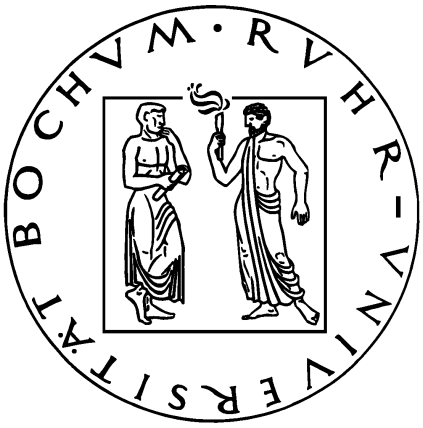
Bluetooth Security & Hacks

Andreas Becker

August 16, 2007

Seminararbeit
Ruhr-Universität Bochum



Chair for Communication Security
Prof. Dr.-Ing. Christof Paar

Supervisor: Dipl.-Ing. Tim Güneysu

Contents

1 Introduction

1.1 Historical Overview

Bluetooth is a technology for wireless connections of short range devices.
Originally invented by Ericsson, it has been called after Harald Blatand, a Danish Viking who lived during the 10th Century and unified Skandinavia.
Bluetooth itself was intented to unify a users devices to her wireless personal area network (WPAN). A Bluetooth WPAN is called *piconet* and may consist of mobile phones, PDAs, printers or personal computers.

In September 1998, the *Bluetooth Special Interest Group* (SIG) was founded with the objective of developing the Bluetooth wireless technology, as well as propagating the Bluetooth brand worldwide. Today, the Bluetooth SIG consists of more than 8000 members, which vary from music industry to telecommunication companies.

1.2 Bluetooth Basics

Bluetooth operates in the license-free ISM band (Industrial, Scientific and Medical Band) between 2.4 and 2.48 GHz.
For prevention of interferences with other devices working within ISM, such as baby phones or microwave ovens, Bluetooth makes use of a technique called *frequency hopping*. Therefore, the base band frequency is switched 1600 times per second to one of 79 frequency steps of 1 MHz band width within the ISM band.

Bluetooth is a connection oriented service. In order to connect two Bluetooth devices, one of them, normally the device initiating the connection, elevates to the master, leaving the second device as a slave (1.1 a).

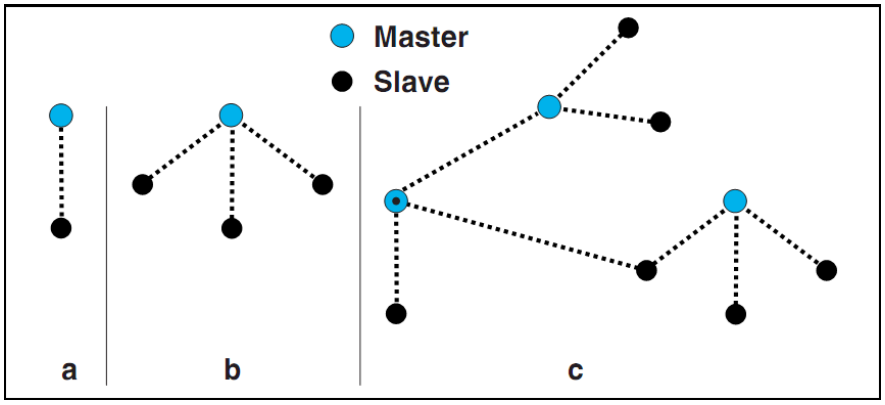


Figure 1.1: Piconet of two (a) or more (b) devices, scatternet (c)

In Bluetooth, connections with up to seven devices, which form a *piconet* are possible, where communication is led by the master device. Finally, a device is able to participate simultaneously in several piconets (in only one of them acting as the master). The resulting topology is called *scatternet* (1.1 c).

The maximum data rate in Bluetooth connections is 1 MBit/s in Version 1.2, while in the most recent version, 2.0+EDR (*enhanced data rate*) data rates of up to 3 MBit/s are reached. A master may reserve SCO (*synchronous connection oriented*) connections to up to three slaves in his piconet, using certain time slots. Those connections, granting symmetric data rates to both communication partners are used for voice transmission primarily.
Unused timeslots may now be used for exchange of point-to-multipoint packets between the master and all slaves in the piconet via ACL (*asynchronous connection less*) packets. These connections grant higher data rates in only one direction.
While three different device classes exist, with a rising power consumption from class I to class III in exchange for a larger operating distance, one of Bluetooths greatest advantages is a generally low power consumption.

Power Class	Max. Power Consumption	Max. Operating Range
1	100 mW (20 dBm)	ca. 100 m
2	2,5 mW (4 dBm)	ca. 20 m
3	1 mW (0 dBm)	ca. 10 m

For a secure communication via Bluetooth, the following security targets are defined:

- confidentiality
- (device) authentication
- (device) authorisation
- integrity

In order to accomplish those security targets, three possible modes of security are defined:

- Security Mode 1: No security efforts
- Security Mode 2: Service level security, applications have to implement needed cryptographic means
- Security Mode 3: Device level security (cryptographic means are implemented in LMP, independent of applications)

Security mode 1 obviously has to be avoided whenever possible. In mode 3, services make transparent use of a secure channel, which is normally established via the Link Managing Protocol (LMP). Therefore, a pairing process between two devices A and B has to be done, in order to establish a shared cryptographic secret for subsequent encryption or authentication.

