Towards a Multidisciplinary Framework for the Design and Analysis of Security Ceremonies

Marcelo Carlomagno Carlos

Thesis submitted to

Royal Holloway, University of London

for the degree of

Doctor of Philosophy

2014

Declaration

I, Marcelo Carlomagno Carlos, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is always clearly stated.

Signed: ....
(Marcelo Carlomagno Carlos)

Date: ....

Acknowledgements

I would like to thank my supervisor, Geraint Price, for all the support and guidance provided during my PhD. I truly appreciate his advices and his always insightful point of view of the research presented. I am grateful to my advisor, Chris Mitchell, whose wisdom, knowledge and enthusiasm I truly admired since our initial chats back in 2008.

I am thankful to the Information Security Group and the Mathematics department at Royal Holloway which provided me with all the support I needed during these years. I will candidly miss the warm and pleasant environment I had the pleasure to work in.

I would like to express immense gratitude to my many fellow graduate students and colleagues for providing a fun, friendly and stimulating environment in which I was able to learn a lot from both academical and social perspectives. To my office colleagues, the "355" guys, and to all other colleagues and friends I have in the department, I would like to express my thankfulness.

A special thanks to my great friend Jean Martina, who played a fundamental role since the beginning of this journey started in 2008. He was the person who gave me a ride to RHUL and introduced me to Chris and Geraint. Throughout this years, his role became even more important, turning into a mix of an important collaborator, advisor and listener who always gave attention to my most varied, and sometimes flawed, ideas and helped me extract the real relevance from them.

I would also like to thank my examiners, Professor Allan Tomlinson and Professor Colin Boyd, who provided encouraging and constructive feedback. Reviewing a thesis is a complex task and I am grateful for their thoughtful and detailed comments.

To my former supervisor, Ricardo Custódio, who always supported me in all imaginable ways. He has always been the greatest supporter of all stages of my PhD, from the initial idea of applying for it, to the end.

To Vania, for the all the moments we shared during this period. Thank you for your support, patience and care.

I cannot end without a very special thanks to my family, to my brother Thiago, who assisted me in many and equally important ways, to my sister Paula, and my loving partner, Ana. My family has always been an endless source of support during my entire life. Without them I would certainly have not accomplished nearly as much as I have. My partner Ana deserves my deep gratitude. Her presence has always been a source of love, encouragement and peace during this period.

My final thanks go to my parents. My father, despite short period I had the pleasure of sharing my life with, has always been my greatest example. My mother, and her unconditional support, took me to the position where I am now. I am extremely thankful for everything she did, and for the several times she put her life and needs aside to support my plans.

Abstract

In today's networked society, security protocols are widely used by the majority of people. From web browsing and instant messaging to cash machines, we are all, directly or indirectly, using protocols in order to ensure that certain security properties, such as integrity or confidentiality, hold. However, a significant number of real world attacks on the implementation of security protocols are against the surrounding components, such as the users, and not directly against the protocol itself. In a protocol specification, the human actions are usually included in the design assumptions, without being explicitly described in the protocol flow. When implemented, these assumptions are then replaced by dynamic user-interactions. It is often the case that these assumptions do not hold in practice. If this happens, the implementation can fail to deliver the security goals that the protocol has been designed to provide.

Security ceremonies, introduced by Ellison, can be described as an extension of security protocols which includes whatever was originally left out-of-band. In a ceremony the node types may include devices and humans, and the communication channels may vary, including not only the traditional network channels, but also human communication (e.g. speech) and human-device communication (e.g. user-interfaces). The increased coverage that ceremonies allow means that assumptions, which previously provided relatively static input to protocol design, are now a more explicit part of the model. This allows a more detailed analysis of their influence on the ceremony's security goals.

In this thesis, we provide a thorough review of security ceremonies and what we can achieve through including them in the design and analysis of a security implementation. First, we propose a taxonomy of human-protocol interaction weaknesses. This is important because the human elements of a ceremony are very hard to model. We outline a taxonomy of the most common human-interaction difficulties that can potentially result in successful attacks against protocol implementations. We then map these weaknesses onto a set of design recommendations aimed at minimising those weaknesses. Such a taxonomy and recommendations are important when modelling the user interaction in a ceremony to prevent an unrealistic expectation of the user's actions. Next, we describe a framework for designing and analysing security ceremonies. We provide a description of the agent types, communication channels, events and an adaptive threat model, designed to accurately reflect real world scenarios. We then analyse existing ceremonies using our framework and present the results. Finally, we discuss how all of our findings are related and could be used and developed further.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Security Protocols in the Real World

Security is very important to today's network based communications. Cryptography, a very active research area, is an essential component for secure communications. However, cryptography alone is not enough. Security protocols, which define sequences of operations and rules among agents, must be correctly defined and implemented to make a communication secure. By secure, we mean achieving relevant security properties within a given context. Security properties are quite varied, normally being integrity, confidentiality and authentication, but not limited to only those services. We all, directly or indirectly, use security protocols on a daily basis. Web browsing, instant messaging, payments using credit cards, cash machines, and many other services that are part of our daily routine rely on security protocols to ensure that certain security properties, hold.

Figure 1.1: Desire Path[1]

[1]image by Alan Stanton - http://flic.kr/p/bNU5cF

Security protocols (also called cryptographic protocols) are widely researched, and methods for verifying whether protocols achieve their claimed security goals are well developed and mature [69, 16]. However, a significant number of real world attacks on the implementation of security protocols are against the surrounding components, such as the users, and not directly against the protocol itself. In a protocol specification, the human actions are usually included in the design assumptions, without being explicitly described in the protocol flow. When implemented, these assumptions are then replaced by dynamic user-interactions. It is often the case that these assumptions do not hold in practice. If this happens, the implementation can fail to deliver the security goals that the protocol has been designed to provide.

A good 'real-world' example of such misalignment between design and practice, which all of us are likely to see very often, is the desire path. Desire paths are those unpaved paths created over time by human and animal footsteps. As we can see in Figure 1.1, they show that the designed path is not aligned with the users' desired path. Therefore, even though there is a paved sidewalk in place, which is likely to be more comfortable, safe, etc., humans tend to use the path where they can easily achieve their goals.

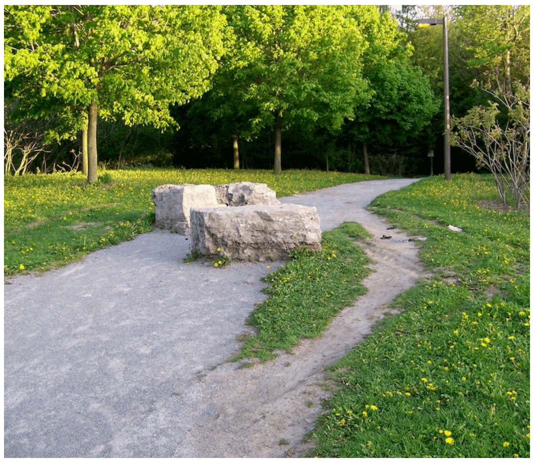


Figure 1.2: Another Desire Path [2]

Such misalignments may happen for several reasons and under different conditions and scenarios, as we will see in this thesis. In security protocols, these misalignments also happen, usually when protocols' assumptions are implemented. As we might expect, predicting human behaviour is a very complex and difficult task. However, leaving human interaction out-of-bounds may increase the chances of incorrect expectations about the user behaviour, and therefore introduce security flaws in practice. As we can see in Figure 1.2, security mechanisms designed to prevent certain actions may be simply ignored or misused by humans when put into practice and easier alternatives are available.

The idea of extending security protocols to include whatever was originally left out-of-bounds was introduced by Ellison [47]. One of the main additions in ceremonies is that now, human interaction is more explicit in the design and specification. In a

[2]image by Will S. - http://flic.kr/p/6o7HAZ

ceremony, the node types include (but are not limited to) devices and humans, and the communication channels may vary, including not only the traditional network channels, but also human communication (e.g. speech) and human-device communication (e.g. user-interfaces). The increased coverage that ceremonies allow means that assumptions, which previously provided relatively static input to protocol design, are now a more explicit part of the model. This allows a more detailed analysis of their influence on the ceremony's security goals.

In this thesis we demonstrate that by extending protocol analysis to ceremony analysis, we can potentially find and solve security flaws that were previously not detectable.

1.2 Contributions

The high-level contribution of this thesis is to reduce the gap between security and human-computer interaction by tackling the problem using security ceremonies. We provide a thorough review of security ceremonies and what we can achieve by including them in the design and analysis of a security implementation. The thesis is structured to sequentially represent all the steps we performed in order to reach a point where we can design and analyse security ceremonies. We used a multidisciplinary approach where we make use of concepts and ideas from different areas such as computer science, social sciences, psychology, web design and HCI design, to be able to develop a comprehensive model for the design and analysis of security ceremonies. We begin with a taxonomy of human-protocol interaction weaknesses. Such a taxonomy highlights the most common human-interaction difficulties that can potentially result in successful attacks against protocol implementations. Then, we map these weaknesses onto a set of design recommendations aimed at minimising their impacts. The development of a taxonomy and recommendations are important when modelling the user interaction in a ceremony. By considering them, we are able to minimise an unrealistic expectation of the user's actions.

After developing the taxonomy and creating a set of recommendations, the natural next step was to move to the design and analysis of security ceremonies. Then we developed a framework for designing and analysing security ceremonies. We provide a description of agent types, communication channels, events and an adaptive threat model, designed to accurately reflect real world scenarios. With this framework, we can start modelling security ceremonies and analyse the outcomes. We then analyse existing ceremonies, such as Bluetooth's legacy pairing ceremony and Simple Secure Pairing, as well as the WhatsApp messenger registration ceremony, using our framework and present the results. Finally, we discuss how all of our findings are related and could be used and developed further.

Part of the contents of this thesis presents revised and updated versions of the following papers:

1. J. E. Martina and M. C. Carlos. Why should we analyse security ceremonies? First CryptoForma workshop, May 2010.

2. M. C. Carlos and G. Price. Understanding the weaknesses of human-protocol interaction. In *Proceedings of the 16th international conference on Financial Cryptography and Data Security*, FC'12, pages 13–26, Berlin, Heidelberg, Mar. 2012. Springer-Verlag.

3. M. C. Carlos, J. E. Martina, G. Price, and R. F. Custodio. A proposed framework for analysing security ceremonies. In P. Samarati, W. Lou, and J. Zhou, editors, *Proceedings of the 7th International Conference on Security and Cryptography*, SECRYPT 12, pages 440–445. SciTePress, July 2012.

4. M. C. Carlos, J. Martina, G. Price, and R. F. Custodio. An updated threat model for security ceremonies. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1836–1843, New York, NY, USA, Mar. 2013. ACM.

1.3 Thesis Outline

The structure of the thesis represents the sequence of steps we performed in order to reach the point where we could design and analyse security ceremonies and discuss the results achieved.

We start, in Chapter 2, by introducing the description of the basic concepts that are used throughout the thesis, as well as fixing the notation used in this work. We also provide a general description of the main protocols, which are extended to ceremonies throughout the thesis, that are relevant for the following chapters.

In Chapter 3 we present our proposed taxonomy of the most common characteristics and behavioural patterns involved in human-protocol interaction. In addition to the taxonomy, we propose a set of design recommendations aimed at minimising the security inherent problems in the human-protocol interaction.

The contents of this chapter are the starting point of our research. The study we conducted led us to better understand why some protocols fail when implemented, and provided grounds that allowed us to analyse and discuss mechanisms to address some of the existing problems.

In Chapter 4 we present a framework that allows us to design and analyse security ceremonies. We present new agent types, communication channels, events and an adaptive threat model designed to accurately reflect real world scenarios.

Then, in Chapter 5, making use of the taxonomy and the framework we proposed, we modelled and analysed ceremonies based on existing and well known protocols. We analyse each scenario by discussing the ceremonies and their possible variations. For each setting, we propose a realistic threat model, informally analyse the possible outcomes and prove the properties of the ceremonies discussed by using an automatic cryptographic protocol verifier (ProVerif). We then discuss the results of the analysis performed and highlight the gains obtained by analysing ceremonies.

Finally, in Chapter 6 we conclude the thesis by providing final remarks and discussing our achievements followed by a discussion of how our ideas could be used and developed further.

Chapter 2

Preliminaries

Contents

In this chapter we introduce the basic concepts that will be used throughout this thesis, as well as the notation which has been used in this work. We describe of the main ceremonies we analyse in this thesis and are relevant to the following chapters.

2.1 Security Ceremonies

We all, directly or indirectly, use security protocols nowadays. From web browsing to cash machines, we all rely on the implementation of protocols to provide assurance that certain security properties, such as integrity or confidentiality, hold in practice. Protocols have been thoroughly analysed since Needham and Schroeder [71] first introduced the idea and several methods and tools have been developed in order to prove protocols' goals. Therefore, we can say that protocol design and analysis is a well established and mature research field.

Nevertheless, during the last decade, the awareness that information security is much more than just a technical issue has been consolidated [15]. Recent research [57, 41, 76, 56, 64] shows that even the most widely analysed and deployed protocols can exhibit security flaws when put into practice. This usually happens when a user of an implemented protocol acts in an unexpected, although plausible, way. This reinforces the claim made by Bella and Coles-Kemp that accomplishing a security goal in practice requires heterogeneous and combined efforts from computer scientists, social scientists, experimental psychologists, cognitive scientists, and web and HCI designers [15].

The technical side of a protocol focuses on the computer level, and therefore, protocols tend to focus on interaction between computers. However, protocols are usually built to facilitate or accomplish a human task and thus we should design and verify such protocols by also focusing on human interaction. The idea of ceremonies was introduced in 2003[1] [46], and elaborated upon further in 2007 [47], extends the concept of a security protocols by including humans as nodes in the network. Ellison states that a ceremony is like a network protocol, but some of its nodes may be human and the network links are not limited to traditional communications channels.

In general terms, a ceremony is an extension of a protocol, where the nodes types include devices and humans, and the communication channels may vary, including not only the traditional network channels, but also human communication (e.g. speech) and human-device communication (e.g. user-interfaces). Figure 2.1 gives a general overview of the association between ceremony and protocol.

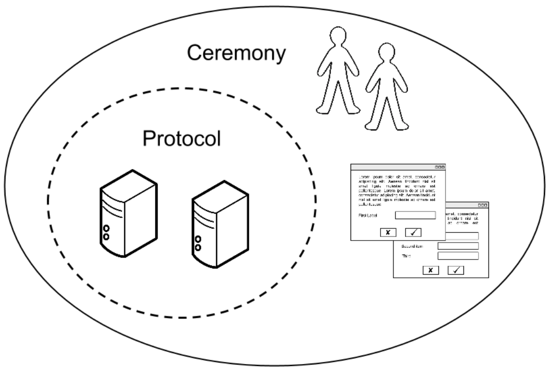


Figure 2.1: Protocol and Ceremony association

Such a detailed level of description introduces interesting characteristics to ceremony descriptions. A protocol leaves some components (e.g. human-device interaction)

[1]Ellison mentions in that document that the term "ceremony" for such a purpose was defined by Jessie Walker.

as 'out-of-bounds' and does not define nor clarify how they should work in practice, leaving them as assumptions. When the assumptions are unrealistic, it becomes extremely difficult to translate the protocol assumption into an implementation that still provides the expected security properties. That is one of the main reasons why we still have practical attacks on well stablished protocols. A ceremony, due to its more detailed description, inherently contain more fine-grained assumptions when compared to protocols. When designing a ceremony, the user interaction (e.g. messages that the user inputs or receives) needs to be explicitly described.

The inclusion of humans into the specification introduces a considerable complexity into the analysis. Humans are a more error prone and unpredictable node. The non-deterministic behaviour of humans is hard to model and analyse. The state machine of a human, their knowledge, skills, strategies and limitations are very complex structures to deal with. We obviously cannot model a human node that matches an exact human being but we can learn some common characteristics empirically (as we see in Chapter 3) and use them in order to develop a more robust ceremony.

In this thesis, we consider that a ceremony has two possible node types: humans and devices. Human nodes represent human beings that are part of the ceremony, and devices could be a device of any type, such as a computer, smart phone, etc. We also consider that a ceremony has three possible communication channels to represent the human-human, human-device, and device-device communication (we will discuss node types and communication channels in more details in Chapter 4).

2.2 Ceremonies versus Protocols

Both security ceremonies and security protocols can be defined as a sequence of interactions among agents to achieve a certain security goal, such as entity authentication, key distribution, confidentiality and anonymity. The difference between ceremonies and protocols is that, as we can see in Figure 2.1, ceremonies are a superset of security protocols. As a result, we can say that all security protocols are ceremonies. However, ceremonies based on the same protocol, due to the extended coverage and context added to its description, are different from each other. In other words, and as described by Radke et al., a ceremony is a (set of) protocol(s) in its context of use [80].

As we have mentioned earlier, ceremonies can potentially include everything that has been left as assumptions in a protocol, such as additional node types, communication channels, internal assumptions and operations which were previously considered out-of-bounds. As examples of these out-of-bounds operations we have user interaction via human-device interface, human knowledge and capabilities, and key provisioning.

A protocol runs among two or more nodes and each node has its own state, its set of possible messages and a state machine. The communication channel (network) between the nodes can be analysed against very powerful attackers that might control the whole channel. Ceremonies, as opposed to protocols, do not necessarily require a (classic) network. We may have additional channels to represent human-human and human-device communication. To analyse and verify security protocols there are several techniques and methods, and all of this analytical power is available to ceremonies [47]. We just need to adapt the threat model to each new channel and verify them accordingly. As we will discuss in Chapter 4, the threats to the new channels can be different to the classic network channel that is used for protocols. Additionally, we may have ceremonies that do not make use of a network channel at all. A ceremony that covers only communication between devices and humans, or only between humans, is an example of a ceremony which does not include traditional communication protocols.

Another relevant difference between ceremonies and protocols is the level of details included in the assumptions. While in a protocol specification we define assumptions to represent out-of-bounds operations, ceremonies turn assumptions into realistic design parameters. The inclusion of the human nodes, and their interaction with the ceremony, enriches the detail and coverage of the analysis. In security protocols we include several assumptions for operations that are considered out-of-scope, such as human-device interaction, human-human interaction, key provisioning, human knowledge, etc.; in security ceremonies we explicitly include such interactions. By including a human node, we have to define and use additional communication channels such as user-interfaces, for human-device interaction, and a human channel, to represent speech, gestures, etc., for human-human interaction. Nevertheless, security ceremonies still require the use of certain assumptions. For example, some initial knowledge regarding the human agent, but the assumptions tend to be more precisely described, fine grained and more realistic.

It is important to emphasize that the gains from designing and analysing security ceremonies are directly linked to the quality and accuracy with which we describe the additional components of a ceremony when compared to protocols. In protocols, it is relatively simple to predict the agents' behaviour and expect a deterministic set of possible outcomes. The communication channel is usually a network channel and it follows the same rules regarding the threat model it is subject to. In ceremonies, we have to deal with the challenge of adding human nodes into the specification. This brings additional communication channels, such as the human-device channel and the device-device channel and the human agent type in addition to the already existing device (or computer) type. All of these new additions have their own possible set of threats, nuances and complexity, making ceremony design and analysis a challenging process.

2.3 Notation

In this section, we describe the basic notation that we will adopt in this thesis. The notation we chose for describing security protocols and ceremonies is very similar to the notation that is commonly used for describing protocols.

2.3.1 Ceremony and Protocol Notation

The notation we use for protocol description consists of describing each step of the protocols in a line identified by a sequential number. Each step is divided into three main blocks: the first block is composed by a step number followed by a dot (“.”); the second block contains the name of the sender on the left side, an arrow that indicates the flow of the message in the center (usually from left to right), followed by the name of the receiver and a delimiter, indicated by a colon (“:”), that represents the end of the second block; the third block contains the message. The components in the message are delimited by a comma (“,”) and may be grouped using braces “{” and “}” to indicate the use of a cryptographic function over the grouped message components. Figure 2.2 shows an example of the notation we use.

$$\begin{array}{lcccc} 1. & A & \longrightarrow & B & : & A, Na \\ 2. & B & \longrightarrow & A & : & \{B, Nb, Na\}_{Kab} \end{array}$$

Figure 2.2: Example of protocol notation

This protocol consists of two steps, where in the first agent A sends to agent B a two component message composed of his identity A concatenated with a nonce Na. The second describes a message from agent B to the agent A. In this message we have the concatenation of B's identity, a nonce Nb and the nonce Na, encrypted with a key Kab. In other words, B sends a encrypted message to A, using the key Kab, containing its identity, Nb and Na.

The notation for ceremonies is very similar to the one used in protocols. The difference exists because of the different communication channels (which may be subject to different threat models), hence the need to differentiate over which channel the message is being sent. To do that, we add a label below the arrows (in the second block) to specify the name of the channel. Figure 2.3 shows an example of the notation being used.

$$\begin{array}{lllcl} 1. & B & \xrightarrow[DD]{} & A & : & B, Nb \\ 2. & A & \xrightarrow[HD]{} & U & : & B \end{array}$$

Figure 2.3: Example of ceremony notation

Similar to the protocol described earlier, this ceremony consists of two steps, where in the first agent B sends to agent A, via the network channel identified by DD, a two component message composed of his identity B concatenated with a nonce Nb. The names used to differentiate the channels and that we will adopt throughout this thesis will be defined in Chapter 4. The second step describes a message from agent A to the agent U via the HD channel. In this message we have B's identity being sent.

2.3.2 Logical Notation

This thesis also contains a few logical expressions, hence in Table 2.1 we present the logical symbols we used, followed by how they are read in English.

Symbol	Logical Context	Reads as
$\wedge$	conjunction	"and"
$\vee$	disjunction	"or"
$\cup$	union	"the union of"
$\neg$	negation	"not"
$=$	equality	"is equal to"
$\in$	set membership	"is in"
$\notin$	negation of set membership	"is not in"

Table 2.1: Logical Notation

2.4 Protocols Studied

In this section, we describe the protocols we will discuss and analyse in this thesis. The focus of the description at this moment is only at the protocol level. In Chapter 5 we then model ceremonies based on these protocols and analyse these ceremonies in more detail.

2.4.1 Bluetooth Pairing

Bluetooth is a short-range communication system intended to replace the cables connecting portable and/or fixed electronic devices [21]. The establishment of such communication has a momentary nature, created for data exchange between the devices.

Bluetooth has been designed with robustness, low power consumption, and low cost in mind.

Bluetooth devices work under two modes of operation, namely, discoverable and non-discoverable. When operating in the discoverable mode, the device responds to inquiries made by other (unknown) devices. On the other hand, when in non-discoverable mode, a device only responds to enquiries from devices with whom it has previously set up communication.

When two devices are communicating for the first time, they do not have a common link key. Therefore, such a key shall be created. To create a new key, a procedure called "pairing" is used. Pairing allows devices to mutually authenticate themselves and create a shared symmetric key, which will provide the basis for all security transactions between those devices. There are two procedures for pairing. The first is currently known as the *Legacy Pairing Protocol* and the second is the *Secure Simple Pairing*.

The legacy pairing protocol, in use from Bluetooth's version 1.0 to 2.0+EDR (Enhanced Data Rate), generally makes use of a user input to establish the connection. User(s) of both devices in the pairing are asked to type a PIN code into each device which is used as part of the connection establishment. For devices with limited input capabilities (e.g. headsets), a fixed PIN number is used (e.g. 0000), whereas for advanced devices, such as mobile phones or computers, a numeric or alpha-numeric PIN may be used.

For recent versions of Bluetooth, a different pairing mechanism is defined. This new pairing procedure is called *Secure Simple Pairing* (SSP) and is available from version 2.1+EDR and above. SSP was designed to solve several problems found in earlier versions of the pairing protocol. First, it simplifies the pairing process from the user's perspective, offering different pairing options and requiring fewer and simpler interactions. In addition to the usability improvements, it adds increased protection against passive (eavesdropping) and active (man-in-the-middle) attacks. This additional protection solves flaws found in earlier versions that allowed attackers to perform offline attacks and deploy man-in-the-middle (MITM) attacks [24, 25, 57, 92, 34].

2.4.1.1 Legacy Pairing Protocol

The legacy pairing protocol, which is the pairing mechanism used from Bluetooth version 1.0 to 2.0+EDR aims to allow devices to create a shared symmetric key called K_{init}, authenticate one device to the other (mutually in most cases), and generate a *link key*, which is a key that will provide the basis for all security transactions between these pairing devices.

The pairing process includes several steps. First, a personal identification number

(PIN) must be set for the both pairing Bluetooth devices. The way that the PIN can be set varies. It can be variable, and therefore be defined via user input (if the devices support such a feature) or it can be fixed (in devices with limited input capabilities, such as headsets). Entering a PIN in to both devices is recommended according to the specification and should be used whenever possible [23]. The PIN must be the same for both devices involved in the pairing process. For most applications, the PIN is a short string of numbers, consisting usually of four digits, but it can be up to 16 octets. In addition to sharing the same PIN, each device has its own (unique) *Bluetooth device address*.

The protocol is composed of 8 messages that can be divided in three different parts. The initial assumption is that both agents involved initiate the protocol run with the same initial knowledge, which is composed by PIN and each others bluetooth addresses (BD_ADDR_A and BD_ADDR_B).

In Figure 2.4 we present the first part of the protocol, which we call initialisation phase. The main goal of this phase is to allow both peers to generate the initialisation key K_{init}. To do that, both peers, which already share a common PIN and know each others bluetooth device addresses (BD_ADDR_A and BD_ADDR_B), initiate a communication by A sending a 128-bit random number IN_RAND to agent B, which replies with an $ACCEPTED$ message. With these values, both sides can now calculate K_{init} by using the algorithm $e22$, which is based on the SAFER+ cipher[2] [67]. The inputs for this function are BD_ADDR, PIN and IN_RAND. In order to define which Bluetooth address should be used (BD_ADDR_A or BD_ADDR_B), the following rule is applied: if one device has a fixed PIN the BD_ADDR of the other device should be used. If both devices have a variable PIN the BD_ADDR of the device that received IN_RAND (B in our example) should be used.

