

forward

a future of reliable wireless ad hoc networks of roaming devices

Analytical Assessment of Bluetooth Security Mechanisms

10th September 2004

Record of Changes

Date	Version	Comment
10-09-2004	1.1	Document made public
13-01-2004	1.0	First Issue

Authorisation

Dr. Sadie Creese
FORWARD Steering Committee Member

Date

Authors

Kamran Azeem, QinetiQ, k.azeem@eris.qinetiq.com

Paul Beechey, whilst at QinetiQ

Sadie Creese, QinetiQ, s.creese@eris.qinetiq.com

Michael Goldsmith, Formal Systems (Europe) Ltd, michael@fsel.com

Richard Harrison, QinetiQ, r.harrison@eris.qinetiq.com

Alan Hood, QinetiQ, a.hood@eris.qinetiq.com

Clive Pygott, QinetiQ, c.pygott@eris.qinetiq.com

Jon Roach, QinetiQ, j.roach@eris.qinetiq.com

William Simmonds, QinetiQ, w.simmonds@eris.qinetiq.com

Paul Whittaker, Formal Systems (Europe) Ltd, paulw@fsel.com

Executive summary

This report forms deliverable D4 of the FORWARD project. It is the first deliverable of Work Package 2 *Securing Bluetooth*.

The purpose of Work Package 2 is to assess the security mechanisms present in the Bluetooth v1.2 standard, as described in [Blu03] and [Geh02], and where necessary develop means of protection for systems utilising Bluetooth as a communications medium. The research is split into three distinct phases:

- Phase One: Analytical assessment of Bluetooth security mechanisms present in Bluetooth v1.2.
- Phase Two: Practical investigation of any issues identified in Phase One.
- Phase Three: Development of mechanisms for protection of systems utilising Bluetooth (if appropriate).

This deliverable reports on Phase One of the research, analytical assessment of the security mechanisms provided by the Bluetooth v1.2 standard. Our contribution being both the results we obtain, and the methodology we have developed for analysing security of systems utilising wireless communications which is generic and applicable to other similiar media.

Our approach is to fuse together the knowledge and intuition possessed by the QinetiQ penetration testing team with the expertise of formal analysts. This enables us to validate the results of the formal analysis with practical testing in a novel approach not formally done within the formal methods community. In addition, we begin our development of a graphical methodology for documenting security cases. This leverages a technique called Goal Structured Notation (GSN), currently utilised by the military for documenting safety cases for critical systems.

We considered a sizeable, but not complete, subset of the possible variants of the bonding procedure. These include most of what could be construed as the 'normal usages' of the procedure. Of these, no definite protocol attacks were found when the Intruder possesses no prior information. We have shown that the Intruder may be able to 'break' or 'compromise' the security when h/she is assumed to have some capability above that which he would routinely expect to have in practice (as described in section 4.2.5). These additional capabilities included:

- **guessing the shared *pin*.**
- **guessing either *lkrandA* or *lkrandB* (as used in the link key).**
- **spoofing the identity of a bluetooth device.**

In the first two capabilities, the Intruder is assumed to be able to make an effective guess on the shared *pin* or random numbers and then verify his guess (cf. question **Q1b**). In the third capability, 'spoofing' concerns the apparent identity of a device as opposed to the origin of a given packet (cf. question **Q3**). This might correspond in the wired world to an attack on a DNS server in which the name to address mapping is altered. The first two capabilities lead to obvious and simple failures of secrecy and authentication while the third leads to more subtle issues relating to authentication.

The outputs of the formal modelling are a series of events which describe successful attacks. These will be validated in the next phase of our research.

A software demonstrator of the CSP models is available from the project website: www.forward-project.org.uk.

Contents

- 1 Introduction** **1**
 - 1.1 Purpose 1
 - 1.2 Background 1
 - 1.3 Structure of the Document 2

- 2 Bluetooth – Potential Vectors for Attack** **3**
 - 2.1 Introduction 3
 - 2.2 Motivating Questions 3
 - 2.3 Attack Vectors 3
 - 2.4 Mechanisms to be compromised 4
 - 2.5 Answer to attack questions 6

- 3 Bluetooth Security Case** **8**
 - 3.1 Introduction 8
 - 3.2 Access 9
 - 3.3 Confidentiality 10
 - 3.4 Availability 11
 - 3.5 Integrity 13
 - 3.6 Bluetooth Security Mechanisms requiring investigation 13

- 4 Verifying the Bluetooth Link-Layer security using Casper** **16**
 - 4.1 Introduction 16
 - 4.2 CSP, FDR, Casper and SPL 16
 - 4.3 Caspose 23
 - 4.4 SPL descriptions of the individual stages of the initialization procedure 25
 - 4.5 Results of analysis 29
 - 4.6 Bluetooth-usage scenario 38
 - 4.7 Summary of analysis results 40
 - 4.8 Conclusions on modelling approach 40

- 5 Conclusions** **43**

- 6 Acknowledgments** **44**

- References** **44**

List of figures

1	The Bluetooth v1.2 Security Case	8
2	The Access component	9
3	The Confidentiality component	10
4	The Secure Link Keys component	11
5	The Availability component	12
6	Protection from Denial of Service Attacks	12
7	The Integrity component	13
8	The roles involved in the generation of the initial link key, K_{INIT}	19
9	The SPL description of the generation of K_{INIT}	20
10	The initial link key generation System	20
11	Specifying the Intruder, <i>Mallory</i>	21
12	Specifying that K_{INIT} is a secret between A and B	23
13	Bluetooth scenario 1: PDA (Alice) getting CIC calendar entry from laptop (Bob)	38

1 Introduction

1.1 Purpose

This report forms deliverable D4 of the FORWARD project. It is the first deliverable of Work Package 2 *Securing Bluetooth*.

The purpose of Work Package 2 is to assess the security mechanisms present in the Bluetooth v1.2 standard, as described in [Blu03] and [Geh02], and where necessary develop means of protection for systems utilising Bluetooth as a communications medium. The research is split into three distinct phases:

- Phase One: Analytical assessment of Bluetooth security mechanisms present in Bluetooth v1.2.
- Phase Two: Practical investigation of any issues identified in Phase One.
- Phase Three: Development of mechanisms for protection of systems utilising Bluetooth (if appropriate).

This deliverable reports on Phase One of the research, analytical assessment of the security mechanisms provided by the Bluetooth v1.2 standard. Our contribution being both the results we obtain, and the methodology we have developed for analysing security of systems utilising wireless communications which is generic and applicable to other similar media.

Our approach is to fuse together the knowledge and intuition possessed by the QinetiQ penetration testing team with the expertise of formal analysts. This enables us to validate the results of the formal analysis with practical testing in a novel approach not formally done within the formal methods community. In addition, we begin our development of a graphical methodology for documenting security cases. This leverages a technique called Goal Structured Notation (GSN), currently utilised by the military for documenting safety cases for critical systems.

1.2 Background

1.2.1 The Next Wave Technologies and Markets vIRC

The FORWARD project is part of the DTI sponsored Next Wave Technologies and Markets programme¹. The aim of the programme is to ensure that UK business is structured and equipped to exploit the new information and communications technologies offered by the pervasive computing paradigm (described in Section 1.2.2 below). In addition, the programme aims to foster viable markets with confident consumers so that U.K. citizens can enjoy the benefits offered by such future technology environments. The programme achieves this by supporting seven themed virtual interdisciplinary research centres (vIRC), integrating and disseminating the research via a knowledge transfer club. The FORWARD project is part of the *City and Buildings Virtual Research Centre*, whose focus is challenges in mobile appliance design, infrastructure integration, delivery architectures and interaction design.

1.2.2 Pervasive Computing

The pervasive computing paradigm foresees communicating and computational devices embedded in all parts of our environment, from our physical selves, to our homes, offices, streets and so forth. Humans will be surrounded by intelligent, intuitive, interfaces capable of providing information and communication facilities efficiently and effectively. Systems will recognise the presence of individuals, perhaps even their mood, in an unobtrusive manner, modifying their functionality according to the users changing needs. The prolific amount of communicating devices will provide and enable multiple dynamic networks at any one location. The systems created from the integration of these large,

¹ www.nextwave.org.uk

complex networks will provide many new ubiquitous services, *nu-services*. In order that users may take advantage of the *nu-services*, users and their autonomous agents will be able to traverse these networks passing seamlessly from one to another, coexisting in many at a single point in time. So creating a truly ubiquitous computing environment capable of supporting ambient intelligence.

1.2.3 The objectives of FORWARD

Trust is at the heart of all successful business relationships, both in the physical world and on-line, ensuring their longevity and stability. User confidence is essential to the development of Next Wave technologies and markets; if people do not believe that the technologies will deliver the desired functionality, then they will not purchase them.

The users of Next Wave technologies will have to be able to rely on them to not only secure their data, protecting it from unwanted eyes, but also to deliver a certain quality of desired basic functionality and services. However, the inevitable complexity of Next Wave technologies and environments means that this will be extremely challenging to achieve. First, there will be more information to secure. In order to take advantage of next wave technologies companies and individuals will not only increasingly store information electronically, but their Cyber-actions (digital behaviour) may also be recorded by third parties. In some cases the stored information will be extremely valuable, perhaps a key business asset or sensitive personal data.

Second, the required communications infrastructures will be far more complex. Dynamic networks, networks that are mobile and can be created in an ad hoc manner, are core to the pervasive computing paradigm. The devices that populate such networks will have to be capable of interaction. The protocols and mechanisms for establishing such forms of communication will have to be scalable, and the infrastructures reliable and re-configurable. Designing systems formed from such networks will be challenging. It is unlikely that a human will be capable of even imagining all of the possible configurations such a system might be in. Such complex systems may possess emergent properties not conceived of in their design, the result of combining many components. These properties may be undesirable, impacting upon the integrity of the system and possibly the trust of the user. Techniques will have to be developed which support the design and implementation of high-integrity systems, capable of evaluating emergent properties, and enabling the assessment and validation of the technologies the systems use. Relevant standards will have to evolve to capture new types of behaviour and interactions.

The aim of project FORWARD is to enable the development of trustworthy and flexible *ambient intelligent* environments. It will achieve this by systematically investigating core issues in the development of trustworthy wireless communications protocols and devices, delivering a methodology based on rigorous tools and techniques. Bluetooth is an example of a peer-to-peer communications protocol which could enable such pervasive communications. It has been chosen as a case study since it could be a core part of future pervasive computing environments, however, all analysis techniques developed within this work-package are generic and could equally be applied to any other similar protocol.

1.3 Structure of the Document

This report is structured as follows. We begin in Section 2 by considering attack vectors on Bluetooth communications from an ethical hackers perspective. Then in Section 3 we present our graphical representation of the Bluetooth security case as documented by the Bluetooth consortium. This produces a list of security mechanisms requiring further investigation, including the link-layer bonding mechanisms which are the subject of our formal analysis, detailed in Section 4. Finally, in Section 5 we present our conclusions.

2 Bluetooth – Potential Vectors for Attack

2.1 Introduction

This section presents potential attack vectors on Bluetooth communications. It has been written by professional penetration testers, and was used both to form the Bluetooth Security Case, presented in Section 3, and in determining the formal modelling approach, presented in Section 4. We consider the concerns expressed by some users, outline the likely practicality of such attacks, and describe current research that could highlight such attacks in penetration testing environments.

2.2 Motivating Questions

Our attack vectors are considered in relation to a set of scenario/implementation based questions. Each question relates to unauthorised 'use of' or 'access to' devices:

- Can an attacker deny access to network services legitimately provided via Bluetooth?
- Can an attacker intercept Bluetooth traffic, e.g. a telephone conversation using a Bluetooth headset?
- Can an attacker eavesdrop on or otherwise obtain data sent via Bluetooth, e.g. recover the last document sent to the printer?
- Can an attacker gain access to information or services on a wired network another user is connected to via a Bluetooth enabled device?
- Can an attacker utilise the camera on a phone via Bluetooth?

2.3 Attack Vectors

While Bluetooth is more difficult to intercept than other wireless mediums (such as 802.11x), it is still easier to intercept than a wired network is to wiretap. To intercept Bluetooth data, an attacker could follow one of four paths:

- Listen on all 79 available channels and recreate the datastream (we will call this 'raw intercept', either blanket intercept on all 79 channels, or using awareness of Frequency Hopping Spread Spectrum (FHSS) and potentially tracking the FHSS through communications means).
- Intercept radio traffic on one of the 79 channels (in the 2.4Ghz range), establish the frequency hopping pattern, and follow the communication (we will term this 'smart intercept', where some knowledge of Bluetooth is assumed).
- Obtain a bond (ie a trusted mode, where link key has been successfully shared) to one or more of the target devices.
- Mimic a bonded device's identity, this would not be as obvious as a separate bonded device, it would essentially be a 'clone' of a headset etc.

All mechanisms require the attacker to establish the 48 bit Bluetooth Device Address (BD_ADDR) of a target host. This and other Bluetooth mechanisms which may need to be compromised are described below.

2.4 Mechanisms to be compromised

2.4.1 Obtaining the BD_ADDR

There are two modes in Bluetooth discovery:

- Discoverable - supply their BD_ADDR to nearby devices searching for partners.
- Non-Discoverable - do not reveal their presence to nearby devices searching for partners.

To obtain the BD_ADDR of a Discoverable device, an attacker needs simply to request it. To establish the BD_ADDR of a Non-Discoverable device, Ollie Whitehouse of @Stake developed the proof-of-concept tool RedFang v0.12. However, although it was useful conceptually, RedFang at the time of early drafts of this paper probed only for devices within a specific manufacturer code range (the value for which is the first half of the BD_ADDR), and it probed slowly, only scanning for one BD_ADDR combination at a time, with each probe taking several seconds. As a result, its use as a practical attack tool was limited.

RedFang 2.5 was recently released with help from QinetiQ and now provides a method of scanning all vendor ranges in good time. For example, RedFang 2.5 now supports up to a theoretical maximum of 127 USB devices which can be used to concurrently scan the bluetooth address space. This makes the attack much more practical.

2.4.2 Frequency Hopping

An attacker is prevented from simple interception of Bluetooth communications by a feature that was not designed for security, but as a method to avoid interference from operational and environmental interference - FHSS. This distributes communication over 79 channels within the 2.4GHz range, typically hopping around 1,600 times per second while communicating. Only the devices involved in a specific communication are meant to know the pattern used to 'hop' between these frequencies.

The hop pattern is derived from the BD_ADDR and precise timing of the Master of the connection. If these can be established, an attacker can potentially align their hop sequence with a communication.

2.4.3 Bonding, Pairing and Encryption

On first use, a Bluetooth device uses its 48 bit BD_ADDR and a random 128 bit value to generate a 128 bit Unit Key using the E_{21} cryptographic algorithm. This key rarely, if ever, changes through the device lifetime. Pairing of two Bluetooth devices occurs as follows:

- The verifier device sends a 128 bit random value to the claimant device
- Both devices use this random 128 bit value (known as IN_RANDOM), the claimant's PIN code (1-16 octets) padded with up to the full BD_ADDR to reach as near 16 bytes as possible (this padded value is PIN', pronounced PIN dash) and the length of the claimant's PIN in octets, to create a common 128 bit Temporary Link Key (known technically as the Initialization Key) using the E_{22} cryptographic algorithm.

Note that the only component not sent in cleartext is the claimant PIN itself, which is entered manually on the verifier, and is obviously known by the claimant device itself. This PIN can potentially be bruteforced (duration depends on length) or predicted (for simple devices such as headphones, which can be hardcoded to values such as 0000).

- To confirm these values match (and thus the correct PIN has been presented), the verifier (A) selects another random value (AU_RAND_A), the BD_ADDR of the claimant (B), and the Link Key, uses the E_1 algorithm to create a 32 bit Signed Response (SRES) value. The random value is also sent to the claimant.

- The claimant repeats the process described in the step above, and its SRES is sent to the verifier.
- The verifier checks the values match. If they do the claimant is authenticated.
- Finally, a 128 bit (semi-permanent) Link Key is agreed, this can be the Temporary Link Key, a Combination Key/a Master Key, or most commonly just the Unit Key of the verifier (see above) - whichever of these is chosen is sent to the claimant encrypted by the Temporary Link Key.

The state where two devices have agreed a Link Key is known as having paired the devices. Those device are bonded when they share that paired status semi-permanently.