In this seminar work, several attacks on weak implementations of certain protocols are presented. In contrast, one attack, *Cracking the Bluetooth PIN*, aims at the Bluetooth Security architecture itself (see Chapter 2.9), while *BlueJacking* has to be interpreted as an undesirable feature of the Bluetooth vCard service (see Chapter 2.6).

Beforehand, a rough overview of the most important services and protocols of the Bluetooth protocol stack is given, in order to establish the reader's understanding for the presented vulnerabilities.

1.3 Bluetooth Services

Bluetooth makes use of a protocol stack, which makes it simple to seperate application logic from physical data connections. In comparison to the *ISO/OSI reference model*, the protocol architecture of Bluetooth allows for straightforward implementation of existing network protocols like HTTP, FTP, etc.

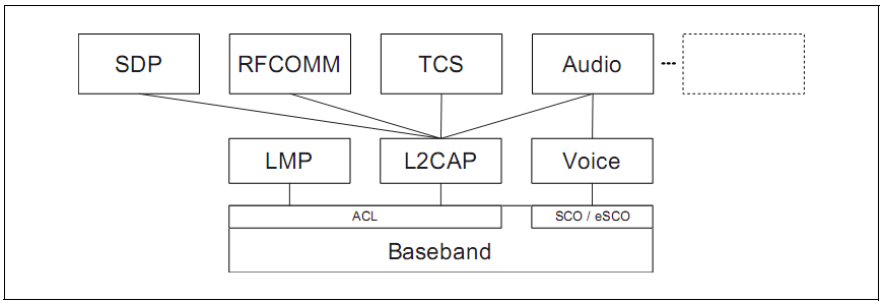


Figure 1.2: Bluetooth protocol stack

1.3.1 SDP - Service Discovery Protocol

Bluetooth is a technology, which is deployed in a dynamical environment. Devices may get out of range or even switched off, while new devices might become activated.

In order to detect services, provided by other devices, a protocol, which detects services makes sense. In Bluetooth, the *Service Discovery Protocol* is responsible for keeping track of services, provided within a device's operating range.

1.3.2 LMP - Link Managing Protocol

Especially interesting for further consideration is the *Link Managing Protocol* (LMP), as one of three possible security modes in Bluetooth is implemented in this layer.

Every Bluetooth device contains a Link Manager Unit, which keeps track of connected devices. Those Link Managers communicate via *protocol data units* (PDU), defined in LMP. New connections are established, using an inquiry routine (detecting device address), followed by a page command (call for a device with known device address), which have to be responded correctly by the opposite device.

Subsequently, devices may initiate the pairing process, where a link key, K_{AB} is established and the opposite device is stored in the "trusted devices" history. The pairing process is considered in detail in chapter 2.9 (*Cracking the Bluetooth PIN*), where a passive attack during the pairing of two devices is presented, in order to eavesdrop the used PIN.

1.3.3 L2CAP - Logical Link Control and Adaption Protocol

The *Logical Link Control and Adaption Protocol* (L2CAP) provides connection-oriented and connectionless data services to protocols in upper layers, while making use of ACL (asynchronous connectionless) packets for communication via the Baseband.

1.3.4 RFCOMM - Radio Frequency Communication

The *Radio Frequency Communication* (RFCOMM) provides emulated serial ports and makes use of the lower-level L2CAP protocol.

Up to 60 simultaneous connections to other Bluetooth devices are possible, called *RFCOMM channels*.

1.3.5 TCS - Telephony Control Protocol

The *Telephony Control Protocol* provides functionality to control telephony applications and makes use of L2CAP connections.

1.3.6 OBEX - Object Exchange Protocol

The *Object Exchange Protocol* (OBEX) provides services for the exchange of binary data objects. To initiate an OBEX session, an optional OBEX authentication is possible.

Therefore, a limited set of commands like PUT, GET or ABORT exist for easy file transfers, comparable to HTTP. In Bluetooth, OBEX is mainly used within the profiles *OBEX Object Push Profile* (OPP) and the *Synchronisation Profile* (SYNCH), which are described below.

1.4 Bluetooth Profiles

In Bluetooth, provided services are composed to a *Bluetooth Profile*. Bluetooth devices communicate via the profiles, that act as "interfaces".

For example, a Bluetooth-enabled mobile phone might make use of a Bluetooth headset, which implements the *Headset Profile* (HSP), which means that SCO (synchronous connection-oriented)-based audio connections and a subset of AT commands for accepting an incoming phone call, etc. are supported.

For further consideration, two Bluetooth profiles are especially interesting, concerning *BlueSnarfing* and *BlueBugging* attacks:

1.4.1 OBEX Object Push Profile (OPP)

The *Object Push Profile* (OPP) provides basic functions for exchange of binary objects, mainly used for vCards in Bluetooth.

vCard is a file format standard for electronic business cards. Since vCards are not worth being especially protected, no authorisation procedure is performed before OPP transactions. Supported OBEX commands are `connect`, `disconnect`, `put`, `get` and `abort`.

1.4.2 Synchronisation Profile (SYNCH)

The *Synchronisation Profile* (SYNCH) provides functions for exchange of *Personal Information Manager* (PIM) data and was adopted from the IrDA infrared specification.

In Bluetooth, especially private data, like the address book, calendar, etc. is sent using the SYNCH profile, thus certain security measures have to be setup in order to make use of SYNCH transactions, namely performing a pairing process to authenticate the opposite device, based on a link key.