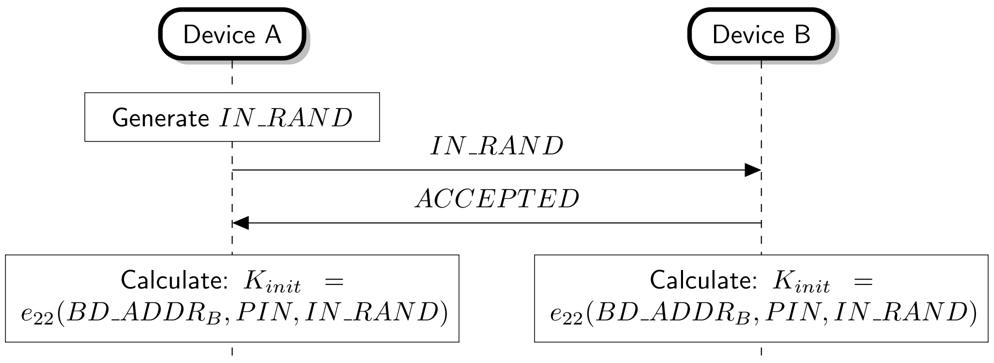

Device A
Device B
Generate IN_RAND
IN_RAND
$ACCEPTED$
Calculate: $K_{init} = e_{22}(BD\_ADDR_B, PIN, IN\_RAND)$
Calculate: $K_{init} = e_{22}(BD\_ADDR_B, PIN, IN\_RAND)$

Figure 2.4: Sequence diagram for the legacy mode pairing - part 1 (Initialisation)

[2]The algorithms $e21$ and $e1$ used in this protocol are also based on the SAFER+ cipher.

After calculating K_{init}, the next part is the authentication phase (Figure 2.5). The goal is to mutually authenticate the devices using a challenge-response mechanism. First, to authenticate B, A generates a 128-bit random number AU_RAND_A and sends it to B. On the other side, B calculates the authentication response $SRES_1$ using the $e1$ algorithm with K_{init}, BD_ADDR_B and AU_RAND_A as inputs[3] and sends it to A. Then A is able to verify if $SRES_1$ sent by B matches the value expected by A. If it does, the protocol continues and the other side and B repeats a similar process to authenticate A.

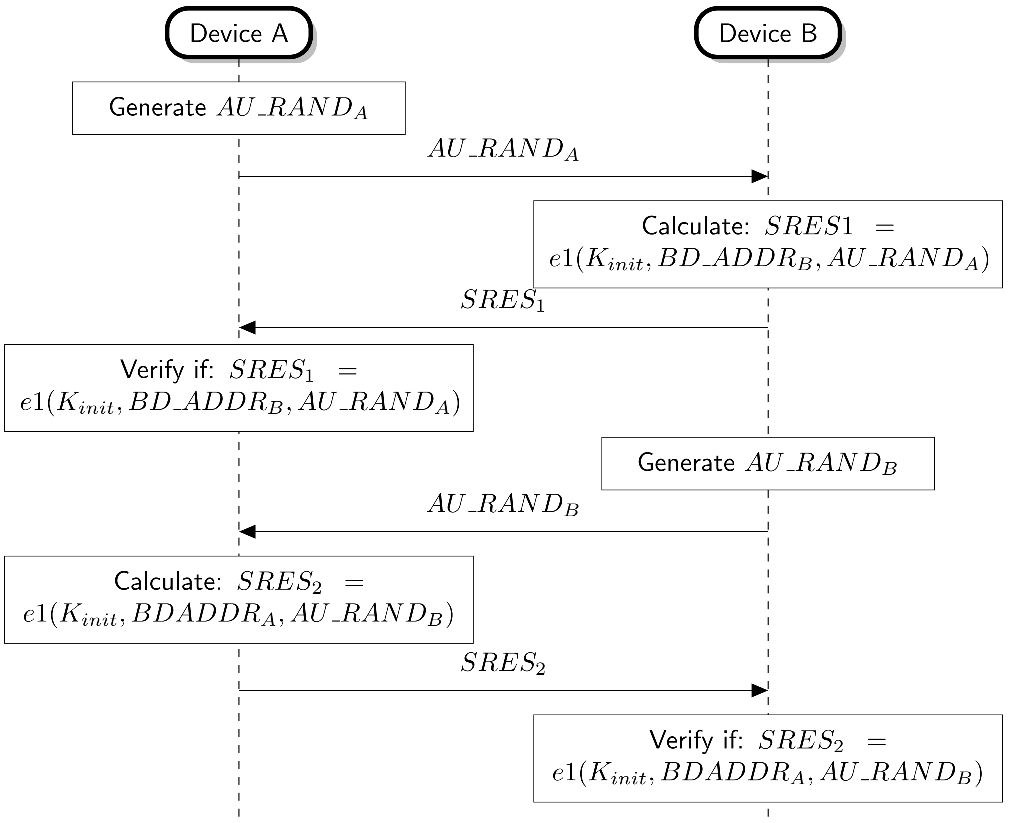

Device A
Device B
Generate AU_RAND_A
AU_RAND_A
Calculate: $SRES1 = e1(K_{init}, BD\_ADDR_B, AU\_RAND_A)$
$SRES_1$
Verify if: $SRES_1 = e1(K_{init}, BD\_ADDR_B, AU\_RAND_A)$
Generate AU_RAND_B
AU_RAND_B
Calculate: $SRES_2 = e1(K_{init}, BDADDR_A, AU\_RAND_B)$
$SRES_2$
Verify if: $SRES_2 = e1(K_{init}, BDADDR_A, AU\_RAND_B)$

Figure 2.5: Sequence diagram for the legacy mode pairing - part 2 (authentication)

Finally, the last part is calculating the link key K_{AB}, as we show in Figure 2.6. To generate this key, A generates a 128-bit random number LK_RAND_A, xors it with K_{init} and sends the resulting value ($E\_LK\_RAND_A$) to B. The other peer, B, obtains LK_RAND_A by xoring $E\_LK\_RAND_A$ with his K_{init} generated in part 1, and repeats the same process as A. It generates a 128-bit random number LK_RAND_B, xors it with K_{init} and sends the resulting value ($E\_LK\_RAND_B$) to A, who obtains

[3]Note that if the link key generation part is run before the authentication, K_{init} shall be replaced by K_{AB} as input for the $e1$ function.

LK_RAND_B by using K_{init}. Since both devices know K_{init}, each device now holds both random numbers LK_RAND_A and LK_RAND_B. At this point, both parties have the information they need to calculate K_{AB} by using the $e21$ algorithm twice and xoring the results.

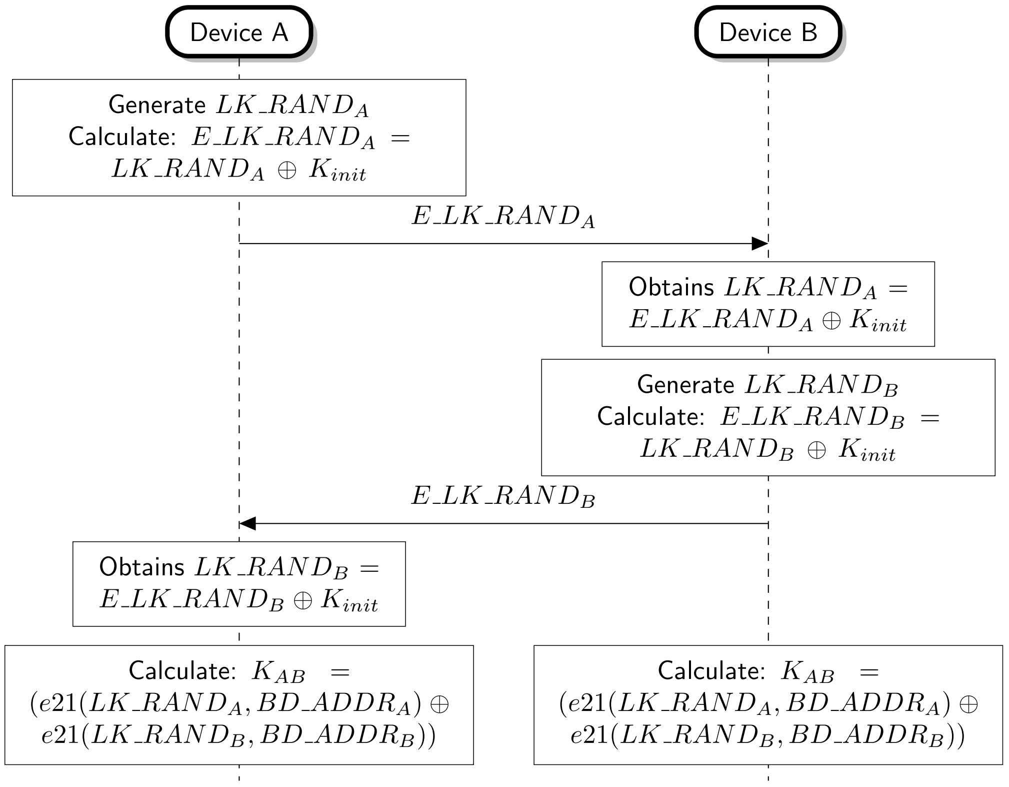

Device A
Device B
Generate LK_RAND_A
Calculate: $E\_LK\_RAND_A = LK\_RAND_A \oplus K_{init}$
$E\_LK\_RAND_A$
Obtains $LK\_RAND_A = E\_LK\_RAND_A \oplus K_{init}$
Generate LK_RAND_B
Calculate: $E\_LK\_RAND_B = LK\_RAND_B \oplus K_{init}$
$E\_LK\_RAND_B$
Obtains $LK\_RAND_B = E\_LK\_RAND_B \oplus K_{init}$
Calculate: $K_{AB} = (e21(LK\_RAND_A, BD\_ADDR_A) \oplus e21(LK\_RAND_B, BD\_ADDR_B))$
Calculate: $K_{AB} = (e21(LK\_RAND_A, BD\_ADDR_A) \oplus e21(LK\_RAND_B, BD\_ADDR_B))$

Figure 2.6: Sequence diagram for the legacy mode pairing - part 3 (Link key calculation)

As we can see, the protocol relies on the PIN. It is clear that if an attacker is capable of obtaining the PIN value, then the protocol properties of confidentiality and authentication are violated. Confidentiality is violated because, by knowing the value of PIN, the attacker will have all the required information to calculate K_{init}. Authentication is also violated because an attacker can successfully impersonate a device by knowing K_{init} and the device's Bluetooth address. Additionally, the PIN is a low entropy input (usually 4-digit long) susceptible to offline attacks, as discussed and presented in related papers [57, 92, 34].

For clarity and simplification, Figure 2.7 presents the full protocol using protocol notation.

$$\begin{array}{llllll}
1. & A & \longrightarrow & B & : & IN\_RAND \\
2. & B & \longrightarrow & A & : & ACCEPTED \\
3. & A & \longrightarrow & B & : & AU\_RAND_A \\
4. & B & \longrightarrow & A & : & SRES1 = e1(K_{init}, BD\_ADDR_B, AU\_RAND_A) \\
5. & B & \longrightarrow & A & : & AU\_RAND_B \\
6. & A & \longrightarrow & B & : & SRES2 = e1(K_{init}, BD\_ADDR_A, AU\_RAND_B) \\
7. & A & \longrightarrow & B & : & E\_LK\_RAND_A = LK\_RAND_A \oplus K_{init} \\
8. & B & \longrightarrow & A & : & E\_LK\_RAND_B = LK\_RAND_B \oplus K_{init}
\end{array}$$

Figure 2.7: Protocol description for Bluetooth legacy mode pairing

2.4.1.2 Simple Secure Pairing

Bluetooth's new pairing procedure, called "secure simple pairing" (SSP), was presented in version 2.1+EDR and, at the time of writing, it is still the current pairing mechanism in use. SSP was designed to mitigate several problems found in the legacy pairing protocol. The two main focuses of this protocol are on improving usability by offering different pairing options and simplifying user interactions, and on mitigating security problems against passive and active attacks.

SSP presents four distinct association modes designed to cover most device types. The **just works (JW)** mode is designed for devices with limited display and input capabilities, possibly without them. In this mode, the associating devices exchange public keys, nonces and a commitment value but nothing is displayed for the user, except in some implementations where the user might be asked whether to accept the connection or not. The **numeric comparison (NC)** association mode, which is our main focus in this thesis, makes use of the same protocol as the just works mode. The difference between them is that the numeric comparison mode is designed for devices capable of displaying digits (a 6-digit number) and accepting user inputs ("yes" or "no"). Due to the additional capability of the device, the protocol includes an additional authentication step, which is performed by the user. Both devices display a number (based on the nonces and public keys shared between the devices) and the users have to check whether the numbers shown are the same for both devices. If they are equal, the pairing is successful. The third association mode is the **out of band (OOB)** and it is designed for scenarios where an out of band mechanism is used for discovering devices as well as exchanging the cryptographic information required for the pairing process. For example, when using two devices that support Near Field Communication (NFC), the user(s) would touch the two devices together to perform the pairing. The last mode is the **passkey entry (PE)**, which is designed for situations where the pairing devices have different input and display capabilities. For example, where one device has only input capabilities, such as a keyboard, and the other has only displaying capabilities, for example, a screen [24].

The SSP protocol is divided into five phases, and phases *one*, *three*, *four* and *five* are the same for all modes/protocols. Phase *two*, which focuses on achieving mutual authentication between the devices, is unique for each association mode. Our focus in this thesis is on the SSP protocol under the numeric comparison mode. Therefore, we will concentrate on phases one and two of the SSP protocol using the NC mode and assume that all following phases are correct.

Phase one, called *Public Key Exchange*, is presented in a protocol description in Figure 2.8. Prior to the beginning of this phase, each device generates its own Elliptic Curve Diffie-Hellman (ECDH) key pair. Note that this key pair is required to be generated only once per device and could be generated at any point prior to the pairing or during the first pairing of the device.

$$\begin{array}{lcccl} 1. & A & \longrightarrow & B & : & PK_A \\ 2. & B & \longrightarrow & A & : & PK_B \end{array}$$

Figure 2.8: Protocol description the SSP - phase one

As shown in the sequence diagram presented in Figure 2.9, in order to start the pairing process, the *initiating* device, sends its public key PK_A to the receiving device. The *responding* device then replies by sending its own public key PK_B. At this point, both devices have the necessary information to calculate $DHKey$, using a $P192$ function that computes a FIPS approved P-192 elliptic curve, which will be used to generate the link key later in the protocol.

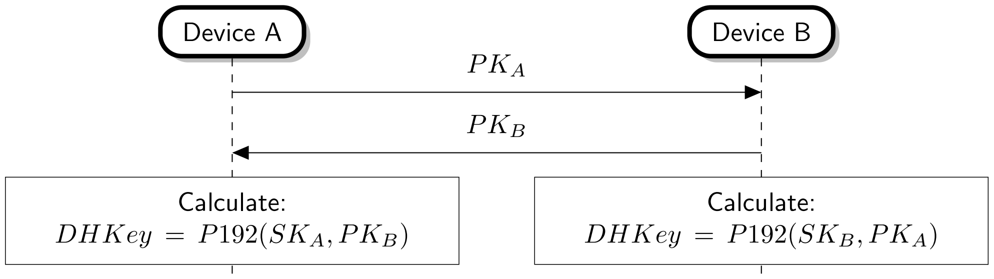

Device A
Device B
PK_A
PK_B
Calculate: $DHKey = P192(SK_A, PK_B)$
Calculate: $DHKey = P192(SK_B, PK_A)$

Figure 2.9: Sequence diagram for the SSP - phase one

Phase two starts as soon as phase one is concluded. In NC mode, considering that N_a and N_b are nonces and the function $f1$ is a function to generate the 128-bit commitment value C_b, the protocol in phase two is presented in Figure 2.10.

As we can see in the protocol description, there are only 3 steps in phase two in the NC mode. However, there are important steps which are not explicitly described. Figure 2.11 presents a detailed sequence diagram of phase two. After exchanging nonces

$$\begin{array}{llcll} 1. & B & \longrightarrow & A & : & C_b = f1(PK_B, PK_A, N_b, 0) \\ 2. & A & \longrightarrow & B & : & N_a \\ 3. & B & \longrightarrow & A & : & N_b \end{array}$$

Figure 2.10: Protocol description for the SSP - phase two

with B, A verifies if the value of C_b sent by B matches the value A calculates. Additionally, there are two confirmation values that must be calculated by the devices A and B. The confirmation values are V_a and V_b which are generated via a known function g using the values PK_A, PK_B, N_a, N_b, that were shared between the devices during the previous protocol messages, as parameters. V_a and V_b are values that will be shown on devices A and B's displays respectively. Finally, the users of these two devices will then compare whether V_a and V_b are equal and confirm by (usually) pressing a button on the device. Without this user verification of V_a and V_b, the authentication goal does not hold for this protocol, making it vulnerable to MITM attacks for example.

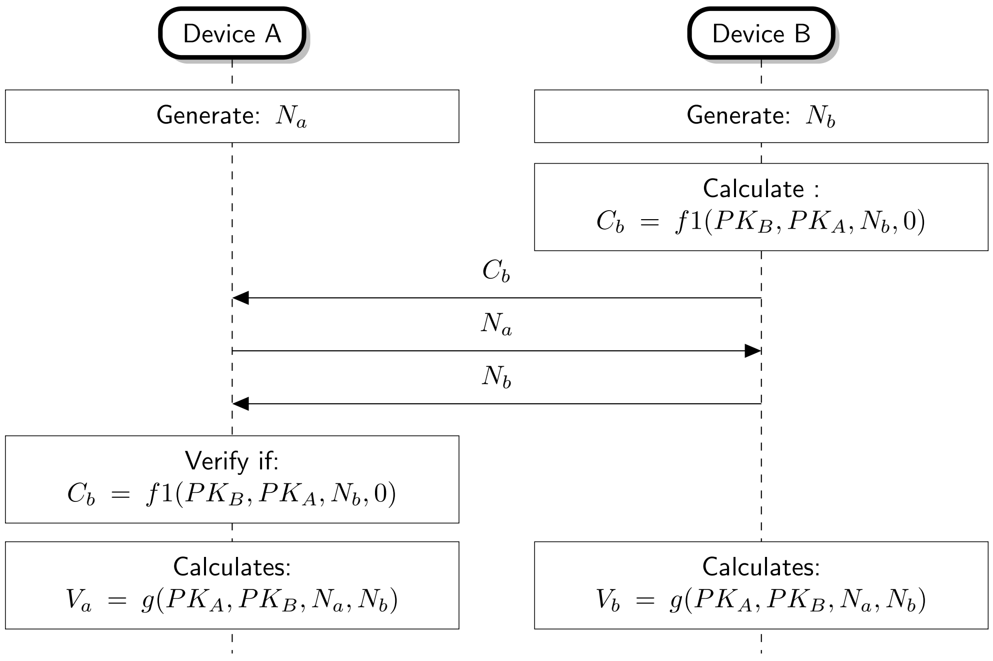

Device A
Device B
Generate: N_a
Generate: N_b
Calculate : $C_b = f1(PK_B, PK_A, N_b, 0)$
C_b
N_a
N_b
Verify if: $C_b = f1(PK_B, PK_A, N_b, 0)$
Calculates: $V_a = g(PK_A, PK_B, N_a, N_b)$
Calculates: $V_b = g(PK_A, PK_B, N_a, N_b)$

Figure 2.11: Sequence diagram for the SSP - phase two

In the following phase, called *Authentication Phase 2*, a new message exchange is used to confirm that both devices have successfully completed all the steps until this moment. A new confirmation value is computed by using the previously exchanged values and this value is exchanged between the devices. Both devices then check if the exchanged values match with the expected value. If these checks fails, it means that the pairing has not been confirmed. Phase four involves the link key calculation. There is no message exchange between peers at this point. Finally, in Phase five, the encryption keys are calculated.

Although major attacks against the protocol have not yet been found, therefore we could consider that the SSP protocol is secure, we still find attacks based on a forced change from a strong association mode to a weaker one [51], or a user's misinterpretation of concurrent pairing sessions [34]. Both attacks focus on possible real-world uses (and on specific association modes) of the protocol rather than its specification. That is, they focus on the Bluetooth SSP ceremony rather than the SSP protocol.

As we can see, there is more involved in this protocol than a classic protocol analysis covers. There is human verification and interaction with the devices and humans communicating with each other. This makes the SSP protocol (and even the legacy mode) an excellent case study for ceremonies.

2.4.2 WhatsApp Messenger Registration

WhatsApp Messenger is a messaging application for mobile devices that allow users to exchange text, images, audio and video messages. It is available for several mobile operating systems such as Android, iOS, Blackberry, Windows and Symbian. It is an extremely popular and fast growing messaging service. According to statistics published by their account on Twitter, in August 2012 the number of messages sent and received via WhatsApp in a day was around ten billion [108]. In June 2013, this number went to twenty seven billion [109] per day, with a database of 300 million of active users [75].

The registration and the communication protocols used by WhatsApp are proprietary. Despite not being available as a public protocol, there are several attacks that exploit the message exchange during the registration process between the application (client) and the server based on reverse engineering the registration and communication protocols.

2.4.2.1 Registration Protocol

WhatsApp user registration uses the user's mobile phone number as basis for identification. The basic registration scenario begins with the user choosing his country and entering his phone number in to the application. Here we note the first curious part of the process. Rather than detecting the phone number, the application asks the user to insert the number. After some investigation, we found that Apple's iOS SDK does not allow applications to access the device's phone number. Android SDK, on the other hand, grants access to the phone number, but this feature is not currently in use by

WhatsApp. There are some potential benefits by asking the user to input the phone number. Firstly, by keeping the user registration process homogenous across different operating systems, it improves usability and user experience. Secondly, it allows the user to register his account using a device which does not have a phone number (e.g. tablet) as long as the user has access to mobile phone capable of receiving SMS during the registration process.

For obvious reasons, asking the user to insert a phone number comes with a security risk. If a user inserts someone else's phone number, he could hijack or create a user account for a number he does not own. To prevent such a problem from happening, an SMS verification mechanism is implemented. After receiving the user input phone number, the application then sends the information entered to the registration server. In response, the server returns an acknowledgement. Additionally, when receiving the request, the server sends an SMS message containing a PIN code to the phone number that requested the registration. The SMS is sent via an alternative channel (separate from the network channel used during the rest of the process) and it is used to verify the possession of the phone number by the device making the request. The PIN received via SMS is then entered by the user in to the application, which is then sent to the server and the registration is confirmed by binding the phone number to the device. Figure 2.12 gives an overview of the registration process. Note that alternatively, in case the SMS verification fails, a method based on an automated phone call to the user is used. The process is the same, the user receives an automated phone call to the number he entered and the recording spells the PIN code that the user has to insert in to the application.

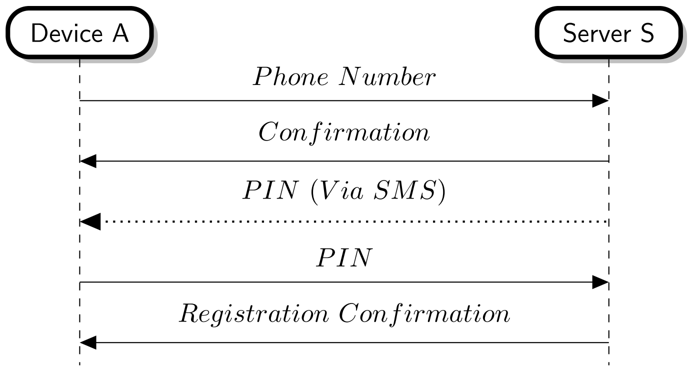


Figure 2.12: Sequence diagram for the general WhatsApp registration protocol

Although the general registration process presented in Figure 2.12 seems to represent the expected protocol specification, we found that the real implementation, at least for earlier versions of the protocol, had some variations and presented serious security flaws.

The first relevant difference is that in earlier versions, the PIN was generated by the application and sent to the server along with the phone number through a HTTPS connection [91, 105]. An important issue to note is that the phone number and PIN sent to the server are not used for registration at this point. These values are only used to allow the server to send an SMS. This SMS is sent to the phone number received and contains the PIN code informed [61]. All the information received at this point is simply discarded after the SMS is sent. This introduces a security flaw where an attacker could simply intercept the HTTPS request sent from the device to the server and read the PIN that was supposed to be delivered via SMS. Despite the fact the communication is SSL-protected, the attacker would only have to intercept the connection between his own device and the server. This can easily be performed using a proxy server and tools such as SSLSniff [65]. By doing that, the attacker can insert any phone number he wants and easily obtain the PIN generated by the application to successfully perform the registration. The attacker could even prevent the request from being sent to the server (this is not necessary for the attack to succeed, but it is useful to prevent the user from receiving an unexpected SMS). According to some tests we performed using Whats-API [102], which allows us to send and receive requests to the server and manipulate the parameters, this flaw seems to be solved now and the PIN is generated on the server side.

As an attempt to fix the problem, the following versions changed the registration and removed the verification of the PIN from the application. Instead of just contacting the server to ask for an SMS to be sent, the application sends a request including the device's phone number and the server responds with a confirmation message. However, rather than just acknowledging that the message has been received, the response message includes the verification PIN code [105]. The most probable reason for that is because the SMS message with the PIN code is not always sent from the server to the device. There are two variations for sending the SMS. The first, called *self*, is designed for Android devices and the device sends an SMS to its own number to confirm the ownership of the number. The application then keeps monitoring incoming SMS messages waiting for the SMS containing the PIN and, when it is received, it automatically reads the number and completes the registration. The second, called *sms*, is the method where the SMS is sent by the server in order to verify the ownership of the phone.