2.4.4 Interception

The four attack methods described are then potentially viable, provided the attacker can achieve the following prerequisites:

- Raw intercept of radio traffic:
 - appropriate RF equipment, potentially with FHSS awareness,
 - appropriate software to recreate the traffic.
- Smart intercept of radio traffic:
 - appropriate RF equipment,
 - knowledge/prediction of the link key,
 - knowledge of Masters BD_ADDR and timing,
 - appropriate software to recreate the traffic.
- Obtaining of paired or bonded status:
 - physical access to device, or,
 - foreknowlege/bruteforcing of PIN and standard pairing as described above.
- Cloning a device's identity:
 - the ability to manipulate the BD_ADDR value,
 - knowledge/prediction of the link key,
 - knowledge of Masters BD_ADDR and timing,
 - appropriate software to recreate/create the necessary traffic.

The ideal, from a hacking perspective, is cloning of another device's identity which may allow a third party device to reduce its visibility on the piconet by remaining concealed in the list of bonded devices, while simultaneously allowing injection of data, exactly as if it were the original device.

If intercept only is required, pure intercept is adequate, and manipulation of the BD_ADDR is unnecessary. Obtaining bonded status is sufficient to intercept data, but is the most visible of the four methods.

2.5 Answer to attack questions

Can an attacker deny access to network services legitimately provided via Bluetooth? There are four existing methods by which an attacker may deny service to the network:

- Band Jamming - by introducing electromagnetic interference in the 2.4GHz band used by Bluetooth it is possible to block all RF communications. The downside of this attack is that it is obvious that the RF frequency is being jammed, and the source of the jamming will be easily detectable.
- Device disruption (eg overflow, resource starvation) - by discovering implementation vulnerabilities in specific Bluetooth implementations, attackers can deny service to those devices. One example of this attack is to send a device an unexpected value in response to a request for information, causing the device to crash, thereby preventing further communication until the device is rebooted/restarted.
- Communications specific denial of service, this theoretical attack would work in a similar way to a network sniffer. Namely, by associating with one of the devices and learning the hop pattern for the communication, the RF jammer can specifically target the channels that the devices are using to communicate. The source of this form of jamming would be much harder to detect, since lower power transmissions would be required, and other Bluetooth communications and RF technologies (such as 802.11b and 802.11g) would remain unaffected.
- The battery draining technique may be useful against remote 'in the field' networks. Bluetooth is carefully designed to use power saving modes to maximise battery life in small devices. However, repeated connections can potentially hold it in 'maximum power' mode, reducing battery life. This needs experimentation to establish the precise impact, and would require the device's BD_ADDR to be known, but does present a potentially valid attack against battery-based Bluetooth devices.

Can an attacker intercept Bluetooth traffic, e.g. a telephone conversation using a Bluetooth headset? In order for an attacker to intercept traffic, one of the above intercept methods would need to be implemented (see 2.4.4). Given this, traffic interception would be possible - the most effective (but difficult) method would be device cloning, for example cloning a telephone earpiece.

Can an attacker eavesdrop on or otherwise obtain data sent via Bluetooth, e.g. recover the last document sent to the printer? Interception of traffic is described above. If an attacker can obtain interactive access, either by cloning or bonding, they can use this to use that access to manipulate connected systems. For example, application level attacks against a printer's ChaiVM13 (embedded JAVA virtual machine) through Bluetooth may be possible. If successful, the printer may well allow activity such as monitoring of the Print Spool Directory. A skilled attacker who was able to compromise the devices through Bluetooth, would be free to install JAVA classes either on such a printer or on a secondary device (such as a laptop) which could copy any 'interesting' print jobs or network activity. Since an increasing number of handheld devices have JAVA engines, it is likely that keyloggers could be deployed on phone handsets. This would enable the attacker to record interesting numbers, including the Bluetooth PIN number entered when bonding with new devices in the future.

Can an attacker gain access to information or services on a wired network another user is connected to via a Bluetooth enabled device? It is likely that once a Bluetooth device is compromised, it would be possible to upload tools that would allow secondary attacks, or even attack directly. If, for example, the phone can communicate with a wired network (either directly or via another device) it could proxy, or even initiate, standard network reconnaissance/attacks against that network. The degree to which these techniques would be successful depend on the security of the network in question. Given compromise of a network-linked Bluetooth device, the attacker would have equivalent access to plugging a laptop into the wired network. From a penetration

testing perspective, internal networks are often the softest part of the network, and compromise of a complete internal network is often viable from such access.

Can an individual utilise the camera on a phone via Bluetooth? Network Layer Access to the camera may be granted as a result of compromise of the application layer of the device. However, it would be a prerequisite to first compromise the network layer (as detailed in the other sections of this document) in order for the attacker to be in a position to lever access from a remote perspective. As mentioned previously, at the application layer the JAVA Virtual Machine (VM) is an essential component of the latest generation of mobile phones, with the JAVA engine handling many functions in place of the traditional firmware. The VM provides a complete API allowing programmers to call all of the phone's functions. This would include access to all traditional functions (placing calls etc), along with more contemporary functions such as cameras and Infrared communications ports. Also, methods exist to allow telephone companies to upgrade SIM software using SMS messages read by the owner of the phone - there is no reason this principle could not be applied inappropriately, allowing arbitrary uploading of software to the phone, but such techniques are outside the scope of this document. A compromise of the underlying Bluetooth infrastructure may allow an attacker to upload software to a Bluetooth device, and perhaps even execute this. This software could potentially connect back to the attacker (either via Bluetooth or a more traditional medium) and allow remote control of the compromised device.

3 Bluetooth Security Case

3.1 Introduction

In this section we present the security mechanisms present in Bluetooth v1.2, as detailed in the Bluetooth SIG White Paper [Geh02]. We construct a *Security Case* using the Goal Structured Notation [Kel04]. GSN has been developed over the last 10 years by the University of York and various industrial partners as a means of capturing and presenting safety arguments, though there is nothing specific to safety in the notation. It is widely used in the safety community, particularly in military aerospace industries, where it is the de-facto standard for the presentation of safety cases.

The security of any system can be expressed as the combination of a number of properties. For the sake of this Bluetooth security study, we will consider the following criteria:

- Access control,
- Confidentiality,
- Integrity
- Availability.

Classically Security is decomposed into *Confidentiality*, *Integrity*, *Availability* and *Non-repudiation*. In our presentation of the Bluetooth v1.2 security case we include *Access Control* as a core property, however, it would normally be considered a component of *Confidentiality*. This mirrors the presentation of the security argument made in the Bluetooth Whitepaper detailing the security mechanisms present in v1.2. We have not included *Non-repudiation* since this is only relevant when considering a particular application or service, and is therefore out of scope of this security case.

Figure 1 details the decomposition of the security case into these four key properties:

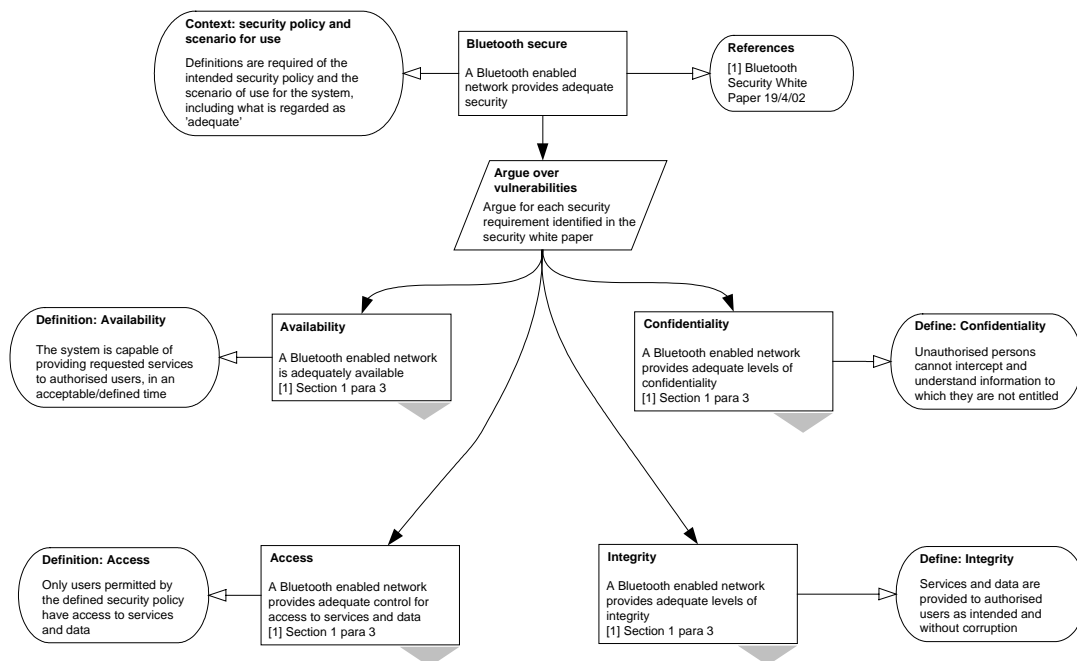


Figure 1: The Bluetooth v1.2 Security Case

The root node shows the overall property that we are arguing has been achieved, that a Bluetooth enabled network is “adequately secure”. This property is decomposed into the four basic properties that we have listed. Each of the basic properties has a separate sub-tree, which represents the security argument that property has been achieved.

The reading of the Goal Structure Notation (GSN) diagram is as follows:

- Rectangles represent “goals” or claims that some objective has been/can be achieved. In the tree structure, a goal is said to have been achieved if all its sub-goals have been achieved (i.e. satisfaction of a goal depends solely upon the satisfaction of its explicit sub-goals). A goal can also be claimed to have been satisfied by direct appeal to some evidence (a Solution).
- Circles represent “solutions”. These are direct evidence (such as a reference to test results) that demonstrates that some goal has been achieved.
- Rounded boxes are “context”, additional information to enhance understanding of the argument. They may for example provide definitions of terms or criteria for abstract requirement such as “adequately secure”. Contexts are usually attached to goals and are assumed to be applicable to all subsequent sub-goals in the hierarchy.
- Trapeziums are “strategies”, essentially comments explaining why a collection of sub-goals is expected to justify their parent goal.

Two additional elements of GSN notation: a goal may be decorated with either a solid triangle or an open diamond. The later represents an ‘unresolved goal’. That is, the argument is not complete, and providing a solution or sub-goals to address this goal is required to complete it.

In the argument presented here, it should be noted that some of the properties are linked at the lower layers of decomposition (specifically, access control and confidentiality both depend on protection of the link keys). This is to be expected in a complex system, where some implementation details have more that one role or function within the overall system when viewed from a higher level.

The following paragraphs will discuss each of these top level properties and their associated security arguments.

3.2 Access

We define access control as the ability of the system to grant access only to properly authorised users. We extend this to say that the access levels granted to authorised users should be in accordance to an agreed security policy. This is illustrated in Figure 2 below:

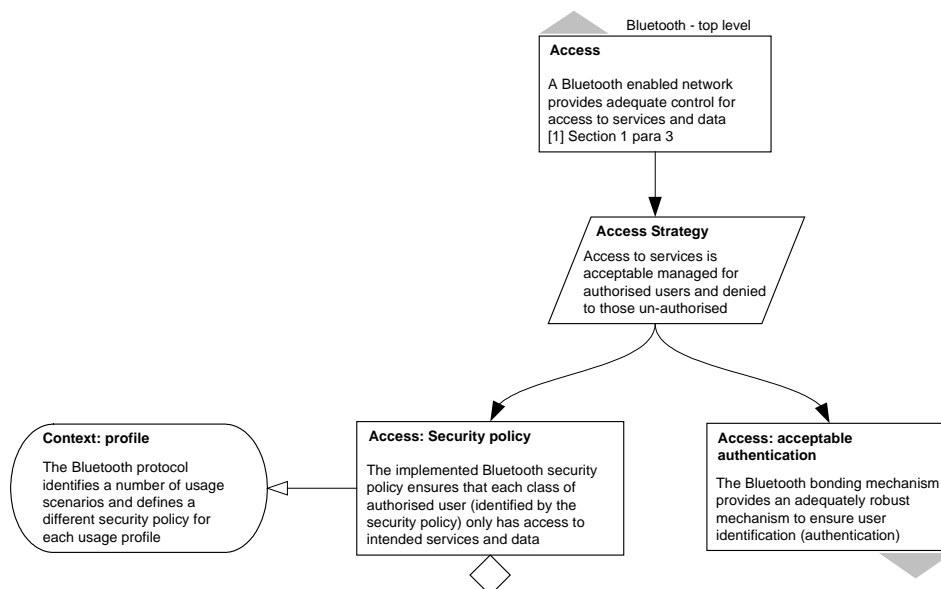


Figure 2: The Access component

Within Bluetooth, the primary method of access control and confidentiality is the authentication mechanism to establish the identity of communicating devices. Authentication is achieved by the bonding mechanism, which establishes a common 'link key' between pairs of devices. We will analyse the elements of this link in the confidentiality section below.

Utilising the core Bluetooth protocol, profiles are defined which seek to apply Bluetooth to a number of application scenarios. For each of these profiles, a different security policy dictates controls over the supply of services to authenticated (in the Bluetooth sense) users. As a result, the diagram places a dependency on 'Service security policy mechanisms' ("Access: security policy").

3.3 Confidentiality

We claim that the confidentiality of Bluetooth is guaranteed by two properties. Firstly, only authorised users should have access to services and secondly, the communications that take place between authenticated parties should be private - meaning that a third party should neither be able to illegitimately acquire the encryption key, nor break the encryption by 'brute force' methods. See Figure 3:

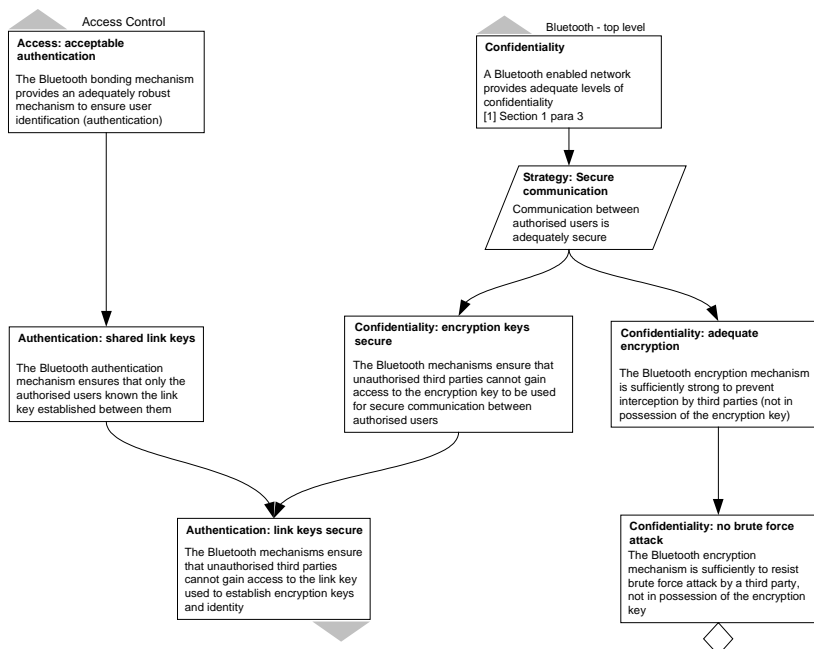


Figure 3: The Confidentiality component

The diagram illustrates a 'single point of failure'. There is an inter-dependency, between "Access: acceptable authentication" and "Confidentiality: encryption keys secure". Both the access control and privacy properties rely on the satisfaction of the "Authentication: link keys secure". As a result, we have expanded this property to show its dependencies in more detail.

When two Bluetooth devices are bonded, the devices partake in a five stage process. These stages are:

- the generation of an initialisation key,
- the generation of a link key,
- an exchange of link keys,
- authentication of the two devices, and finally

- the creation of an encryption key, where encryption is to be used.