1.5 Trusted Devices

The Bluetooth specification defines several possible relations between Bluetooth devices, mainly based on whether a link key has been established previously (cf. section 2.9).

A paired device may get explicitely marked as a *trusted device*, which means that e.g. no authentication based on the link key is required for that device, in order to access certain services.

A weak implementation of the trusted devices feature leads to the *HeloMoto* vulnerability, which ocurred on some Motorola mobile phones and is described in section 2.7.

2 Bluetooth Security

2.1 Attacks against Bluetooth - Introduction

In the last few years, with growing popularity of the Bluetooth wireless technology, also the interest in abusing (wireless) network connections has risen.
Often an adversary's goal is to eavesdrop personal data, to use third parties' devices for his own purpose, or just the drive of hacking into a computer system, PDA or cell phone.

Whatever an adversary's goal may be, at first it might be helpful, to gather some information on his potential victim. Is it a cell phone? Is it perhaps vulnerable to a known software vulnerability? Which services does the aimed device provide, which might get exploited within any further stages of the attack?
Such questions might get answered via some portion of social engineering, but the adversary has also access to a growing number of useful tools. The most important tools are introduced below.

2.2 Attacker's Tools

2.2.1 Generic Tools

Making use of services provided by the Bluetooth architecture itself might seem the easiest way to collect information about the victim. The *Service Discovery Protocol* (SDP) may grant some useful information to the adversary. Which "ports" are open with programs listening to them, waiting for example for OBEX requests, serving voice data and so on.
Therefore, the tools *BTScanner* and *hcitool* should be mentioned as frontends for the *bluez-utils* Bluetooth stack on Linux platforms, providing basic information about devices within operating range, such as device addresses, device classes and names. The most direct way to detect provided services via SDP is *sdptool*, another frontend for bluez-utils.

On Microsoft Windows platforms, the *BlueScanner* should be mentioned, collecting information like device type and provided services of detected devices. BlueScanner uses, in contrast to the console programs above, a graphical user interface.

All those tools share a significant problem: The attacker normally depends on a laptop, which might become noticed by a potential victim. One quite successful attempt to turn an adversary's mobile phone into a suitable auditing tool has been done with Blooover, an application, which is presented as follows.

2.2.2 Blooover

Blooover, a Java application for mobile phones, is developed by the trifinite group [tri07]. It is available for some mobile phones and requires J2ME (*Java 2 Platform, Micro Edition*). Blooover is able to do security audits on mobile phones, checking known vulnerabilities. Additionally, it is capable of executing a *BlueSnarf* and a limited *BlueBug* attack.

Those attacks are described within the next few chapters. Roughly, BlueSnarfing enables an adversary, to retrieve private data from the targeted mobile phone, such as pictures, the calendar or address book, while BlueBugging aims at taking remote control over a mobile phone. The latter one would potentially cause financial damage for the victim, thus it is not possible to write SMS or initiating phone calls with Blooover.

2.2.3 BackTrack

BackTrack, a Slax-based Linux distribution [bac07], available as a live-cd goes one step further, providing several more complex automated attacks on Bluetooth devices.

Figure 2.1: BackTrack livecd with fluxbox interface

The BackTrack disc is packed with a large amount of security-related software, ranging from network analysis tools like *ettercap* and *wireshark* to a series of (partly) automated attacks, including attacks on Bluetooth devices, such as BlueSnarf or BlueBug.

The most relevant of those attacks and/or vulnerabilities of Bluetooth or insecure implementations will be presented, closing with a conclusion on Bluetooth security.

2.2.4 BTCrack

BTCrack [nru07] has been published at 23C3, a German hacker congress. It serves a graphical interface for a passive attack, which aims at eavesdropping messages in a pairing process between two devices, in order to retreive the Bluetooth PIN and generated link key.

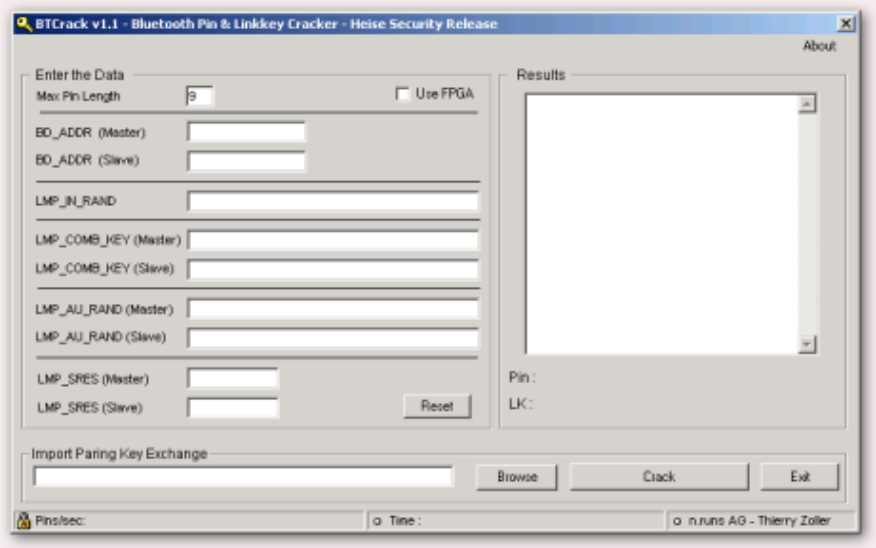


Figure 2.2: BTCrack interface

An adversary is able to enter previously discovered Bluetooth device addresses, in order to enforce re-pairing between the two target devices. Subsequently, the whole pairing communication is eavesdropped, allowing the program to do an exhaustive search on the Bluetooth PIN used within the pairing process. The underlying attack is described in chapter 2.9: *Cracking the Bluetooth PIN*.