However, both *self* and *sms* methods were still vulnerable to attacks. Since the PIN can be discovered prior to the reception of the SMS, an attacker could use a simple variation of the attack presented above, and sniff the SSL connection between his own device and the server to get the PIN. Then, for the *self* method he could simply spoof an SMS sent from his own device, using the PIN he captured and complete the registration. For the *sms* method, it would be even simpler, we could just enter the PIN in to the application and complete the registration.

The latest version of Whatsapp changed the registration process. It is now similar to that we present in Figure 2.12. According to some tests we performed using Whats-API [102], the PIN is no longer displayed when the registration is performed using the *sms* method. However, the PIN, which is now a longer code, is still returned when the *self* method is requested. Although we could not test whether the attack described above would work in this new version, we believe that the attack is still possible, since the attacker would still have access to the PIN in advance when requesting the *self* method.

In Chapter 5, we will discuss more details of some of the scenarios above from a ceremony point of view.

Chapter 3

Understanding the Human-Protocol Interaction

Contents

In this chapter we propose a taxonomy of the most common characteristics and behavioural patterns involved in human-protocol interaction. Along with the taxonomy, we present a set of design recommendations to minimise the problems inherent in the human-protocol interaction. This was the starting point of our research, which led us to understand why some protocols fail in practice and how we can address some of the existing problems. The content of this chapter is a revised and updated version of the paper "Understanding the weaknesses of human-protocol interaction" by the author and Geraint Price published in the proceedings of the 16th international conference on Financial Cryptography and Data Security [33].

3.1 Introduction

The volume of data that flows through modern networks is immense. People use many different applications and tools to store, process and manipulate this data on a daily basis. Among this intensive data flow lies sensitive information. To ensure that this sensitive information is adequately protected, different security mechanisms and protocols have been developed to provide security services such as confidentiality, integrity and authenticity.

There is a wide range of protocols, each one with its own goals and characteristics. To check whether those protocols are correct or not, that is, they provide the properties they claim, verification techniques have been developed. However, there are situations where even some of the most secure and robust protocols are vulnerable to attacks when implemented. A significant number of these attacks are against the non-cryptographic components, such as the human-protocol interaction.

In protocol design and analysis, the human interaction is usually part of the assumptions and not specifically included in the description (e.g. an assumption that a user is able to verify a digital certificate, or will generate strong passwords). However, user behaviour is often unpredictable, making the assumptions not precise enough [47, 66]. Despite that unpredictable nature, there are some common design errors and weak assumptions that can be avoided if taken into account.

We conducted a thorough study of the existing work, among different areas, to identify and understand the most common characteristics and behavioural patterns involved in the human-protocol interaction. Based on the results of this study, we propose a unified set of often overlooked human-protocol interaction components that merges the findings from different research areas into a harmonised taxonomy. In particular, we focus on human characteristics that are usually overlooked during the protocol design process.

Based on the proposed taxonomy, we then discuss ways in which we can factor in these weaknesses and minimise their impacts. Our analysis is based partly on related research findings, but is also based on our own proposals which evolved from our taxonomy of weaknesses. Ultimately, this allows us to present a set of design recommendations to address the problems inherent in human-protocol interaction. What is interesting is that this set is not simply a linear evolution of the taxonomy of weaknesses. What we can see is that a second independent layer of structure emerges when we separately consider the categorisation of solutions to the problems we collate and identify in the taxonomy of weaknesses.

3.2 Overview

Human computer interaction is a topic which spans several different areas, including computer science, sociology and psychology. A large portion of research in this subject is related to design and usability of security systems. Each of these research areas independently address different layers of systems security. Figure 3.1 gives us an overview of the layers involved. The specification layer (I), represents the protocol specification; the application layer (II) contains the implementation of the specified protocol in an application; the interface (III) brings a point of interaction between the application and user layers; finally, the user layer (IV) represents a user of the application.

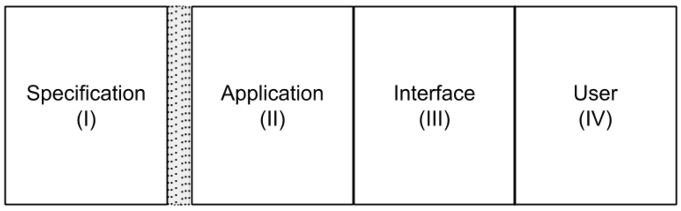


Figure 3.1: Human-protocol interaction layers

Software engineering is focused on the application's design and implementation (application layer); humancomputer interaction (HCI) focuses on the interface and usability aspects (interface and user layers); computer security design focuses on the design and implementation level (specification and application). Most of the existing research within these areas focuses on a specific layer rather than the whole set. This creates a gap, particularly between the specification and application layers where the human-protocol interaction involved has received little attention. The human interaction within a ceremony context, which we focus on this work, crosses all the layers, such as specification, application, interface and user.

Within the scope of human-protocol interaction we consider any type of action performed by a user that might impact on the security properties of a system. Since the interaction is usually made via a software interface, we have to consider usability issues. Additionally, we need to look at the implementation and specification levels. In a protocol specification, the human-protocol interaction is usually part of the design assumptions, that is, static components where actions are assumed to happen without being explicitly included in the specification. When implemented, these static choices are then replaced by dynamic user-interactions. It is often the case where the assumptions are inaccurate, as a result, it is very unlikely that an implementation can provide the expected security properties. Our work focuses on detecting patterns of problems that occur during the human-protocol interaction and highlight human characteristics that are often overlooked during protocol design and implementation. We explore the findings of research within these layers to construct a broader point of view.

As mentioned above, different research fields address security issues in different ways. However, we were able to observe overlaps in the way those issues are dealt with. These overlaps occur due to the use of different terminologies and distinct research goals. For example, there have been studies of phishing attacks [56, 76, 111, 40, 39]; users' susceptibility to attacks [44, 87]; factors exploited to allow an attack to be successful [99, 41, 48, 88] amongst others. The findings from these experiments are a very good source of analysis and comprehension of users' weaknesses. Among this wealth of studies, we found that similar findings are labelled in different ways. We also found that some findings, applied to a specific context, can be extended to different contexts. Each of these existing areas of research independently contributed to the construction of the set of human-protocol interaction weaknesses and recommendations we propose.

3.3 Frequently Overlooked Components of Human-Protocol Interaction

The interaction between computers is relatively easy to define and the results are, in general, predictable. However, defining or predicting human characteristics or behavioural patterns is a challenging task. It requires a more subtle approach, commonly based on empirical results [66]. Despite this complicating factor, we can create a generic (but not perfect) human-protocol interaction model by using empirical and statistical information to improve the human-protocol interaction process.

Therefore, it is necessary to study and analyse the most common characteristics and behavioural patterns regarding human-protocol interaction. As we will see throughout this Section, there is a significant amount of research which maps human characteristics (or principles, weaknesses, etc.). This existing body of work offers important and relevant insights that constitute the basis of the set of overlooked components of human-protocol interaction we discuss during this section. By analysing existing work and detecting the common components among them, we define our list of five main overlooked components of human-protocol interaction: user knowledge, authentication capabilities, decision making influencing factors, bounded attention, and inherent characteristics.

The taxonomy we present here is necessarily incomplete, simply because new technologies, new attacking tactics and new human-computer interaction methods may arise. Nevertheless, this taxonomy reflects the current situation for overlooked components of human-protocol interaction.

3.3.1 User Knowledge

We define knowledge as familiarity, awareness, experience or understanding of a certain subject. Within the human-protocol interaction context, users' knowledge certainly is an important factor to be considered. We have seen several situations where this factor (or the lack of it) is exploited by attackers. In general, most users do not have extensive knowledge of how computers, operating systems and software work. Furthermore, even users who are familiar with computing do not necessarily have knowledge about computer security, cryptography or security protocols. Therefore, a secure human-protocol interaction should carefully consider users' knowledge.

Phishing attacks provide a very interesting case study from which we can evaluate human-protocol interaction. The main reason is that, in many cases, the user has to interact with the implementation of the SSL/TLS protocol [42]. Dhamija et al. [41] developed a usability study to understand how phishing attacks work. In their research, a set of hypotheses was developed and tested. Within this set, they identified that, indeed, users are not familiar with computer systems, security, security indicators and risks (such as web frauds). Similar findings are presented by Wu et al. [111], Downs et al. [44], Sasse et al. [87] and Jakobsson [56], among several others.

The obvious approach to minimise problems related to untrained users is training them. It does, indeed, improve some aspects of user practices, such as strong password generation [101] and not revealing private information in phishing websites [60]. However, there are also side effects, such as a greater likelihood of writing down passwords due to human memory limitations (related to the component we describe in Section 3.3.5) [101] and falling for spoofs of security indicators [44] because of human limited authentication capabilities (see Section 3.3.2).

When an attacker is able to perceive a weakness in the victim's level of knowledge, it is relatively easy to manipulate and exploit it to successfully attack a protocol. In phishing attacks, for example, the attacker exploits the victim's lack of knowledge of computer systems (e.g. inability to recognise URLs and redirections [111, 41]), security risks (e.g. many users do not know that websites can be cloned [41]) and lack of knowledge in computer security (e.g. when they erroneously decide to trust a website solely on the fact that they see the picture of a padlock [56]).

In summary, we can list the knowledge-related issues that are most commonly exploited by attackers:

Lack of knowledge of computing – Despite the fact of being computer users, many people do not (and probably should not be required to) have a proper understanding of how operating systems, networks and protocols work [41, 44, 111]. Techniques and attack methods such as web site redirections, URL masking, malware and others are often used by attackers to exploit this fact. Systems often have security failures because they are too difficult to be used. Even educated and careful users sometimes cannot understand security-relevant user interfaces [94].

Lack of knowledge of security – Most users do not have knowledge and understanding about digital certificates, encryption mechanisms and most of security technologies [41, 111, 56, 44]. In a test that required users to detect phishing websites, 95% of users did not know the meaning of the browser warning about an untrusted server certificate [41]. Additionally, users often do not know how to identify a spoofed website or email, and do not understand the meaning of security indicators [111].

Lack of knowledge of security threats – Many users do not know they can be attacked or attacks such as spoofing websites are even possible [41, 111]. Furthermore, even those with a better understanding of the risks, are not usually familiar with many techniques used by attackers.

Inaccurate mental models – There are many indicators that people are likely to construct their own concepts about computing and security. Jakobsson [56] observed that people judge relevance of content before checking for indicators of authenticity. We have also seen situations where users create their own (and frequently inaccurate) concepts about how the authentication process works [4], and about the likelihood and impacts of attacks ("They could not do much damage anyway" [87]).

Human-protocol interaction cannot rely on users' computing/security knowledge and awareness. A secure human-protocol interaction must consider users knowledge and, preferably not require higher training levels. Taking into account the level of knowledge and understanding relevant to the target users is very important to avoid security flaws due to an overlooked target audience. Also, the more generic the audience, the lower level of understanding should be required. It is a design flaw to assume that the users are knowledgeable in something they are not. Thus, when designing a human-protocol interaction, giving tasks to users where they might have insufficient knowledge to perform that task should always be avoided.

3.3.2 Authentication Capabilities

An authentication performed by a user is a task where the user verifies that the authenticating party is whom it is expected to be. People frequently make use of visual cues as an important authentication tool. However, there are studies [98, 99, 41, 111, 56] that show that this visual authentication mechanism is weak and unreliable in some cases. Stajano and Wilson [98, 99] performed several experiments as attempts to detect common exploited patterns in scam scenarios. The limited authentication capabilities inherent in populations of users was often an important (if not the most important) factor to the success of their attacks. Their study demonstrated that people are usually very good at recognising people they already know, however they are not good at authenticating strangers (someone they do not know) or objects. The more visually equivalent to the original a spoofed element is, the more likely it is to be accepted as authentic.

Other research [41, 111, 56] achieved similar outcomes, corroborating Stajano's results. Many additional techniques were used to deceive users, such as fake email messages, spoofed websites and untrusted certificates. In most cases users were fooled, and consequently attacked, because they failed to authenticate objects. Therefore, human authentication should be avoided for authenticating strangers or objects (either real or digital).

In a human-protocol interaction scenario, the attacker may exploit the user's limited authentication capabilities to make the victim believe the messages haven been exchanged with a trusted party. Once the user authenticates the attacking party as valid (by accepting the attacker's digital certificate, or simply ignoring security indicators), the attacker is extremely close to achieve their objectives.

In general, we found four different users' authentication skills that are well known by attackers, but not always correctly addressed by designers:

Users are good at authenticating people they know – In general, users are very good and efficient at authenticating people they know [98]. Indeed, facial recognition for computer systems still faces several challenges, while the human visual system works very well at recognising other people. Even when not in person, humans can accurately recognise familiar faces displayed in computer displays or photos (even in low-resolution images) [93, 52].

Users are not good at authenticating objects – Users usually have problems authenticating objects [98, 39, 111]. Objects are easy to spoof, with it often being possible for an attacker to produce exact copies of objects. When facing a spoofed object, it is very likely that the spoofed copy will be perceived by the user as original. A fake card reader machine given to a user in a restaurant, for example, is likely to be unnoticed by the user who will type his PIN in to the machine.

Users are not good at authenticating strangers – People are not usually good at recognising unfamiliar people or people they do not know [93]. Bruce et al., for example, demonstrated that humans perform poorly when matching different photographs of an unfamiliar person [27]. Additionally, people are not good at establishing whether someone belongs to a designated class (e.g. confirm whether a person dressed as a police officer belongs indeed to a police squad) [98]. When authenticating strangers, users have to make use of other authenticating factors, such as documents or references given by someone else (e.g. physical attributes). This shifts the authentication type to an object-based authentication, which, as we have seen, is not precise enough to be used in security protocols. In addition to that, other factors, such as social conditioning and time pressure (discussed in Section 3.3.3) may be used to weaken user's authentication capacity even further.

Users are not good at authenticating digital objects – In the same way as real objects, digital objects, such as websites, software and email are not easily authenticated by users. Also, the use of deceptive texts (exploiting similar drawing patterns of some characters to avoid users from detecting relevant information) and masking information (using images or windows on top of legitimate text or other elements) are successfully used by attackers [41]. By creating visually identical (or very similar) copies of the original source, attackers can fool users into believing they are contacting the entity they trust. Finally, personalised content increases the chance of a spoofed virtual object being accepted as real [44, 56].

Directly or indirectly, most scams and/or attacks exploit the false acceptance of spoofed content by users and almost all scams are forms of deception [99]. Asking a user to authenticate an object (e.g. an online banking website) by checking its elements (e.g. digital certificates, padlocks, etc.) does not represent a proper translation from the authentication design goal to its implementation. In fact, it is very likely that the authentication task, despite technically feasible, will not be performed properly, introducing a security breach.

3.3.3 Decision Making Influencing Factors

There are different factors that need to be taken into account when considering users' decision making capability and its influencing factors. These aspects include personal and environmental issues. Despite the differences, the core concept behind this component is that users can be influenced to make different (and potentially damaging) decisions to those they would usually make. Thus, when designing human-protocol interaction, security engineers must be aware of which factors may influence the user's decisions and check whether this decision under influence can introduce security breaches or not.

The most common influencing factors are:

Social conditioning – When people receive commands from strangers, they are unlikely to follow that command without questioning the request. However, when the command comes from a recognised authority (or someone mimicking an authority), people are very likely to obey such a command. This happens because people are trained to accept commands from certain people, such as police officers, their bosses, etc., without further rationalisation [99, 87, 79]. An attacker can make use of social conditioning to force people to behave in a predictable manner and use it to deploy a successful attack. Nevertheless, before using social compliance, an attacker needs to make the victims believe they are receiving orders from an authority figure. The manipulation of decision making is often exploited in conjunction with another component we discussed previously: users' authentication capabilities. Therefore, an authentication failure is often a pre-requisite for exploiting social conditioning.

User's principles – Victims' principles such as guilt, dishonesty, need and/or greed, when exploited by an attacker, are powerful ingredients to increase the effectiveness of an attack. People's needs and desires make them vulnerable because the attacker can use them to force the victim to behave in a predictable manner. Once attackers know what the victim wants, they can easily manipulate the victim's behaviour [99, 79]. One classical example of an attack exploiting this principle is the 419 scam (or Nigerian fraud) where a significant amount of money is offered to the victim, but before receiving the money, they are persuaded to advance a certain sum of money in the hope of receiving a significantly larger gain afterwards. In this attack, victims' greed, need and also dishonesty are manipulated (not necessarily all of them), making the victims more vulnerable than they usually are. Another example is related to guilt. An attacker could manipulate the victim's feelings by approaching the target and pretending that they desperately need some help. In such a scenario, it is very hard to refuse the request, otherwise the victim will feel guilty for not helping someone who desperately needs help [79]. These violations (deviations from safe operating practices, procedures, standards, or rules) are different from user errors (discussed in Section 3.3.5). Errors arise from informational problems (inattention, lack of knowledge), while violations are motivational, the actions - despite being unsafe - are intended [82].

Time constraints – The main idea behind this factor is to push the victim into making a decision without sufficient time to rationalise the decision. Consequently, the actions taken by the victim tend to be more predictable and easier to manipulate [99]. In attacks that exploit time constraints, the victims believe they must act quickly, otherwise they might lose the opportunity. Also, the decision strategies used under time pressure are typically based on affective and intuitive heuristics, rather than on a reasoned examination of all the possible options [48].

Shared risk – This aspect can be easily seen in real-world situations where someone is worried about taking a certain action but accepts the risk because there are many others sharing the same risk [99]. An attacker can exploit this by including other people (other attackers) faking that they accept the risk. When the victims see other people accepting the risk, they tend to accept the risk, allowing the attacker to be successful. In online reputation services, we can see an example where shared risk can be used. The attacker can create fake profiles and insert as many reviews as he wants to improve the 'reliability' of the service or product offered. Additionally, offering additional 'verification' options (such as a customer service number or chat in a website) also creates trust. Subjects stated that they would not call the number to verify the authenticity, but "someone else would" [56, 79].

Fear – Many techniques such as scareware are effectively used by attackers to scare people and make them fall into attacks. Warnings presenting scary messages that your computer is infected, or that you might be attacked by a hacker, etc., makes users act without proper rationalisation to solve the problem (e.g. Mac defender malware[1]). Fear changes the decision strategies of the users, making them more likely to behave in a predictable manner. Another type of fear exploited by an attacker is the fear of getting into trouble. For example, the fear of receiving a negative reaction from their superiors because a routine security check was applied (even though it should be applied) to an important user and that person (boss, or an important visitor) felt offended [79]. This second type is related to social conditioning since, in addition to the fear of having problems with their bosses, the victim is dealing with an important person or authority.

Users' decision making factors involve many different factors that should be carefully analysed. Even trained users might have their decision strategies manipulated under certain circumstances. People will make errors and will eventually make wrong decisions. To prevent an attacker from exploiting this, it is important to identify potential situations where this component might be exploited and make the system insensitive to them. Alternative methods to avoid unreasonable decisions by users should be considered. These possibilities may include extra security checks, additional verifications, or a reduction in the impact that the user interaction has on the security properties of the protocol.

3.3.4 Bounded Attention

Users focus on their main task, and consequently, most of their attention is bound to the activity of performing that task. According to Stajano [99], people tend to forget the task of protecting themselves when it is not their main activity at a given moment. Security protocols are frequently used as part of a computational system or software. Consequently, from the users' perspective the protocol used and the security aspects of the protocol are a secondary concern. As a result, they may not notice security indicators and warnings.

[1]http://support.apple.com/kb/ht4650

The users' bounded attention can potentially be exploited in the case of lack of attention to security indicators, or even in the lack of attention to the absence of security indicators [41]. Wu [111] conducted an experiment to check the efficiency of security toolbars when trying to prevent phishing attacks. About 45% of users who were spoofed said that the reason was they were focused on finishing their main task (i.e. dealing with email requests). Some of the spoofed users explicitly mentioned that reason, and even having noticed the security warnings, they decided to take some risks to complete their task.

Users have a tendency to notice only what they are interested in and ignore the fact those security mechanisms were created to protect them from attacks [99, 41]. For example, in the SSL/TLS protocol implementation in web browsers, when an untrusted server certificate is sent to the client's browser, the user is asked whether to trust the certificate presented or not. In this case, users' bounded attention principle can be successfully exploited because users tend to dismiss the security warning of the untrusted server certificate presented by the server. In this case, the security decision is presented in a context were the focus is on accessing a website. Consequently, a warning informing users that an untrusted certificate had been presented and asking whether to continue or not is very likely to be quickly dismissed since it is stopping the users from completing their main tasks. In other research, Herley [53] argues that users rejection of warnings and security messages is entirely understandable and rational from a economical point of view. He states that it is necessary to prioritize the important security messages and remove the others that do not do much to address security threats. By doing that, it is possible to improve the cost-benefit tradeoff that currently exists regarding security advice.

If a warning is required to be presented to a user, this should implemented using an active interruption. Egelman et al. [45] conducted an investigation into the effectiveness of passive and active warnings. Active interruptions, that is, indicators that stop users from performing their tasks and only allow them to proceed after certain steps are executed, were far more efficient than passive warnings. The trade-off between usability and security must also be considered. An excessive number of interruptions may train users to dismiss all security warnings without proper rationalisation.

We found four main factors which can potentially weaken the security aspects of human-protocol interaction:

Lack of attention to security – The user's focus is not on the security aspects of the system, and consequently, the security checks tend to be executed less carefully. Using the SSL/TLS protocol as example, users who are aware of the security indicators of an SSL connection in the web browser, despite the fact that they understand the need to look for security indicators, may simply look for the image of a padlock on the webpage. We have seen examples of attacks that exploit such a factor [41, 44, 64, 63]. In this case, the users were fooled by a spoofed icon appearing in the body of the web page rather than in the browser's chrome. This is closely related to the user knowledge, users that are knowledgeable of security may still fall for such attack if they are not aware of the security threats (e.g. spoofing a security indicator).

Lack of attention to the absence of security – Another factor is the lack of attention to the absence of security indicators. In the same way that security checks and indicators are not the users' main focus, they are likely to be ignored [41]. Attacks such as SSLStripping, presented by Marlinspike, exploit the fact that most requests to HTTPS websites are performed via links or redirections (HTTP response status code 302) [64, 63]. With that, an attacker could just switch url links and/or location that contains https to http. The indicators in the web browser (usually a padlock) are not shown to the user in this case. However, the lack of such indicators are very likely to go unnoticed. Even some additional tricks exploiting users' knowledge can be employed to make the attack even more efficient (e.g. sending a image of a padlock as a favicon, which mimics a security indicator in the address bar). Additionally, in some cases, users might not even know that security is a relevant part of the interaction [107].

Security in a secondary workflow – Users are more likely to finish their main task rather than stop it due to a security warning. Therefore, security should not be part of a secondary workflow. Security checks that occur outside of the main task, interrupt the user's focus, and are more likely to be dismissed without much consideration. Usually, users' primary goal when using a computer is not security per se, but communicating with friends, using online services, etc. Users will try to finish their main tasks if they believe they are more important than the security tasks, even if there are potential risks [111, 35].

Conditioning – An excessive number security interruptions ends up training users to dismiss warnings, pop-up boxes and any other security interruption in a insecure way because this is the only (or the simplest) way to finish their tasks. Some attacks successfully exploit this factor because it is known that people will have this behaviour even when they know they should not [6]. We have several examples of an excessive number of warnings. Older versions of the web browser Internet Explorer, for example, used to warn users for every change in the security context, even when successfully changing from an insecure to a secure context. This excessive number of warnings can make users become less inclined to take them seriously in the future [73, 35].

Security tasks are more effective when they are included in the main workflow. Simply warning users by stating that something is wrong is not sufficient: they need to be provided with a safe alternative to achieve their goals [111]. Additionally, security activities should be designed to disrupt user's workflow as little as possible to minimise the impact of these additional tasks on the user's motivation to perform the security activity [35]. Finally, bounded attention may also be affected by user's knowledge. For example, a user may not know what security cues they should look for or whether the operation being performed is insecure or not.

3.3.5 Inherent Limitations

Human skills are vast and vary significantly. It includes proficiency or ability that is acquired or developed through training or experience. Overlooked inherent characteristics encompasses situations where human skills might not be sufficient to perform an activity or task as intended. It also includes particularities of human behaviour. We cannot expect that humans behave in a similar manner to a computer, nor believe they share similar skills. By equating these two different components during the human-protocol interaction, a series of security threats may arise.

A usable and secure design must consider human abilities and check what people can, and more importantly, what they cannot do well [40]. We cannot expect people to store large amounts of data, as well as we cannot assume they will 'erase' data from their memories once that information is not needed anymore [87]. In the same way, we cannot expect a human-protocol interaction to be performed in the same way, or under the same timing constraint on every run. Thus, skill limitations should always be considered within the scope of human-protocol interaction. There are several skills we should consider when designing secure systems:

Memory limitations – Sasse et al. listed the most important issues related to human memory (focusing on password memorability) [87]. From her list, we must highlight that human capacity for working memory is limited and decays over time, meaning that they may may not be able to recall information when needed or not recall it accurately. Items used often (such as passwords you use on a daily basis) are easier to remember than those that are rarely used. Additionally, people cannot 'forget on demand', so even undesired items will remain in memory even when they are no longer needed. Finally, Sasse et al. highlight meaningful items (e.g. words) are easier to recall than non-meaningful ones (e.g.

randomly generated passwords). Other research corroborates Sasse's list, showing that users cannot remember large and random keys or recall dozens of different passwords [35, 112]. Dhamija and Perrig [38] also pinpoint that precise recall is not a strong point of human cognition. They state that the main weakness of knowledge-based authentication is that it relies on precise recall of the secret information. An authentication process does not allow variations on the secret, therefore, it fails even if the user makes a small mistake when entering the secret (e.g. password).