From a consideration of the specification, it is clear that each step relies on the previous for its security. As a result, a compromise at any step is sufficient to allow an attacker to circumvent the access control and encryption. Figure 4 below outlines this property:

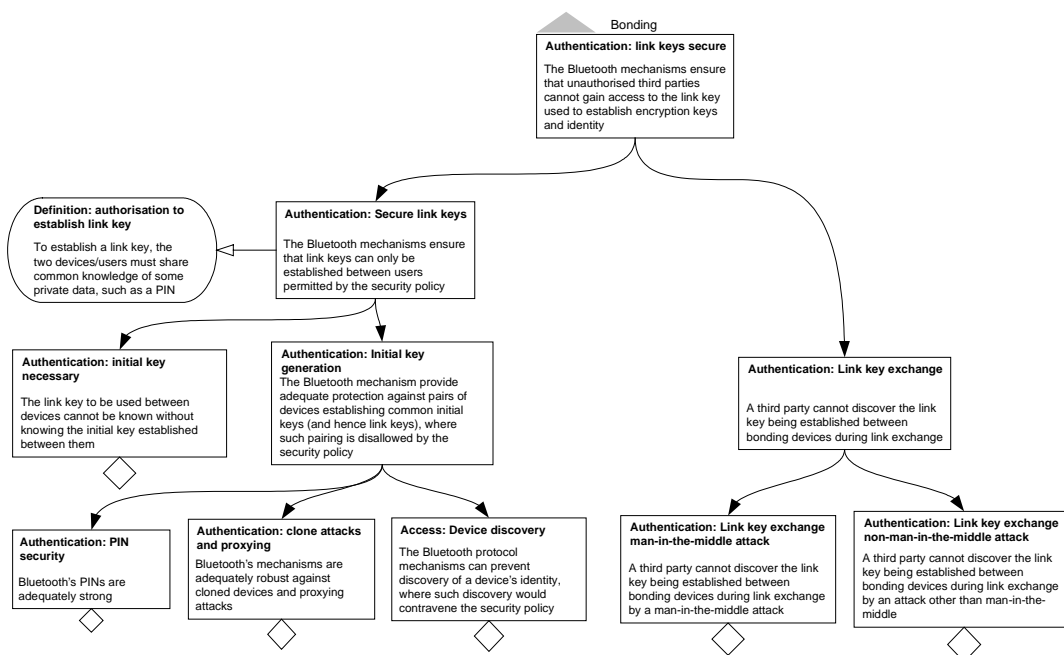


Figure 4: The Secure Link Keys component

The initialisation key is generated from a number of parameters. These parameters are:

- The physical address of one of the devices, designated the BD_ADDR
- A PIN code. This may be fixed, for example in the case of a mobile phone headset, or variable (user supplied), for example in the case of a PDA or mobile phone.
- The length of the PIN in octets (8 bit bytes)
- A random number, IN_RAND, agreed between the parties.

These parameters have to be agreed between the parties before an initialisation key can be generated. Depending on which device initiates the connection and the type of the PIN each parties possess, either an agreement on the IN_RAND and BD_ADDR is reached, or in the case when both devices have fixed PINs, no further communication is possible.

3.4 Availability

The availability property is expressed as the Bluetooth service being available for use by legitimate users in a timely fashion. The availability property is further decomposed into three features: Underlying hardware reliability and performance, Availability in a noisy environment and Denial Of Service protections for authentication. See Figure 5 below:

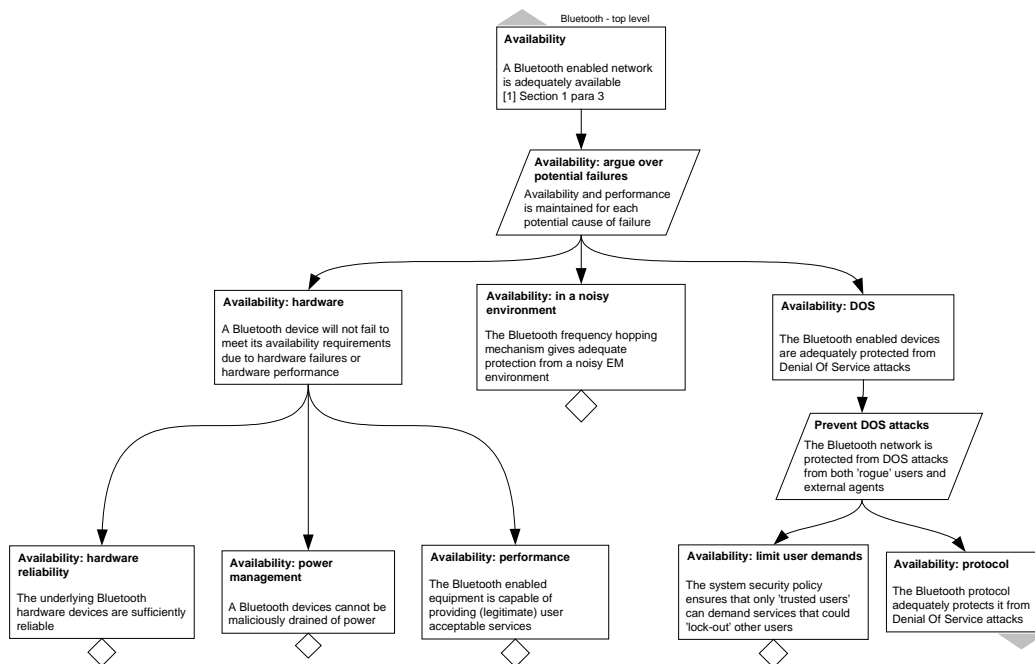


Figure 5: The Availability component

In addition to basic hardware reliability and performance, the main vulnerability related to hardware is its susceptibility to power consumption attacks. In mobile devices, battery power is conserved by placing the device in standby mode whenever possible. A power consumption attack would seek to either prevent the target device from utilising standby mode or seek to wake up the device as often as possible.

Measures are put in place in the Bluetooth v1.2 specification to protect against Denial of Service attacks. See Figure 6 below:

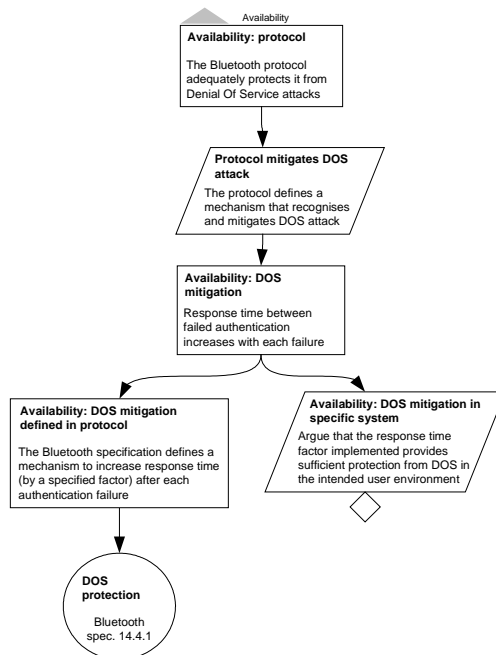


Figure 6: Protection from Denial of Service Attacks

Denial of Service for authentication: The security part of the Bluetooth specification, Volume 2 Part H, details a defence against denial of service attacks on the authentication mechanism (5.1 Repeated Attempts). The basis of this scheme is that if a device fails to authenticate successfully, then the rejecting device should not consider another authentication attempt from the same device before a specific period of time has elapsed. Furthermore, the length of time between attempts should be increased exponentially for each repeated failed attempt.

The purpose of this scheme is twofold. Firstly, it protects the requested device from being swamped by authentication requests from a malicious device, and secondly it prevents a malicious device from attempting to force an authentication by applying brute force techniques.

3.5 Integrity

Integrity of information in the Bluetooth arena focuses on ensuring that data transmitted is received unmodified. If data is corrupted, the Bluetooth service should be capable of detecting the error and possibly retransmitting this information:

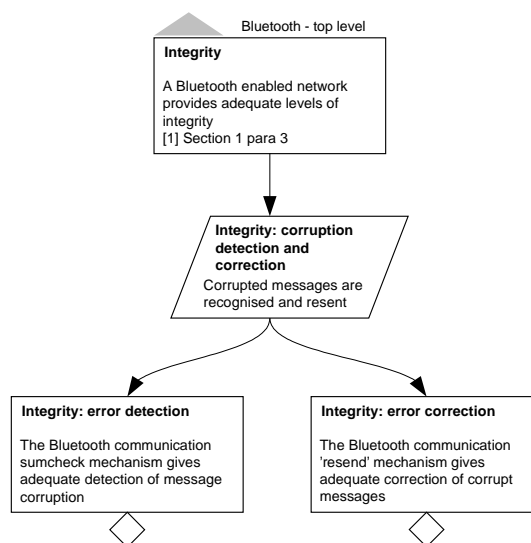


Figure 7: The Integrity component

3.6 Bluetooth Security Mechanisms requiring investigation

The Security Case details the set of security mechanisms which must be true, and therefore should be the subject of further investigation. They appear at the bottom of the tree, and we list them here incorporating the outputs of Section 2:

- **Access: Security Policy** To test the service security policy mechanisms, each of the claimed security policies for the profiles should be analysed and a set of tests constructed and implemented.
- **Confidentiality: no brute force attack** One possible attack against Bluetooth would be to brute force the cryptography. A study is required to determine the strength of the Bluetooth under normal operating conditions. One observation is that the brute force technique, requires interception of the message in a complete fashion. The interception device would have to follow the frequency hopping sequence of the participating devices, which is practical, due to the way in which the 'hop' algorithm is specified.
- **Authentication: initial key necessary** The security of the key generation and exchange process is dependent on two values. Firstly, the BD_ADDR physical address and secondly, the

PIN values of the two devices. Interestingly, the algorithm to generate initialisation keys does not depend strictly on both the BD_ADDR and the PIN values, but rather a combination of the BD_ADDR and PIN. If the PIN is less than 16 bytes, it is padded to 16 bytes by using bytes from the BD_ADDR value. Therefore, in these circumstances, it may be possible to use knowledge of the BD_ADDR to compromise the key.

- **Authentication: PIN security** Since the security of the initialisation key generation is dependent on a PIN and this is user controlled, it can be seen that this is a weak point. As a result, one area of investigation is PIN strengths. Standard login passwords can be made more secure by applying certain techniques to choosing them (for example not using real names, mixing case, using numerics etc). Although the PIN is believed to be limited to a number, this is not the case. The values that PINs can take is limited only by the interface available to input the PIN. As a result, policy should be devised to ensure the choice of strong PINs. There is a trade-off, however, between PIN strength and usability. A further issue is to discover whether the use of a 16 octet PIN eliminates completely any use of the BD_ADDR address in generating the initialisation key. If this is the case, then a complete class of attacks based around discovering the BD_ADDR could be eliminated, or at least drastically reduced.
- **Authentication: clone attacks and proxying, Access: Device discovery** Also connected to the secure access property, is the issue of device discoverability. In order to communicate with a device, that device must be capable of being discovered and addressed. Bluetooth provides a mechanism for setting the mode of a device to limit the way that the device exposes its services. Whilst these modes provide a level of defence against an attacker, tools such as RedFang have shown that even devices in a non-discoverable mode can in fact be discovered. Additional work should be conducted in this area to investigate whether the RedFang techniques can be guarded against.
- **Authentication: Link key exchange man-in-the-middle attack** The possibility of such attacks are the subject of our formal analysis, detailed in Section 4.
- **Authentication: Link key exchange non-man-in-the-middle-attack** It is conceivable that via some empirical means certain crucial information may be compromised.
- **Availability: hardware reliability** The impact of unreliability can only be considered with respect to a particular implementation.
- **Availability: power management** According to the Bluetooth specification, to save power, slave nodes that do not wish to participate in communications with a master node, but which wish to remain synchronised to the master, may enter a state known as parked. This facility is also used to allow a piconet to contain more than 7 slaves. If non-participating slaves are parked, then the piconet can contain a large number of slaves (although only 7 can be active at any one time). Whilst parked, the slave device will wake up periodically to re-synchronise with the master and to check for broadcast messages. The specification states that it can be woken up by sending a specific message to either its parked address PM_ADDR (designated when the slave goes into parked state), or its BD_ADDR. Since we have already shown that the BD_ADDR address can be determined by an attacker, it should be possible for an attacker to keep a target device from remaining in the parked state and therefore from conserving power. A test should be devised to determine both the practicability of such an attack and the impact on the battery lifetime of a range of typical devices.
- **Availability: performance** For a particular implementation availability must be considered in relation to the specified acceptable level of service.
- **Availability: in a noisy environment** Availability in a noisy environment: The frequency hopping component of the Bluetooth specification is designed to enable Bluetooth devices

to operate in noisy RF environments and not designed specifically to provide any security properties. Each Bluetooth device is capable of switching frequency 1600 times a second. The spectrum allocated to Bluetooth is split into 79 distinct channels. A master device will select a pseudo-random sequence of 32 channels. The initial seed for the sequence is given by the BD_ADDR of the master device. Since the BD_ADDR address of the master device is discoverable, it is possible to determine the hopping sequence, and hence intercept the Bluetooth communications for the purpose of eavesdropping. If a denial of service attack was planned, then the channels in the hopping sequence could be selectively blocked, effectively preventing any transmission. It should be noted, however, that the equipment required to selectively block channels in this way is far more complex and sensitive than a device to simply jam the entire 2.4GHz frequency band (i.e. all 79 Bluetooth channels) in the location of the communicating devices. Since the power requirements for Bluetooth are deliberately low, the amount of power required to jam the Bluetooth signals would be very low.

- **Availability: limit user demands** For any implementation the security policy must be assessed to ensure that it only enables appropriate users to make demands which might 'lock out' other users. A mechanism for quickly assessing access control policies must be constructed.
- **DOS protection, Availability: DOS mitigation in specific system** Unfortunately, the adoption of this technique itself leads to a denial of service attack. Imagine that a laptop wishes to bond with a printer to allow documents to be printed. A malicious device located in the vicinity could impersonate the laptop, attempt to authenticate to the printer and deliberately fail. If the malicious device then persistently continued to attempt failed authentications, then length of time between attempts would increase exponentially and the laptop would never be able to authenticate to the printer. As a result, no documents could be printed and so a successful denial of service attack is possible. In order to impersonate the laptop, the BD_ADDR address of the laptop would need to be determined by the attacker. After this, a simple piece of software would be needed to generate the spoofed Bluetooth transmissions to impersonate the victim device. Since tools exist to determine the BD_ADDR address of Bluetooth devices (even ones which are not in discoverable mode), this attack is believed to be entirely possible.
- **Integrity: error detection, Integrity: error correction** Mechanisms exist within the Bluetooth specification to provide for checksums and retransmission. These mechanisms should be assessed to ensure that they provide the necessary protection for integrity.

4 Verifying the Bluetooth Link-Layer security using Casper

4.1 Introduction

In this section we report on our verification of the Bluetooth Link-Layer security using Casper ('Casp'=Compiler for Analysing Security Protocols) [Low98, Low99], see *Authentication: link key secure* of the Security Case presented in Section 3. Casper is a Communicating Sequential Processes (CSP) [Ros98] front-end well suited to the verification of two-party security protocols, such as the stages that comprise the Bluetooth *initialization procedure*.

Casper takes, as input, descriptions of security protocols written in a de facto "standard notation", together with specifications that assert the security properties claimed for the protocol. Casper compiles a given protocol description to a CSP process representing the possible behaviours of that protocol when run in the presence of a general *Intruder*. The specifications are also compiled to CSP processes. These processes - representing the implementation and specifications of the protocol - can then be automatically compared using the CSP refinement checker, Failures-Divergences Refinement (FDR) [For].

The main reason for choosing Casper is that it does all the hard work of modelling the general Intruder for us - for a human, writing a bespoke CSP model of the Intruder can be a time-consuming (and error prone) process. However, Casper does have its limitations when it comes to the modelling of multi-party protocols - as may be evinced from our model of Bluetooth service usage (including usage of the *service discovery protocol*), as reported on in section 4.6.

The remainder of this section is structured as follows. In section 4.2 we give an introduction to CSP, FDR, Casper, and the Security Protocol Language (SPL) that Casper uses. This introduction is intended to be sufficient for a reader unfamiliar with Casper to follow our translations of the original Bluetooth protocol specifications to Casper models (as detailed in section 4.4). In section 4.3, we describe our tool *Caspose*, which allowed us to take individual stages of the Bluetooth initialization procedure and compose them in different ways. Potentially, *Caspose* could be used to generate a suite of SPL models that collectively describe all possible variations of the initialization procedure. In section 4.4 we consider the individual stages of the Bluetooth initialization procedure, and describe how we specified each of those stages in SPL. In section 4.5, we look at a number of questions regarding Bluetooth Link-Layer security which were answered through our Casper modelling, or other 'off-line' arguments short of automated proof. We also quote some interesting questions for possible follow-on work, which were outside the scope of our current analysis. In section 4.6, we describe an example Bluetooth scenario that one could well expect to see in the business world. The scenario involves a device using the Bluetooth service discovery protocol, this scenario has been modelled using Casper. Finally we summarise the results of our analysis, and detail areas for further research.