2.3 BlueSnarf

The *BlueSnarf* attack exploits a weak Bluetooth implementation on some mobile phones, especially the implementation of *OBEX* (object exchange protocol) which is generally used by the OBEX PUSH profile, as by the Synchronisation Profile. Normally, no authentication or authorisation is required in OPP, as only vCards are transferred through OPP, if implemented correctly.

The vulnerability in insecure implementations on certain mobile phones now provides an OPP and also a SYNCH profile, which both have access to the same OBEX stack, which maps requests through the secured SYNCH profile and those through the OPP profile to the same filesystem. A possible attack may be done as follows.

At first, the adversary connects to an OBEX push profile, which he might have detected within a previously done SDP scan. Now, in insecure OBEX implementations, he is able to execute a successful OBEX GET request for known filenames, like telecom/pb.vcf or whatever the adversary might expect to find on the target device. This results in a hidden download of private data.

As for the use of the OPP, no pairing process with the adversary's device is necessary, the user is not aware of being spied out via Bluetooth. Combined with the advantages of a *long distance attack*, the adversary does not even have to be within range of sight to his target. This physical extension of the effective operating range of a Bluetooth device makes all of the attacks presented in this seminar work, much more effective.

Instructions to modify a Bluetooth Dongle in such a way to reach operating ranges around several hundred meters are available on the internet. The trifinite group [tri07] has succesfully done a *long distance snarf* over a distance of 1.78 km.

A list of mobile phones vulnerable to BlueSnarf attacks is available in [the07].

2.4 BlueSnarf++

BlueSnarf++ is an enhancement of the previously presented *BlueSnarf* attack, where the adversary had to deal with the OBEX push daemon in order to gain private data from the target device. In the BlueSnarf++ attack, the adversary instead connects to an OBEX FTP server, which is normally used for easier file transfers and, to the adversary's pleasure, is available on some of the vulnerable

mobile phones.

To fix the security vulnerabilities concerning BlueSnarf, respectively BlueSnarf++, the user is normally dependent on the applicable firmware update from the manufacturer (mainly affected were mobile phones by Nokia and Sony Ericsson who, today, cooperate with the trifinite group in security concerns).

2.5 BlueBug

For a better understanding of how the *BlueBug* attack works, a short introduction into the *RFCOMM* protocol will be given.

RFCOMM is a protocol on top of L2CAP within the Bluetooth protocol stack, which emulates serial RS-232 interfaces via Bluetooth connections. Up to 60 connections via RFCOMM may get established simultaneously for one device, called *RFCOMM channels* . While L2CAP and, on a lower level, the base band, are responsible for establishing a synchronous or asynchronous data connection, RFCOMM emulates a "virtual" serial connection between two communication endpoints.

Now, what the adversary needs to know of a vulnerable device [the07], is the Bluetooth Device Address, *BD_ADDR*. The adversary connects to RFCOMM–channel 17, where vulnerable mobile phones provide an open backdoor, in the form of an AT-parser which requires no authentication procedure. AT commands are a powerful tool for remote control of communication devices, like analog or ISDN modems, or, like in this case, mobile phones with a Bluetooth interface.

The adversary is able to execute most tasks a normal user of the mobile phone were able to do, like initiating phone calls (especially numbers like 0190..., which could be of economic interest for the adversary), reading or writing SMS or gathering the victim's private data.

On a linux computer, a session for reading a single address book entry from a mobile phone might look like this (cf. [HW06]):

```
# scan for bluetooth devices:

oscar@darkside $ hcitool scan
Scanning ...
00:0E:6D:10:1D:B6   Nokia 6310i
00:05:7A:01:A3:80   Airbus A380
```

```
00:06:6E:21:69:C2   Bluespoon AX
00:0F:DE:6C:61:04   T610


# bind channel 17 of target device to /dev/rfcomm42:

oscar@darkside $ rfcomm bind 42 00:0E:6D:10:1D:B6 17


# connect to AT terminal via, for example, cu:

oscar@darkside $ cu -l /dev/rfcomm42
Connected.
AT+CPBS="ME"
OK
AT+CPBR=1
+CPBR: 1,"",,"Paris Hilton"
OK
~.
Disconnected.
```

This example was only presented to show, what simple set of commands on the console answer an adversary's purpose of retrieving private data or executing any AT commands on a vulnerable mobile phone. The bounds of BlueBug are only limited by the adversary's creativity and the capability of the supported set of AT commands on the target device.

An overview of AT commands for GSM devices can be found in [Tra07]. The solution to RFCOMM-based attacks is, as in most cases, a firmware update fixing the insecure RFCOMM-channel.

2.6 BlueJacking

BlueJacking is not an attack against Bluetooth or any implementation in terms of breaking into another device, stealing private data or otherwise dealing damage to the "victim". Instead, it uses the ability of Bluetooth mobile phones to send so-called vCards, a file format, which is intended to exchange personal information in the form of an electronic business card.

Now it has come quite into fashion to abuse this mechanic to send—free of charge—messages via Bluetooth, such as "You were BlueJacked!". Users who

are not familiar with BlueJacking might draw the conclusion, their mobile phone were infested with a mobile phone virus, or they were followed by a stalker.