Lapses, Slips and Mistakes – Lapses, slips and mistakes are very common human error types. A lapse happens when the plan to achieve a certain goal is correct but an error happens when a required action is forgotten [82, 35] (e.g. a required step in a sequence of actions is skipped). A slip, in the same way as a lapse, occurs when the plan is correct, but in this case, an action is performed incorrectly [82, 35]. For example, when a user wants to type "password", but instead of typing the correct word, a slip happens and the word "passwprd" is typed because the wrong key was accidentally pressed. A mistake, on the other hand, occur when the plan to achieve a certain goal is inadequate, that is, even if the the actions performed go as planned, it will not achieve the desired goal [82, 35] (e.g. a user checking if he/she trusts a website by looking for a picture of a padlock in its contents).

Problem solving limitations – Some problems can be easily solved by some users but the same problem can be a complex task for others. This limitation can be influenced by many of the previously presented weaknesses, such as lack of knowledge or bounded attention. Even in the case of the user being capable of understanding the task received, knowing how and when it should be applied, a failure may still happen if the user does not have the capability to perform the appropriate actions [35].

Task termination – A user may decide to terminate the interaction. When users finish their main task, they might leave the subsidiary tasks incomplete. For example, a user accessing a webmail, after reading the messages, may leave the computer without logging out. This is acceptable on private computers, but it is a security risk in public environments. In a similar manner, users may terminate the interaction if they assume there is no alternative to proceed due to a fault or an unexpected system state [84].

Non-deterministic behaviour – As opposed to the previous limitations, this issue is not related to a limited set of capabilities, in fact it is the opposite situation. According to Ruksenas [84], in any situation, any one of several cognitively plausible behaviours might be taken. It cannot be assumed that any specific plausible behaviour will be the one that a person will follow where there are alternatives.

We cannot expect that users have skills or abilities they do not have. Human-protocol interaction should be designed taking the user's inherent characteristics into account and checking whether the task given to the user is feasible or not. Even educated users may fail when performing a security task if their capabilities (e.g. physical skills) are not properly considered [35]. For example, despite understanding the importance of creating strong passwords, a policy that requires a password length of 10 or more random characters is not feasible for most users due to memory limitations. Also, limitations on the number of password length and attempts (e.g. a maximum 3 attempts before blocking the account), despite being important to prevent password guessing attacks, might overload the user's memory and reduce compliance with security rules.

3.4 Minimising the Weaknesses in the Interaction

After merging several research findings into a harmonised and limited set of often overlooked human-protocol interaction components, a set of design recommendations to minimise the impact of these components is an obvious next step. Despite the wide range of users' characteristics and behavioural patterns, we were able to propose a taxonomy of frequently overlooked characteristics of human-protocol interaction, and from these characteristics, we can develop a set of design recommendations.

To construct a set of recommendations on how to reduce the effectiveness of attacks exploiting these human-protocol interaction weaknesses, we initially attempted to make a one-to-one association between a weakness and a corresponding recommendation where, for each weakness, we proposed a recommendation. We independently analysed each component of the taxonomy presented in Section 3.3, and by making use of our findings and results from related work we drew a recommendation aiming at minimising the impact of that specific component on the security of the human-protocol interaction.

What we found was that some factors, even those belonging to the same taxonomy item, have to be treated in different ways. However, the opposite situation was also found, when factors from different taxonomy items could be handled in a similar manner.

The correspondence between the taxonomy of human-protocol interaction weaknesses that we proposed and the design recommendations is not linear. There is a separate layer of abstraction that makes the associations between our taxonomy to the set design recommendations. In this section, we discuss how we tackled each weakness and its subdivisions and we highlight their associations to the design recommendations we will outline in Section 3.5.

3.4.1 User Knowledge

To perform a task correctly, users need to have a certain level of understanding of the key components that make up the task. If they do not have sufficient understanding, one or more sub-tasks are likely to fail. Within the context of human-protocol interaction, there are some fields of knowledge which are pre-requisites in most cases, such as computing and security. When the user's understanding of computing and security is weak, other problems may also arise, such as the lack of knowledge about security threats, and also, the construction (by users) of inaccurate concepts.

To minimise problems related to the **lack of knowledge in computing** human-protocol interaction should not require tasks that require advanced knowledge about computing, such as configuration or filesystem management tasks. The same idea should be applied to the **lack of knowledge of security** problem. Users should not have to understand how encryption works, how to verify a digital signature, or how a certificate chain is built. We should only rely on security tasks which are feasible for ordinary users.

On the other hand, to address the issues related to the **lack of knowledge of security threats** and **inaccurate mental models**, it is necessary to make use of methods to prevent users from making incorrect decisions. We cannot expect users to be aware of potential threats, or that they understand the risks of making incorrect decisions. Additionally it is dangerous to assume that users will have an accurate idea of the protocol's workflow (e.g. authentication processes).

Training users, although recommended and effective if used correctly [101, 60, 112, 4], is mostly feasible in smaller environments, such as companies and other institutions. A protocol designed to be used by the general public within different environments should not rely on training users.

As we can see in Figure 3.2, there are three main types of approaches to minimise problems related to users' knowledge. The first is keeping realistic and low user knowledge requirements (as we will discuss in Section 3.5.1). The second is making use of methods to prevent users from performing unsafe actions, which we will discuss in Section 3.5.5. Finally, there is training, which we will not focus on in our design recommendations, since we will assume that we should not design protocols that rely on training users.

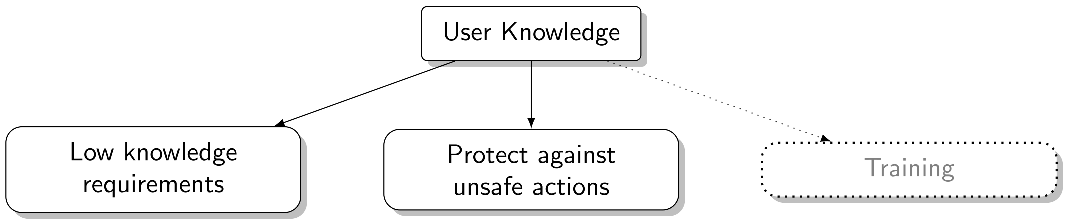


Figure 3.2: Minimising user knowledge issues

3.4.2 Authentication Capabilities

In the previous section, we highlighted that users have limited authentication capabilities. That is, they can easily make mistakes when asked to authenticate objects (real or virtual) and unknown people. Attackers make use of several techniques to fool users by using spoofed objects, such as identities, text, documents, and by doing that they are able to deploy their attacks.

It is necessary to understand and acknowledge that **users are not good at authenticating objects (real and digital)** or **strangers**. Consequently, objects and text, which are easy to spoof, should not be used as a component in a authentication process. By reducing the relevance of objects and text as authentication factors, masking information would become less effective for attackers. However, when an authentication performed by a user is a required component in the protocol workflow, this task should be designed to be feasible by ordinary users, that is, the user can perform the task accurately.

As we can see in Figure 3.3, there are two main types of approaches to tackle the limitations of users' authentication skills. First, we need to recognise the limitations of users' authentication skills and should not rely on such skills when designing protocols (as we will discuss further in Section 3.5.2). Second, we need to respect user faculties and provide them with feasible tasks, such as authenticating people they know or images they are familiar with (presented in Section 3.5.1).

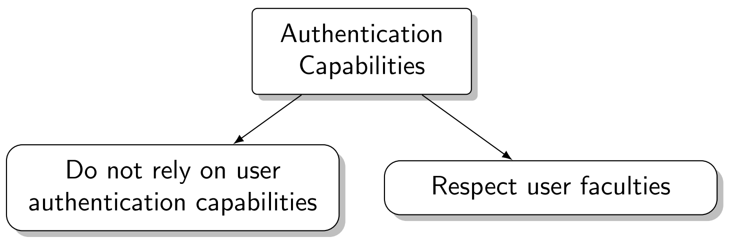

Authentication Capabilities
Do not rely on user authentication capabilities
Respect user faculties

Figure 3.3: Minimising authentication capabilities issues

3.4.3 Decision Making Influencing Factors

When a user has to make a decision during the interaction with the protocol, it is important to identify whether this decision might be affected when the user is under external influencing factors. In the Section 3.3.3, we listed social conditioning, user's principles, time constraints, shared risk and fear as the most relevant factors of influence during human-protocol interaction. Each one of them should be considered as a potential threat to the interaction's security.

Users influenced by **social conditioning**, by the manipulation of their **principles**, **sharing risk** with others or **afraid** of some kind of threat, change their perception of risk making their decision strategies different (and often less careful). Attackers exploit these factors by manipulating users' perception of risk, weakening the reliability of their decisions. When **time** is a limiting factor of an activity, users once again change their decision strategy making them more vulnerable. Additionally, when security is not the main activity, the decisions under timing constraints tend to be performed even more carelessly.

All five factors are exploited by attackers to change users' decision strategies. However, we found that for some factors, such as timing constraints, bounded attention is another determining factor that should be dealt with. To avoid that, security features should always be part of the main task. Furthermore, users' limited authentication capabilities are also often used by attackers when exploiting all of the five factors. A spoofed authority or entity that inspires some level of trust is often a pre-requisite for the attacker to be able to successfully deploy an attack based on decision making influencing factors.

As we can see in Figure 3.4, by considering that users' decision strategies are likely to change under different situations (as we will discuss in more detail in Section 3.5.4), making use of methods to prevent users from performing unsafe actions (which will be discussed in Section 3.5.5) and bringing security into the main workflow (discussed in Section 3.5.3), we can minimise the effects of decision making influencing factors on protocols. Since users' limited authentication capabilities are closely related to the decision making factor, we represented the approaches to tackle this factor by using dotted rectangles and arrows in the Figure 3.4.

3.4.4 Bounded Attention

Users are focused on their main activity, leaving secondary tasks in low priority. Therefore, protocols' implementation might present weaknesses during the users' interaction due to the lack of attention to the security features, lack of perception of absence

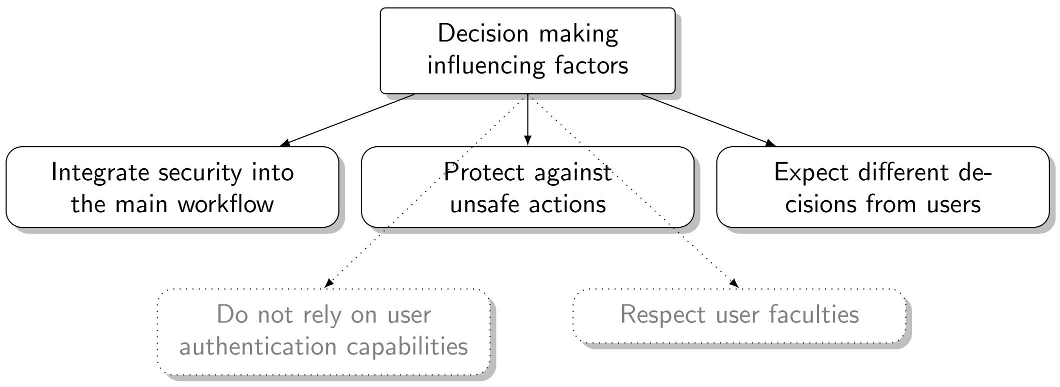

Decision making influencing factors
Integrate security into the main workflow
Protect against unsafe actions
Expect different decisions from users
Do not rely on user authentication capabilities
Respect user faculties

Figure 3.4: Minimising decision making influencing factors issues

of security, the fact of security is treated as a secondary task in the workflow, and conditioning.

To minimise the problems related to the **lack of attention to security**, it is necessary to make use of mechanisms to change the priority of the security activities within the users' main task. An effective way to deploy it is by bringing security activities into the users' main task [111]. In this way, we can bring the users' focus to the security aspects of the activity. This solution might also minimise the problems related to **lack of perception to absence of security**, since users will be more likely to detect the lack of a feature that was part of their main task. Changing the relevance of security activities in the main task also changes users' perception of the relevance of security. Since security is in the main workflow, **security will not be a secondary activity** anymore. Ultimately, by differentiating warnings and avoiding unnecessary security messages, we can minimise the effects of **conditioning** users to ignore security interruptions.

As we present in Figure 3.5, to minimise the impacts of user's bounded attention, we need to focus on integrating security into the main workflow. Including security actions as part of the users' main workflow rather than just notifying and warning in a secondary flow is a good way to approach the problem (we will discuss the integration of the security concerns into the main workflow in more details in Section 3.5.3).

3.4.5 Inherent Limitations

Inherent limitations, such as **memory**, **lapse, slips, mistakes** and **problem solving** skills are often underrated. In the same way that we have discussed the users' lack of knowledge, we cannot give users a task that they are not capable of performing, whether by physical or cognitive limitations. We cannot overload humans' memory by requiring users to 'store' large amounts of data, especially when it is random. Users might also

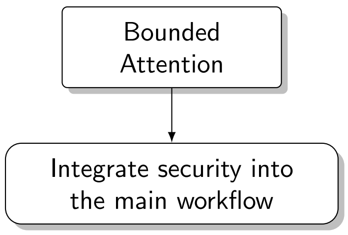


Figure 3.5: Minimising bounded attention issues

make mistakes due to motor issues (press the wrong key by accident). Additionally, if a problem is given to users to solve, we should also consider humans' problem solving limitations. All these issues should be considered before being added as part of the protocol design or implementation, and if this type of task is relevant to the protocol's scope, it should be feasible to general users.

Thus, tasks given to users should be feasible considering humans' limitations. There are other factors that influence the human limitation weakness, such as **human's unpredictable behavior** and **task termination**. Protocol implementations cannot assume that a task given to user will be completely performed (a user can potentially leave a task incomplete) or that a user will always make the correct decision (user decision is non-deterministic). In these situations it is necessary to use mechanisms to prevent incorrect decisions from being made.

As we can see in Figure 3.6, there are two main types of approaches to minimise problems due the inherent human limitations. First, we need to respect user faculties and provide them with feasible tasks (which we discuss in more details in Section 3.5.1). Second, it is necessary to make use of methods to prevent users from performing unsafe actions (that we will discuss in Section 3.5.5)

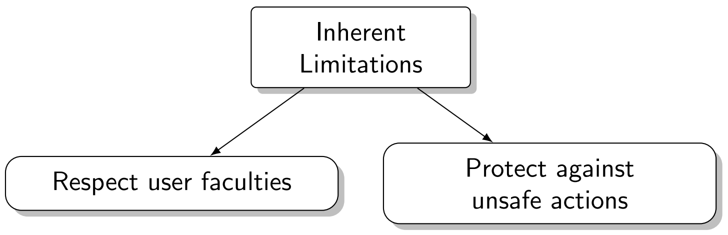


Figure 3.6: Minimising inherent limitations issues

3.5 Design Recommendations

By analysing the human-protocol interaction weaknesses we presented in Section 3.3 and by exploring related research findings, we were able to discuss ways to tackle these weaknesses and minimise their impacts. The associations among the weaknesses and solutions to minimise their impact were a key factor that allowed us to identify the design recommendations we discuss in this section. By proposing a set of design recommendations, we introduce guidelines to help designers overcome the problems presented in Section 3.3. In the following we describe each of our five design recommendations.

3.5.1 Respect User Faculties

Humans have different levels of knowledge and skills in a wide range of areas. Some people have stronger abilities in subjects such as logic or mathematics, others are better dealing with human sciences and so on. Some protocols might be expected to be used only by specialists and consequently could require a higher level of skill or knowledge. However, there are other protocols that are designed for general purpose use and, consequently, used by people that have different levels of skill and areas of expertise. In both cases, protocols should be designed and implemented keeping in mind the level of knowledge and skills of the people who will interact with it. In this recommendation we are merging two approaches we described in Section 3.4 that are related to human skills and knowledge: "low knowledge requirements" and "respect user faculties".

The SSL/TLS protocol implementation in web browsers, for example, is designed to be used by a wide range of people. In this protocol, which we will use as example throughout this Chapter, advanced knowledge about security technologies should not be required. During the protocol handshake, there is an assumption that the server certificate is previously known and trusted. However, there are situations where the certificate is unknown, untrusted, or does not match some requirement (e.g. the certificate content's "common name" field does not match the server's name). In these situations, when the server certificate is sent to the client, the browsers' implementations often ask the user whether to accept the certificate or not.

Taking a university's webmail service as an example, there are several cases where self-signed digital certificates are used to setup the SSL Server. Consequently, this certificate will not be initially included as a trusted certificate in the web browser's certificates lists, and subsequently the user will be asked whether to accept the server's certificate or not. Most users do not know what the decision presented to them means. Additionally, most of them do not know what a digital certificate is (and they probably should not have to know). Thus, the decision given to the user is an example of an infeasible task. It is necessary to create alternatives when tasks that require higher levels of knowledge are required. By simply leaving users unassisted, security flaws might be introduced.

Another example of a task that, depending on the context, may be considered unfeasible, is to require the user to generate input data to be used as part of the protocol workflow (e.g. random passwords). Certain inputs can represent fundamental elements of the whole system security and should be carefully analysed. For example, in Bluetooth Legacy Pairing (presented in Section 2.4.1.1), as well as in Kerberos [72] protocol, a user input (password) represents a fundamental part of the protocol security. A key that is used to encrypt some of the protocol's messages is derived from the user generated password in these examples. If we simply allow the user to create a password, a weak password might be generated and consequently compromise the protocol security [18]. On the other hand, defining password policies (e.g. minimum length, use of special characters, etc.) can also generate other types of problems such as information disclosure [55].

Humans don't have strong memorisation capabilities when dealing with random information, especially when dealing with various different sets of random information (e.g. a user that has many passwords). To deal with this memorisation issue, humans tend to disclose this information by using notes, which might allow other users to discover this data and use it for impersonation attacks. Thus, it is necessary to find alternatives to produce input data to a protocol which has sufficient quality to be used as a trustworthy source, as well as make its generation and use feasible to ordinary humans. Interesting results in password generation are described in [112], where they performed an experiment with 288 participants and concluded that training users to produce mnemonic phrases make them as easy to remember as the naively selected passwords and significantly improve security. Additionally, they showed that although educating users is important, it does not dramatically improve the compliance rate if not accompanied by monitoring and enforcement. Therefore, training users on how to produce random and/or mnemonic phrases and enforcement of (adequate) policies are equally important.

The recommendations about not giving users an unfeasible task can be summarised in the following list:

- Identify where the security conditions of the protocol relies on a task performed by users and identify the level of knowledge of the target audience.

- Check whether the task requires specific types of knowledge or skills. If it does require, check whether the target audience attend/possesses the pre-requisites. The more generic the audience, the lower level of understanding and skills should be required.

- Avoid using user input as a main part of the establishment of security properties of the protocol. If user input is unavoidable, check if user training and/or enforcement policies, to guarantee the quality of the input, are necessary.

- Focus on reducing the chance of mistakes, lapses, and slips. Always minimise the number of steps necessary to complete the task [35].

Perceiving the knowledge level of the target users is very important to avoid security flaws due to an overlooked target audience. Also, the more generic the public, the lower level of understanding should be required. It is a design flaw to assume that the users are knowledgeable in something they are not. Thus, when designing a human-protocol, giving tasks to users where they might have insufficient knowledge to perform that task should always be avoided.

Even educated users may fail when performing a security task if their capabilities (e.g. physical skills) are not properly considered [35]. For example, despite the importance of creating strong passwords, a policy that requires a password length of 12 random characters may not be feasible even for trained users due to their memory limitations. The same applies for policies that limit the number of password attempts before blocking a user's account. Despite being an important security feature to prevent password-guessing attacks, such a feature is likely to increase administrative costs, user's mental load, as well as decrease productivity and reduce compliance with security rules, as shown by Brostoff and Sasse [26]. In their work, they demonstrated that by simply adjusting the policy to a more feasible alternative (i.e. 10 attempts before blocking the account), all these negative effects are significantly reduced.

3.5.2 Do Not Rely on User Authentication Capabilities

People are very good at recognising people they already know, but they are not good when authenticating strangers or objects [99, 93, 52]. Thus, except for particular cases, such as human-human interaction between people that know each other, we cannot rely on a user's authentication capabilities and consequently should not include this task in the human-protocol interaction.

In the same example presented earlier during the SSL/TLS protocol handshake, when an unknown server certificate is presented to the client's browser, users are asked whether they trust that certificate or not. By being asked that question, users are receiving an authentication task. However, asking users to authenticate an object (a digital certificate in this case) is not recommended because humans are not capable of authenticating digital objects properly, and therefore, this authentication process becomes insecure. In this specific cause, user's knowledge is also not properly considered, since this authentication task requires a high level of knowledge.

In the university's webmail example, an attacker could easily create a spoofed copy of the webmail's website, maintaining the same visual attributes. To detect the spoofed webmail service, users would have to notice the difference in the URL (which might also be spoofed by a DNS poisoning attack, being the same as the original in this case) or deny the spoofed server's certificate when presented by the browser.

As we discussed in Section 3.3, users can be easily fooled by deceptive text, and more importantly, users are not good at authenticating objects, especially if the authentication task requires technical knowledge. In our example, authenticating a digital certificate is a task that includes these two problems. It requires an authentication of digital objects (digital certificates and security indicators), and also, knowledge about how to differentiate a real from a spoofed certificate. Thus, this task cannot be given to users, it is necessary to find alternative ways to obtain the required result.

The designer, to avoid security failures due to authentication mistakes, should:

- identify where the security conditions of a protocol relies on an authentication task performed by humans.
- check whether the authentication task includes authenticating unknown people or objects.
- verify if the authentication task given to the user is feasible for a ordinary verifier, not requiring specific technical knowledge [99].

There have been some attempts to provide more feasible authentication methods for users. Dhamija and Tygar [40, 39] presented an idea of dynamic security skins, which allows a remote web server to prove its identity to a user in a verifiable manner, and at the same time is difficult to spoof. The main idea is to provide users with a way of authenticating a server without requiring additional technical knowledge. They only need to compare two images, one that they already know and have already associated to their authentication task, and another which is presented by the server. Since the image is customised (and previously defined) for each user, it is easy for the user to detect whether the server is spoofed or not. This is an example of where we can rely on users authenticating objects, and at the same time, have a good level of security since humans are good at recognising and remembering images [100, 38]. A similar approach has been proposed by Gajek et al. [49, 50] focusing on the SSL/TLS protocol, again using images to provide feasible server authentication for users.

3.5.3 Integrate Security Into the Main Workflow

The idea that the user's protection goals must match the security mechanisms implemented, has been discussed for a long time. Saltzer and Schroeder [86], for example, state that it is essential that the human interaction with a protocol is designed for ease of use. In this way, users will routinely and automatically apply the protection mechanisms correctly. If the translation of the security goals to the actual interaction is not clear, the chance of inaccurate human interactions is likely to increase.

When security is a secondary activity for the user, it tends to be ignored or underrated. Warnings, messages and prompts asking users whether to accept a certain change in the security context are likely to be ignored by users, compromising the protocol security [88, 111]. Moreover, most current implementations are plug-ins or amendments to existing designs which are attempts to overcome inherited design problems (such as the dozens of browsers toolbars to provide additional security). Security concerns about human-protocol interaction should be part of the design and included into the main path of the protocol's flow.

The following recommendations summarise considerations to be used during the protocol design:

- If a decision is critical to the security of the protocol, integrate the security concerns into the critical path of their tasks. By doing it, users will be forced to interact with it, and will not be able to ignore it. Additionally, asking users to switch to a safe mode other then just reminding them has been found to be more effective [111].

- Use active interruption other than passive warnings. This change can produce more effective results. However, it is necessary to consider the usability impact of the new design. An excessive use of warnings or employment of unnecessary prompts and interruptions reduce the usability and consequently reduce the attention given to the them over time [111].

- Incorporating security decisions into the users' workflow, and, whenever possible, infer authorisation from acts that are already part of their primary task [113] (e.g. when a user types the URL in the web browser, we can assume that if there is an SSL connection, server certificates issued to a different domains should be rejected).

- Respect user intentions. Warning users that something is wrong and advising them not to proceed (but still giving them the option to continue) is not the right approach. They will accept the risks if they believe it is worthwhile. Warnings that propose an alternative path allowing users to finish the task safely would probably be more effective [111].

In the SSL/TLS protocol implementation, we could apply these recommendations by changing the message presented to the user regarding the untrusted server certificate. Although the protocol's specification does not explicitly consider the user-computer interaction, the SSL/TLS protocol implementation in web browsers has to include the communication between these nodes. Currently, a message from the browser to the user is sent via an active warning (a window asking whether the user wants to accept the certificate). Despite some recent changes in the implementation of these warnings (which made them more effective [45]) we still believe that once users learn how to dismiss these warnings, the efficiency of this type of warning is likely to be reduced. Thus, a third warning type might be needed. We call "interactive warning" a new type of warning that instead of informing or interrupting users, it makes the user interact with the protocol.

If we analyse the webmail example, the user will always receive the warning and then decide whether to proceed afterwards. If an attack occurs, the passive warning will be ineffective, especially because users are already conditioned to dismiss these warnings. In this case, a change to an interactive interruption could be a replacement of the warning message with a question to the user containing an input for the web address confirmation. The users would be required to type the web address they want to have access. After typing the URL, the system would check if the "Common Name" field in the server's certificate matches the address typed by the user, and, based on that, decide whether to continue or not. By making this change, attacks that exploits users' bounded attention, or even users under influence of external factors, would be less effective. Additionally, the effects of deceptive URLs and also, the limited authentication problems would be reduced in this example. This solution needs further analysis, including user testing. However, the idea behind it, is to remove the security decision (presented as a passive interruption) from the user's responsibility and replace it by a request for a piece of information (using an interactive interruption) that allows the system to infer the appropriate security decision. By doing that, the security will be integrated in to the main path of the protocol's flow. Finally we will be converting a complex activity into a task that an ordinary user can perform (making the task feasible).

By doing that, we should be able to reduce the problems related to the lack of attention to security, as well as the problems with the lack of attention to the absence security, since users will be more likely to detect the lack of a feature that was part of their main task (e.g. inserting a card in a ATM and not having to type the PIN).