4.2 CSP, FDR, Casper and SPL

4.2.1 CSP and FDR

CSP is a process algebra which is useful for describing systems that interact by communication. A system is modelled as a *process* (itself possibly constructed from a collection of processes) which interacts with its environment by means of atomic *events*. Communication is synchronous; an event takes place precisely when both the process and the environment agree on its occurrence. The syntax of CSP provides a variety of operators for modelling processes, and the associated algebra provides rewrite laws. Primitive operators include process prefix, sequential composition, deterministic and non-deterministic choice, parallelism, hiding, recursion, deadlock and successful termination:

$$P ::= a \rightarrow P \mid P ; P \mid P \square P \mid P \sqcap P \mid P \parallel P \mid P \parallel\!\!\parallel P \mid \\ P \setminus b \mid \mu X \bullet F(X) \mid STOP \mid SKIP$$

The collection of mathematical models and associated semantics that make up CSP facilitate

the capture of a wide range of process behaviours. The traces model captures the observable traces of events which a CSP process might exhibit. The failures model captures not only the traces of a process, but also those events it can refuse to perform after a particular trace. The failures divergences model also captures information about events which a process may diverge on (perform infinitely often) from a particular state.

The theory of refinement in CSP allows correctness conditions to be encoded as refinement checks between processes. If process P refines process Q , then all of the possible behaviours of P must also be possible behaviours of Q (although Q may also possess many other behaviours). Therefore, P is a correct, and more deterministic, implementation of Q . This notion of refinement holds for all three of the semantic models, where the possible behaviours of processes are interpreted in terms of the semantic model under consideration.

Refinement is transitive:

$$R \sqsubseteq S \wedge S \sqsubseteq T \Rightarrow R \sqsubseteq T$$

If process R is refined by process S (written $R \sqsubseteq S$), and S is refined by T then R is refined by T . All CSP operators are monotonic with respect to refinement:

$$R \sqsubseteq T \Rightarrow C[R] \sqsubseteq C[T]$$

where $C[\cdot]$ is a context built from CSP operators and constants. This facilitates compositional development of systems. Imagine we want to prove:

$$Spec \sqsubseteq C[System]$$

we can break this into two parts: Find a process which does refine the desired specification:

$$Spec \sqsubseteq C[P]$$

and prove that:

$$P \sqsubseteq System$$

By monotonicity:

$$C[P] \sqsubseteq C[System]$$

and by transitivity:

$$Spec \sqsubseteq C[P] \wedge C[P] \sqsubseteq C[System] \Rightarrow Spec \sqsubseteq C[System]$$

The Failures Divergences Refinement Model Checker (FDR) tool takes a machine-readable dialect of CSP_M as its input syntax, and can be used to check refinements as well as determinism, deadlock freedom and livelock freedom of processes. CSP_M is the combination of a rich data language, based on functional programming, and the CSP process algebra. The various CSP_M operators which are used in this thesis are given below. See [Ros98] for detailed descriptions of CSP_M .

The CSP_M processes that we use are constructed from the following:

- STOP is the simplest CSP process; it never engages in any action, nor terminates. It is equivalent to deadlock.
- $a \rightarrow P$ is the most basic program constructor. It waits to perform the event a and then behaves as process P . The same notation is used for outputs ($c!v \rightarrow P$) and inputs ($c?x \rightarrow P(x)$) of values along named channels.
- $P \mid\sim\mid Q$ represents internal choice. It behaves as P or Q *nondeterministically*.

- $P \square Q$ represents external choice. It will offer the initial actions of both P and Q to its environment at first; its subsequent behaviour is like P if the initial action chosen was possible only for P , and like Q if the action selected Q . If both P and Q have common initial actions, its subsequent behaviour is nondeterministic (like $| \sim |$). $STOP \square P$ behaves as P .
- $P [| a |] Q$ represents parallel composition. P and Q evolve concurrently, except that events in a occur only when P and Q agree to perform (i.e. *synchronise on*) them.
- $P ||| Q$ represents interleaved parallel composition. P and Q evolve separately, and do not synchronise on any events. Equivalent to $P [| \{ | \} |] Q$
- $P [a || b] Q$ represents alphabetised parallel. P and Q have to agree on events in the intersection of their alphabets.
- $| | x:a @ [A(x)] P(x)$ represents replicated alphabetised parallel. This constructs the parallel composition of $P(x)$ processes, one for each x in a , over their respective alphabets.
- $P \setminus A$ is the CSP hiding operator. This process behaves as P except that events in set A are hidden from the environment and are solely determined by P ; the environment can neither observe nor influence them.
- $P [[a \leftarrow b]]$ represents the process P with a renamed to b .
- $P [a \leftrightarrow b] Q$ is the linked parallel operator. P and Q synchronise on channels a and b , which have been renamed to the same and hidden. There are also straightforward generalisations of the choice operators over non-empty sets, written $| \sim | x:X @ P(x)$ and $[] x:X @ P(x)$.

4.2.2 Casper and SPL

Casper is a compiler that was devised specifically to simplify the process of CSP-based analysis of security protocols.

Casper inputs are `.spl` files, where 'spl' stands for *security protocol language*. Each `.spl` file describes: (i) a security protocol, (ii) a *system* of several honest *agents* running that protocol in the presence of an *Intruder* who tries to 'break' the protocol with respect to a set of required security properties, and, (iii), a formal *specification* in terms of the set of security properties that should hold of the system.

Casper compiles `.spl` files to `.csp` files containing CSP processes representing the system (including the Intruder) and the security properties. These processes can then be compared using the CSP refinement checker, FDR.

In addition to `.spl` and `.csp` files - which one would expect to be present in any Casper-based validation project - we also have `.cspl` files. These require a bit of explanation. The suffix 'cspl' stands for 'Caspose-spl', where 'Caspose' is our `.spl` composition tool described in 4.3. The `.cspl` suffix is used merely to differentiate `.spl` files with some additional macros from pure Casper compile-able `.spl` files. The macros allowed us to write slightly more natural and less cumbersome scripts, but which are still generally in accord with the Casper syntax. The Caspose tool maps the macros to pure Casper compile-able SPL instructions, and so each `.cspl` file is semantically equivalent to a `.spl` file. In the reading of this document, '`.cspl`' and '`.spl`' are safely interchangeable.

For the remainder of this section, we shall show, by way of example, how a protocol is described in Casper. Our example `.spl` script is 'init.cspl', in which we modelled the Bluetooth protocol for generating an *initial link key*, K_{INIT} ².

²The generation of an initial link key is the first stage of the Bluetooth initialization procedure.

4.2.3 Processes

In Casper, a protocol is split up into a number of named parts, or *roles*. These are introduced in the *Processes* section of the .spl file. For example, in figure 8 we have the Processes section of our file init.spl. From this, we see that there are two parts to this protocol, namely *INITIATOR* and *RESPONDER*. The first parameter of each role - in this case *A* and *B* - will later to be instantiated as *actual* Bluetooth devices.

Note, also, the *knows* keyword. Following the *knows* keyword are a list of 'facts' that are assumed to be known *a priori* to any device in that role such as cryptographic functions. In this case, *A* and *B* both know all about Bluetooth Addresses, as returned by the function $Addr : BlueToothDevices \rightarrow BDADDR$.

```
#Processes
```

```
INITIATOR(A,inrand) knows Addr
RESPONDER(B,secret) knows Addr
```

Figure 8: The roles involved in the generation of the initial link key, K_{INIT}

4.2.4 Protocol description

Perhaps the most important section of an .spl file is the *Protocol description* section. In this section, the analyst describes each role of the protocol in terms of the sequence of *messages* that an agent in that role is expected to communicate to other agents during the course of a run of the protocol. These messages may depend on messages that have been sent or received earlier, and on inputs from the *environment* - the latter may be regarded as 'user inputs'.

The syntax used in the Protocol description section is a formal grammar based on a de facto 'standard notation' used in academic literature. There are three types of protocol description line, these are as follows.

A line of the form:

$$label. sender \rightarrow receiver : f_1, f_2, \dots, f_n \quad (1)$$

indicates a message with label *label*, meaning *sender* is to send to *receiver* the messages f_1, f_2, \dots, f_n . An empty *sender* field means that the message is *from* the environment; an empty *receiver* field means that the message is *to* the environment. Messages sent from or to the environment are taken to represent user inputs and outputs and are treated specially by Casper.

A line of the form:

$$\langle var_1 := expression_1 ; var_2 := expression_2 ; \dots ; var_n := expression_n \rangle \quad (2)$$

indicates local variable assignments by the *sender* of the immediately *preceding* message.

A line of the form:

$$[predicate] \quad (3)$$

indicates a test (or *guard*) made by the *receiver* of the immediately *preceding* message, where *predicate* is an FDR predicate, typically over variables cited in the preceding message, which must be satisfied in order for the receiver to accept the message.

By way of example, in figure 9, we have, verbatim, all of our protocol description lines from the file init.spl. These describe the processing of messages sent and received during the generation of the initial link key, K_{INIT} .

With regard to figures 8 and 9, *A* is an agent playing the role of *INITIATOR*, and *B* plays the role of *RESPONDER*. The first line, labelled 0a, says that *A* receives from the environment a PIN (personal identification number), *pin*, and the identity of some responder, *B*. The second line,

labelled 0b, says that B receives a PIN from the environment (not necessarily the same pin as *A*'s). The line labelled 1, says that *A* sends to *B*, in plain-text, a (previously defined) random number, *inrand*. This line is actually the last line of the protocol, such as it is. At this point, both *A* and *B* can calculate the initial link key $K_{INIT} = E22(addrA, pin, inrand)$. The line labelled c. is not part of the protocol per se. Rather, it is a contrivance that will allow us to formally assert that the initial link key, K_{INIT} , is a 'secret' known only to *A* and *B*. We shall come back to this line when describing Casper security specifications in section 4.2.6.

```
#Protocol description

-- initial phase (generation of K_INIT - in which A gives B 'inrand')

0a. -> A : pin, B
0b. -> B : pin

< addrA := Addr(A) >
1. A -> B : inrand

-- A and B calculate kinit as E22(addrA,pin,inrand)

< addrA := Addr(A) >
c. B -> A : secret (+) E22(addrA,pin,inrand)
```

Figure 9: The SPL description of the generation of K_{INIT}

4.2.5 The Casper model of the System + Intruder

Having described the different roles of a protocol, we can now define our *System*. A *System* consists of a number of *processes* corresponding to roles of the protocol (as defined in the Processes section) instantiated using actual variables for each process variable. For example, the process *INITIATOR* is instantiated using actual variables *Alice* and *INRAND* for process variables *A* and *inrand*. These processes are specified in the *System* section of the .spl file. Casper uses a newline to separate processes running concurrently, and semicolons to separate processes running sequentially. For example, in figure 10 we have our *System* from init.spl which consists of an actual Bluetooth device called *Alice* running the *INITIATOR* process concurrently with an actual Bluetooth device named *Bob* running the *RESPONDER* process³.

```
#System

INITIATOR(Alice, INRAND)
RESPONDER(Bob, Secret)
```

Figure 10: The initial link key generation System

Hitherto, there has been little mention of modelling the malicious 'Intruder' in Casper (in particular, we note that there is no user-defined role of 'Intruder'). The reason is that most of the work involved in modelling the Intruder is done for us by Casper. In fact, the analyst has only to name the Intruder (in this case, it will be one of the actual Bluetooth devices) and specify the Intruder's *initial knowledge* - i.e. what is known to the Intruder before a protocol run begins - and Casper does

³We have followed convention by using the names *Alice* and *Bob* for our two honest Bluetooth devices, and *Mallory* for the (malicious) Intruder device. In addition, we needed a fourth (compromised) device, which we named *Colin*.

the rest. The naming of the Intruder, and the specification of its initial knowledge is done in the *Intruder Information* section of the .spl script. We shall follow Casper convention by naming our Intruder *Mallory*. In figure 11, we have our Intruder Information section from init.spl. Here *Mallory* is named as the Intruder, and his initial knowledge includes, among other facts, the identities of *Alice* and *Bob*, the Bluetooth address function *Addr* and a random number *RANDM*.

```
#Intruder Information

Intruder = Mallory
IntruderKnowledge = {Alice, Bob, Mallory, Addr, RANDM}
```

Figure 11: Specifying the Intruder, *Mallory*

We shall not go into the technicalities of how Casper compiles the Intruder device, *Mallory*, to CSP. Instead, we shall describe the capabilities that Casper empowers its Intruder with. This is an important consideration, because we want to be sure that the Casper model of the Intruder is sufficient, in that it is as powerful as any malicious device operating in a real-world Bluetooth environment. If it is not, then we want at least to be clear about what type of attacks are not covered by our Casper analysis.

In order to understand *Mallory*'s capabilities, it is important to first understand how Casper models the successful communication of a message, m , say, from one honest device, A , to another honest device B . This is realised through two distinct atomic CSP events which translate into English precisely as: " A sends message m intended for B ", and " B receives message m , ostensibly from A ".

Immediately after A sends message m to B , m is 'overheard' by *Mallory*. Then, using m together with previously overheard messages and his initial knowledge, *Mallory* may be able to *deduce* some new facts. Here, the rules of deduction are formally defined algebraic equivalences between expressions involving user-defined data types, and Casper's built-in operators such as exclusive-or, hash functions, and encrypt and decrypt. This model adheres to the Dolev-Yao model, in which encryption⁴ is assumed to be perfect, and hash-functions are collision-free.

Now suppose *Mallory* has just deduced some new fact, f . Then he may send f to any honest party that is willing to accept it. Moreover, the receipt of *Mallory*'s message is immediate, or *hasty*. This means that, relative to the sending and receipt of messages between any two honest parties, *Mallory* may forward, insert, and replay any number of deduced messages, subject only to the proviso that there is always at least one honest party willing to accept them.

How does this Casper model of the general Intruder differ from a malicious device operating in a real-world Bluetooth environment? It seems that the only way in which the theoretical Intruder, *Mallory*, is weaker than a real-world counterpart is in the fact that an honest party is not obliged to take receipt of any message - either from *Mallory* or an honest party - unless it is actually willing to do so. This is a subtle point, which we will now expand on. As mentioned above, in the CSP world, the 'receipt of a message' is modelled as an atomic CSP event, but the intended recipient may choose whether or not to engage in that event. However, in the real wireless world an agent would, of course, have no choice in the matter - buffer-size permitting, it will receive all messages transmitted in the bandwidth it is listening over at the time, although it may subsequently choose to discard some of those.

The net effect of all of this is that, in the Casper model, the receipt of a message sent by one honest party to another honest party cannot be delayed indefinitely with respect to the sending and receipt of any other messages. Let us consider how this impacts on the completeness of our Casper-based analysis.

First, it means that the Casper model of a protocol run abstracts away from the most primitive of denial of service attacks, in which an honest party is simply bombarded with junk messages to

⁴ Including Vernum encryption, i.e. bit-wise xor-ing with a nonce.

the extent that one or more legitimate messages never get through. Ultimately, though, there is nothing that one could do in practise to counter this basic type of denial of service attack with 100% effectiveness (although measures can be taken to alleviate the problem), and, certainly, Bluetooth does not specify measures to guard against such attacks. Thus there would seem to be little lost in Casper abstracting away this problem. That said, we do still want our analysis to show up more intelligent DOS attacks. This though, raises the question of what we mean by an 'intelligent DOS attack'. If we simply want 'not a primitive DOS attack as above', then our definition of what constitutes an 'intelligent DOS attack' could be as follows. It is one in which either: (i) the Intruder *deadlocks* the System before it can successfully terminate, and only a finite number of visible actions are performed by any agent in the process, or, (ii), the Intruder *livelocks* the System (i.e. there is always one honest agent ready to perform some action) so that it never successfully terminates, but all messages between honest agents *eventually* get through. Certainly any attack expressible in Casper falls into that category. Crucially, the converse is also true - with the proviso that we consider Systems involving only a bounded number of runs of the Bluetooth bonding process. The proviso, although not a serious limitation⁵, would, for example, rule out analysis of the interesting DOS attack of 3.6. There the Intruder forces the bonding process to repeatedly fail, with the interval between each new attempt subject to exponentially increasing delays. That attack would not be expressible in Casper, although there is no reason why it could not be expressed in a bespoke CSP model.