Users who feel disturbed by the way, primarily teenagers, abuse the vCard service for BlueJacking, should turn off Bluetooth connectivity on their mobile phones, only reactivating it, whenever needed.

2.7 HeloMoto

HeloMoto is a combination of BlueSnarf and BlueBug attacks, exploiting a flawed implementation of "trusted devices" in some Motorola mobile phones, which led to the attack's name.
At first, the adversary connects to an OBEX push profile, as it is done in BlueSnarf. If there is no vulnerable implementation of OBEX that would allow a BlueSnarf attack, HeloMoto makes use of the "trusted devices" feature, defined in Bluetooth.

The adversary now attempts to send a vCard to the target device, as it is done in BlueJacking and immediately cancels the request. In consequence of the HeloMoto vulnerability, his device remains within the "trusted devices" history, while the owner of the target device is not aware of being attacked. Finally he uses the status of a "trusted device" to execute AT commands, as it is done in BlueBugging.

Again, owners of a vulnerable device depend on a firmware update by Motorola, or otherwise should simply deactivate Bluetooth.

2.8 BlueSmack

BlueSmack is a Denial of Service (DoS) attack, hence it is directed at the availability of a Bluetooth device. The attack is done similarly to the "Ping of Death" against IP-based devices. The adversary sends an L2CAP echo request (ping) of large size, approximately 600 bytes to a Bluetooth device with limited hardware resources.

Those devices (especially known for such a behaviour is the iPaq) reserve an input buffer of fixed length (around 600 bytes). When receiving such a malicious ping request, the input buffer overflows, which normally leads to a seqmentation fault and, by this, to the immediate knock-out of the target device.

On a linux computer, this can be done simply using the bluez-utils' `l2ping` command with `-s <num>` option, defining the packet length.

2.9 Cracking the Bluetooth PIN

Without exception, the previous attacks were focused on certain insecure Bluetooth implementations which could usually been fixed by firmware update. The following attack instead exploits the Bluetooth security architecture itself and was presented by Yaniv Shaked and Avishai Wool in [SW05].

Roughly spoken, the attack aims at eavesdropping the whole communication through the pairing process between two Bluetooth devices, in order to extract the initialisation key, K_{init}, using it for an exhaustive search on the Bluetooth PIN.

Generally, Bluetooth aspires confidentiality and authenticity. We will take a closer look at the cryptographic functions used to accomplish those security targets within the Link Managing Protocol, while discussing the attack. In order to establish a secure Bluetooth connection between two devices, an inquiry routine has to detect the opposite's Bluetooth device address, BD_ADDR.

Subsequently, in both devices have to be entered the same Bluetooth PIN, depending on the device type. Some devices with limited memory or user interface capability have a fixed PIN, which is in most cases 0000, which will be discussed later on. Important for the establishment of a connection is, that at least one device must have a variable PIN, that has eventually to be set to the second device's fixed PIN.

Now everything is set up correctly for the following steps within the pairing process:

1. Creation of an initialization key, K_{init}, which is used to confidentially exchange random values
2. Creation of a link key K_{ab}, which is a function of the input of A and B; discarding of K_{init}
3. Mutual Authentication via simple challenge-response scheme, based on K_{ab}

Note: All algorithms used during the pairing process, namely E_{22}, E_{21} and E_1 are based on the *SAFER+* block cipher. SAFER+ operates with a block size of

128 bits and three different key lengths: 128, 192 or 256 bits. Bluetooth uses the SAFER+ with 128 bit key length.

As the inner design of SAFER+, respectively the derived Bluetooth algorithms mentioned above, are not essentially important for the following attack, they are used as "black boxes" in this seminar work. For further information see [SW05].

2.9.1 Creation of K_{init}

We assume, that the same PIN has been entered into both Bluetooth devices correctly. Now the Master device A (usually the device which initiated the session) chooses a pseudorandom number of 128 bits length (IN_RAND) and sends it to the Slave B.
Subsequently the K_{init} key is created on both devices, using the E_{22} algorithm:

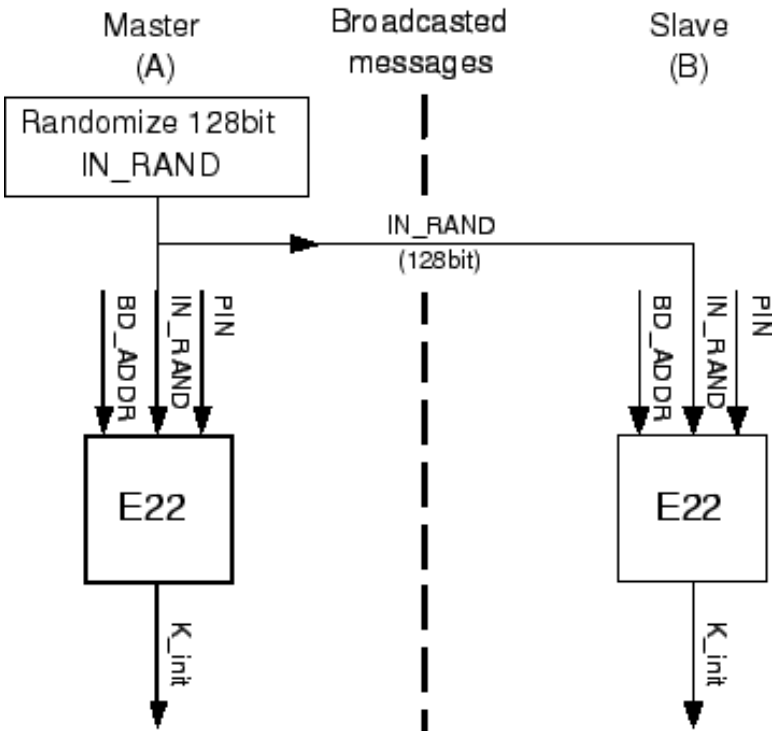


Figure 2.3: Generation of K_{init} using E_{22}

The K_{init} key generated by E_{22} is only used in the following to confidentially exchange random values. Those values are used to agree on a link key, which is a function of inputs from both devices, A and B. After the creation of K_{ab}, the link key, K_{init} is discarded, leaving K_{ab} as the shared secret for subsequent communication.