3.5.4 Consider that the Expected Behaviour Might Change Under Different Circumstances

Human behaviour is likely to change under different circumstances (as we have seen in Section 3.3). The ability to influence the users' decision making, includes several factors that might influence users' behaviour. Factors such as social conditioning, user's principles, time constraints, shared risk and fear are efficiently exploited by attackers. It is necessary to avoid situations where a user interaction might be made under influenceable conditions. The recommendations for protocol designers are:

- check whether external and internal changes might influence user decisions.
- avoid asking users for decisions when they might be under influence.
- support the users' goal in making a predictable choice, based on the goal of their task.

An example of an implementation that does not provide proper mechanisms to deal with user's change of behaviour is the web browser implementation of SSL/TLS protocol. In this implementation, the dialog that asks the users whether they want to accept an unknown server certificate does not include an effective protection to user's change of decision strategies. If the users are under time pressure, for example, they are more likely to dismiss warnings and informative messages, making them more vulnerable. In this case, the server's certificate authentication message, is more likely to be ignored.

Returning to the webmail example, if a student has just a few minutes to check his email before going to a class, his decision strategy would potentially change considerably. All the warnings and messages would be dismissed as fast as possible to allow him to check his email. A spoofed webmail service would be quickly (and carelessly) accepted as real by the student. The certificate verification, or even the URL would be ignored due to the time pressure. To avoid this situation, the warnings should require further checks, such as the domain name confirmation presented in the Section 3.5.3. By implementing those changes, the student would be forced to 'authenticate' the URL, and by doing that, avoiding some attacks. An attack exploiting time pressure, in this case, would be less effective.

We should also take into consideration situations where a user generates input that affects the system's security. A change of decision strategy, in this case, might also impact the security properties of the system. Using the Kerberos protocol example, if the user has to generate a password under time pressure, the quality of input is likely to be lower than in a 'normal' situation. Additionally, it is likely to increase the chance of re-using passwords.

It is important to identify potential situations where this component might be exploited and make the system insensitive to them. Alternatives should be created to avoid unreasonable decisions by users. These possibilities may include extra checks, additional verifications, or a reduction in the impact that the user interaction has on the security properties of the protocol.

3.5.5 Design Should Prevent User From Performing an Inappropriate Interaction

The system should prevent the user from performing an inappropriate interaction. Norman [74] introduced the concept of a "forcing function", which aims to prevent a user from behaving in any other way than the correct way. Basically, the forcing function prevents users from progressing with their task until they perform an action which must be taken to avoid a failure. Additionally, the forcing function will only enable the safe options when an action is being performed.

Forcing functions prevent errors where a user skips an important step and condition users to progress with the correct (safe) behaviour. To be effective, efforts (cognitive and physical) required to follow the forcing function must be less than the effort required to circumvent it [59]. Thus, protocol designers should:

- attempt to provide only safe options to users, and avoid giving unnecessary (and unsafe) options when not needed.
- avoid drastic changes in the usability due to use of forcing functions. If the impacts are too high, users will try to find ways to avoid the 'safe paths'.

Following the previous examples, in the web browsers' implementation of SSL/TLS protocol, the way that the decision of accepting a certificate is implemented does not protect users from making an inappropriate decision.

In the case of a spoofed website presenting a certificate, the invalid option (accepting the fake certificate) is still available. On the other hand, predicting users' intentions is, for obvious reasons, unfeasible. However, it is possible to 'ask' users for their intentions and then check if the actions match the intentions (Brustolini and Salomon implemented such mechanism in [28]) before proceeding. The domain name confirmation presented in the Section 3.5.3 is an example of a forcing function in this case. The user would only be able to proceed if the certificate presented by the server matches with the server the user wants to have access.

Another example of preventing the user from making an inappropriate interaction can be seen in cash machines (ATMs). When the user is withdrawing cash, for example, the sequence of operations performed prevents the user from forgetting his card in the machine. In this case, the cash is not given to the user until he removes the card from the machine. That is, the only way for the user to achieve its main goal (withdrawing money) is by clearing a security step, which is removing the card from the machine.

As we can see, forcing functions are very useful. However, we must take care with the usability impacts and trade-offs, otherwise users will attempt to find ways of dismissing this feature whenever possible.

3.6 Associations Between Interaction Weaknesses and Design Recommendations

The associations among the weaknesses and the design recommendations show us that, to minimise the problems of human-protocol interaction, we sometimes have to consider multiple safeguards to prevent a single weakness from being exploited.

Figure 3.7 represents how the set of weaknesses can be mapped in to the design recommendations. As we can see, to reduce problems related to users' knowledge in a protocol implementation, we have to consider not only that user faculties must be respected, but also make use of mechanisms to prevent inappropriate interactions from being performed. The same recommendations should be applied to deal with problems related to human inherent limitations. Despite being different types of weaknesses, users' lack of knowledge and human limitations present similarities when we attempt to minimise the threats that arise from them.

Decision making influencing factors are linked to three recommendations, which reinforces our impression that this is a complex weakness to handle. For this weakness, it is necessary to make use of techniques to prevent inappropriate decisions from being performed by users, to integrate security into protocols' implementation main workflow, and also, to maintain the predictability of the workflow, that is, minimise the impacts of the users' changes of behaviour.

To deal with users' limitations regarding authentication, we first have to consider that we cannot rely on the accuracy level of an authentication task performed by an user. However, when this task is unavoidable, it should respect the capabilities of ordinary users.

Users' bounded attention deals with users' scant regard for security tasks. The main reason for the lack of attention to security is because security activities are usually secondary concerns in users' activities. The recommendation connected to this issue represents advice to change the focus of security tasks, and bring them to the main workflow. Consequently, users' focus will also include security as a component of the main activity.

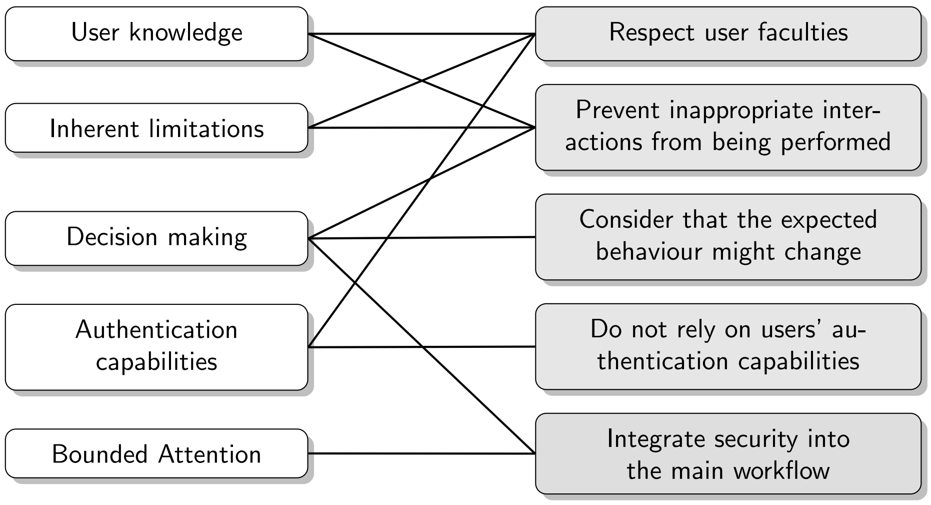

User knowledge
Inherent limitations
Decision making
Authentication capabilities
Bounded Attention
Respect user faculties
Prevent inappropriate interactions from being performed
Consider that the expected behaviour might change
Do not rely on users' authentication capabilities
Integrate security into the main workflow

Figure 3.7: Mapping overlooked components into design recommendations

It is crucial to consider the importance of not relying on user authentication capabilities for objects and unknown people. Directly or indirectly, most components presented in our taxonomy are related to this factor. Most attacks exploit the false acceptance of a spoofed content by users.

3.7 Summary

In this chapter, we have shown that there are many factors that should be taken into account when considering human-protocol interaction. At the same time, there is a wealth of research involving human behaviour analysis and detecting human characteristics that might be exploited by attackers in specific contexts, such as phishing scams and authentication systems. However, despite the existence of similar findings, there is a lack of harmonisation regarding the definitions of human characteristics and weaknesses. We have not found a broad and general taxonomy of human-protocol interaction weaknesses. Therefore, we proposed our own unified set of human weaknesses that merges different research findings into a well defined set of weaknesses.

From this set of weaknesses, we built a set of recommendations to assist designers in the complex task of minimising security threats from user interaction. The recommendations are based on our findings, recommendations found in related works, empirical analysis, and extrapolation from the set of weaknesses presented earlier.

A better understanding of users' characteristics and behaviour should lead to more reliable protocol design and reduce the number of threats found in the human-protocol field. In this chapter, we highlighted those characteristics and proposed ways to deal with them. From a ceremony perspective, the taxonomy and design recommendations we presented are even more relevant. In a ceremony, the human interaction is a more explicit part of the design. The lessons learnt from analysing human interaction in existing implementations of security protocols are of great use to provide guidelines for designing and analysing the interaction that will be performed between users and other agents in a ceremony. A user interaction in a ceremony should be designed to comply with the recommendations we presented and take into account the components of the taxonomy we presented.

Chapter 4

A Framework for Designing and Analysing Ceremonies

Contents

In this chapter we present the foundations of a framework for designing and analysing security ceremonies. We describe the communication channels and agent types involved and their respective characteristics. We also present a threat model tailored for security ceremonies. Part of the content of this chapter appears in two papers: "A Proposed Framework for Analysing Security Ceremonies" by the author, Jean Everson Martina, Geraint Price and Ricardo Felipe Custódio published in the Proceedings of the 7th International Conference on Security and Cryptography [32]; and "An Updated Threat Model for Security Ceremonies", by the same authors, published in the Proceedings of the 28th Annual ACM Symposium on Applied Computing [31].

4.1 Introduction

Security ceremonies, as well as security protocols can be seen as sequences of steps that its peers must take to establish a secure communication amongst themselves. In each step, a message is transmitted between peers. Each message can be sent in clear (not encrypted) or encrypted. The contents of these messages may vary according to the type of channel used. For a traditional protocol channel (network) the contents may include (but not limited to) names, random numbers, ciphertexts and cryptographic keys. For other channels, such as those used for expressing human-computer or human-human communications, message contents can be user input via keyboards, touch screens or speech. Additionally, a message may be formed by more than one component, for example, in a single message we may have the concatenation of a random number, an identity and a key. At its completion, a ceremony as well as a protocol, attempts to achieve certain goals, usually composed of a set of security properties. A security ceremony is considered flawed if it fails to provide its claimed goals.

Protocols have been designed and analysed for a long time and several methods have been developed to prove protocols' claims. Since Needham and Schroeder [71] introduced the idea of using encryption to achieve authenticated communication in computer networks, we have seen a lot of research in the security protocols area. Particularly in developing formal methods and logics to check and verify protocols' claims.

We must cite Burrows et al. [29] for giving a formal representation to describe the beliefs (and its derivations) of the parties involved in the protocol during its execution; Bellare and Rogaway [17] for the provable security, that allows for probabilistic study of the confidentiality goals; Lowe [62], Meadows [68], Schneider [89] and Ryan [85] for works on state enumeration and model checking; Abadi [1, 2] for extending π-calculus [70] for the description and analysis of cryptographic protocols and thefore creating the spi-calculus; Paulson and Bella [78, 12] for their inductive method to verifying protocols, that allows proving the existence of security properties over an inductively defined set of traces (communication events). We have also seen the creation of a number of tools to verify and check security protocols automatically such as ProVerif [19] and Scyther [37]. These techniques and tools have evolved in such a way that nowadays we can check and analyse complex and extensive protocols.

Meadows [69] and Bella et al. [16] in their area survey gave us a broad coverage of the maturity in this field of protocol verification. They also point to trends followed by methods, pinpointing their strong and weak features. They give propositions for research ranging from open-ended protocols, composability and new threat models; something that has changed very little since Dolev and Yao's proposal [43]. These problems seem very well covered. Current research is, in general, aimed at optimising the actual methods in speed and coverage. An important issue regarding these works is that they do not discuss the extension of protocol verification and description to encompass better assumptions as a valid way of extending research in this area. Therefore, extending protocol verification and description to include fine-grained assumptions and derivations is a new and unexplored research path.

Although Ellison [47] proposes the possibility of using formal methods for security protocol analysis, no major work is found today in the ceremony formal-analysis field. Most existing research on human-protocol interaction is focused on the human aspects, based on empirical analysis, which is very important at a design level, but can be difficult and error-prone during analysis, as the history of protocol analysis shows us.

An advance in the reasoning about ceremonies was introduced by Rukšėnas et al. [83, 84]. They developed a human error cognitive model, initially applied to interaction on interfaces. They show that, normally, security leaks come from mistakes made when modelling interfaces, not taking into account the cognitive processes and expectations of human beings behind the computer screen. They successfully verify problems on an authentication interface and a cash-point interface. They showed that the normal lack of consideration in the human peers cognitive processes is one of the weakest factors in these systems. Their proposal comes with a powerful implementation using a model-checker.

Our approach is different, we do not focus on a specifically difficult to describe limitation of human beings, but on giving to the protocol and ceremony designers a better way to define human actions in a ceremony. By making the assumptions more explicit, and requiring a description of the ceremony's security, we can enable designers to experiment with different ceremony techniques. By stating fine-grained assumptions and analysing their absence, we can get insights in to potential break points for security ceremonies. The extension we propose aims to be richer in details than what we currently have, and compatible with established protocol analysis techniques we currently have.

To try to achieve this complex task of verifying security ceremonies we need to first understand what we cover with our framework, then we discuss the communication channels and agents involved and finally the threats such channels are subject to.

4.2 Overview

In traditional protocol specifications we have one communication channel. When moving to ceremonies we add additional agent types into the specification. Consequently, there is a need for new communication channels. By adding human agents, we need to create channels to represent human-computer interaction (e.g. user interfaces) and another to symbolise human-human interaction (e.g. speech). Figure 4.1 gives an overview of the agents and the types of communication channels involved. The area bounded by the dotted line represents the traditional protocol overview, while the complete figure represents the point of view from a ceremony perspective.

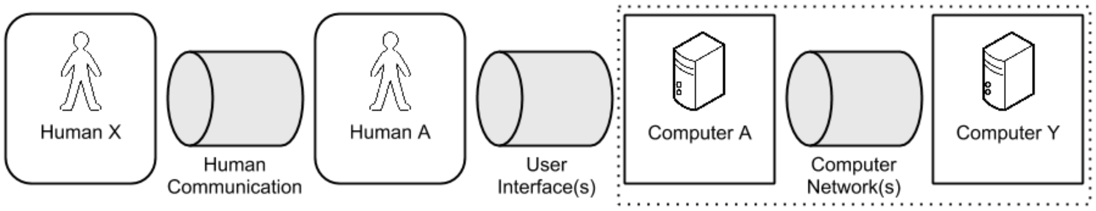


Figure 4.1: Ceremony communication overview

As we can see, in a protocol design and specification, the human-protocol and human-human interaction are assumed to happen out-of-band, and therefore become part of the design assumptions. All the actions outside the dotted area are not included as messages in a protocol specification. As we will see in Section 4.3, in our framework, the human agents and the additional communication channels are explicitly included and the messages exchanged are now part of the ceremony specification.

The idea of having different channel types is not necessarily new. In fact, especially in ad-hoc networks and ubiquitous computing, we have seen some relevant ideas on using special channels that are used by devices and humans to solve existing issues (e.g. authenticating devices that do not pre-share a secret). One of the first references to the idea of using additional channels was presented by Stajano and Anderson where they use secret data exchanged over a contact-triggered channel to initiate an authentication and key exchange protocol [96, 95, 97]. Once the secret data is exchanged, this data is used for subsequent authentication of the parties on regular network communication channel. This approach already makes some assumptions on the channels' properties. First, it requires that the channel is location-limited, that is, it is physically restricted to a certain space (e.g. a room). Second, is that the channel is authenticated, meaning that the attacker cannot transmit data on the channel. Finally, the channel also requires confidentiality, therefore the attacker cannot eavesdrop this restricted channel.

Balfanz et al. [11] further developed Stajano and Anderson's idea by removing the requirement for the additional channel to be secret. They do that by making use of public key cryptography. The agents use the location-limited channel for exchanging their public keys. In this case, the attacker can eavesdrop the channel and still will not be able to do anything with the information obtained. The participants authenticate each other over the network channel by proving possession of their corresponding private keys. Since the attacker does not have access to those private keys, he cannot impersonate any of the participants. Creese et al. [36] also describe the use of different channels where the attacker has a limited set of capabilities. They describe the use of a regular network channel with the addition of a low-bandwidth channel, which can either be uni or bi-directional. An interesting contribution of their work is that they consider human-interaction with the protocol. Although they do not explicitly define a human agent in the protocol, they consider that the user might need to input some information during the protocol run using the low-bandwidth channel.

Hoepman [54] also comes up with the idea of using a low bandwidth communication channel over which two agents can exchange a limited amount of information. This extra channel is either authenticated, meaning that (although an attacker can eavesdrop) the receiver can be sure that a message he received was by the sender he expects; or private, meaning that only the receiver will be able to read the message (however the sender is not authenticated). In other words, the attacker is capable of eavesdropping on the authentic channel (but not insert or modify messages) or insert and modify messages on the private channel. Vaudenay [106] describes commitment schemes and makes use of authentication channels with stronger authenticity properties, such as stall-free transmission, transmission with acknowledgment by the receiver and listener-ready transmission (in this last one the sender can check if the receiver is currently listening to the authenticated channel). Such properties are relevant for us because they are all present in face-to-face (human-human) conversations. In phone conversations, only the last two properties are achieved. Other less interactive communications, such as voice mail messages do not provide these properties [106]. Ĉagalj and Ĉapkun [30] also make use of humans when tackling the problem of key agreement over a radio link. They use the human's ability of authenticating each other by visual and verbal interaction. They base their proposal on three different techniques: on visual comparison of short strings; on distance bounding (where the distance between the devices is displayed to the user); and on integrity codes. They do not explicitly include the human nodes in the protocol specification, but they do include some human-performed tasks into the design assumptions (such as the comparisons of strings).

Wong and Stajano [110] propose protocols which make use of the multichannel approach and present a practical implementation for such protocols. By making use of a visual channel, users possessing camera phones would take pictures of short nonces embedded in QR Codes. With information shared using a radio channel (susceptible to an attacker who can intercept, stop, modify, and insert messages at will) and making use of a low-capacity channel that provides data-origin authentication, they describe a protocol for mutual authentication that does not assume confidentiality and resists eavesdropping on the auxiliary channel. They also propose another protocol that resists a more powerful attacker by assuming a unidirectional visual channel and a one-bit-per-message data-origin authenticated channel. In fact, this variation of the protocol makes use of two low-capacity channels. One from the receiver to the sender (who sends/displays an image) and another one from the sender to the receiver, which consists of a button that is pressed (in this case, the button would be physically part of the receiver, but pressed by the sender).

Kainda et al. [58] performed a comparative usability study of the application of similar methods to those presented above. They argue that most of these proposals fail to take into account factors that may seriously harm the security and usability of a protocol. They performed a usability study of pairing methods and outlined recommendations for designing user interfaces that minimise human mistakes. They also discuss security failures that such methods might be subject to, such as the users not performing a string verification properly, or choosing an incorrect option when checking an authentication string, image or sound. In summary, they show that the methods of comparing and typing short strings into devices are still preferable despite claims that new methods (such as using QR codes) are more usable and secure. Finally, they state that the interface design alone is not sufficient for mitigating human mistakes on the additional channels.

Our approach focuses on a more generic context than those presented. Most existing work focuses on a specific problem (e.g. authentication is most cases). We, on the other hand, focus on building the basis for tackling a wider range of different problems. With the additional channels we propose, along with the new agent types and a dynamic threat model, we will be able to approach a larger set of problems from a ceremony point of view. In Section 4.3, we describe the channels and agents we use. Next, in Section 4.4, we present a threat model that can be adapted to reflect several different real world scenarios.

4.3 Communication Channels and Agents

One of the main characteristics that differentiates a ceremony from a protocol is the additional number of channel and agent types. By extending protocol analysis to ceremony analysis, we can potentially find and solve security flaws that were previously not detectable. However, the design and verification of ceremonies requires the definition of additional communication channels, nodes, and consequently new threat models.

With the addition of humans to the specification, we also have to consider human skills and capabilities. They are different from computers in a number of ways. Therefore, we need a new agent type which we call *human*. By adding such an agent, the communication between this agent and other agents in the ceremony must also be described. Thus, new communication channels must also be included to represent the exchange of messages between the *human* amongst themselves and with the already existing *device* agent (also known as *computer*). In Figure 4.2, we have two new communication channels called *human-device* and *human-human* channel. The first represents the communication between the human agents and devices. The second denotes the communication between humans. The area bounded by the dotted line represents the classic protocol overview, where we have the *device* agent type, that represents general devices (e.g. computers, smartphones, etc.) and a *device-device* channel which is the traditional network channel.

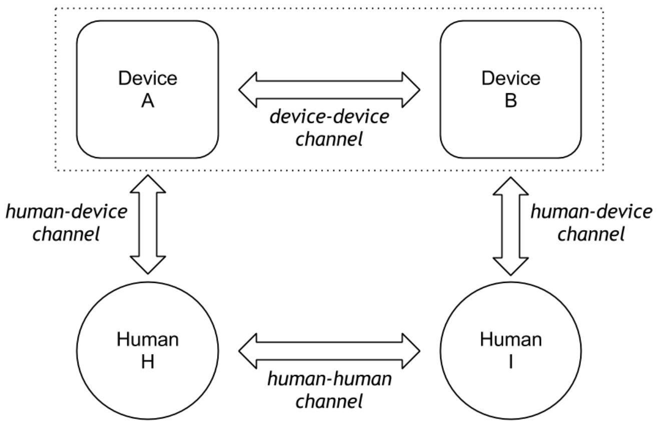


Figure 4.2: Ceremony communication channels and agents

In the same way as *devices*, a *human* agent is capable of sending and receiving messages on a channel, but in this case, such a channel must comply to human capabilities and constraints. This *human* agent should be capable of storing knowledge and sending messages on the mediums it is capable of operating. The agent should also be able to use knowledge conversion functions to be able to operate its devices. Humans can also be related to devices they operate or own, and some of the physical constraints existent in the real world can also be present in this relation. For example, for a one-time-password generator token, the relationship between the human and the one-time-password generator device is based on the device's uniqueness.

In addition to the *human* agent, we create two communication channels, one for the interaction between *humans* and *devices* and another for interaction between *humans* themselves. To represent the new human-device interaction, we create a channel called *human-device* channel. This channel represents the message exchange between *humans* and *devices*. Such an interaction is usually performed via a user interface. A *device* usually displays (or emits) and receives messages, whereas the *human* reads (or hears) and inputs messages.

To represent human-human interaction, we define a channel called human-human. This channel represents the message exchange between humans. Such an interaction is usually performed via speech, chats, user interface. This is a very complex channel which includes a large set of possible actions and virtually impossible to translate into a finite set of actions. However, we consider a limited set of basic actions and analyse a ceremony against them. The taxonomy we present in Chapter 3 is important to give us insights in to what we can, and what we cannot, expect from that interaction. The recommendations given in the same chapter can provide clues to how to avoid incorrect and unsafe expectations from the human interaction. Merging this with a simplistic, but formal definition of the human-human and human-device communication, we can achieve interesting findings as we will see in Chapter 5.

The addition of humans to the specification brings interesting properties to the ceremony. Human communication can be performed in different ways, as for example, in person, via phone or even via some type of recorded material. Each one of them providing different properties. The four main properties we found (which corroborates with some ideas proposed by Vaudenay [106]) are:

Authentication – The channel involving humans is usually authenticated, meaning that the recipient of a message is able to be sure of who sent the message.

Stall-free communication – Human communication can not be stalled, that is, from the time an authenticated message is released, it is treated by the receiver immediately.

Communication with acknowledgment – In communication in human-related channels the sender is capable of checking whether the receiver has received the message or not.

Listener-ready communication – The sender can check whether the receiver is currently listening to the channel.

It is important to highlight that such properties are not always achievable. We may find scenarios and communication types that may achieve a subset of these properties, or even not achieve any. For example, face-to-face conversations usually achieve all these properties. As a counter-example, as we have seen in Chapter 3, if one of the parties is unknown (e.g. an unfamiliar person), we may not achieve authentication.

There will still exist assurance of the sender in this communication (the recipient can easily spot the sender of the message), but the recipient might not be sure whether the sender is the person (or authority) he claims to be (e.g. a real police officer). Telephone conversations usually achieve the last two properties and possibly authentication. If an agent starts communicating with another one, the first is aware that second is listening, and subtle human senses assure her that the latter has heard her message [106]. In addition to that, familiarity with the sender may allow the receiver to authenticate the sender by their voice. Other communication types, such as voice mail and recordings, do not provide these properties. There is no immediate reception, no confirmation of reception and no insurance that the message was recorded in the first place. There still may be some form of authentication. If there is familiarity with the sender, there might be authentication of the sender by their voice.

After discussing the human agent and its channels and messages, we have to discuss knowledge distribution. In a classic protocol framework, knowledge distribution deals with the contents of the message flow. Whatever goes through the network media, the sender may learn the messages and also break the messages into their subcomponents and learn them. The attacker may also learn (depending on the threat model) the contents. With the additional channels and agents, we keep the knowledge distribution the same. Whatever flows on the human-device and human-human channels is susceptible to be learned by the attacker, again depending on the threat model.

With the inclusion of new agents and channels, another point that needs to be covered is the need for a tailored threat model. We need a model that encompasses active threats, as we have in protocols, as well as passive threats. The classic threat model for security protocols, as we will show, is not realistic for our human-device and human-human channels. The presence of an omnipotent and omnipresent being in human interactions is not always realistic in practice and may imply some unnecessary and complex solutions to prevent attacks from such a powerful entity. Therefore, before starting analysing ceremonies, we need to review the possible threats we have on each channel we propose. The existence of a single worst-case scenario threat model is justifiable in security protocols. However, the same cannot be said for security ceremonies. Human agents executing security ceremonies are constrained by the laws of physics and usual capacities expected from human beings. The existence of such a powerful agent in a setting involving human-human communication is not plausible and is likely to demand solutions that are not tailored to reality.