Second, it means that the Casper model of the Intruder cannot both overhear a message, but at the same time delay the delivery of that message to the intended recipient indefinitely. This model of an intruder has been investigated as part of the work-package 1 study in authentication models. Such an assumption is reasonable for peer-to-peer systems where parties are within broadcast range of each other, assuming that it is not physically possible to build a device that can both electronically jam a transmission, and eavesdrop on that transmission at the same time.

However, it is entirely feasible for one (malicious) device to physically block a wireless transmission from one other device to another, perhaps relaying that transmission at some later time - although, arguably, it is unlikely that the Bluetooth user would not notice his device being physically blocked from another device in this manner.

Where devices are out of direct broadcast range we can envisage the following, quite advanced, scenario which one might refer as a case of 'malicious boosting'. Device *A* knows of, and wants to bond with device *B*, but *B* is just outside of *A*'s transmission range. In the proximity is a malicious device, sitting about mid-way between *A* and *B*. At the very least, the malicious device may act as a kind of 'booster', selectively relaying any transmission it chooses from *A* and *B*. It is not clear to what extent this may be exploited, and this is out of scope of our current analysis.

4.2.6 Specifying security properties in Casper

Casper has a number of pre-defined security specifications that can be asserted of a System. There are three sorts of specification, namely *secrecy*, *agreement* and *liveness*. We used the pre-defined Casper secrecy and agreement specifications, we also wrote one bespoke CSP specification (to determine whether the Intruder could influence the generation of an encryption key - see Q6, section 4.5.3.).

Specifications are listed in the Specifications section of the .spl script. In figure 12, we have the secrecy specifications for our init.spl script. These assert that $(B)ob$ is right in believing that *secret* is a secret known only to himself and $(A)lice$, and, similarly, that $(A)lice$ is right in believing that *secret* is known only to herself and $(B)ob$.

Some explanation is required here as to what, exactly, *secret* is. One of the idiosyncrasies of Casper is that it does not allow some fact *f* (in this case the initial link key K_{INIT}) to be asserted a 'secret' between parties *A* and *B*, unless *f* has appeared itself as a field in some message that has been transmitted during a run of the protocol. Not surprisingly, though, the initial link key, K_{INIT} , itself never appears in a message transmitted as part of the initial link key generation stage

⁵ If necessary, we could always try to write a pen-and-paper proof to extrapolate from machine-tested bounded cases to unbounded cases

of the Bluetooth initialization procedure (although it is used in the next stage, where it is used as a temporary encryption key in order to securely generate the link key). So, in order to assert that K_{INIT} is a secret, we need an additional message, at the end of the *natural* run of the protocol, in which our fact f (here the expression $E22(addrA, pin, inrand)$) is transmitted as part of a message from $(B)ob$ to $(A)lice$. Care must be taken though, because this last message is *not* part of the protocol and we do not want it either artificially enhancing or constraining behaviours possible under the real protocol.

A naïve solution would be to have B send f to A in plain-text, but then *Mallory* would overhear f immediately, even if he would not otherwise have been able to deduce f before the natural end of the protocol run.

Our solution is to introduce a variable, *secret*, that appears only as a fact known a priori to all honest parties, but not to the Intruder, *Mallory*. This variable *secret* appears nowhere else in our protocol description other than in our final transmitted message, which is:

$$B \rightarrow A : secret \oplus f \quad (4)$$

where ' \oplus ' is the bit-wise exclusive-or operator. We can then specify in Casper that *secret* is known only to A and B . Given that *Mallory* will have overheard $secret \oplus f$, but should never have been able to deduce *secret*, the assertion that *secret* is a secret between A and B is equivalent to saying that f is a secret between A and B - as required. It is a bonus that the former is a generic specification that does not explicitly mention our real 'secret', f . As a consequence, we may have many lines of the form (4) at various points of our protocol description, each referring to some different f , but only the two lines of the specification of 12 are required in order to assert that all of those f s are secrets between A and B .⁶

#Specification

```
-- the product K_INIT = E22(addrA,pin,inrand) should be a secret between
-- A and B
```

```
Secret(B, secret, [A])
Secret(A, secret, [B])
```

Figure 12: Specifying that K_{INIT} is a secret between A and B

4.3 Caspose

This section describes the motivation, requirements and design of a simple tool, *Caspose*, developed for use with our analysis of the Bluetooth initialization procedure but potentially useful for future work.

The motivation for *Caspose* arises from our desire to model and analyse the Bluetooth initialization procedure within Casper. As Casper was only designed to handle fairly static protocols, and the initialization procedure is highly schematic in nature, modelling the procedure within Casper is fraught from the outset. Actual runs of the initialization procedure may vary substantially, and the questions we wish to ask (see section 4.5) may only be applicable to particular configurations. To capture this variance we chose to separate out the configurations into separate Casper scripts - an argument must then be made as to the coverage of the scripts. In this context, *Caspose* enables the

⁶ However, if the FDR check were to fail, then it tells us only that one of the facts, f , that we wanted to remain a secret was learnt by *Mallory*, but it does not tell us which one. In order to ascertain which fact was learnt by *Mallory*, we would need to replace the two *secret* assertions by a series of individual assertions explicitly citing the different facts that we wanted to be secrets. FDR's trace counter-example would then tell us precisely which of the facts *Mallory* learnt, and how, rather than just that one of them was learnt.

separation of the specification of individual stages from the specification of configurations (or rather scenarios).

When writing Caspose, we were mainly concerned with issues of flexibility and reliability. In terms of flexibility, the tool should facilitate the production of a large number of configurations, each a valid and meaningful casper script. Here, a configuration is phrased in terms of a sequence of selected protocol sections and the parameterisation of individual sections. We also invariably want to specify details particular to a configuration that are hard to formulate, for example the System and Specification sections. In terms of reliability, the tool should not significantly diverge from the reliability of Casper itself. To this end the tool resembles an intelligent filter, processing and producing casper scripts.

For the Bluetooth initialization procedure, we identified four areas of parameterization:

1. the assignment of agents to roles for a given stage, i.e. reversal of agents A and B in protocol steps.
2. the System section, e.g. number of protocol runs per role, parallel or sequential, actual variables.
3. the Specifications section, e.g. what is expected of a given configuration.
4. the Intruders initial knowledge at the beginning of each stage of the protocol, e.g. could Mallory possibly deduce a combination key by later learning the initial link key?

In our experience of using Casper we were presented with certain limitations in the Casper language, namely with respect to the use of Casper assignments. So, finally, we wanted a capability in Caspose to overcome such limitations, for instance a powerful preprocessing capability.

4.3.1 Design

As hinted to above, the current design of *Caspose* is based upon the concept of a filter that takes a sequence of protocol sections and glues them together with a little intelligence to produce a complete Casper protocol description. If we assume a common environment of free (session) variables, and if the System and Specification sections are written by hand it is a simple matter to then produce a complete Casper script given a header, a footer and a System + Specifications file.⁷

The header file contains the generic script commentary and free variables section while the footer file contains the actual variables, Intruder section and as well as any function or equivalence sections. These two files form the start and end of all Casper scripts produced and are currently simply copied verbatim. As the footer currently includes the Intruder section, any configuration requiring a special Intruder section must use a modified footer file.

The protocol section files (.cspl) each specify an individual stage of the initialization procedure, for example the authentication stage. Each file just contains a Casper *Processes* and *Protocol Description* section as described in sections 4.2.3 and 4.2.4. The Processes section should cover the Protocol Description, mapping the agent identifiers to processes. All session variables given to an agent should either be listed in the process arguments or passed via environmental inputs (Caspose will handle any redundancy). In this way, the protocol section files are intended to be self-contained.

To address certain limitations in Casper we have added macro variable assignments to the Protocol Description language. These are specified analogously to normal Casper assignments but are prefixed with a "!" as in "! < a := b ; c := a (+) z >". While macros are clearly dangerous in the wrong hands, they provide a powerful mechanism to enable us, for example, to define the expression denoting the link key *k* for later stages such as authentication. Macros are applied to messages in the order they are declared and their declaration commented in the resulting Casper script.

When composing protocol sections, Caspose keeps a record of the set of facts a given agent currently knows (including facts learnt through process definitions and received messages). This

⁷ It should be noted that the Processes section is produced automatically.

allows Caspose to use some intelligence over initialisation issues and results in better flexibility when reversing protocol descriptions, swapping them round and so on. For instance, Caspose assumes that prior knowledge of a session variable, perhaps through an earlier message⁸, obviates the need to input it via the environment. Any alterations are highlighted in comments.

4.4 SPL descriptions of the individual stages of the initialization procedure

This section describes the modelling of the Bluetooth initialization procedure, also known as *pairing* - in which two devices establish a shared *link key*. The procedure assumes a shared - presumably secret - PIN which both users enter into their respective devices on request. An initial link key, K_{INIT} , is then derived from the PIN, a random number and the master's Bluetooth address. Then, using this initial link key, and depending on preferences, a *unit* link key or *combination* link key is generated and exchanged. Both parties then achieve mutual authentication of the resulting link key by a simple challenge-response protocol. Finally, if required, an *encryption key* may be generated and exchanged.

In the following descriptions of the individual stages, each stage is assumed to operate in a common environment of free variables (pin , $inrand$, $Addr$ etc) so that it becomes a fairly simple matter to piece together protocol sections (as in figure 9) to create a complete protocol. Stages may be parameterised by reversing roles ($A \leftrightarrow B$) or pre-defining inputs (such as the link key, k , referred to in the authentication stage).

Throughout our scripts, we have endeavoured to keep our naming convention as close as possible to that of the original Bluetooth specification document, [Blu03]. For each stage of the protocol, we cite the pages of the specification which acted as our source for the modelling.

4.4.1 Generation of the initial link key, K_{INIT}

In the following protocol both agents are supplied their PIN, pin , while agent A is instructed to initiate with agent B . Agent A then generates and sends a random number, $inrand$, to agent B in plain-text.

To check the secrecy of the resulting initial link key K_{INIT} , $E22(addrA, pin, inrand)$, we add an additional protocol step not present in the real protocol. Here we tangle the initial link key with an assumed secret $secret$, as described in section 4.2.6.

```
#Processes
```

```
INITIATOR(A,inrand)    knows Addr
RESPONDER(B,secret)   knows Addr
```

```
#Protocol description
```

```
-- initial phase (generation of K_INIT - in which A gives B 'inrand')
```

```
0a.  -> A : pin, B
0b.  -> B : pin
```

```
< addrA := Addr(A) >
1. A -> B : inrand
```

```
-- A and B calculate kinit as E22(addrA,pin,inrand)
```

```
< addrA := Addr(A) >
```

⁸ The parsing of Casper messages uses the BNF syntax as given in [Low99].

c. B → A : secret (+) E22(addrA, pin, inrand)

Note that the assignment to the free variable *addrA* - a parameter to *E22* - must be performed locally for each agent. Recall that the assignment statements, $\langle \dots \rangle$, immediately precede a protocol step in which the assigning agent is the sender. Recall also that the operator \oplus denotes bit-wise exclusive-or.

See [Blu03, page 756] for the original Bluetooth protocol specification.

4.4.2 Generation of a unit key, K_A

In the following protocol, agent *A* generates a unit link key using the Bluetooth cryptographic hash function *E21* and sends it (XOR encrypted) to agent *B* using the established initialization key, K_{INIT} . The link key, *k*, is defined using a macro assignment as the expression denoting *k* involves the use of Casper features incompatible with normal assignments (hash functions and masking). Again, an extra artificial protocol step is added at the end to assert the secrecy of the resulting unit link key.

#Processes

```
INITIATOR(A, addrA, inrand, pin, lkrandA)
RESPONDER(B, addrA, inrand, pin, secret)
```

#Protocol description

```
-- generation of unit link key (K_A = K_BA)
-- (in which A generates then sends the key E21(...) encrypted by K_INIT)
```

```
0. → A : B
```

```
-- note the use of 'kba' as B's mask variable.
!< k := E21(lkrandA, Addr(A)) % kba >
```

```
1. A → B : (k) (+) E22(addrA, pin, inrand)
```

```
-- 'kba' is B's mask variable.
!< k := kba % E21(lkrandA, Addr(A)) >
c. B → A : secret (+) (k)
```

Note in the above the use of the Casper operator $\%$ in the two macro assignments (i.e. $\langle \dots \rangle$). The $\%$ operator (known as 'masking') is used when the sender or receiver are required to treat a message as opaque, i.e. as a sequence of bits. Here, agent *B* interprets the unit key *k*, $E21(lkrandA, addrA)$, as simply the variable *kba*. Agent *B* does not know the random number *lkrandA*, and, therefore, cannot check for the correct unit key, but it can, nonetheless, still accept any message with matching type (e.g. number of bits, fields).⁹

See [Blu03, pages 756–757] for the original Bluetooth protocol specification.

4.4.3 Generation of a combination key, K_{AB}

In the following protocol, agents *A* and *B* exchange (XOR encrypted) their contributions, respectively *lkrandA* and *lkrandB*, to the combination link key. Again the link key, *k*, is defined using a macro assignment, and again an extra artificial protocol step is added at the end to assert the secrecy of the resulting combination link key.

⁹ It is a minor, but important, point that *B* will *only* accept messages of type $E21(RAND, BDADDR)$ - attacks exploiting type flaws will not be detected.

```
#Processes
```

```
INITIATOR(A,addrA,inrand,pin,lkrandA)      knows Addr
RESPONDER(B,addrA,inrand,pin,lkrandB,secret) knows Addr
```

```
#Protocol description
```

```
-- generation of combination link key (K_AB = K_BA)
-- (in which A and B swap 'lkrandA/B' encrypted by K_INIT)

0.  -> B : A

-- B sends A lkrandB (using K_INIT = E22(addrA, pin, inrand))
1. B -> A : lkrandB (+) E22(addrA, pin, inrand)

-- A sends B lkrandA (using K_INIT = E22(addrA, pin, inrand))
!< k := E21(lkrandB, Addr(B)) (+) E21(lkrandA, Addr(A)) >
2. A -> B : lkrandA (+) E22(addrA, pin, inrand)

-- A and B calculate K_AB/BA as E21(lkrandB, addrB) (+) E21(lkrandA, addrA)

!< k := E21(lkrandB, Addr(B)) (+) E21(lkrandA, Addr(A)) >
c. B -> A : secret (+) k
```

See [Blu03, pages 757–758] for the original Bluetooth protocol specification.

4.4.4 Authentication of current link key

The following protocol represents one half of the mutual authentication of the current link key, k , for agents A and B . Here agent A , the *verifier*, issues a challenge, $aurandA$, in plain-text, which agent B , the *claimant*, must correctly respond to, thus demonstrating knowledge of the current link key, k . In the response, agent B uses the Bluetooth cryptographic function $E1$ to produce a signature of the challenge, but only sends the last 32 bits of the result (known as *SRES*).

```
#Processes
```

```
VERIFIER(A,aurandA)      knows Addr
CLAIMANT(B,A)           knows Addr
```

```
#Protocol description
```

```
-- one-way authentication (B to A) of the current link key K.
-- (in which A challenges and B responds using K)

-- give A & B their current link key.
0a. -> A : k, B
[ A != B ]
0b. -> B : k

-- send challenge.
!< e1 := E1(k, aurandA, Addr(B)) >
1. A -> B : aurandA
```

```
-- send response (just SRES of e1).
-- note:
--   SRES(e1) (a hash value) corresponds to the last 32 bits of 'e1'.
--   if you also know ACO(e1) (first 96 bits), then you can deduce 'e1'.
!< e1 := E1(k,aurandA,Addr(B)) >
2. B -> A : SRES(e1)
```

In the above, we model the Bluetooth function *SRES* (that returns the last 32 bits of its argument) as a hash function. This provides the basic semantics (everyone knows *SRES*, can apply it and so on). We can then add additional information, such as the deduction of *e1* from *SRES(e1)* and *ACO(e1)* (the sister function to *SRES* that returns the first 96 bits of its argument). If the cryptography allows, we may even add Casper Equivalences such as $SRES(e1) = SRES(e2)$; then agent *A* may accept *SRES(e2)* as a valid response, and the authentication fails.

See [Blu03, page 756] for the original Bluetooth protocol specification.

