but 1-4 channels. While jamming devices for the 2.4 GHz band are already available as consumer products, these are not configurable and mostly aim to disrupt all service.

Another obstacle to Bluetooth compatibility of our Bluetrial project is the **time synchronization** between the USRP and the target Bluetooth device.

Data Whitening in Bluetooth is also used as an optimization of the physical layer, and not as a security mechanism. The whitening „key“ depends on the internal clock value like the frequency hopping pattern, and this clock value will be sent by any discoverable device when queried.

VIII. Related Work

A. Attacks on Bluetooth

Bluetooth 1.1 was the first major release. Bluetooth 1.2 was an update, and current versions are still compatible to it. Bluetooth 2.0 added Enhanced Data Rate (EDR). All these versions use the same insecure pairing process, which is examined in [2], [3], [4].

Bluetooth 2.1 introduced a new security architecture called Secure Simple Pairing, so the old attacks do not work anymore. Lindell [1] described an attack on the Passkey Entry pairing method that we implemented in this paper. Lindell also proposed a fix of the protocol: Encrypting the whole authentication stage 1 using *DHkey*. No implementation was done. We suggest a different fix in Section III-D2.

The Numeric Comparison pairing method was proven secure in [6]. The other pairing modes do not offer any security.

B. Tools for Bluetooth Analysis

USB Protocol Analyzers (software or hardware) can be used to sniff on the traffic between a consumer Bluetooth USB dongle and the PC. However, it does not help in attacking Secure Simple Pairing, since the pairing itself is executed on the dongle and not in the driver on the computer. The same applies to analysis of a Bluetooth driver within the OS.

GR-Bluetooth [8] by Spill and Ossmann is a GNU Radio implementation of the Bluetooth baseband layer. It is intended for experimentation and teaching, but it does not include a complete software stack. Sniffing on packets is supported, but there is no frequency hopping. The software was not updated since two years. We did not reuse any code from this project.

The Frontline FTS4BT [9] is a commercial closed-source wireless analysis tool in USB dongle format that comes with a proprietary Windows software priced around $5000. It allows monitoring of Bluetooth channels and sniffing packets. Apparently, it is possible to manipulate certain cheap consumer Bluetooth dongles to make them work with the commercial wireless analysis software of Frontline [10].

Ubertooth One [8] is a $100 open-source LPC175x ARM Cortex-M3 processor-based USB dongle with a CC2591 RF front end and CC2400 wireless transceiver. It supports Bluetooth monitoring mode, which is the equivalent to Wi-Fi promiscuous mode, allowing the user to observe traffic and read data sent over the air. Ubertooth One supports basic data rate of 1 Mbit/s, and currently no packet injection or attacks on pairing, although it could be possible with later revisions.

IX. Conclusion

In this paper we showed that a man-in-the-middle attack on the Passkey Entry method in Bluetooth's Secure Simple Pairing process is practical. In particular, in our implementation the attack succeeded with a probability of 90%. While our Bluetrial implementation of the attack is currently not interoperable with off-the-shelf Bluetooth devices, this is only an issue of the radio capabilities offered by the USRP platform. It is only a matter of time until the speed of this platform will be improved, or a workaround is found, or new devices for Bluetooth analysis appear. The attack should therefore be taken seriously and the countermeasures suggested in this paper should be taken into account.

X. Acknowledgments

This work has been supported by the UMIC Research Centre, RWTH Aachen University.

References

[1] Andrew Y. Lindell. Attacks on the Pairing Protocol of Bluetooth v2.1. In *Blackhat USA*, 2008.

[2] Yaniv Shaked and Avishai Wool. Cracking the Bluetooth PIN. *MobiSys '05 Proceedings of the 3rd international conference on Mobile systems, applications, and services*, 2005.

[3] Markus Jakobsson and Susanne Wetzel. Security Weaknesses in Bluetooth. *CT-RSA 2001 Proceedings of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, 2001.

[4] Dennis Kügler. Man in the Middle Attacks on Bluetooth. *Financial Cryptography: 7th International Conference, FC 2003*, 2003.

[5] Bluetooth Special Interest Group. Specification of the Bluetooth System Version 2.1 + EDR, July 2007.

[6] Y. Lindell. Comparison-based key exchange and the security of the numeric comparison mode in bluetooth v2.1. In *The Cryptographers' Track at the RSA Conference*, 2009.

[7] Dominic Spill. Implementation of the Bluetooth stack for software defined radio, with a view to sniffing and injecting packets. Master's thesis, 2007.

[8] Project ubertooth. online, http://www.webcitation.org/662Sy58V2, 2012.

[9] Frontline FTS4BT. online, http://www.webcitation.org/66E6p6eGV, 2012.

[10] Max Moser. Busting the bluetooth myth - getting raw access. online, http://www.webcitation.org/662MPjVom, 2007.