4.4 A Threat Model for Security Ceremonies

A realistic analysis of security protocols must account for a realistic threat model, and the protocols' goals must hold against it [12]. Security protocols are generally secure against passive attackers who eavesdrop the communication medium. However, since Needham and Schroeder introduced the notion of an active attacker [71] a lot of research has been conducted in this area in order to prove protocols' security against active attackers. Needham and Schroeder's attacker model assumed that the attacker could alter, copy, replay and create messages (or parts of messages) in all communication paths. Dolev and Yao [43] further developed this attacker model by formalising it and adding new assumptions. In general, we can say that the Dolev-Yao threat model defines the most hostile environment for protocols, and that the attacker has complete control of the network but is not able to perform cryptanalysis [12].

Currently, the Dolev-Yao threat model is the most widely accepted model to analyse security protocols [12]. Consequently, there are several security protocols considered secure against Dolev-Yao's assumptions. In general, we assume that if a protocol is secure against such a powerful attacker, it is secure against less powerful variations. When we move to ceremonies, however, a realistic threat model may not be Dolev-Yao anymore since its focus is only on a networked environment. In ceremonies we have new communication channels, new nodes, and consequently possible new attacker variations. For that reason, an appropriate threat model must be designed to fit into this new architecture. We argue that, even though Dolev-Yao's threat model can represent the most powerful attacker in a ceremony, the attacker in this model is not realistic in certain scenarios.

As we discussed earlier, one of the reasons that certain protocols fail when implemented is because their assumptions are either not well specified or not realistic, forcing implementations to create mechanisms to circumvent these problems. Consequently, these workarounds may introduce security problems, making the implementation of the protocol, in certain contexts, flawed. In this case, despite the fact that the problem was created during the implementation, the flaw was caused by an inaccurate assumption at design level. Therefore, we must revisit Dolev-Yao's threat model so we can have a tailored threat model for ceremonies, allowing more aligned design and implementation components for ceremonies.

The definition of a threat model for security ceremonies is not a straightforward process. In the same way that a ceremony allows a more detailed analysis of a protocol, the threats, or the capabilities of an attacker under a ceremony scope requires finer granularity in their description. Even though the assumptions made by Needham and Schroeder [71] and extended by Dolev and Yao [43] are the current standard for protocol analysis, for ceremonies they are not always consistent with real world threats. For example, an attacker capable of modifying (or replaying) a 'speech' packet in a human-human medium is unrealistic if this communication happens in person.

By specifying and verifying security ceremonies we will be able to encompass a more human-centric security view. The definition of a realistic threat model for ceremonies will help us to design ceremonies which will assist the human peers to assess the threat level they are subject to. By not overstating assumptions we inherently make them plausible and achievable.

4.4.1 Abstract Threat-Models for Protocols

Security protocols were initially designed to be safe in the presence of a passive attacker. That is, an attacker able to eavesdrop the communication channels and try to make use of its contents [43]. The idea of the existence of an active attacker, also called intruder, saboteur or spy, was first mentioned in the classic Needham-Schroeder paper [71]. They assumed that an attacker can intervene between parties in all communication channels. Such an attacker is capable of altering, copying, replaying and creating messages. Needham and Schroeder claim that, although it is a very pessimistic scenario, the only way an authentication protocol can be considered safe is if it resists against such a powerful attacker. In the following years, this definition became widely accepted and applied when discussing whether a protocol is correct or not. In summary, assuming that the machines involved in the protocol are safe and that cryptography cannot be broken by brute force or cryptanalysis, Needham and Schroeder assume that an attacker can:

- obtain any message that goes through the communication channel;
- modify messages and its subcomponents;
- copy messages and its subcomponents;
- replay messages;
- create messages.

In the following years, Dolev and Yao [43] formalised Needham-Schroeder's attacker and described the attacker capabilities in more details, in order to allow protocols to be analysed more precisely, with less assumptions on the attacker's behaviour. They added the following assumptions to the attacker capabilities:

- The attacker is a valid agent of the network, and therefore can initiate communication with any other agent.
- The attacker can prevent the receiver from receiving a message.
- The attacker, despite the fact that he cannot read the content of an encrypted message (to which he does not know the key), can forward its contents to another agent.
- The attacker can perform any operation over a message except cryptanalysis. For example, he can encrypt and decrypt messages using a key he knows.

The threat model we present above, known as Dolev-Yao, is a de facto standard for symbolic protocol analysis at present [12]. In summary, in this model, the attacker controls the communication channel. From the Dolev-Yao threat model, we have seen the development of two lines of research. The first considers that the Dolev-Yao model should be extended because such an attacker cannot perform cryptanalysis. They take a probabilistic-reasoning based approached, which has been described by Bellare and Rogaway [17]. This research thread has evolved during the years in parallel with the Dolev-Yao research line [8], although there are a few efforts in reconciling them [3, 10]. The second follows the idea that if a protocol is secure against a Dolev-Yao attacker, it is secure against a less powerful variation. Furthermore, the latter argues that subtleties of protocol attacks can still be found even after a protocol is proved correct against Dolev-Yao. By adjusting the powers of the attacker to adhere to the real world and using the symbolic approach, such subtleties may be discovered [8].

Our main focus is on the second line of research. A first example of a threat model that consider variations of Dolev-Yao's attacker capabilities is *BUG* [14]. In *BUG*, the agents are no longer classified as attacker/non-attacker. In this model, the agents of the protocol are partitioned in three groups: *Bad, Ugly and Good. Bad* agents may (or may not) collude with each other in an attempt to break the protocol for their own (illegal) benefits. *Ugly* agents have an intermediate type of behaviour. They may follow the protocol or may deliberately not, letting the *Bad* principals exploit them. *Good* are principals who follow the protocol and its rules. The *BUG* threat model is interesting because of its novelty in having attackers that may not share their knowledge and may change their behaviour during the protocol run, depending on the risks of retaliation.

A direct derivation from *BUG* is the *Rational Attacker* [13]. The rational attackers model drops the triple distinction of the agents' types (due to its complexity) and simplifies *BUG* by considering that any principal makes cost/benefit decisions at any time whether to behave according to the protocol. The attacker must decide if the gain outweighs the risks of being caught. The Rational Attacker was followed up by the *General Attacker* [8]. In this model, the cost/benefit function is removed, but still, any agent may behave as a Dolev-Yao attacker. In fact each peer is a potential attacker that can use the information lawfully acquired to subvert the protocol. This threat model is more realistic than the BUG and the Rational Attacker in an Internet scenario and has the benefit of not having to deal with the gain/loss function.

Finally, we have a refinement to the *General Attacker* model, called *Multi-Attacker* [9]. In this model, each principal may behave as a Dolev-Yao attacker but will never reveal his long-term secrets to other agents. The *Multi-Attacker* adds some rationality to the *General Attacker* in a way that it avoids trivial impersonation attacks.

The *BUG* family of threat models can be seen as a relevant attempt to represent protocols' execution environments in a more accurate manner. On the human-centric security and ubiquitous computing area we see works from Stajano [96], Balfanz [11] and Creese et al. [36] as interesting starting points where we can see variations of the Dolev-Yao attacker applied to the analysis of human-interactive security protocols. As mentioned in Section 4.2, their models include two different communication channels to encompass different threats and environments. In general, they consider a traditional network channel which is susceptible to a Dolev-Yao attacker and a restricted channel where a weakened version of the attacker is used to represent the threat model involving human agents and devices. These examples are relevant because they introduce the idea that communication channels involving human peers and human communication need to be analysed against realistic attacker actions.

In our work, we will discuss how to re-arrange the attacker capabilities to analyse security ceremonies under a realistic threat model. Although initially simple, such a threat model hides subtleties that can validate (or invalidate) claims regarding the achievement of security goals. Attacks can be seen as the marriage between weak goal achievements and the misunderstanding of the correct threat model and therefore, the definition of a realistic and accurate threat model is extremely important.

4.4.2 Premises for Ceremony Threat Modelling

It seems obvious that developing a ceremony that is secure against a Dolev-Yao attacker will imply that the same ceremony will be secure against any weaker real-world attacker. However, it is often the case that, to guarantee that a certain ceremony is secure against a such powerful attacker, we have to include very complex mechanisms which may degrade usability. By doing that, a new threat is introduced, which is the fact that the user will try to circumvent the security mechanisms in order to accomplish his/her tasks. If we consider a more realistic threat model, where the attacker might not be as powerful as the Dolev-Yao attacker, but such an attacker is powerful enough to encompass realistic threats, we can prevent the user from being overloaded, and consequently make the ceremony more usable and secure.

Since the focus of ceremonies is not only on the network channel, we need to define a set of premises for threat models that involve human agents. One important premise for a reasonable threat model is that **no being is omnipotent in human-human channels**. This premise is easily verifiable by humans. The detection of powers beyond usual human capability is straightforward in the setting of security ceremonies. The impact of such a premise is that, depending on the situation, the presence of an active attacker is not realistic. For example, in a network channel, it is realistic to consider that the attacker may block and replay messages, as defined by Dolev-Yao. However, in a human-human channel, an attacker with such power is not reasonable. Replaying or blocking 'speech' in human communication certainly involves the use of powers that are not feasible for a human peer.

Following a similar aspect, we have a premise that **omnipotency in the human-device channel is not always realistic**. Although we have scenarios where we expect that an attacker has full control over the human-device channel (therefore a Dolev-Yao attacker), in some situations such a powerful attacker is not consistent with the real threats. For example, if we assume that the operating system (OS) might be corrupted, we should then analyse the human-device channel against a Dolev-Yao attacker. However, if we assume the OS is safe, we may consider that the attacker does not have all capabilities that the Dolev-Yao attacker has. Another example is when a ceremony makes use of single-purpose devices (e.g. one-time-password generators). When these devices are used, the capabilities of the attacker over the human-device channel is very limited. Thus, the threat model used on such a channel is ceremony and context-specific.

The next premise we have is that **a threat model including human peers should be constrained by the laws of physics**. In several cases, it is unrealistic to assume an omnipresent attacker in human-human channels. In public locations, we may need to consider that an attacker (or several attackers working together) is present during the human-human and human-device communication. However, if we consider that some ceremonies are executed in a suitable location that takes into account the verifiable presence of a potential attacker, we have to respect these physical restrictions. For instance, there are ceremonies that run in a physical context where human peers have strict physical access control. A real world example of such a premise is the execution of security ceremonies for PKIs in safe rooms. These rooms have very strict physical and electromagnetic controls that prevent attackers from being physically present in the environment.

Another important premise for a security ceremony threat model is that **humans are capable of performing basic information recall or mathematical operations**. As we discussed in Chapter 3, some security protocols and their related security ceremonies are designed to encompass unrealistic human capabilities regarding the recall of information or the execution of mathematical operations. In a realistic threat model, human peers are required to recall just fresh information and to execute basic mathematical operations. This premise impacts on how the personification of the attacker in the human-human channel behaves. Without support from a device, a human peer has limited memory and limited mathematical capability. The presence of external aids is detectable and can be used to verify an expected behaviour. An example of such a premise is the verification of possession of a device in an authentication scenario to generate one-time-passwords.

Finally, we have the premise that **one should never use more crypto than needed**. Using more crypto than it is necessary often impacts on usability problems or may introduce inaccurate assumptions from a human-device interaction perspective. Although this is not a ceremony specific problem, encryption should be used only when it is necessary and for clear purposes. Just because a protocol uses encryption, for example, it does not mean it is secure [7]. Additionally, the inclusion of extra layers of crypto, that do not address the threat model, may induce the human who is taking part in the ceremony to misunderstand the threat level he is subject to. An example of such an extra layer not addressing the threat model is the use of one-time-password devices by banks. The extra layer of crypto may induce users into believing that an attacker could not access his account without the device. However, the use of this device does not address the active man-in-the-middle attacks [90], although establishes a strong device possession premise.

With the premises above in mind we propose a threat model that encompass the characteristics of each specific channel. For every channel, we analyse the threat model looking at Dolev-Yao's premises, but we dynamically add and remove capabilities in order to define an attacker model that matches real-world threats.

4.4.3 An Adaptive Threat Model for Ceremonies

A proposition for a new threat model for security ceremonies is justified because no protocol is executed without context. It is known that even if a protocol is proven secure against a powerful attacker (e.g. Dolev-Yao), it might still fail due to some reasons, which may include:

- Usability problems – despite the fact that user interaction is usually part of a protocol's assumptions (and not an explicit part of the specification), in some cases, when these assumptions are implemented, they may require an unrealistic set of user capabilities to achieve the expected goals. Therefore, a user may not be able to perform his tasks correctly and/or not even able to execute them at all.

- The assumptions are too big/strong or generic – it is often necessary to assume that previous steps were successfully performed, or that the user is capable of performing some kind of operation. However, in some scenarios, converting an assumption into an implementation that achieves the same (expected) security properties might be extremely challenging.

While those reasons are not directly related to the network channel, and therefore one could state that the protocol achieves its security goals, we cannot make the same statements when the protocol is implemented. When put in practice, assumptions that involve human-device and human-human interaction have to be implemented somehow. These assumptions must be replaced by dynamic user-interactions. By doing that, we introduce two new possible communication channels, as we have shown in Figure 4.2. In this case, we cannot be certain that the expected security properties assumed in the protocol design will hold in the ceremony.

Another important issue regarding threat models for security ceremonies is that humans make different decisions, regarding their security, based on a dynamic evaluation of the environmental threat level that they are subject to [77]. An example of such an embodied decision making strategy is the evolutionary pressure humans suffered when deciding the trade-offs between whether to engage in attacks and become hunters or to keep a way of life as gatherers, and thus being exposed to less risk [5]. This inherent faculty of human nature is usually not taken into account when we always assume the worst case scenario as in a Dolev-Yao setting. Some attacks may be thwarted by using a very pessimistic threat model, but inherently this action may provoke human nature into acting and finding an easier and plausible solution if the user does not see the alternative path as risky.

With the above in mind we stress that for a security ceremony, the threat model must be adaptive. Even the same protocol might need to run under distinct threat models and achieve its goals in different but still reasonable ways. Considering the worst case is not always the best option since it can degrade usability. The adaptive model we propose applies mostly for the human-device and human-human channels. For network communication (device-device channel), in the majority of cases, we will assume a Dolev-Yao attacker since it is the de facto standard. This is important since it is very well studied and developed.

In addition, having different threat models for different environments can potentially 'teach' users to be more aware of threats and to better (more intuitively) understand the threat model for each circumstance. For example, a user pairing two Bluetooth devices at home is under a different threat to when in an airport. The same may happen for an ATM cash withdrawal ceremony in America and Europe, or even when the ceremony takes place in the same location but at different times of the day. All of these scenarios are subject to different threats.

Based on that, the most challenging step in designing and analysing ceremonies is to define the threats and the conditions where the ceremony will be used. A threat model for ceremonies must be ceremony and context-dependent. However, we can define a limited set of threats that encompasses the great majority of those cases. The existence of a standardised threat model scenario is paramount to the establishment of security goals of ceremonies and for the comparison between the efficacy of different ceremonies.

Therefore, our proposal for a threat model for ceremonies is based on Dolev-Yao's attacker set of capabilities. We enlist and describe each capability and we dynamically add and remove them from the threat model we define. For example, we can have a threat model where the attacker may have the *eavesdrop* and *initiate* capabilities only to enable the fulfilment of our premises stated in Section 4.4.2. Our final goal is to measure the security of ceremonies against a realistic attacker, whose capabilities may be a subset of Dolev-Yao's attacker capabilities. This approach will also help us to reuse some of the abstract verification techniques and tools already in use for security protocols.

To describe our threat model approach, we first enlist a set of attacker's capabilities based on the Dolev-Yao attacker. By using this set and defining a threat model, we will be able to design and verify ceremonies that are secure against a realistic attacker with different capabilities under different channels (e.g. Dolev-Yao attacker on the network channel, while an attacker with *eavesdrop*, *initiate* and *block* on the human-device channel, and finally a passive attacker, who only *eavesdrops*, on the human-human channel).

To describe the powers of the attacker on a specific channel, we will use a simple notation. We start with no threat model, or simply, a model where the attacker has 'no capabilities'. From that point on we add to the attacker the desired capabilities, such as E for *eavesdrop* only, or EB for *eavesdrop* and *block* only. Another way of approaching the notation is by presenting a weakened version of Dolev-Yao's attacker model instead of using a composition of capabilities. We would consider that "DY" is a Dolev-Yao attacker, and everything followed by a "–" symbol will represent the capabilities removed from this attacker. For example, a "DY-BR" means a Dolev-Yao attacker without the *blocking* and *replaying* capabilities. We would like to stress that this is more a denotational problem and using both strategies we will always achieve an equivalence. However, before making use of the secondary notation, we would need a formal and precise definition of Dolev-Yao's set of capabilities, which we will not include here. We will therefore use only the first type of notation in this thesis.

Using the same initial assumptions of Dolev and Yao about the attacker, that is, that the attacker is a valid agent of the network and that he cannot perform cryptanalysis, we list below the attacker's set of capabilities which we include in our adaptive threat model. All the capabilities we list are based on Dolev-Yao's attacker and their informal definitions are shown below:

Eavesdrop (E) – Represents the capability of the attacker to passively listen to the communication media. That means that the attacker will learn the contents of any message M that is sent through the communication channel. Even if the message is not intended for the attacker, he is able to capture and learn M.

Initiate (I) – Another important capability of an attacker is the capacity to initiate a communication with another peer using the knowledge the attacker possesses. The definition of initiate is that the attacker can use any information he knows (from his initial knowledge or learnt during the ceremony run) to initiate a communication with an agent A by sending message M, the contents of which are part of, or derived from, the attacker's knowledge.

Block (B) – Is the attacker's capability to block messages, that is, preventing the receiver from learning the contents of a message sent to them. Its definition is that when A sends a message M to an agent B, the content of the message will not be learnt by B.

Atomic Break Down (A) – Defines the capability that allows the attacker to break down messages in to its subcomponents. This capability is relevant so that the attacker can use atomic components of previously learned messages to produce new ones. The definition for Break Down is that for all pairs composed by some X and Y elements in the knowledge of the attacker I, the element X and Y are also in the knowledge of the attacker I individually.

Derive (D) – An important capability for the attacker is the usage of publicly known functions to produce new messages. Examples of such functions can be cryptographic hashes, encryption and decryption, or any other function publicly available to the execution of the ceremony. Derive may be an n-ary function. The capability of deriving messages can be described as for all messages X that are part of the set of knowledge of the attacker I, the result of the application of a publicly available function F is also part of the set of messages known by the attacker I.

Some of the well known attacker's capabilities (based on Dolev-Yao threat model) are not directly shown here, since they can be achieved by the combination of our definitions. For example, the capability of **Modifying (M)** messages on the communication channels could be defined as the use of **Block** + **Initiate**, while **Replaying (R)** messages can be represented as **Eavesdrop** + **Initiate**. It is important to remember that the list we presented is necessarily incomplete, simply because new attacks and/or attacking tactics can be discovered or exist under different scenarios that we initially do not cover. Nevertheless, our dynamic and adaptive threat model inherently allows additions to the set of attacker's capabilities.

The adaptive threat model we propose here can be applied on all the communication channels in a ceremony. Although we recommend that the device-device channel should be verified in the majority of cases against a Dolev-Yao attacker, it is important to always consider whether the Dolev-Yao attacker is realistic for every channel, even for device-device communication. In Chapter 5 we will present and discuss some real-world examples and the threat model for each channel in more details.

4.4.4 Case Study: Bluetooth Pairing Protocol

To demonstrate how the threat model for security ceremonies we proposed may be used in the ceremony analysis, we will use the Bluetooth pairing protocol as a case study. Considering the Bluetooth Legacy Pairing (described in Section 2.4.1.1) and Bluetooth Simple Secure Pairing (SSP) (described in Section 2.4.1.2) as examples, when we thoroughly analyse the protocol specifications, we find that the association modes are designed under assumptions that imply a weaker threat model for the pairing protocol. In this Section, we focus on showing the impacts of changing the threat model of the communication channels. We demonstrate that an unrealistic threat model may lead us to find impractical attacks. In Chapter 5, we provide a more thorough analysis of the Bluetooth ceremony, using additional tools, results from our taxonomy and set of recommendations, and additional granularity of the threat model.

In the legacy pairing, the device-device channel (DD) is designed considering a DY attacker, while the human-device (HD) and human-human channels (HH) are assumed to have no attackers. Although there are other flaws in this protocol [57, 51], a relevant, simple and effective passive attack can be found if we add the capability of eavesdropping to the attacker on either HD or HH channels. In this case, the attacker would learn the PIN by just eavesdropping those channels (hearing the PIN value) and with that, he could decode all messages. This works similarly to the attack described in [92], but without the need of deploying a brute force attack on the PIN. This attack would easily be captured when verifying this protocol as a ceremony.

In the case of the SSP protocol, which is the main focus of our case study, each association mode needs to be analysed under a different threat model, and more importantly, each implementation should respect the specified threat model. In our examples, we will use the **numeric comparison** against a Dolev-Yao attacker and also under other variations of the attacker capabilities. The following specification describes phase two of the SSP protocol using the numeric comparison mode specified as a ceremony. A and B represent the devices used and U_A and U_B represent the users of each device respectively. The notation follows the format discussed in Chapter 2, and the functions $f1$ and Va are the same as presented in Section 2.4.1.

$$\begin{array}{lcccccc}
1. & B & \xrightarrow[DD]{} & A & : & C_b = f1(pk_B, pk_A, Nb, 0) \\
2. & A & \xrightarrow[DD]{} & B & : & N_a \\
3. & B & \xrightarrow[DD]{} & A & : & N_b \\
4. & A & \xrightarrow[HD]{} & U_A & : & V_a = g(pk_A, pk_B, N_a, N_b) \\
5. & B & \xrightarrow[HD]{} & U_B & : & V_b = g(pk_A, pk_B, N_a, N_b) \\
6. & U_A & \xrightarrow[HH]{} & U_B & : & V_a \\
7. & U_B & \xrightarrow[HH]{} & U_A & : & V_b
\end{array}$$

Figure 4.3: Protocol description for Bluetooth SSP phase two under the Numeric Comparison mode

In our analysis, we considered the ceremony using the **numeric comparison** (NC) mode under different variations of the threat model. Table 4.1 presents a summary of threat models and association modes we use.

Channel	**NC + DY**	**NC + ATM V1**	**NC + ATM V2**
DD	DY	DY	DY
HD	DY	E	E
HH	DY	DY	E

Table 4.1: Threat models and SSP modes

As we can see, we created three different scenarios. In the first, we consider a Dolev-Yao (DY) attacker for all three channels. In the second, we use an Adaptive Threat

Model (ATM V1) that changes the threat model for the human-device (HD) channel by having an attacker capable of *eavesdrop* only. Finally, in the third (ATM V2), we have a DY attacker only for the device-device (DD) channel. For the HD channel and for the human-human (HH) channel, we have an attacker capable of *eavesdrop* only.

Given the notation presented in Section 2.3.2, $knows(Y)$ representing the set of knowledge of Y, M_n symbolising the message n of the ceremony, DY as the Dolev-Yao attacker, E a threat model where the attacker possesses only the eavesdrop capability, and $\emptyset$ representing that no attacks have been found, we present the theorems below that describe the results of our analysis.

Theorem 1 (Numeric Comparison + Dolev-Yao)**.** *If the ceremony messages M_1 to M_7 are run against a DY attacker, the attacker can prevent U_A from learning V_a or V_b and U_B from learning V_b or V_a, forcing them to learn V_i instead.*

$$\frac{M_{1\ldots7} \cup DY}{\begin{aligned} V_a \wedge V_b \wedge V_i \in knows(I) \wedge \\ V_a \notin knows(U_A) \wedge V_b \notin knows(U_A) \wedge \\ V_b \notin knows(U_B) \wedge V_a \notin knows(U_B) \wedge \\ V_i \in knows(U_A) \wedge V_i \in knows(U_B) \end{aligned}}$$

Proof. Assuming that the attacker I, acting as a man-in-the-middle, initiated two parallel pairing sessions with A and B during Messages M_1 to M_3. The authentication from A to B starts on M_4 where the value V_a is sent to U_A. The equivalent message from B to U_B occurs in M_5. A DY attacker I, by using his block (B) and initiate (I) capabilities, can prevent the message M_4 and M_5 from being delivered to U_A and U_B respectively, and instead, send them any chosen value V_i. In M_6 and M_7, A and B would complete the protocol by sending V_i to each other, successfully concluding the pairing and allowing the man-in-the-middle attack to be deployed. □

Using an alternative threat model, which we term the Adaptive Threat Model V1, we assume the attacker can only eavesdrop the HD channel. In the specific case of Bluetooth pairing, the assumption is that the device is free from malware and the display is presenting the correct information.

Theorem 2 (Numeric Comparison + Adaptive Threat Model V1)**.** *If the protocol messages M_1 to M_3 are run against a DY attacker; the messages M_4 to M_5 are run against an E attacker; and messages M_6 to M_7 are run against a DY attacker, the attacker can prevent U_A from learning V_b and U_B from learning V_a, forcing them to* *learn the repetition (replay) of V_a and V_b (respectively) instead.*

$$\frac{(M_{1\ldots3} \cup DY) \wedge (M_{4\ldots5} \cup E) \wedge)(M_{6\ldots7} \cup DY)}{V_a \wedge V_b \in knows(I) \wedge V_a \notin knows(U_B) \wedge V_b \notin knows(U_A)}$$

Proof. Again, assuming that the attacker I, acting as a man-in-the-middle initiated two parallel pairing sessions with A and B during Messages M_1 to M_3. The authentication from A to B starts on M_4 where the value V_a is sent to U_A. The equivalent message from B to U_B occurs in M_5. In this case, the E attacker can only learn the values V_a and V_b. In M_6 and M_7, A and B complete the protocol by sending V_a and V_b respectively, to each other. A DY attacker in messages M_6 and M_7 can perform a similar attack to the one described in Theorem 1. By preventing V_a from being delivered from U_A to U_B in M_6 and V_b from U_B to U_A in M_7, and then replaying the values V_b and V_a (respectively) instead, the protocol run would be successfully finished and would allow a man-in-the-middle attack to be deployed. □

Finally, when using another variation of the threat model, which we call the Adaptive Threat Model V2, the attacker can eavesdrop both the HD and the HH channels, that is, the attacker can eavesdrop on any communications in the pairing process that involve the humans.