4.4.5 Generation of an encryption key, K_C

The following protocol represents the optional generation of an encryption key, *kc*, prior to the use of encryption. Here agent *A*, the *master*, sends a random number *enrandA* to agent *B*, the *slave*. Both agents then calculate the encryption key, K_C , using the Bluetooth cryptographic function *E3*, the current link key, the random number and the first 96 bits (known as *ACO*) of the *e1* previously established in the authentication stage.

```
#Processes
```

```
MASTER(A,k,e1,enrandA)
SLAVE(B,k,e1,secret)
```

```
#Protocol description
```

```
-- generation of encryption key K_C.
-- (given shared link key 'K' and product of E1 'e1').

0.  -> A : B

-- give SRES to Intruder (can't do so through IntruderKnowledge)
-- 0a. A -> B : SRES(e1)

-- leak ACO to Intruder.
-- 0b. A -> B : ACO(e1)
-- 0c. A -> B : secret (+) e1

-- A sends enrandA to B.
-- note: the BT spec (1.2) does not state how enrandA is sent to B.
-- we have not been able to determine how, and therefore assume plain-text.
1. A -> B : enrandA

-- both A & B calculate K_C as E3(k,enrandA,ACO(e1))

-- test secrecy of encryption key K_C = E3(k,enrandA,ACO(e1))
!< kc := E3(k,enrandA,ACO(e1)) >
c. B -> A : kc (+) secret
```

N.B. The original Bluetooth Specification [Blu03] is ambiguous as to how the random number, $enrandA$, is transmitted to the slave. However, at least one source [Yan, section 5.1] claims that it is transmitted in plain-text, and we have assumed that in our model. We have raised this issue with the Bluetooth user group, and will correct our models if and when it is appropriate to do so (see question **Q1a**. for a discussion of the possible implications).

N.B. There is also ambiguity over which authentication the product of $E1$, $e1$, originates from given that the devices may undertake a mutual authentication, i.e. two authentications. We have not raised this issue with the Bluetooth user group and do not consider it a major issue given that we have not analysed mutual authentication as yet.

See [Blu03, page 758–759] for the original Bluetooth protocol specification.

4.5 Results of analysis

In this section we present the findings of our Casper-based analysis. The analysis is question-driven. Bluetooth does not formally assert any properties of its Link-Layer security that we could verify of our models, so we instead looked at a number of questions motivated by the same scenarios/implementations considered in Section 2. Our report is structured as follows:

First, a question is quoted. When a question was posed by someone other than a member of the analyst team, it is recorded here almost exactly as it was originally phrased. For many of those questions, the first job of the analyst was to ascertain what it was, precisely, that was being asked.

Precise questions were translated into formal CSP specifications that we could check our model of the System against by way of FDR refinement tests. As we shall see, some of those tests failed, implying that the System under scrutiny can reach some undesirable state, possibly as a result of malicious behaviour.¹⁰ If a refinement test has failed, then FDR offers (in order of length) all the possible traces of events - each corresponding to the sending or receiving of some message - leading up to that failure state. In other words, each trace cites the exact details of some particular ‘attack’.¹¹ In turn, we can then input any of those traces to Casper’s built-in interpreter, which translates pure CSP traces into a more readable format (cf. 4.5.2). We have included the Casper interpretations of some of the more interesting failed tests. The intention here is that the undesirable behaviour can be duplicated in our Bluetooth test-bed and be subject to further scrutiny.

It turned out that some of our questions could not be suitably translated into CSP specifications (e.g. Q6). In such cases, we endeavoured to address the question through an ‘off-line’ formal argument short of automated proof.

Some of the questions remained unanswered at the time of the completion of this report, usually this was because of time, or because our modelling at present could not satisfactorily capture the question. Those questions are listed as ‘open questions’, one of our recommendations being that they be duly addressed in follow-on work.

4.5.1 Caveats

The Bluetooth security white paper makes a point of advising against a device running the pairing procedure with two other devices in parallel (cf. [Blu03, pages 773–774]). Unless stated to the contrary, all of our System models involve two honest Bluetooth devices both running all, or part of the initialization procedure *sequentially* in the presence of the malicious agent described in 4.2.5.

We make no claims about the behaviour of Systems subsequent to our runs of the initialization procedure as stated. If subsequent practical investigations suggest this is a significant area of risk then future research will address behaviours after the initialization procedure.

¹⁰ For “undersirable”, read “a state contrary to the states allowed for by the specification”.

¹¹ Which is considerably more helpful than merely being informed that it is possible (somehow) to reach some undesirable state as a result of some (unspecified) attack. This is often cited as one of the advantages that model checking has over theorem provers - the information can be a great help to the analyst or developer in finding a solution to the problem, or to replicating the undesirable behaviour in a test-bed, for example.

4.5.2 Casper interpretations

Some explanation is required as to Casper's very particular interpretation of FDR/CSP trace-failures. Each such 'interpretation' consists of a number of lines, each being one of the four forms below.¹²

A line of the form:

$$label. \rightarrow receiver : f_1, f_2, \dots, f_n \quad (5)$$

means, precisely, a message (labelled *label*) with payload f_1, f_2, \dots, f_n sent by the environment (user) just having been received by *receiver*. The Intruder cannot overhear this message.

A line of the form:

$$label. I_sender \rightarrow receiver : f_1, f_2, \dots, f_n \quad (6)$$

means, precisely, a message (labelled *label*) with payload f_1, f_2, \dots, f_n sent by the Intruder in *sender*'s name just having been received by *receiver*. (In particular, we may infer from this that the Intruder must have been capable of deducing f_1, f_2, \dots, f_n .)

A line of the form:

$$label. sender \rightarrow : f_1, f_2, \dots, f_n \quad (7)$$

means, precisely, a message (labelled *label*) with payload f_1, f_2, \dots, f_n just having been sent by *sender* to the environment (user). The Intruder cannot overhear this message.

A line of the form:

$$label. sender \rightarrow I_receiver : f_1, f_2, \dots, f_n \quad (8)$$

means, precisely, a message (labelled *label*) with payload f_1, f_2, \dots, f_n just having been sent by *sender* and intended for *receiver*. The intended receiver is yet to receive it, although the Intruder has overheard it. (In particular, the Intruder now knows all of f_1, f_2, \dots, f_n .)

Lastly, the standard notation:

$$\{x_1, x_2, \dots, x_n\}\{k\}$$

is used to mean the items of data x_1, x_2, \dots, x_n encrypted using the key k .

4.5.3 Results

Our analysis is presented as a series of security questions and answers. Before presenting the results some introduction to the method of analysis is required. For each question, we briefly cover our interpretation, detail the particular configurations and systems used in the analysis, and then present the results. As the choice of configurations and systems are very important, the following paragraphs discuss the major decisions we took and their rationale.

While it would be desirable to include all stages of the initialization procedure for each configuration, this is not currently practical as the model checker FDR cannot handle a complete run. This is mainly due to the size of the expression denoting the encryption key, and is more a feature of Casper than of the complexity of the full protocol run.¹³ More importantly, we can, and do, model the procedure up to the authentication stage. When modelling the encryption stage, we currently proceed from the authentication or link key generation stages.

When asked to determine the information required by the Intruder to break security properties, we have taken the approach of adding the candidate information to the Intruders initial knowledge. An inspection of the traces of any failures then reveals 'when' the Intruder first requires the information. Alternatively, we could have provided the information dynamically via a CSP process. We have not explored this option as the simple approach through Casper features suffices for our purposes.

To simplify and reduce the number of resulting failures raised we have chosen to use 'strong' secrecy, i.e. secrets must remain secret even over incomplete protocol runs. Of course, a given

¹² We took liberties in our paraphrasing of Casper interpretations only in that very minor syntactic changes were made to some tags/names so as to ease readability further.

¹³ The process modelling agents handling the encryption key generalises over all possible session parameters (*pin*, *inrand* etc) and as a result FDR has difficulties compiling the process.

failure must then be interpreted with regard to the success of failure of the complete protocol run. Due to time constraints we have only reported failures that we find interesting. For example, after investigating basic failures resulting from knowledge of atomic facts we investigated weakening the knowledge to compound facts. As Casper only allows the analyst to specify atomic facts in the Intruders initial knowledge, this was achieved via dummy messages.

When considering the appropriate system of honest agents to analyse, Casper provides several options as described in section 4.2.5. Normally, the system will simply consist of one initiator and one responder running in parallel. However, we may add extra initiators or responders when checking for multiplicity attacks (on authentication) or when an attack depends on an honest agent reusing critical knowledge such as a secret key. We may choose between parallel or sequential processes and experiment with the reuse of actual variables. In particular, the use of sequential composition enables us to explore scenarios in which an honest agent may withdraw and restart after the failure of a previous run.

The configurations and systems used and referred to in the following questions are as follows. The basic configurations are:

- `auth` - initial + combination key + authentication
- `unit_auth` - initial + unit key + authentication
- `auth_encrypt` - authentication + encryption

To these, 'auth' and 'unit_auth' add three more variants 'var1', 'var2' and 'var3' corresponding to a reversal of the link key stage, reversal of the authentication stage and reversal of both stages respectively. In addition to the standard system of one initiator and one responder there are also three systems 'sys1', 'sys2' and 'sys3'. In total there are sixteen combinations for both 'auth' and 'unit_auth', managed by a make file.

The variant systems 'sys1', 'sys2' and 'sys3' each add extra copies of either the initiator or responder, reusing *inrand* (the nonce used in the initial stage) and the nonces used in the link key stage. However, the challenge used in the authentication stage (which is assumed to be fresh) is never reused. As the *pins* are decided by the environment, these may also be reused and we may expect security failures as a result. The first system 'sys1' adds an extra copy of the initiator (in parallel), 'sys2' adds an extra copy of the responder (in parallel) and 'sys3' is as 'sys1' but using sequential composition and the option of withdrawl. Clearly, there are many other systems but our preliminary analysis considers these three.

For the 'auth_encrypt' configuration we add similar variants over the reversal of the encryption stage and similar systems. Although the configuration includes the authentication stage, we do not currently assert that authentication is maintained through the encryption stage as the sending of the random number, *enrandA*, is assumed to be plain-text.

Q1a. Is it possible that an initial link key, K_{INIT} , can ever be deduced by the Intruder without any initial information?

In the question above, we interpret the term 'ever' to range over the scope of the initialization procedure, and interpret the deduction of the initial link key in terms of a secrecy specification over (complete) protocol runs. In other words, either party must have completed a protocol run before they claim a given fact as secret. It is arguable whether failures of secrecy for incomplete runs should be considered as attacks, as the agent involved should not normally use the secret after an incomplete (i.e. failed) run.

In terms of configurations, we have used the 'auth' and 'unit_auth' configurations including all variants and systems as described above. This does not include the final optional encryption stage for the reason cited at the beginning of the section. However, it is unlikely that the inclusion of the stage would have any effect, as in terms of secrecy it simply involves the transmission of a random number, *enrandA*. It should be noted that the means of transmission is not specified in the Bluetooth Specification (raised in section 4.4.5). Our assumption of plain-text transmission does

not effect secrecy but other means could conceivably. For instance, an ostensibly secure use of XOR encryption and the initial link key could well give the Intruder additional unforeseen capabilities, i.e. this would allow the Intruder to re-use encrypted messages from the link key generation stage and conceivably learn something about the link key from a genuine response.

Our analysis of the 'auth' and 'unit_auth' configurations given the four variants and four systems detailed above did not produce any failures of secrecy, implying an inability of the Intruder to deduce the initial link key, K_{INIT} from observing the protocol run.¹⁴

- **Result 1.1: We were unable to demonstrate any failure of secrecy (deduction of the initial link key, K_{INIT}) over the course of the initialization procedure for the range of situations given by our four systems.**

Q1b. What information would be required by the Intruder, and when, in order for him to be able to deduce the initial link key, K_{INIT} ?

Our interpretation of the question above follows on from the previous question. In particular, the term 'when' is taken to be relative to the steps of the initialization procedure and information is taken to include basic session parameters such as the *pin* or random number *lkrandA*. We assume the Intruder has access to all previous messages and perform our analysis by providing the Intruder with candidate information from the outset (looking at the traces of any failures then tells us 'when' the Intruder first requires the information).

In terms of configurations, we again use the 'auth' and 'unit_auth' configurations including all variants and systems as described above. As described at the beginning of the section, when asked to determine the information required by the Intruder to break security properties we have taken the approach of adding the candidate information to the Intruders initial knowledge. For the candidate information we can choose from atomic facts such as the *pin*, random numbers *lkrandA*, *lkrandB*, *aurandA* or compound messages, although most compounds can simply be deduced by the Intruder given the appropriate atomic facts.

At this point it is important to discuss how the Intruder may verify his deduction of the initial link key given that the deduction may rest upon a probable 'guess'. Although the initial link key is used within the link key generation stage, the Intruder must wait until the authentication stage to confirm his knowledge of the initial link key. This is because of the nature of the XOR encryption used in the link key generation stage - guessing the initial link key, K_{INIT} provides only a guess for the unit key or combination key.

The following are common-sense observations over simple runs of the initialization procedure.

- **Result 1.2: Knowledge of *inrand*, the initial random number used within the initial link key, clearly does not comprise secrecy as it is sent in plain-text and the Intruder has no opportunity to use it prior to its release.**
- **Result 1.3: Knowledge of the shared *pin* clearly compromises secrecy as it is the only unknown in the expression, $E_{22}(addrA, pin, inrand)$, denoting the initial link key.**
- **Result 1.4: Knowledge of either *lkrandA* or *lkrandB* again clearly compromises secrecy as it allows us to decrypt the messages of the link key generation stage. Again, it is fairly easy to see that *prior* knowledge (and active use) of these random numbers will not help.**

Q2a. Is it possible that a link key (either combination key or a unit key) can ever be deduced by the Intruder without any prior information?

In the question above, we use the same configurations and systems as in **Q1a**.. Again, we interpret the term 'ever' to range over the scope of the initialization procedure and interpret deduction of the link key in terms of a secrecy specification over (complete) protocol runs.

¹⁴ It should be mentioned that for variants 'var1' and 'var2' of the configuration 'unit_auth' Casper produces CSP scripts that do not compile, seemingly due to a bug in the tool over mask variables. The scripts affected have been patched to fix the bug and as such are included in the analysis.

There is one important difference from the deduction of the initial link key regarding the reuse of link keys. While the Bluetooth specification insists that initial link keys should be discarded once a common link key has been established, for link keys each repeat connection between two devices should proceed from the authentication stage reusing the common link key. There is an obvious and well known weakness¹⁵ with unit link keys in this respect as the unit link key is usually created once by the device and re-used for each connection with a new device.

In our analysis we have not considered scenarios in which the Intruder plays a legitimate role (with the exception of the scenarios of section 4.6), and as such the obvious attack on unit keys above is not covered. However, we do consider other scenarios pertaining to the reuse of unit keys and combination keys in the various systems described at the beginning of the section, i.e. reuse of random numbers. For instance, the system 'sys1' involves adding a copy of the initiator while reusing session variables (except of course the authentication challenge *aurandA*). Unfortunately, we have not had time to directly approach the issue of reuse of link keys and skipping of previous stages - the system 'sys1' models this indirectly by re-using the same session variables for previous stages, i.e. it produces redundant messages we may ignore.

Again, our analysis of the various configurations given did not produce any failures of secrecy, implying an inability of the Intruder to deduce either a combination or unit link key. This result is to be expected given the apparent inability of the Intruder to deduce the initial link key, K_{INIT} . Indeed, the initial link key and link keys are connected intimately through simple XOR encryption and then only used later within the Bluetooth cryptographic functions $E1$ and $E3$. For instance, having overheard the generation of the link key and learning the initial link key the Intruder may then deduce the link key itself. Similarly, learning the *unit* link key allows deduction of the initial link key.

In our analysis of the initialization procedure we have not considered typing attacks where one message type may masquerade another, perhaps even originating from a different protocol. Such issues may be of relevance to the generation of unit and combination link keys as both stages contain similar messages - a random number XOR encrypted with the initial link key. In the case of the unit key stage this is the unit link key itself (an unknown to the recipient) while for the combination key stage it is taken to be a contribution to the resulting combination link key. Although this potential for confusion is interesting and quite possibly within the scope of Casper we have not investigated it but reserve it for our recommendations of section 4.8.

- **Result 2.1: We were unable to demonstrate any failure of secrecy (deduction of the link key - combination or unit) over the course of the initialization procedure for the range of situations given by our four systems.**

Q2b. What information would be required by the Intruder, and when, in order for the Intruder to be able to deduce a link key (either combination key or a unit key)?

Our analysis of the question above follows on from questions **Q1a.** and **Q1b.**. Again we use the same configurations and systems, using the same approach. All the common-sense observations/results of question **Q1b.** hold as deduction of the initial link key allows deduction of the link key.