2.9.2 Creation of K_{ab}

After exchanging the initialization key, K_{init}, the devices agree on the link key K_{ab}. Both devices choose a 128 bit random number, LK_RAND_A, respectively LK_RAND_B, xor it bitwise with K_{init} and send it to their communication partner.

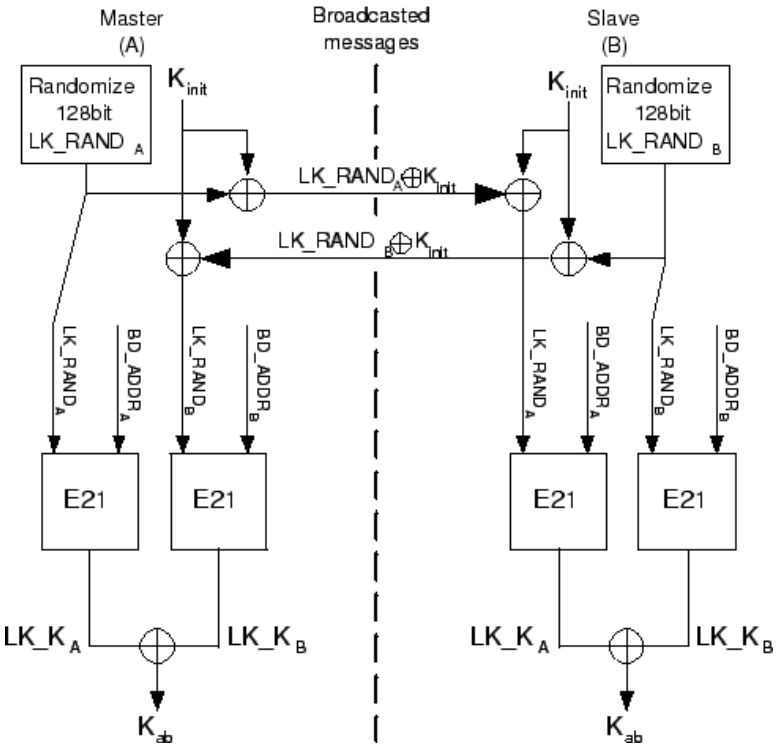


Figure 2.4: Generation of K_{ab}using E_{21}

Using the self-inverse xor function again, the random values are decrypted and used to generate the link key using the E_{21} algorithm. Now both devices share a link key and $K_{init}$is discarded. Finally a mutual (device) authentication, based on K_{ab} is done.

2.9.3 Mutual authentication

In order to authenticate the opposite device, a mutual authentication based on a challenge-response scheme is performed. Therefore, one of the devices is the verifier A, leaving the other device as the Claimant B. Now, B claims to know the corresponding link key which A likes to verify.

The verifier A choses a 128 bit random number AU_RAND_A as a challenge and

sends it to B, who responds with the 32 bit word $SRES'_A$, using AU_RAND_A, BD_ADDR_B and K_{ab} as input values for the E_1 algorithm. Then, A computes a value $SRES_A$, using the same input for E_1, as B is expected to take.

If $SRES'_A$ and $SRES_A$ are different, A immediately halts the session, leaving B the possibility to authenticate again, after a certain amount of time. After each unsuccessful attempt, this amount of time is increased exponentially, which is done to prevent brute force attacks. Otherwise, roles for verifier and claimant are switched and the entire process is repeated.