Theorem 3 (Numeric Comparison + Adaptive Threat Model V2)**.** *If the protocol messages M_1 to M_3 are run against a DY attacker and the messages M_4 to M_7 are run against an E attacker, the attacker cannot produce any relevant attack.*

$$\frac{(M_{1\ldots3} \cup DY) \wedge (M_{4\ldots7} \cup N + E)}{\emptyset}$$

Proof. Once again, assuming that the attacker I, acting as a man-in-the-middle, initiated two parallel pairing sessions with A and B during Messages M_1 to M_3. The authentication from A to B starts on M_4 where the value V_a is sent to U_A. The equivalent message from B to U_B occurs in M_5. In this case, the E attacker can only learn the values V_a and V_b. In M_6 and M_7, A and B complete the protocol by sending V_a and V_b respectively, to each other. In messages M_6 and M_7, the E attacker can only learn the values V_a and V_b. Therefore, V_a received by U_B in M_6 and V_b received by U_A in M_7 would not match the V_b and V_a in $knows(B)$ and $knows(A)$ respectively, not allowing the attack to succeed. □

Although the attack described in Theorem 1 is plausible in real world scenarios, it is very difficult to deploy. An attacker would have to corrupt both devices as well as start parallel sessions with both users during a short period of time. This is a good example of a technically feasible attack but highly unlikely to happen in practice. By removing capabilities B and I of the attacker, we can analyse the protocol further, and possibly find other (more) relevant attacks. In Section 5.4, we implement variations of this scenario using an automatic cryptographic protocol verifier tool (ProVerif) and discuss each variation in more detail.

The second attack, demonstrated in Theorem 2 is utterly unrealistic. To be deployed in practice, the attacker would have to block a communication between two humans and then replay some data over a channel where the user would easily notice if some other party wanted to spoof the identity of the sender. In this case, the attack does not exist in practice.

In theorem 3, which we found to be free of attacks, will be further discussed in Section 5.4. We will demonstrate that even when a protocol is proven to be free of attacks, inaccurate assumptions on the human components may introduce security problems.

The idea is similar for the other association modes. Each one of them must consider the real-world scenario and define the threat model. In addition to that, it should not be possible to use an association mode under a different threat model to the one specified.

4.5 Summary

Protocols are, by design, implemented to attend to human demands. The method we propose approaches real world concerns on the design level. It is impossible to represent all possible human characteristics in a limited set of operations, but by including the human node in the specification and adjusting the threat model to represent realistic threats to every communication channel we use, we can thoroughly study interactions and factors which were previously included in the set of assumptions for each protocol.

As we can see, a ceremony adds some complexity to the analysis due to the additional agents and channels. On the other hand we do not want to change the way we analyse protocols today, since the formal methods available are mature and powerful for their intended purposes. Our framework approaches the problem from an extended point of view. The extensions we propose allow adding support to ceremony analysis in existing methods and tools (as we will see in Chapter 5). Our objective with this model is to extend the coverage from the verification of security protocols to ceremonies. Human behaviour is indeed unpredictable, but by including humans in the formal models we can, at least, predict what happens if some action is performed by the user in an incorrect manner.

Our approach for describing a threat model for security ceremonies is based on a well established model for security protocols. In our approach we weaken the attacker to conform to the premises governing human-device interaction and human-human interaction. This strategy seems plausible because it will help security protocols and ceremony designers to develop ceremonies with reasonable assumptions and tailored to the real capacities of the attacker.

Chapter 5

Designing and Analysing Ceremonies

Contents

In this chapter we present examples of the design and analysis of security ceremonies. We make use of existing real world protocols that require user interaction and model their ceremony versions. Furthermore, we analyse each ceremony presented and prove its properties by using an automatic cryptographic *protocol verifier (ProVerif). We then discuss our findings and highlight the achievements obtained by analysing these ceremonies. Part of the content of this chapter appears in the paper "An Updated Threat Model for Security Ceremonies", published in the Proceedings of the 28th Annual ACM Symposium on Applied Computing [31].*

5.1 Introduction

The next step to confirm the benefits of designing and analysing security ceremonies and evaluate the results, is to analyse real world situations where protocols that involve human interaction are used. In this chapter we start by describing the ceremony design process we consider ideal. Next, we present three examples of such protocols and we demonstrate how we could analyse them as ceremonies. We develop different scenarios and use different threat models in order to dissect possible outcomes.

First, we design and analyse the Bluetooth Legacy Pairing ceremony. Although legacy pairing is deprecated and has been replaced by Simple Secure Pairing (SSP), the analysis of the Bluetooth Legacy Pairing ceremony is relevant due to the authentication mechanism used, which is based on user input. Such a mechanism and its analysis could then be extended to other ceremonies and therefore it is worth analysing. We describe the threat model for the ceremony and provide both informal and formal analysis of the ceremony.

Our second ceremony is SSP, which replaces Legacy Pairing in recent versions of Bluetooth. In our analysis we focus on the Numeric Comparison (NC) pairing mode, which represents an interesting case study for ceremonies due to the user interaction involved. We again describe the threat model for the ceremony and provide both informal and formal analysis of the ceremony. Furthermore, we propose a variation of the ceremony, which relies on the same protocol with an additional comparison that should be performed on the device after the user interaction. This new ceremony improves the authentication process of SSP under the NC mode by changing a task that the user might not perform adequately to another that makes use of a forcing function, preventing the user from performing the incorrect interaction.

Finally, we analyse a ceremony that involves the registration process of a very popular messaging application called WhatsApp. We present different registration ceremonies and use different registration methods. The threat model for each ceremony is discussed and we provide both informal and formal analysis of the ceremonies.

The formal analysis we perform uses a cryptographic protocol verifier tool called ProVerif [19]. This tool is used for automatically analysing security protocols. ProVerif handles different languages as input types. The most commonly used, and considered to be state-of-the-art is the language known as typed pi calculus [20], which is the language we use for our implementations. Cryptographic primitives, such as symmetric and asymmetric encryption, digital signatures and hash functions are supported and ProVerif is capable of proving reachability properties, correspondence assertions, and observational equivalence. Such capabilities allow the analysis of confidentiality and authentication properties, which are our goals in the examples we provide in this chapter.

By default, ProVerif's communication channels are assumed to be controlled by a Dolev-Yao attacker. It also provides support for private channels, where the attacker has no capabilities. A passive attacker, who is only able to eavesdrop the communication channel is also supported. However, once the attacker is set as passive, the whole threat model is switched to such a threat model. At the moment, there is no direct support for modelling both Dolev-Yao and passive attackers on different channels in the same protocol. In our ceremony framework, we need such a feature, and we can simulate such attacker behaviour by using some implementation tricks. We describe in this chapter how we implemented this feature.

5.2 The Ceremony Design Process

The ceremony design process is more complex than the protocol design process. It involves more agents, more channels, different threats, more variables and more steps. A more thorough ceremony analysis can be achieved by evaluating ceremonies in a loop, as presented in Figure 5.1.

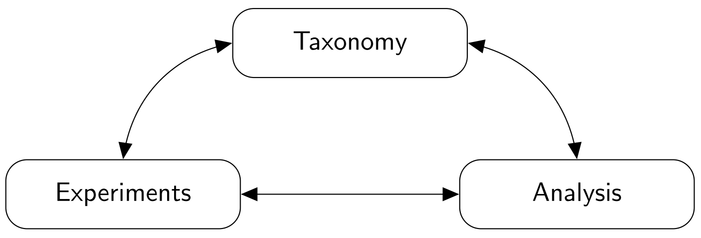


Figure 5.1: The ceremony design process

The analysis component involves the process of formal and/or informal analysis, similar to the work we presented in Chapter 5. Every human interaction must be modelled to respect the taxonomy components we presented in Chapter 3 and follow the design recommendations. User experiments may also be relevant to evaluate whether the expected interaction happens in practice and whether the assumptions regarding the human agent capabilities are accurate. In this thesis, the experiment phase was based on related work that performed experiments similar to those required for the ceremonies we analysed. It is important to highlight that every step in this cycle generates new information that might help to refine the other steps performed. Whenever a failure or a security flaw is found in a phase, the ceremony designer must go back to the design phase, correct it, and perform the analysis in the loop again.

This design process loop provides a systematic approach to identifying different possible failures in different aspects of a ceremony. More importantly, it assists the identification of root causes of different failures. After completing a loop, designers may need to revisit the whole loop to try to further improve the ceremony and reduce the risk of failures. The number of rounds in the loop can be as many as needed to reach a point where no relevant issue can be found.

5.3 Bluetooth Legacy Pairing Ceremony

Although it is well known that this protocol presents security flaws, the Bluetooth Legacy Pairing ceremony is worthy of the analysis because we can develop several insights into what ceremony analysis gives us. By systematically presenting variations of the ceremony model, and the threat model involved, we can reach different results and discover clues as to what are the benefits of this approach.

The first change we note when we move from a protocol description (as seen in Section 2.4.1.1) to Bluetooth's legacy pairing ceremony, is that the definition of the PIN code cannot be ignored, and therefore it is not part of the design assumptions any more. In fact, for each type of method for setting the PIN value, a different ceremony must be specified. For a fixed PIN, we could still assume that the PIN is pre-defined and therefore the ceremony would be similar to the protocol. However, in the variable PIN setting, which is the recommended use according to the specification [22], the PIN is now a relevant part of the specification. The PIN request is now an explicit message sent from the device to the user after the $ACCEPT$ message, as shown in the protocol description presented in Figure 5.2. Also, the request of the PIN and its value being shared between the U_A and U_B are also explicit messages in the specification. Finally, the user entering the PIN in to the device is also presented in the ceremony description.

The remainder of the protocol stays the same. In this ceremony we consider two different users pairing their respective devices. The moment where the PIN is generated by the user does not affect the ceremony. However, the way they share the PIN with the other user is relevant, since an attacker could eavesdrop the PIN value. Note

$$
\begin{array}{rccccc}
1. & A & \xrightarrow[DD]{} & B & : & IN\_RAND \\
2. & B & \xrightarrow[DD]{} & A & : & ACCEPTED \\
3. & A & \xrightarrow[HD]{} & U_A & : & PIN\ Request \\
4. & B & \xrightarrow[HD]{} & U_B & : & PIN\ Request \\
5. & U_A & \xrightarrow[HH]{} & U_B & : & PIN \\
6. & U_A & \xrightarrow[HD]{} & A & : & PIN \\
7. & U_B & \xrightarrow[HD]{} & B & : & PIN \\
8. & A & \xrightarrow[DD]{} & B & : & AU\_RAND_A \\
9. & B & \xrightarrow[DD]{} & A & : & SRES1 = e1(K_{init}, BD\_ADDR_B, AU\_RAND_A) \\
10. & B & \xrightarrow[DD]{} & A & : & AU\_RAND_B \\
11. & A & \xrightarrow[DD]{} & B & : & SRES2 = e1(K_{init}, BD\_ADDR_A, AU\_RAND_B) \\
12. & A & \xrightarrow[DD]{} & B & : & E\_LK\_RAND_A = LK\_RAND_A \oplus K_{init} \\
13. & B & \xrightarrow[DD]{} & A & : & E\_LK\_RAND_B = LK\_RAND_B \oplus K_{init}
\end{array}
$$

Figure 5.2: Ceremony description for the legacy mode pairing

that there is also a ceremony setting where a single user could be pairing two devices they own.

The ceremony we will focus on in this section is a legacy pairing ceremony using a variable PIN that involves two users pairing their devices. As mentioned earlier, the changes from the protocol description to ceremony description are clear in the initialisation steps in legacy pairing. Figure 5.3 shows a sequence diagram of the initialisation phase of the ceremony. In the other parts, authentication and link key calculation are similar for the protocol and ceremony (see Figure 2.5 and 2.6 for more details).

5.3.1 Threat Model

Now that we have defined the ceremony, it is important to define a realistic threat model for its channels. It is reasonable to assume that for the DD channel, we will have a DY attacker since it represents a very powerful (and realistic) attacker for a network channel. The communication channel between the humans and the devices (HD) in this ceremony could possibly be under three different threat models. First, we could use DY and therefore assume that we do not trust what the devices A and B display and receive via input (possibly due to a corrupted application and/or operating system). Second, we could assume an attacker capable of only eavesdropping (E) on the messages, meaning that we have a passive attacker. Third, we could assume

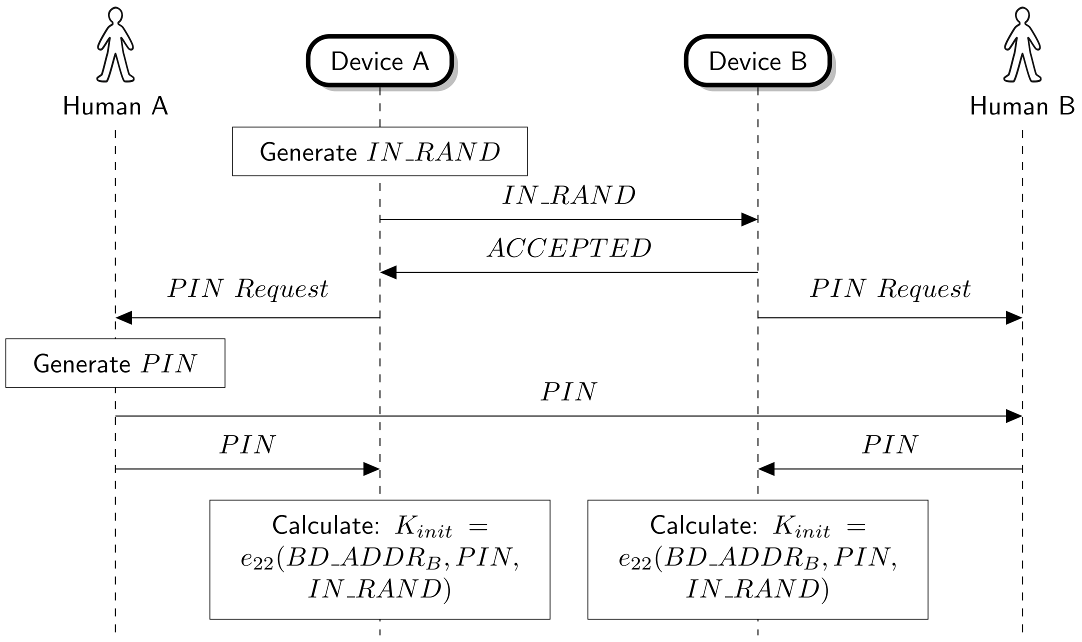


Figure 5.3: Sequence diagram for the legacy mode pairing ceremony (Initialisation)

that there is no attacker on the HD channel. Finally, for the HH channel, a DY attacker is unrealistic since an attacker cannot perform any other operation rather than eavesdropping the channel without being detected in this Bluetooth pairing setting.

As we can see, defining the threat model for the DD and HH channel for this ceremony is straightforward. However, the attacker capabilities on the HD channel requires further analysis. Although possible, a DY attacker on the HD channel in this ceremony is extremely unlikely to exist in real world scenarios. An attacker would have to corrupt both devices in order to manipulate the messages exchanged between the users and their devices to deploy an attack. A threat model that does not consider any attacker capability for the HD channel is also questionable. Despite that, the protocol description for the legacy pairing protocol assumes that there is no attacker on the HD channel. Such a threat model does not seem to be accurate in the case of the ceremony. Since Bluetooth is designed to be used in short range communications between (mobile) devices, the pairing ceremony could be run anywhere, from private places (e.g. home) to public ones (e.g. airport lounges). It is too optimistic to assume that no one else apart from the users involved in the ceremony could see (eavesdrop) the user entering PIN in to the device. Finally, considering that the attacker is capable of eavesdropping on the HD channel seems realistic for the reasons we presented. Over-the-shoulder is a very well known threat and fits well in this scenario. Thus, an eavesdropping attacker is appropriate and realistic for the HD channel.

5.3.2 Informal Analysis

As described in Section 2.4.1.1, it is clear that if an attacker is capable of obtaining the PIN value, the protocol properties of confidentiality and authentication are violated. The same situation applies for the legacy pairing ceremony. Although the specification allows for $PINs$ up to 16-digit long, in practice only 4-digit long $PINs$ are used. This represents a very low entropy input. That is, an attacker is likely to be able to guess the PIN value and verify it offline, as already discussed in [57, 92, 34]. In fact, for 4-digit long $PINs$, Shaked and Wool demonstrated that brute-forcing the PIN could be done in near real time [92].

To analyse the legacy pairing ceremony in more detail, we will divide the threat model in to four possible settings:

i) the ceremony is similar to the most common version of protocol, that is, the PIN is short (e.g. 4-digit long) and we assume that the attacker has no control over the HH and HD channel. In this scenario, the attacker is DY on the DD channel, and has no capabilities on the HD and HH channels.

ii) we consider the PIN as the longest possible (16 digits). Therefore, we assume that it is long enough not to be broken by the attacker in a reasonable time. Again, the attacker is DY in the DD channel, and has no capabilities on the HD and HH channels.

iii) we still have a long PIN (16 digits), but the attacker now has eavesdrop (E) capability on the HD and HH channels.

iv) we reduce the PIN back to 4 digits, and keep E for the attacker on the HD and HH channels.

The ceremony is considered broken if an attacker is able to break the confidentiality of the protocol by either obtaining PIN or deriving K_{init}, or if the attacker is able to violate mutual authentication between the parties involved. For scenario i, we can see that by eavesdropping the DD channel, the attacker cannot learn the PIN, making the offline attack the only viable option due to the low-entropy of the PIN. This attack is similar to the attacks presented in [57, 92, 34] and violates confidentiality for the current session and authentication for future sessions. Confidentiality is violated because the attacker will have all the information needed to calculate K_{init}, then derive the link (K_{AB}) and encryption keys. Authentication is violated, possibly only in future sessions, since pairing happens only for the first time when two devices connect. After that, they use K_{AB} as the base for calculating encryption keys between them. Thus, by knowing K_{AB}, an attacker can impersonate the devices in future sessions. Note that, depending on the length of PIN, it could be broken by the attacker in a very short time (in milliseconds, as demonstrated in [92]) and therefore authentication could be violated even in the first session (during pairing) when short $PINs$ are used (e.g. less than 6 digits).

If we increase the length of PIN, as proposed in scenario ii, breaking or brute-forcing its value would be unreasonable for the attacker. Since the attacker has only access to the DD channel, there is nothing much he could do to break the ceremony. Indeed, this scenario is considerably more secure than the previous one and no relevant attacks can initially be found. However, another problem is evident in this ceremony: usability. Asking users to generate and share 16-digit long random numbers is very unusable and unfeasible as we discussed in Chapter 3, and as some experiments have demonstrated [103, 104, 58]. The result of this setting is two-fold. First, if users decide to proceed and enter the 16-digit long number, there is a high probability that the numbers will not have enough randomness and be predictable, significantly reducing the entropy. In this case, the attacks we discussed for scenario i, are now possible in this scenario. Second, as Uzun et al. [103] and Valkonen et al. [104] demonstrated, entering 4-digit long $PINs$ already require a significant effort from users (taking around 20-25 seconds to be performed). By increasing this number to 16 digits, there is a high probably that the ceremony would be unusable and users would give up on using the mechanism. In this case, the setting would be relatively safe (if users could choose random $PINs$), but useless, since no one would use it in practice.

In scenario iii, we still assume a high-entropy PIN, making it relatively unbreakable by the attacker. However, in this setting, there are at least two additional points where an attacker could learn the PIN value in the ceremony without even having to brute-force it. By eavesdropping the HH or HD channels in messages 5, 6 or 7, he could learn the PIN and therefore calculate K_{init}, K_{AB} and finally the encryption key. This violates both confidentiality and authentication for the current session in a very simple and straightforward way, making the attack even more powerful than the offline attacks presented in scenario i. Note that the attacker would only need to eavesdrop either the HH or HD channel to be able to lean the PIN, which is, in practice, completely feasible since Bluetooth is designed to run on mobile devices, and therefore be used in public places such as airport lounges, restaurants, etc.

Finally, in scenario iv we have the most realistic setting for this ceremony. We merge the case where we have the low-entropy 4-digit PIN, which is the most used setting, with the E capabilities of the attacker. In this scenario, the attacks we found in i and iii are possible.

In the analysis we present we consider the case of two users pairing two devices they own. Another situation that could happen is a setting where one user is pairing two devices they own (e.g. a smart phone and a laptop). In this case, there would be no HH channel in the ceremony. However, the only change in our analysis would be that the attacker would have the eavesdrop capability only on the HD channel. The attacks would still be the same on the scenarios presented.

5.3.3 Formal Analysis

In our formal analysis, we modelled the same scenarios presented in the informal analysis we performed earlier (Section 5.3.2). In ProVerif however, there are some details that are relevant to the scope of our analysis. First, we need to model the secrecy of PIN, in order to represent its variations in low and/or high entropy settings. Second, we need to model the additional channels used in the ceremony. Third, we have to define different threat models for the channels, which proved to be quite tricky. Finally, it is necessary to model the confidentiality property of the protocol.

Modelling the secrecy of the PIN value is subtle. As discussed earlier in the informal analysis and in Section 2.4.1.1, the PIN used in legacy mode is usually a low entropy input and therefore is guessable. ProVerif provides a mechanism for modelling low entropy values, such as human-memorable passwords. It has already been demonstrated and proved that legacy pairing is subject to offline attacks in its protocol version [57, 92, 34]. To model such a characteristic of the PIN in ProVerif, and therefore verify the absence (or otherwise) of offline guessing attacks against the PIN, we make use of the query: **weaksecret** PIN. When this query is used, ProVerif tries to prove that the adversary cannot distinguish a correct guess of the PIN from an incorrect guess [20]. In the scenario where we consider longer $PINs$ and therefore wish to analyse when a PIN provides higher entropy, we can simply remove the **weaksecret** query from the model.

Modelling the additional channels is quite straightforward. It is as simple as modelling a network channel. However, in our implementation, one important difference is that it is semantically important to differentiate between the channels used for the communication between user U_A and device A and user U_B and device B. By using different channels for them, despite being of the same type (HD) we can clearly differentiate the messages between the users and the devices they possess rather than mixing them all on the same channel (which does not happen in practice).

Defining the threat model for each channel is a tricky part of our modelling. By default, ProVerif provides either a Dolev-Yao or a passive attacker for all channels. The only additional differentiation we could have for a channel is between defining it as public or private. In a private channel the attacker has capability on the channel. In a public channel we can have either Dolev-Yao or passive only. Thus, we kept the threat model for all channels (except for private channels) as Dolev-Yao, and declared the channels that we will add eavesdrop capability as private. Then, to allow the attacker to eavesdrop these private channels, we broadcast any communication on these channels over a public channel. With this trick, we are able to simulate the passive-only channels needed along with the Dolev Yao threat model for the other channels. Then, the definition of the channels in ProVerif is the following:

free DD : channel. *(* device-device channel *)*
free $\mathsf{HD_A}$: channel [*private*]. *(* human-device channel A (no attackers) *)*
free $\mathsf{HD_{AE}}$: channel. *(* human-device channel A (eavesdrop) *)*
free $\mathsf{HD_B}$: channel [*private*]. *(* human-device channel (no attackers) *)*
free $\mathsf{HD_{BE}}$: channel. *(* human-device channel B (eavesdrop) *)*
free HH : channel [*private*]. *(* human-human channel B (no attackers) *)*
free $\mathsf{HH_E}$: channel. *(* human-human channel (eavesdrop) *)*

As we can see, we defined a public channel for device-device communication (DD), private channels between the users and their respective devices (HD_A and HD_B) and a human-human private channel (HH). In addition to these channels, we defined the three additional public channels (HD_{AE}, HD_{BE} and HH_E). To broadcast the messages sent on the private channels HD_A, HD_B and HH, to the public channels HD_{AE}, HD_{BE} and HH_E respectively, and therefore simulate the attacker's eavesdrop capability, we add the following to the process model:

(!**in**($\mathsf{HD_A}$, x:bitstring); **out**($\mathsf{HD_{AE}}$, x))
| (!**in**($\mathsf{HD_B}$, x:bitstring); **out**($\mathsf{HD_{BE}}$, x))
| (!**in**(HH, x:bitstring); **out**($\mathsf{HH_E}$, x))

The modelling above duplicates everything on the private channel to the respective public channel enabling a passive attacker on the channel.

To model the secrecy property of the protocol, we then simulated a scenario where A sends a secret to B using the initialisation key K_{init} they established during the pairing process. To do that, we defined secret s, which is sent from A to B at the end of the protocol run, and added the query "**query** *attacker*(s)" to the model, in order to test the secrecy of s. Internally, ProVerif attempts to prove that a state where the secret s is known by the attacker is unreachable. If a state where s is known by the attacker is found, it means that the protocol specified fails to provide the desired secrecy property.

We modelled in ProVerif each scenario presented in Section 5.3.2 using the mechanisms discussed above when needed. The results were similar to those presented in our informal analysis. For scenario i, we tested the secrecy of PIN using the query **weaksecret** PIN. As expected, ProVerif found an attack which is fundamentally the same as the attacks found and described in [57, 92, 34]. In the attack, the attacker eavesdrops a successful pairing session between A and B and uses the values BD_ADDR_B, IN_RAND, AU_RAND_A and $SRES_1$. Figure 5.4 shows how the attacker proceeds after collecting the information needed. The attacker tries an initial value for PIN', then he calculates K'_{init} and $SRES'_1$. He then compares whether $SRES'_1$ matches $SRES_1$. If it does, the value he guessed for PIN' matches PIN, otherwise he tries a new value for PIN, until he finds a matching value.