Q3. What knowledge is required by the Intruder to cause the authentication of a link key (either combination key or a unit key) to fail?

In the question above, we interpret 'authentication' in terms of agreement over session variables - specifically the random numbers constituting the link key and authentication challenge as well as the apparent identities and roles of initiator and responder. In terms of protocol sessions, we further require that whenever the verifier thinks he or she has successfully completed a run of the protocol the claimant has previously been running the protocol and there is a one-one relationship between the runs of the verifier and claimant.

¹⁵ A malicious device having learnt the unit link key legitimately may later pose as another device as repeat connections do not have to demonstrate knowledge of the shared PIN. For this reason and others, the use of unit keys is deprecated.

The above definition may be stronger than necessary, especially with regard to the apparent identity of the verifier¹⁶. In practice this is not a major concern as the link key generation stage serves to establish the (apparent) identities of the principals through knowledge of the shared *pin*. As such, we have made no assumptions over the identities of the agents. When authentication fails we must then interpret the failure with such considerations to determine whether the failure represents a real attack on the initialization procedure. The failure over unit key based authentication, below, is a good illustration of these issues.

As for question **Q1a.**, we have used the 'auth' and 'unit_auth' configurations including all the variants and systems. For the 'auth' scripts, agreement occurs over *lkrandA* and *lkrandB*, the combination key contributions, and *aurandA*, the challenge while for the 'unit_auth' scripts agreement occurs over just *aurandA* (the slave device never learns *lkrandA*). It should be noted that we have only examined one half of the mutual authentication, as the addition of an extra authentication stage currently has a considerable effect on the memory usage of the model checker FDR. Attacks depending on a previous authentication attempt will not be covered by our analysis. Again, we reserve this for a future analysis.

For our analysis, we initially tried checking for failures of authentication without providing any candidate information. This alone revealed some interesting failures with the unit link key authentication, i.e. 'unit_auth' configuration. However, no failures were found for the combination link key authentication - implying an inability of the Intruder to defeat authentication in this case.

For the unit link key authentication Casper produced several classes of failure. The first and simplest uses the 'var3' variant configuration (responder initiates the link key generation and authentication stages) and the ordinary system of one initiator and one responder. In the scenario, the Intruder *Mallory* manages to make *Bob* think he has completed a run of the protocol with *Alice* while *Alice* has actually been running with *Mallory*. It is arguable as to whether this constitutes an attack for two reasons. Firstly, our definition of authentication as given above may be too strong, specifically regarding the apparent identity of the verifier (i.e. *Bob*). Secondly, the scenario depends on the Intruder initially convincing *Alice* that *Bob* is *Mallory* (i.e. 'spoofing' as described later in question **Q5b.**). However, it is certainly interesting and highlights the known weakness of unit keys.

The failure produced (and interpreted) by Casper is as follows. The messages are labelled as in the script 'unit_auth_var3.spl' where *Pete* denotes a PIN, *AddrA* and *AddrB* denote the Bluetooth addresses of *Alice* and *Bob*, and *INRAND*, *RAND1* and *RAND2* are random numbers. Note that the contrived protocol lines 3 and 5 declaring the secrecy of the initial link key and the unit link key have been filtered out for clarity.

Alice's user (the environment) initiates a connection with *Mallory*¹⁷ having decided upon a PIN, *Pete*.

0. \rightarrow *Alice* : *Pete, Mallory*
2. *Alice* \rightarrow *I_Mallory* : *INRAND*

Bob's user (the environment) receives a connection request from *Mallory* claiming to be *Alice*. *Mallory* uses the same initial random number, *INRAND*, that *Alice* chose.

1. \rightarrow *Bob* : *Pete*
2. *I_Alice* \rightarrow *Bob* : *INRAND*

Having established the initial link key, K_{INIT} , *Bob* now initiates the generation of the unit link key by deriving the unit key using the Bluetooth cryptographic function *E21* and sending the key (XOR encrypted) to *Mallory* who duely relays the message to *Alice*.

4. *Bob* \rightarrow *I_Alice* : $E22(AddrA, Pete, INRAND) (+) E21(RAND1, AddrB)$
4. *I_Mallory* \rightarrow *Alice* : $E22(AddrA, Pete, INRAND) (+) E21(RAND1, AddrB)$

Although the unit key has been derived by *Bob* and contains his Bluetooth address, *Alice* treats

¹⁶ It was the conclusion of the analysts that the Bluetooth use of the word 'authentication' more concerns the link key than the identities of Bluetooth users or devices.

¹⁷ It is important to note here that *Mallory* need not know the shared pin, and may have simply spoofed *Bob's* identity, i.e. *Alice* thinks that *Bob* is *Mallory*.

the key as a sequence of bits and cannot see this. When *Bob* requires authentication, he initiates the authentication stage sending a challenge, $RAND2$, to *Alice* which *Mallory* relays to get the required response to send back to *Alice*.

6. $Bob \rightarrow I_Alice : RAND2$

6. $I_Mallory \rightarrow Alice : RAND2$

7. $Alice \rightarrow I_Mallory : SRES(E1(E21(RAND1, AddrB), RAND2, AddrA))$

7. $I_Alice \rightarrow Bob : SRES(E1(E21(RAND1, AddrB), RAND2, AddrA))$

At this point, *Bob* is confident that he has been talking to *Alice* and that *Alice* has the unit key although *Alice* has actually been running with *Mallory*. This may not be a problem - authentication of *Mallory* to *Alice* will probably fail as *Mallory* does not know the unit key and will have difficulty getting hold of an appropriate response.

The other failures exploit similar weaknesses under scenarios such as reuse of pins and random numbers, withdrawal and restarting of agents, the addition of extra parties and so on. These premises may or may not be valid, but they do allow us to analyse “what-if” scenarios similar to the above in which the Intruder initially spoofs an identity causing confusion and potential security failures.

- **Result 3.1: We were unable to demonstrate any failure of authentication when using combination link keys but were able to demonstrate several failures of authentication when using unit link keys. As discussed, it is arguable as to whether the failures constitute real attacks as our interpretation of authentication may be stronger than necessary, especially regarding the apparent identity of the verifier.**

The issue of what ‘authentication’ means is further complicated by the nature of Bluetooth’s wire-less environment. Research into authentication in pervasive environments is being undertaken by [Fwd05].

Q4a. Is it possible that an encryption key can ever be deduced by the Intruder?

Q4b. What information would be required by the Intruder, and when, in order for him to be able to deduce an encryption key?

Our analysis of the questions above again follows on from the preceding questions regarding the initial link key, link keys and authentication. All the common-sense observations raised in questions **Q1b.** and **Q2b.** still hold, as the secrecy of the encryption key depends primarily on the link key, all other constituents known to the Intruder ($aurandA$, $enrandA$). Further, from our view point restricted to the initialization procedure, the secrecy of the encryption key reduces to secrecy of the constituents as the encryption key is not used by either party within the scope of the procedure. As such, our answers to the questions above simply reference our analysis performed for questions **Q1.** and **Q2.**

As raised in section 4.4.5, there are ambiguities as to the transmission of the random number, $enrandA$, and as to which authentication (of a possible two) the product of $E1$, $ACO(e1)$, originates from. In both cases, the Bluetooth Specification simply does not detail where the values originate from. We have assumed plain-text transmission for the random number $enrandA$ (see question **Q1a.**) and have not resolved the later ambiguity as we currently only model one authentication.

Given that there are ambiguities in the encryption stage, we have prepared a configuration including the encryption stage to explore the effect of any amendments to our understanding. We use the ‘auth_encrypt’ configuration that proceeds directly from authentication given a previously established link key, k . If needed, we may include the previous link key generation stage allowing us to talk about the generation of the link key given a previously established initial link key. We have not used this configuration (‘comb_auth_encrypt’) save for question **Q6.** where we wish to talk about the influence the Intruder may have on the generation of the encryption key given that he may actively attack previous stages.

- **Result 4.1: In our analysis of the simple configuration ‘auth_encrypt’, we did not find any failures of secrecy. The analysis did not use any variant systems and the issue of mutual authentication is still unexplored.**

In exploring how the Intruder could deduce the encryption key without recourse to the link key it is interesting to analyse the cryptographic properties of the Bluetooth functions E_{21} , E_{22} , E_1 and E_3 . All the functions seem to essentially come down to the same algorithm (a slightly modified version of SAFER-SK128), each differing in its invocation. This leads to the interesting question of whether the Intruder could exploit any cryptographic equivalences. For example, there is a trivial equivalence between the functions E_1 (used in the authentication response) and E_3 (used in the encryption key) when the same key, random number and Bluetooth address are used (cf. [Blu03, pages 777–783]):

$$E_1(K, RAND, BD_ADDR) = E_3(K, RAND, BD_ADDR \cup BD_ADDR) \quad (9)$$

where \cup denotes the append operator. Unfortunately this application of E_3 and BD_ADDR occurs during the generation of master keys used in point-to-multipoint sessions, which we have not analysed. This equivalence alone would be unlikely to reveal any new behaviour in the Intruder as it not obvious the other constraints over K and $RAND$ can ever be met. However, a future analysis concerning such cryptographic equivalences would be very interesting. Indeed, Casper already provides a facility to add such equivalences to an analysis.

Q5a. Can the Intruder break the bonding process?

In the question above we interpret the term ‘break’ as a failure of authentication (of the link key) as defined in question Q3.. However, we recognise that ‘break’ may also cover the Intruder inducing a protocol session to fail. In the first sense, to answer the question we reference the conclusions of question Q3. regarding authentication.

- **Result 5.1: In the second sense of ‘break’ we simply note that the Intruder can indeed induce the bonding process to fail (ignoring simple denial of service attacks). Given that the authentication challenges are sent in plain-text, the Intruder may simply interject at the appropriate point with a fake challenge of his own and the claimant’s response will fail. The Intruder may then repeat the response any number of times until the protocol fails (given an exponentially increasing back-off scheme [Blu03, page 775]).**

Q5b. Can the Intruder spoof one party in the bonding process?

In the question above we interpret the term ‘spoof’ again primarily as a failure of authentication as defined in question Q3., although the term is most often used at the transport level for packets with fake sources. We already assume the Intruder has the ability to create fake messages in our analysis (i.e. mimic a Bluetooth address), and cannot address the second more usual interpretation. Specifically, we take ‘spoof’ to concern the issue of apparent identity (after complete protocol runs) covered by our stringent definition of Bluetooth authentication (as given in question Q3.). Therefore, our notion of ‘spoof’ and Bluetooth authentication are linked but not equivalent.

- **Result 5.2: With reference to the conclusions of question Q3., we found that authentication (in our stringent definition) failed for unit link keys but not for combination link keys. In fact, the simple failure we document in question Q3. fails because and only because there is a disparity in apparent identities; agent *Alice* believes she has set up a connection with the Intruder *Mallory* but has actually been running with *Bob*. This is clearly an example of ‘spoofing’ - its usefulness depends on what *Alice* does next.**

Q6. Can the Intruder influence the derivation of an encryption key without knowledge of the initialization or link keys?

At first sight this might seem too vague a question to formulate coherently in the context of an algebraic-based analysis. After all, in every day usage the word ‘influence’ is used to mean many different things.

In fact, though, there has been much insightful research in the CSP community into what exactly 'influence' means in terms of malicious 'interference' in a system [Ros98, Ch 12.4] and [PR01, Ch 9.7]. For our purposes, it seems that a good starting point would be to use a so-called *lazy abstraction* of our Casper-compiled System in order to specify that the derivation of an encryption key is truly independent of any choice or action that the Intruder can make. However, that specification would almost certainly be too strong, because the Intruder *can* influence the derivation of an encryption key without knowledge of the initialization or link keys insofar as he can prevent even a link key being successfully generated (c.f. **Q5a**).

Thus, we believe that what is actually wanted here is the following, very precise, specification. To an outside observer, it should be non-deterministic as to whether: (i) the System halts without an encryption key being generated, or (ii) a process occurs resulting in an encryption key being generated, and that process is deterministic in the environment (user) inputs to the honest agents. In other words, the Intruder can either stop an encryption key being generated if he wants to, but if he chooses not to prevent one being generated, then he cannot influence its form in any way. This is the subject of ongoing research and we cannot answer the question at this time.

Q7. If we assume the PIN is secret, can we compromise the bonding process?

In the question above we interpret the term 'compromise' as failure of secrecy of the link key (combination or unit).

- **Result 7.1: With reference to the conclusions of question Q2a. regarding secrecy of the link key we can certainly answer in the negative with good confidence. Of course, for unit link keys it is assumed that the Intruder does not play a legitimate role and thereby learn the (common) unit link key for devices with limited memory (cf. question Q2a.).**

4.5.4 Open questions

Q1. Does a long PIN (of 16 octets) prevent the Intruder making use of the BD_ADDR to compromise the bonding process?

This question seems to pertain to the fact that shorter PINS are augmented using the BD_ADDRS (which we are anyway assuming are known to the Intruder). To answer this question, we could use a type representing a 16 octet string that would be split into two parts. Each instance of this type would either be a short PIN concatenated with a BD_ADDR, or the concatenation of two shorter 'PIN's representing a true 16 octet PIN. We could then check that even if the Intruder has knowledge of the 'first half' of a 16 octet PIN, then this would not allow him to 'compromise' the process. (Whereas knowledge of a short PIN plus BD_ADDR would almost certainly allow the Intruder to break the process.)

Q2. Following the initial link key generation stage, can parties get confused as to whether a unit key or combination key is being generated?

Q3. In some cases a master key can temporarily replace a link key. Is it possible for an agent to be spoofed into accepting a dud encryption key on a false premise of temporary use of a master key? If not, might a device broadcast anything 'of interest' before turning down the dud master key?

Q4. Where do the *IN_RANDS* come from? If they are clock-based, then can they be compromised?

Q5. Suppose it is possible to trick one Bluetooth device, *A* say, into generating a unit key with another device, *B*, but have *B* believe that a combination key is being generated. Can the Intruder then learn any secret information?

Q6. What attacks, if any arise from the 'malicious boosting' scenario described in our discussion of the Intruder model in section 4.2.5? Recall that this envisages two honest devices, *A* and *B*, just out of transmission range of each other. A malicious device sits about midway between *A* and *B*, and is thus in a position to overhear any attempted transmissions between *A* and *B*, relaying them or not as it chooses, those that it does not relay, fail.

Q7. Bluetooth recommends that each time a fraudulent device tries and fails to bond with an honest device, then the honest device impose an exponentially increasing delay before it allows the fraudulent device to try and bond with it again. Might the fraudulent device try to get around this by claiming a different BD_ADDR each time? How much of a problem is it if the honest device fails to deal with the issue?

4.6 Bluetooth-usage scenario

In this section we will consider a Bluetooth-usage scenario which one might expect to see in the real business world. It has a relatively simple topology that involves up to four devices, including the malicious device *Mallory*. One of the three honest devices is a client device that is either seeking, or is actively using, the services offered by one of the other devices. In the scenario one of the honest devices, *Colin* is 'compromised', in that the device *Mallory* knows everything that *Colin* knows.

A PDA, *Alice*, uses the Bluetooth Service Discovery Protocol (SDP) to find a device that can return to it a commercial-in-confidence (CIC) calender entry for some date, *date*. The calender applications used are those of MS Outlook and MS Exchange. In the immediate vicinity of the PDA are an honest laptop, *Bob*, and a malicious laptop, *Mallory*. Only laptop *Bob* has the pertinent calender application. There is a fourth device, *Colin*, that is honest, but which may have had its PINs compromised to laptop *Mallory*. The device *Colin* is currently not in the vicinity. PDA *Alice* trusts the laptops *Bob* and *Colin*, but not necessarily the laptop *Mallory*. The set-up is illustrated in figure 13.