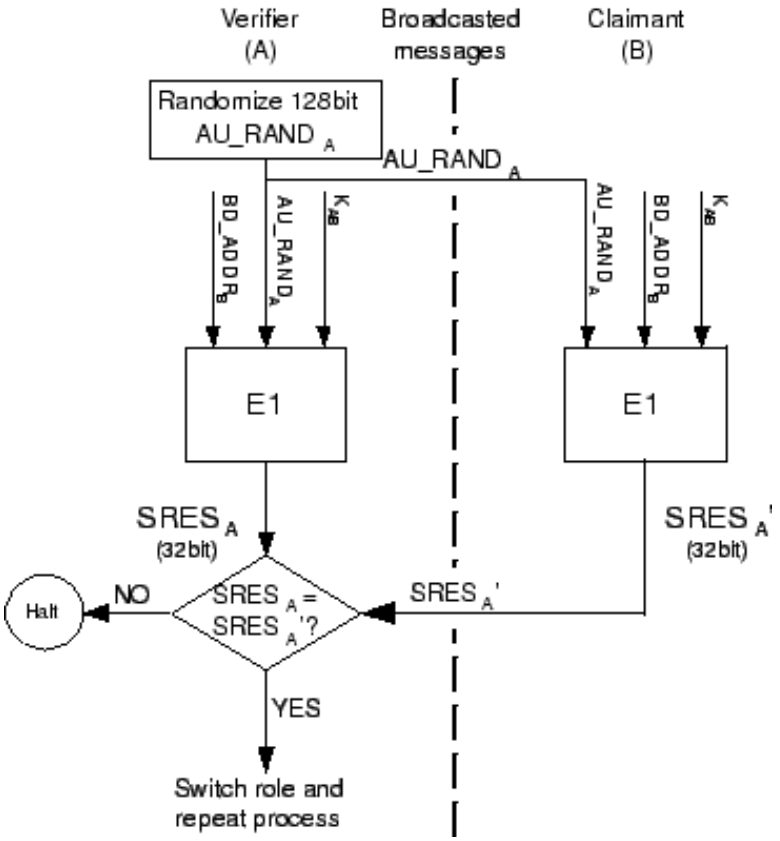


Figure 2.5: Mutual authentication using E_1

2.9.4 Attacking the pairing process

In [SW05], the following passive attack against the pairing process between two Bluetooth devices is documented. Let us assume the adversary was able to eavesdrop the whole communication within the pairing process between two devices A and B:

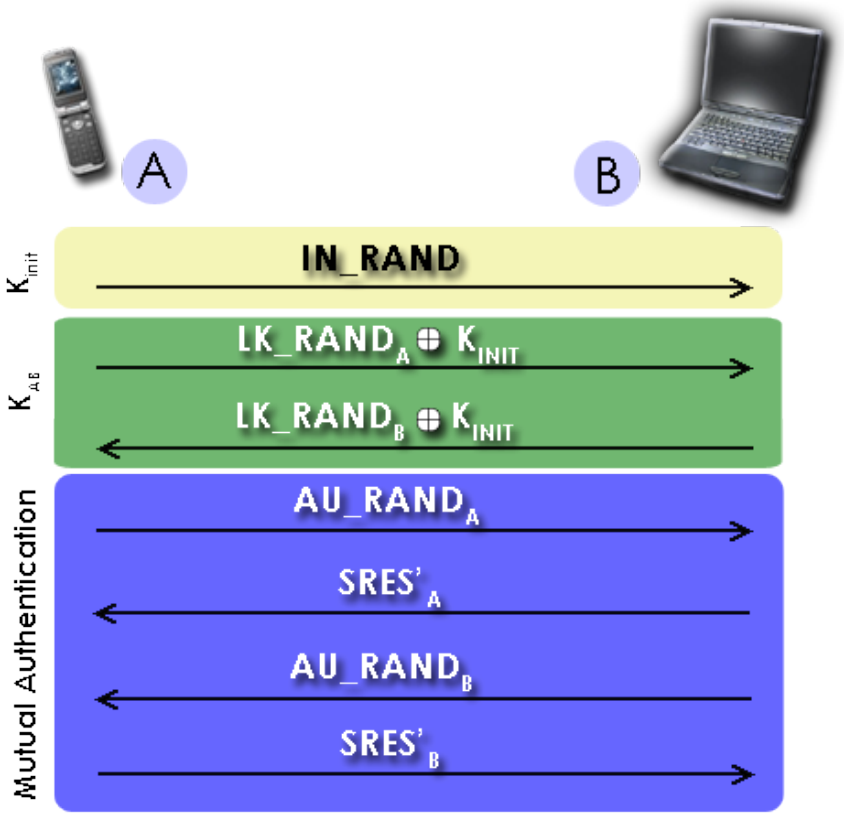


Figure 2.6: Eavesdropping of the pairing process

Now the adversary is able to do an exhaustive search within the space of possible Bluetooth PINs. Knowing IN_RAND and BD_ADDR, the adversary feeds E_{22} with those values, together with the PIN "candidates", chosen by his brute force algorithm, receiving a hypothesis for K_{init}.

Subsequently, the adversary decrypts the first two messages within the mutual authentication process, resulting in a hypothesis for LK_RAND_A and LK_RAND_B. Due to LK_RAND_A and LK_RAND_B being the only secret information used in E_{21} to calculate the link key, the adversary computes a hypothesis for K_{ab}.

Now the adversary is able to use the last few messages to prove, whether his hypothesis for K_{ab} is correct or not. He feeds E_1 with the challenges AU_RAND_A, respectively AU_RAND_B and compares his result with the corresponding $SRES_A$ or $SRES_B$ message.

This process is repeatedly done until the correct Bluetooth PIN used by A and B is found. The entire attack looks like this:

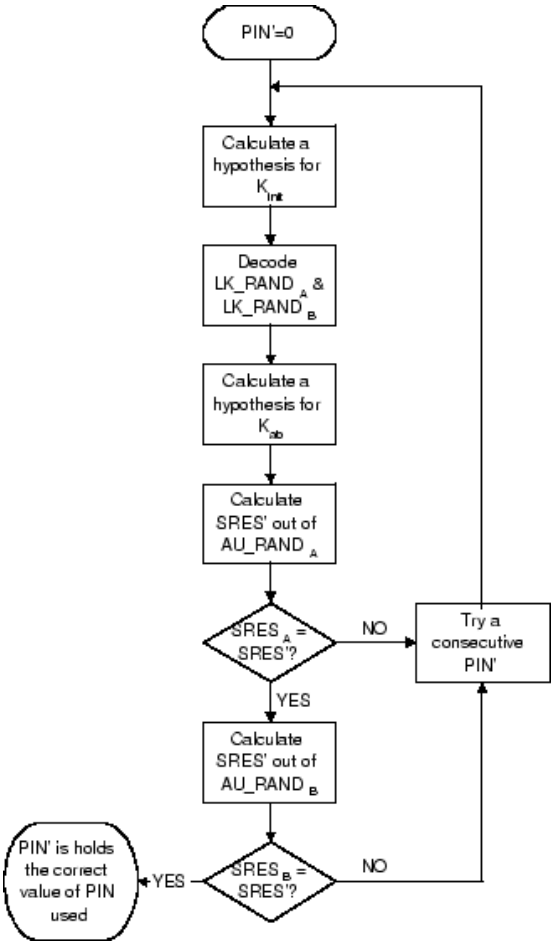


Figure 2.7: Cracking the Bluetooth PIN - algorithm

Note that this attack is more powerful than it might appear at a first glance. While generally the pairing procedure for two devices is not frequently repeated, Bluetooth grants the possibility to "forget" a link key, due to limited memory. By this, the adversary is able to enforce a repairing process, which still requires to inject a specific message at a certain point in the protocol run, what needs a custom device in general.

Additionally, the attack is rather efficient. Due to some algebraic optimisation of

needed computations, cracking a 4-digit Bluetooth PIN with a Pentium IV 3GHz takes approximately 63 milliseconds.

Since this attack is aiming directly at the Bluetooth security architecture itself, instead of against a vulnerable implementation, it is not possible to fix this problem through a simple firmware update. Instead, the user should be aware of the possible risks when being prompted to reenter the PIN for an existing connection and, whenever possible, use a PIN of considerably more than 4 digits.

Finally it should be mentioned, that devices using a fixed PIN(which is generally 0000) are - due to obious reasons - an open door for any adversary. This holds for devices like headsets, printers, etc. In this case, the user should be aware of the fact that no effective method for providing confidentiality is applied to such connections, because further authentication and encryption is based on the insecure link key K_{ab}, or even a fixed unit key.

2.10 Conclusion on Bluetooth Security

The most important practical attacks against Bluetooth have been presented in this seminar work. Those attacks either exploit the Bluetooth architecture itself, a short PIN, or a weak implementation. While insecure software on embedded devices generally leads to compromised private data, gaining remote control over a cell phone, or, at least, a simple knockout due to a DoS attack like BlueSmack, the power of an attack grows with the complexity of the target.

When a buffer overflow or similar vulnerabilities occur in Bluetooth protocol stacks or application software for a personal computer or MAC, the adversary might even be able to execute arbitrary code (cf. [pen04]).

Concerning the Bluetooth specification [SIG04], some security-by-obscurity elements have been promoted, as things like frequency hopping or the limitation to short operating ranges have been advertised as security features. When considering previous attacks like the long distance snarf or successful attempts of mounting an embedded bluetooth device on a modified rifle, for large distance attacks, those features seem to create a false awareness of security.

Additionally, the key management in Bluetooth seems to draw its security partly from the big issues of perfoming practical attacks. The link key, which is used to authenticate an opposite device and to compute encryption keys, is promoted as a shared secret of 128 bits length, which is not true. Since it is based on a Bluetooth PIN, the effective key space of K_{ab} can generally be reduced to 10^4

(approximately 13 to 14 bits), which essentially is, what Shaked and Wool did in their attempt to attack the pairing process.

Finally, the weakest link within the system "Bluetooth" often seems to be the user, who is not always aware of potential security risks and chooses short PINs or permanently leaves Bluetooth connectivity switched on.

Combined with the fact that auditing tools and tools for automated attacks on Bluetooth devices get improved steadily, Bluetooth seems to be an interesting issue in the future, especially, if Bluetooth will go on receiving an increasing popularity.

List of Figures

Bibliography

[bac07] BackTrack, Slax-based Linux Distribution for Security Audits, 2007. http://www.remote-exploit.org/backtrack_download.html.

[Ho04] Heise online. Sicherheitsluecke gefaehrdet Bluetooth–PCs, 2004. http://www.heise.de/security/result.xhtml?url=/security/news/meldung/50068.

[HW06] Marcel Holtmann and Christoph Wegener. Oft erlauben Sicherheitsluecken einfachen Datenklau via Bluetooth, 2006. http://www.linuxmagazin.de/heft_abo/ausgaben/2006/02/zahnstatus/(kategorie)/368.

[Mul01] Nathan J. Muller. *Bluetooth*. mitp, 2001. ISBN: 3826607384.

[nru07] n.runs AG - security tools and security advisories, 2007. http://www.nruns.com/security_tools.php.

[pen04] Widcomm bluetooth connectivity software buffer overflows, 2004. http://www.pentest.co.uk/documents/ptl-2004-03.html.

[SIG04] Bluetooth SIG. Specification of the Bluetooth System, 2004. http://www.bluetooth.com/Bluetooth/Learn/Technology/Specifications/.

[SW05] Yaniv Shaked and Avishai Wool. Cracking the Bluetooth PIN, 2005. http://www.eng.tau.ac.il/~yash/shaked-wool-mobisys05/.

[the07] The Bunker, 2007. http://thebunker.net/resources/bluetooth.

[Tra07] Alexander Traud. AT commands for GSM, 2007. http://www.traud.de/gsm/index.html.

[tri07] The trifinite group, 2007. http://www.trifinite.org.

[wik07] Bluetooth — Wikipedia, the free Encyclopedia, 2007. http://en.wikipedia.org/w/index.php?title=Bluetooth.