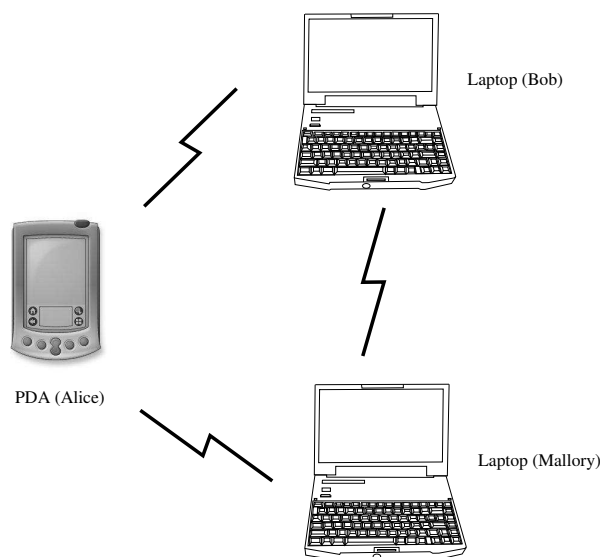


Figure 13: Bluetooth scenario 1: PDA (Alice) getting CIC calender entry from laptop (Bob)

Unless otherwise stated, all communications between honest parties are assumed to be encrypted using the relevant encryption keys. However, we know from the results of question **Q4b.** in section 4.5.3 that if *Mallory* has somehow learnt of the PIN shared between two devices, then it is possible for it to have deduced the encryption key that they are using.

Our objective is to run through different instances (with respect to initial knowledge of agents and System definition) of the above set-up in order to gauge what information *Mallory* would require in order to obtain a CIC calender entry. Also, we want to consider the robustness of different application-defined algorithms that the PDA might use in order to choose from a number of competing service provider devices.

In the first instance of the above Bluetooth set-up, we assumed the following. The PDA, *Alice*, knows some *dates*, the Bluetooth addresses of all devices, and encryption keys k_{AB} , k_{AC} and k_{AM} which it shares with *Bob*, *Colin* and *Mallory* respectively. The laptop, *Bob*, knows the calender

application, *calender*, the Bluetooth addresses of all devices, and the encryption keys k_{AB} , k_{BC} and k_{BM} that it shares with *Alice*, *Colin* and *Mallory*. The malicious device *Mallory*'s initial knowledge is of the Bluetooth addresses and the encryption keys k_{AM} , k_{BM} and k_{CM} that it shares with *Alice*, *Bob* and *Colin*. We also assume that *Mallory* knows the keys k_{AC} and k_{BC} of *Colin*, perhaps having deduced them from the compromised PINs of *Colin*. Our System comprises one PDA *Alice* process running in parallel with two copies of the laptop process *Bob* process (although, for the purposes of this example, it could just as well be that the laptop *Bob* process be run twice sequentially). As always, all runs are in the presence of the malicious device *Mallory*, here a laptop.

Using FDR, we compared the Casper-compiled model of the above system to a specification that said that all CIC calender entries were secrets between the PDA and laptop *Bob*. FDR showed that the assertion failed. Below, as interpreted by Casper, is the first of the 'attacks' discovered by FDR in which the laptop *Mallory* learns of a commercial-in-confidence calender entry.

Alice's user (the environment) wants an MS Outlook calender application:

1. $\rightarrow Alice : MSOutlook$

The PDA sends a service search attribute request to *Bob*,

2. $Alice \rightarrow I_Bob : \{Alice, MS Outlook Cal App\}\{k_{AB}\}$,

it is overheard by the Intruder.

The service search attribute request is received by *Bob*:

3. $I_Alice \rightarrow Bob : \{Alice, MS Outlook Cal App\}\{k_{AB}\}$.

Bob decrypts the message, sees that it must have at least have originated from *Alice*, and sends a service search attribute response back to *Alice*:

4. $Bob \rightarrow I_Alice : \{1, MS Outlook Cal App\}\{k_{AB}\}$.

The response indicates that *Bob* has one application satisfying the stated attributes.

The PDA sends a second service search attribute request, this time to *Colin*:

5. $Alice \rightarrow I_Colin : \{Alice, MS Outlook Cal App\}\{k_{AC}\}$,

it is overheard by the Intruder.

Bob's service search attribute response is now received by *Alice*:

6. $I_Bob \rightarrow Alice : \{1, MS Outlook Cal App\}\{k_{AB}\}$.

Mallory, in the guise of a server, *Colin*, sends a service search attribute response back to *Alice*:

7. $I_Colin \rightarrow Alice : \{1, MS Outlook Cal App\}\{k_{AC}\}$.

It claims that *Colin* has one application satisfying the stated attributes - although he does not.

As far as *Alice* is concerned, both *Bob* and *Colin* can satisfy the request. The PDA chooses to use the service offered by *Colin* (the 'choice' is modelled as a free enviromental choice between *Bob* and *Colin*):

8. $\rightarrow Alice : Colin$

Alice's user (the environment) wants the CIC calender entry for 02.11.03:

9. $\rightarrow Alice : 02.11.03$.

The PDA sends the date 02.11.03 to *Colin* encrypted using k_{AC} :

10. $Alice \rightarrow I_Colin : \{02.11.03\}\{k_{AC}\}$.

Bob receives from *Mallory*, in the guise of client *Colin*, a search attribute request:

11. $I_Colin \rightarrow Bob : \{Colin, MS Outlook Cal App\}\{k_{BC}\}$.

Bob decrypts the message, believes that it must have at least have originated from *Colin*, and sends a service search attribute response back to *Colin*:

12. $Bob \rightarrow I_Colin : \{1, MS Outlook Cal App\}\{k_{BC}\}$,

The response indicates that *Bob* has one application satisfying the stated attributes.

Bob receives from *Mallory*, in the guise of client *Colin*, the date 02.11.03 encrypted using k_{BC} :

13. $I_Colin \rightarrow Bob : \{02.11.03\}\{k_{BC}\}$.

Bob returns to *Colin* the calender entry for 02.11.03 encrypted using k_{BC} :

14. $Bob \rightarrow I_Colin : \{calender entry for 02.11.03\}\{k_{BC}\}$.

At this point *Mallory* knows the calender entry.

We considered three other instances of this system, the following are summaries of the analyses.¹⁸

In the second instance of the system we simply deleted the encryption key k_{BC} from *Mallory's* initial knowledge (so that the only thing that *Mallory* knows, but shouldn't, is k_{AC}). *Mallory* was still easily able to deduce the CIC calendar entry for 02.11.03.

In the third instance of the system we assumed that the laptop *Bob* could only offer MS Outlook. The PDA *Alice* prefers the better MS Exchange application, but is prepared to make do with MS Outlook if and only if no MS Exchange is available. Then, given that *Mallory* can always lie about the quality of service that he (or rather his unwilling 'proxy' *Colin*) can offer, it follows that *Mallory* could always ensure that, in the guise of *Colin*, he is chosen by *Alice* in preference to *Bob*. He could do that by claiming to offer MS Exchange in the search attribute response to *Alice*, whereas *Bob*, who is honest, would only ever claim to offer MS Outlook.

In the fourth instance, we assumed that the SDP service attribute requests and responses were not encrypted. Then *Mallory* did not need any of *Colin's* encryption keys in order to deduce a CIC calendar entry.

4.7 Summary of analysis results

This is a summary of the findings of our analysis to date of the Bluetooth initialization procedure.

We considered a sizeable, but not complete, subset of the possible variants of the bonding procedure. These include most of what could be construed as the 'normal usages' of the procedure. Of these, no definite protocol attacks were found when the Intruder possesses no prior information. We have shown that the Intruder may be able to 'break' or 'compromise' the security when h/she is assumed to have some capability above that which he would routinely expect to have in practice (as described in section 4.2.5). These additional capabilities included:

- **guessing the shared *pin*.**
- **guessing either l_{krandA} or l_{krandB} (as used in the link key).**
- **spoofing the identity of a bluetooth device.**

In the first two capabilities, the Intruder is assumed to be able to make an effective guess on the shared *pin* or random numbers and then verify his guess (cf. question **Q1b.**). In the third capability, 'spoofing' concerns the apparent identity of a device as opposed to the origin of a given packet (cf. question **Q3.**). This might correspond in the wired world to an attack on a DNS server in which the name to address mapping is altered. The first two capabilities lead to obvious and simple failures of secrecy and authentication while the third leads to more subtle issues relating to authentication.

The outputs of the formal modelling are a series of events which describe successful attacks. These will be validated in the next phase of our research.

4.8 Conclusions on modelling approach

The Casper-based analysis completed for this report provides us with a firm basis with which to complete a thorough and comprehensive analysis of the usage of the Bluetooth Link-Layer security. Our scripts cover in detail all the stages of the Bluetooth bonding procedure up to and including the generation of encryption keys. Only the generation of master keys used in point-to-multipoint sessions has been omitted.

Our strategy has been to write a small number of detailed, highly configurable scripts covering the different stages of the bonding procedure. The Caspose tool was then written to allow us to combine those scripts in any number of ways in order to analyse the robustness of the bonding procedure over

¹⁸ It is worth noting that it took only a matter of minutes to change the model to correspond to the different instances. If, however, we had written bespoke CSP models, the time taken to make the corresponding changes would likely have been measured in hours. Also, all the FDR tests quoted in this section were compiled and run in a matter of seconds.

many different assumptions. Those assumptions notably include the malicious Intruder's assumed abilities (including 'guesses') and process topology (i.e. who is running what in sequence or in parallel). These scripts should allow us to formulate a meta argument to the effect that the secureness of *any* envisaged run of the bonding procedure reduces to the question of the assuredness of a small number of simpler runs of the procedure between two agents that have been machine-checked by Casper/FDR. This 'meta argument' would also cover point-to-multipoint sessions (which was why we omitted point-to-multipoint in this stage of the work).

For any test that fails, FDR and Casper provide us with an easily interpreted trace of events representing the Intruder's attack leading up to the breaking of that protocol. With that information, further work may be done on the testbed to ascertain the severity of the problem in practise. In such cases, we may also use our models to develop fixes to the problem, which may, again, be implemented in the testbed.

We ran some forty different variants of the Caspose/Casper compiled test scripts through FDR so as to answer a variety of questions pertaining to the robustness of the bonding process. Some of the more interesting issues that arose from those tests are presented in this report in the form of annotated Casper interpretations. The forty or so different tests represent a sizable, if not complete, analysis of the set of different variations possible when running the bonding procedure. We can claim that the forty tests cover most of the 'normal usages' of the bonding procedure up to encryption.

Unfortunately, we have not had time to address more complicated runs of the initialization procedure, e.g. serial runs of the procedure. As the Intruder is assumed to know all the Bluetooth cryptographic functions, previous messages will only be of use to the Intruder if they contain facts pertaining to the current protocol session. This entails a reuse of session variables, for instance the reuse of a fixed PIN. However, this analysis has already been performed in part as the systems 'sys1', 'sys2' and 'sys3' reuse various parameters.

Using our scripts we considered a few interesting 'what if' scenarios. For these, we presuppose the Intruder has somehow manipulated the agents (by some feasible means possibly outside the scope of our modelling - e.g. by 'brute force' crypto-code breaking) into a potentially 'dangerous' state. Then we could check our model using FDR to see whether, in fact, the protocol could be broken given that the agents start in that state

We also presented an example usage of Bluetooth subsequent to the bonding process (compromised or not), including use of the Service Discovery Procedure. The idea here was to see how problems relating to the bonding process raised by our analysis could impact on the use of higher-level Bluetooth protocols, and, in so doing, we might comment on certain aspects of the specification that Bluetooth leaves to the application to resolve (e.g. choice of competing service).

Throughout our work on the Link-Layer modelling, Casper has proved to be an invaluable tool, reducing significantly the effort that would have been needed to write bespoke CSP models. Only in our modelling of the higher-level Bluetooth usage 1 (4.6) might it have been better to write a bespoke CSP model. That said, writing that model using Casper was still a worthwhile exercise, if only to assess the applicability of Casper to modelling outside its *raison d'être* (i.e. the analysis of crypto protocols).

Ideally, the ultimate aim of our modelling should be to make some assertion as to the robustness of the Link-Layer security under *all possible* usages - whether they be 'reasonable' usages or not. This we would achieve by a standard formal methods two-stage argument that goes as follows: (1) Formally verify, i.e. using Casper, a number of 'basic usages'. (2) Formulate a meta argument, i.e. a 'hand proof', to the effect that the secureness of any other usage reduces to the question of the secureness of a finite combination of the 'basic usages'. However, the sheer number of 'basic usages' that we had to consider for Bluetooth (forty three have been run to date) meant that we were unable to complete (2) confidently in the time allocated for this study. Having acknowledged this, we endeavoured to be thorough in our caveats - carefully spelling out the areas not addressed by our modelling¹⁹. Nevertheless, we recommend completion of (2) as a priority in future work. When (2),

¹⁹ In particular, the consequences of running various stages of the initialisation procedure in parallel.

above, has been completed, the open questions listed in 4.5.4 should be addressed.

A *typing attack* is an attack in which the Intruder - possibly exploiting weak typing - tricks an honest agent into interpreting a type correct message differently from what was intended. Typing attacks (and, more generally, crypto attacks) are not always suitable to formal modelling, due to the complexity of the types involved. However, during our validation effort, we identified two interesting scenarios that could be construed as potential typing attacks (cf. questions **Q2** and **Q5**, section 4.5.4). Both these scenarios should be tractable to Casper-based analysis, as the types involved in these cases would be relatively straightforward to model in SPL (by defining *equivalances* between fields of a message). We would recommend that those analyses be undertaken; if successful, that work might then be extended to a more systematic study of the susceptibility of Bluetooth to typing attacks.

A *guessing attack* is an attack in which it is assumed that the Intruder can feasibly 'guess' some supposedly private piece of data, and use that information to somehow break the protocol. We have implicitly modelled some of the more obvious Bluetooth typing attacks insofar as letting the Intruder 'know' some supposedly private data (cf. question **Q1b.** of section 4.5.3 for example). However, recent research has been undertaken into developing a formal Casper-based way of analysing guessing attacks. This might be exploited in future work.

There are some concerns of a probabilistic nature in Bluetooth. Arguably, one of the most pressing is the possible malicious exploitation of the exponentially increasing delays that the Bluetooth specification recommends be imposed on successive failed bondings [Blu03, page 775]. Although CSP and FDR are not really suitable for analysing such probabilistic issues, other modelling tools, notably PRISM [<http://www.cs.bham.ac.uk/~gxn/publ.html>] could be used in future analysis of probabilistic aspects of the Bluetooth Link-Layer security.

5 Conclusions

This report has shown the benefit of a multi-disciplinary approach to the analysis of the security properties of the Bluetooth protocol.

The GSN modelling approach has proved useful in providing a clear structure to a security argument. GSN allows a complex problem to be decomposed into simpler discrete elements. In the Bluetooth arena, this has been helpful to highlight areas to concentrate further analysis on.

The input from the QinetiQ Security Health Check team provided real world guidance to the formal analysis process, making that analysis far more 'grounded'. Additionally, the real world information from the Health Check team was used to verify that the GSN model had captured all the important elements of the protocol.

The combination of input from the GSN and the real world experience of the Health Check team, the formal analysis could be conducted in a very focused manner. In addition to standard formal protocol analysis, a number of specific real world security related questions were answered.

The formal modelling considered a sizeable, but not complete, subset of the possible variants of the bonding procedure. These include most of what could be construed as the 'normal usages' of the procedure. Of these, no definite protocol attacks were found when the Intruder possesses no prior information.

All of the issues listed in Section 3.6 require further investigation. The next stage in our research is to further validate the results of the formal analysis, and investigate other issues via practical penetration testing techniques.

6 Acknowledgments

We would like to thank Ollie Whitehouse of @stake for valuable collaboration.

References

- [Blu03] Bluetooth Consortium. *Bluetooth Core Specification*, 1.2 edition, November 2003. Available from:<http://www.bluetooth.org/spec>.
- [For] Formal Systems (Europe) Limited. *FDR Failures-Divergences Refinement: User Manual and Tutorial*.
- [Fwd05] Authentication for pervasive computing. Report D15, QineitQ, Due: March 2005.
- [Geh02] Christian Gehrman, et al. BluetoothTM Security. White paper, Bluetooth SIG Security Expert Group, April 2002.
- [Kel04] T. Kelly. *Safety Case Management: A systematic approach*. John Wiley and Sons Ltd, to be published May 2004.
- [Low98] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.
- [Low99] Gavin Lowe. *Casper: A Compiler for the Analysis of Security Protocols – User Manual and Tutorial*. University of Leicester, Department of Mathematics and Computer Science, University of Leicester, LE1 7RH, England, 1.3 edition, July 1999.
- [PR01] Steve Schneider et al. Peter Ryan. *Modelling and Analysis of Security Protocols*. Addison-Wesley, 2001.
- [Ros98] A.W. Roscoe. *The Theory and Practice of Concurrency*. Computer Science. Prentice Hall, 1998.
- [Yan] Tao Yang. Bluetooth security. <http://www.cs.utk.edu/~tyang/wireless/blue.htm>.