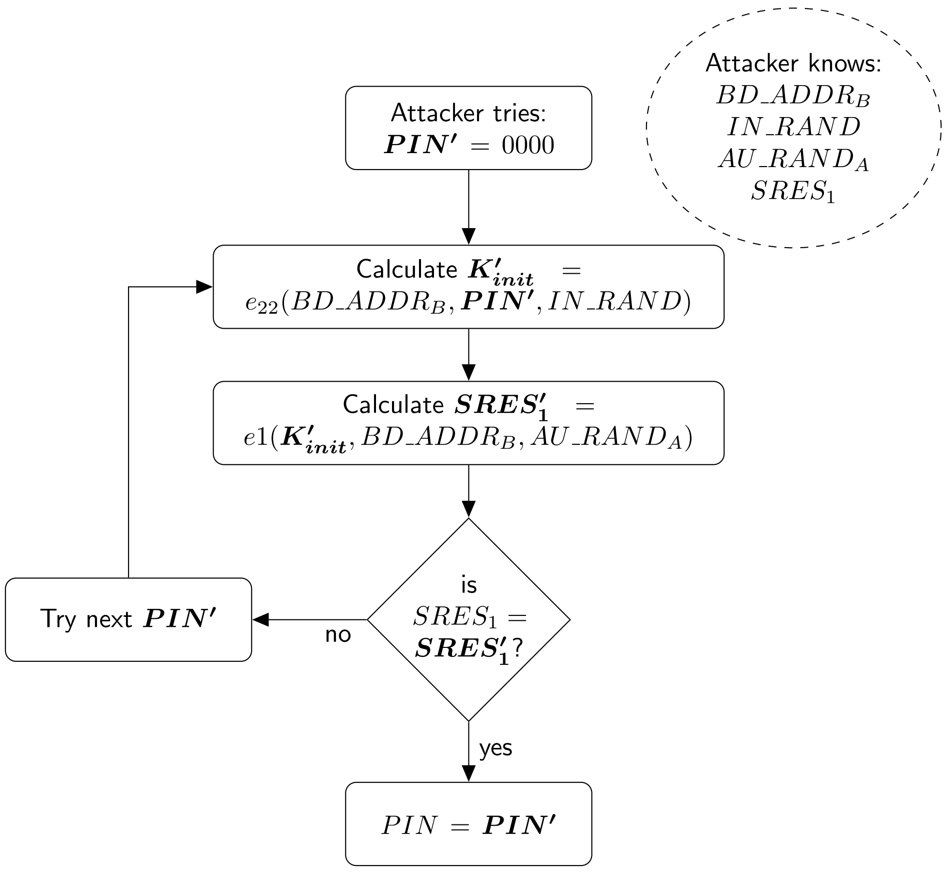

Attacker knows:
BD_ADDR_B
IN_RAND
AU_RAND_A
$SRES_1$
Attacker tries: $\boldsymbol{PIN'} = 0000$
Calculate $\boldsymbol{K'_{init}} = e_{22}(BD\_ADDR_B, \boldsymbol{PIN'}, IN\_RAND)$
Calculate $\boldsymbol{SRES'_1} = e1(\boldsymbol{K'_{init}}, BD\_ADDR_B, AU\_RAND_A)$
is $SRES_1 = \boldsymbol{SRES'_1}$?
no
Try next $\boldsymbol{PIN'}$
yes
$PIN = \boldsymbol{PIN'}$

Figure 5.4: Brute-forcing PIN in the legacy pairing

For scenario ii, we removed the **weaksecret** query from the model and added a query for the secret s, as described earlier. In this case, no attack was found, since PIN was assumed to have high entropy. Note that the analysis we presented in Section 5.3.2 already pointed to the same results. However, it highlights the low probability of a such setting being feasible in practice.

Next, we modelled scenario iii, where the entropy of PIN is still assumed to be high, but the attacker now has the eavesdrop capability on both HD and HH channels. As expected, ProVerif found an attack where the attacker could eavesdrop the PIN value from the HH channel, and then violates the confidentiality of s by being able to obtain K_{init} from the values eavesdropped from the DD channel and the PIN from HH. Although this attack seems very simplistic, the assumption that the attacker will never learn the PIN value by eavesdropping the communication channels between humans and between human and devices is too optimistic. As mentioned earlier, Bluetooth pairing is expected to be used in public places and therefore should take into account such a realistic setting in its design.

Finally, in scenario iv, we analyse the scenario that covers the most (realistic) threat among the variations we presented. As expected, the results demonstrated the existence of attacks on the entropy of PIN as we found by analysing scenario ii, and on the confidentiality of s, as found in scenario iii.

5.4 Bluetooth Simple Secure Pairing Ceremony

As discussed earlier, the Bluetooth Simple Secure Pairing (SSP) ceremony is designed to mitigate the problems found in the legacy pairing protocol. The most relevant phase for the ceremony analysis is the second phase. It is in this phase where mutual authentication is achieved, which is the main goal of this ceremony. Therefore, our focus on the ceremony analysis will be on phase two using the numeric comparison (NC) mode.

The second phase of SSP, described as a ceremony, includes two more agents U_A and U_B and two additional communication channels HD and HH to represent the human-device channel and human-human channel respectively. Figure 5.5 presents the ceremony of phase two using the NC mode.

When compared to the protocol description (presented in Figure 2.10), we have 6 additional steps. These steps start at 4, which represents device A sending the value V_a to user U_A; then 5, which represents device B sending the value V_b to user U_B; 6 which represents user U_A sending the value V_a to user U_B; 7 which represents user U_B sending the value V_b to user U_A; finally 8 and 9 where the users confirm whether the values received and generated match. It is important to note that in our ceremony we assume two users pairing two devices. The main reason for using this assumption is that it allows us to introduce the idea of the HH channel in a realistic scenario. However, there is also the possibility of a single user pairing two devices, which would

$$\begin{array}{llcll}
1. & B & \xrightarrow[DD]{} & A: & C_b = f1(PK_B, PK_A, N_b, 0) \\
2. & A & \xrightarrow[DD]{} & B: & N_a \\
3. & B & \xrightarrow[DD]{} & A: & N_b \\
4. & A & \xrightarrow[HD]{} & U_A: & V_a = g(PK_A, PK_B, N_a, N_b) \\
5. & B & \xrightarrow[HD]{} & U_B: & V_b = g(PK_A, PK_B, N_a, N_b) \\
6. & U_A & \xrightarrow[HH]{} & U_B: & V_a \\
7. & U_B & \xrightarrow[HH]{} & U_A: & V_b \\
8. & U_A & \xrightarrow[HD]{} & A: & Ok \\
9. & U_B & \xrightarrow[HD]{} & B: & Ok
\end{array}$$

Figure 5.5: Ceremony description for the SSP ceremony phase two

remove the need of the HH channel. In this case, we would just assume that messages sent by both devices via the HD channel would be sent directly to the same human who would then compare their values. Figure 5.6 shows the sequence diagram for phase two of the pairing ceremony under the NC mode.

By adding these six additional steps, the pairing ceremony encompasses the whole pairing process and covers all the messages sent and received among the peers in order to achieve mutual authentication. Furthermore, when analysing the specifications, we find that both legacy mode and SSP are designed under assumptions that the attacker possesses a restricted set of capabilities during the pairing process.

5.4.1 Threat Model

A SSP ceremony is considered flawed if an attacker is able to violate the mutual authentication between the devices A and B. Although no explicit mention is made of these additional channels or variations of threat models in the specification, it is clear that the SSP protocol and its association modes are designed under assumptions that imply a weaker threat model. If they assumed a Dolev-Yao attacker, several attacks would be possible (as we briefly presented in Section 4.4.4). However, since there is no HD and HH channels specified, it is not clear which threat models were assumed instead.

When analysing the SSP ceremony we need to define a realistic threat model. A DY attacker is reasonable for the DD channel, however it is not appropriate for the HD and HH channels. If we assume a Dolev-Yao attacker for the entire ceremony, unrealistic attacks will be found. These unrealistic attacks may incur countermeasures that are extremely likely to make the protocol more complex and less usable in practice.

In a similar manner to the threat model we discussed for legacy pairing, the threats

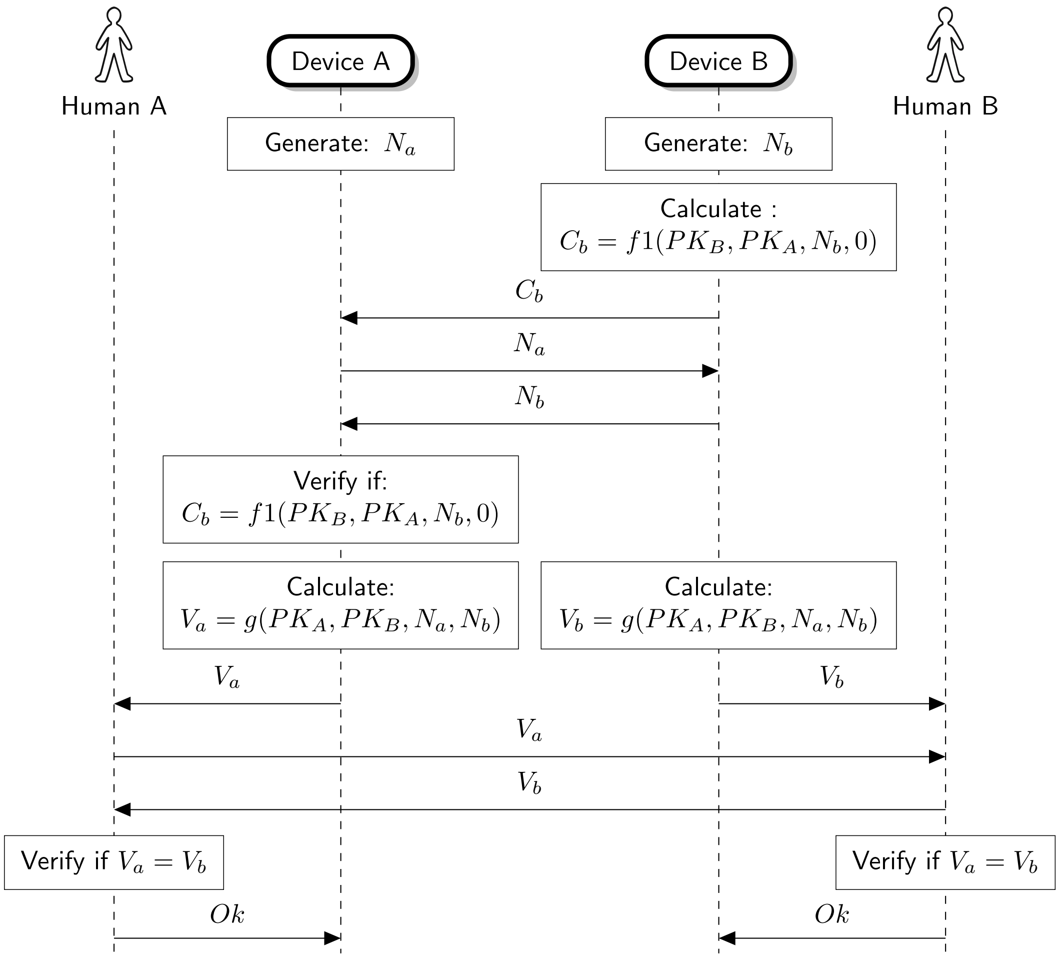

Human A
Device A
Device B
Human B
Generate: N_a
Generate: N_b
Calculate : $C_b = f1(PK_B, PK_A, N_b, 0)$
C_b
N_a
N_b
Verify if: $C_b = f1(PK_B, PK_A, N_b, 0)$
Calculate: $V_a = g(PK_A, PK_B, N_a, N_b)$
Calculate: $V_b = g(PK_A, PK_B, N_a, N_b)$
V_a
V_b
V_a
V_b
Verify if $V_a = V_b$
Verify if $V_a = V_b$
Ok
Ok

Figure 5.6: Sequence diagram for phase two of the ceremony of SSP under the NC mode

that the HD channel is subject to could vary. We could assume DY and therefore not trust what A and B display and receive. Another option would be that the attacker is capable of eavesdropping (E) the messages, or even no attackers on the HD channel. For the HH channel, an attacker capable of eavesdropping is very realistic. The presence of no attackers on the HH channel for this ceremony could only be assumed in very strict scenarios.

In our analysis, the threat model we consider consists in DY for the DD channel and E for both HD and HH channels. Since Bluetooth is designed to be used in both private and public locations for short range communications between (mobile) devices, assuming that no one else, apart from the users involved in the ceremony, could eavesdrop the messages between the users and the users and devices is too optimistic, and would ignore some realistic threats. On the other hand, active attackers on the

HH channel for the SSP ceremony is not realistic either, since both users agree to pair their devices in the first place. Finally, assuming an E only attacker for the HD channel rather than DY is reasonable since the attacker would need to corrupt both devices in order to succeed. Still, it would be interesting to analyse what happens in the case of one device being corrupted, that is, having a DY attacker for only one of the HD channels we have in the ceremony.

5.4.2 Informal Analysis

Although no major attacks currently exist against the protocol, analysing the SSP ceremony is relevant since it brings different scenarios into perspective and new possible threats. We divided our analysis in six different settings, where we systematically change the threat model and user behaviour so that we can simulate several possible scenarios, as follows:

i) In a similar manner to the most common version of the protocol specification, we assume that the attacker has no control over the HH and HD channel. Thus, in this scenario the attacker is DY on the DD channel, and has no capabilities on the HD and HH channels.

ii) In this setting, we add eavesdrop capabilities to the attacker on both HD and HH channels. On the DD channel the attacker is kept as DY.

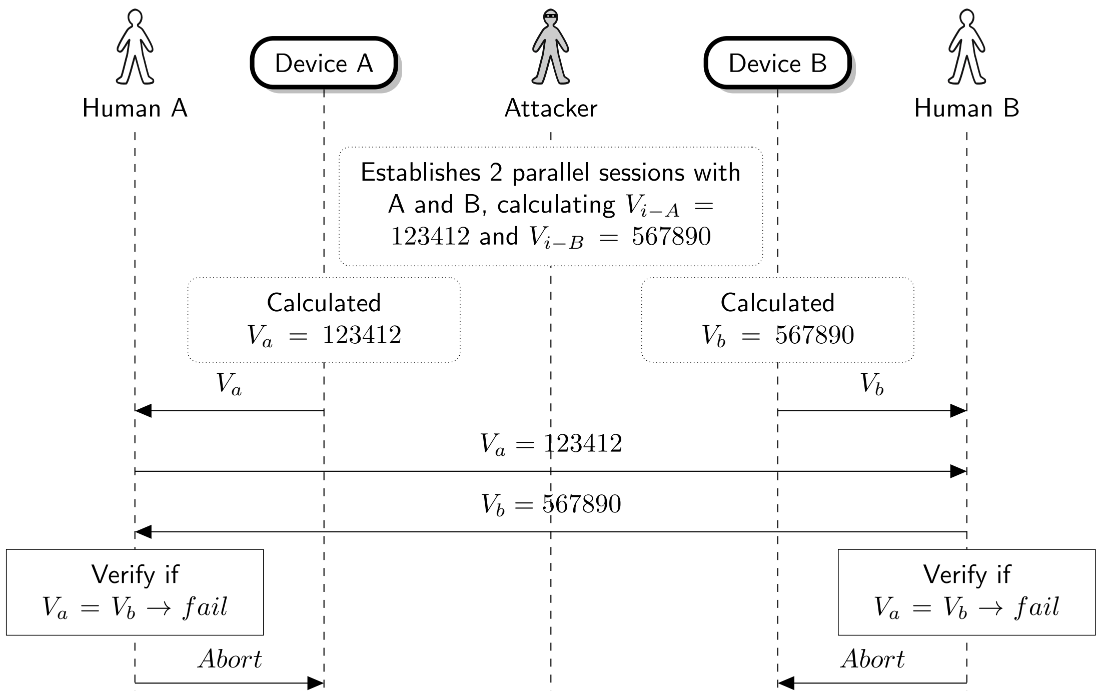
iii) For this variation, we still consider DY for the HD channel and E for both HD and HH channels. The difference now is that we will make an assumption about the behaviour of human U_A. Rather than behaving as expected and comparing the number on his screen to the one he received from the human U_B, he will proceed even if the numbers do not match.

iv) In this case, we assume the opposite of we assumed in iii. Now, U_A behaves as expected, but U_B will proceed in any case.

v) Complementing the previous two settings, we now assume that both U_A and U_B proceed without checking if V_a and V_b match.

vi) Finally, we again change the threat model on the HD channel. In this case we will analyse what happens in the case where one of the devices is corrupted. That is, the HD channel of one user is under a DY attacker. In this case, we will assume that the HD channel between U_A and his device A is under a DY attacker and that both U_A and U_B behave correctly.

Figure 5.7 shows an attacker attempting to deploy a man-in-the-middle (MITM) attack between devices A and B. In the figure, we present the expected execution of the ceremony when this situation occurs. As we can see, what prevents the attacker from succeeding is the human verification of whether V_a and V_b match and if it does not, the human ends the ceremony and aborts the pairing.



Human A
Device A
Attacker
Device B
Human B
Establishes 2 parallel sessions with A and B, calculating $V_{i-A} = 123412$ and $V_{i-B} = 567890$
Calculated $V_a = 123412$
Calculated $V_b = 567890$
V_a
V_b
$V_a = 123412$
$V_b = 567890$
Verify if $V_a = V_b \rightarrow fail$
Verify if $V_a = V_b \rightarrow fail$
$Abort$
$Abort$

Figure 5.7: Sequence diagram for an MITM attempt in the SSP in NC mode

For scenarios i and ii, no attacks happen if the user verification is performed correctly. Even in ii, where the attacker has E, he will not be able to prevent the users from detecting the difference between V_a and V_b. This generates a scenario similar to the one presented in Figure 5.7. Although being simple and easy to use when using 4-digit long numbers, the authentication method used, based on users comparing values and confirming whether they match is susceptible of a high error rate, going up to 20% error rate in some scenarios [103, 104, 58]. Therefore, attacks such as the one presented in Figure 5.7 could succeed in many cases. If the users make a mistake when checking the values, or do not check the values at all, and therefore just proceed confirming the pairing, the MITM attack would be successful, even though the ceremony is designed to prevent such an attack. Furthermore, if consider the taxonomy we presented in Chapter 3, this user-ceremony interaction is likely to be susceptible to user-conditioning, as presented in the bounded attention taxonomy item (Section 3.3.4); lack of knowledge of security threats and inaccurate mental models (Section 3.3.1) from the user knowledge item; and the decision making influencing factors (Section 3.3.3).

Users can easily be conditioned to just click on “confirm” for the pairing process, especially if the pairing is repeated several times by the user. In this case users can predict the outcome, and it may result in them anticipating the matching hash values task and simply pressing “confirm” without properly checking the values [58]. Another factor that may impact on the failure rate of this ceremony is the users’ lack of knowledge of security threats and inaccurate mental models, since by not appreciating the possible threats to the pairing process and not evaluating the importance of authentication processes correctly, the importance of comparing the values displayed are likely to be underrated and consequently poorly performed. Finally, factors that influence decision making are more difficult to simulate in lab experiments such as those in [103, 104, 58]. Factors such as time constraints are likely to negatively influence the user performance when comparing strings, making the error-rate even higher, as mentioned by Kainda et al. [58].

For iii and iv, we consider that just one of the humans involved behaves as expected by the ceremony. In iii, for example, A would be vulnerable to the attacker since A would have concluded the pairing, even though B has not accepted or finished the pairing. This problem could be solved by U_B warning U_A that the numbers did not match, but it could be too late and A would have to abort by closing the application they were using to create the pairing. The same applies for the opposite scenario, iii. In the case where the two devices perform the pairing process prior to any other action (e.g. sharing a file), there would be little or no harm if U_B notifies U_A about the not matching values. However, if the pairing was started in order to immediately share a file between A and B, the file transfer is likely to happen before U_B can warn U_A, causing a more serious security problem. Another issue with this scenario is that at the end, the attacker will have created a link key with the device that mistakenly accepted the pairing, and future connections with this device would not require a new pairing procedure. Thus, to prevent that from happening, A would have to manually delete the link key stablished with the attacker, which is not simple or easily understood by some users.

In scenario v, both U_A and U_B behave inadequately, and therefore a MITM attack is trivial. In this case, an attack would be even harder to detect because the pairing session, and the subsequent actions (e.g. file sharing), would work normally. The attacker would then have link keys established with both devices, making future attacks against the devices trivial.

Finally, in vi, we assume a DY attacker on one of the HD channels. In our example, we assume that we have DY on the HD channel between A and U_A. The attacker, in this case, could simply replace the number sent from A to U_A by the number generated between the attacker and B. In this case, the numbers would match, and even if both U_A and U_B perform the authentication task correctly, the attack would still succeed, since the numbers displayed to the users would match. Figure 5.8 shows the attack. Another variation of an attack on the HD channel would be in case the attacker changes the confirmation message. In this case, even if U_A sends an *Abort* message to A, the attacker would replace *Abort* by *Ok* and therefore A would confirm the pairing. However, this second case, would only affect A's side, since U_B would abort after verifying that the number sent by U_A does not match the one he sees on his display (similar to scenario iii).

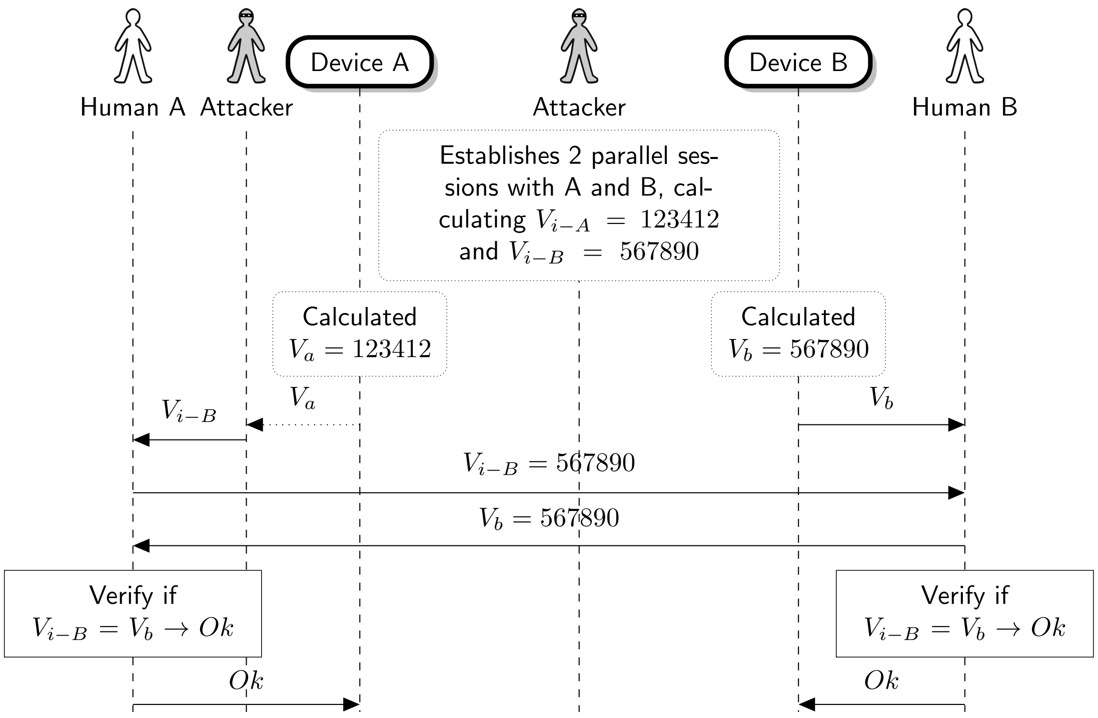

Human A
Attacker
Device A
Attacker
Device B
Human B
Establishes 2 parallel sessions with A and B, calculating $V_{i-A} = 123412$ and $V_{i-B} = 567890$
Calculated $V_a = 123412$
Calculated $V_b = 567890$
V_{i-B}
V_a
V_b
$V_{i-B} = 567890$
$V_b = 567890$
Verify if $V_{i-B} = V_b \rightarrow Ok$
Verify if $V_{i-B} = V_b \rightarrow Ok$
Ok
Ok

Figure 5.8: Sequence diagram for an MITM attack in the SSP in NC mode

In the analysis we presented, we assumed the case where two users pair two devices they each posses. The pairing process may be also run in a setting where one user pairs two devices (e.g. a smart phone and a laptop). In this case, there is no HH channel in the ceremony. In this case, scenarios iii and iv are not likely to exist, being replaced just by v. The other scenarios would still be similar. Note that, in this case, user-conditioning could have more influence and be more likely to happen. The repetitive task of verifying numbers would be performed twice by the same user per pairing session, accelerating the user-conditioning process. Another factor that may happen, but would be difficult to measure, is the possible increased level of confidence when the user pairs two devices they own. In this setting, the user could feel more ‘secure’ and be less careful when checking the numbers on both displays.

5.4.3 Formal Analysis

Our formal model of the Simple Pairing ceremony focuses on phases one and two of the ceremony and implements similar scenarios to those we have seen in the informal analysis we performed. The basic ceremony is the same as we have shown in figure 5.6 with the addition of a public key exchange (phase one) that happens prior to the sequence presented in the figure. We modelled the channels HD and HH using the same approach we discussed in Section 5.3.3, where we have separate channels for the HD for A to U_A and B to U_B. The threat model for these channels in our analysis varies from no attacker (N), to eavesdrop (E) and finally Dolev-Yao DY. The approach for modelling these variations is similar to the one we discussed in Section 5.3.3. Our focus is on modelling authentication, which is the most important and interesting part of this ceremony.

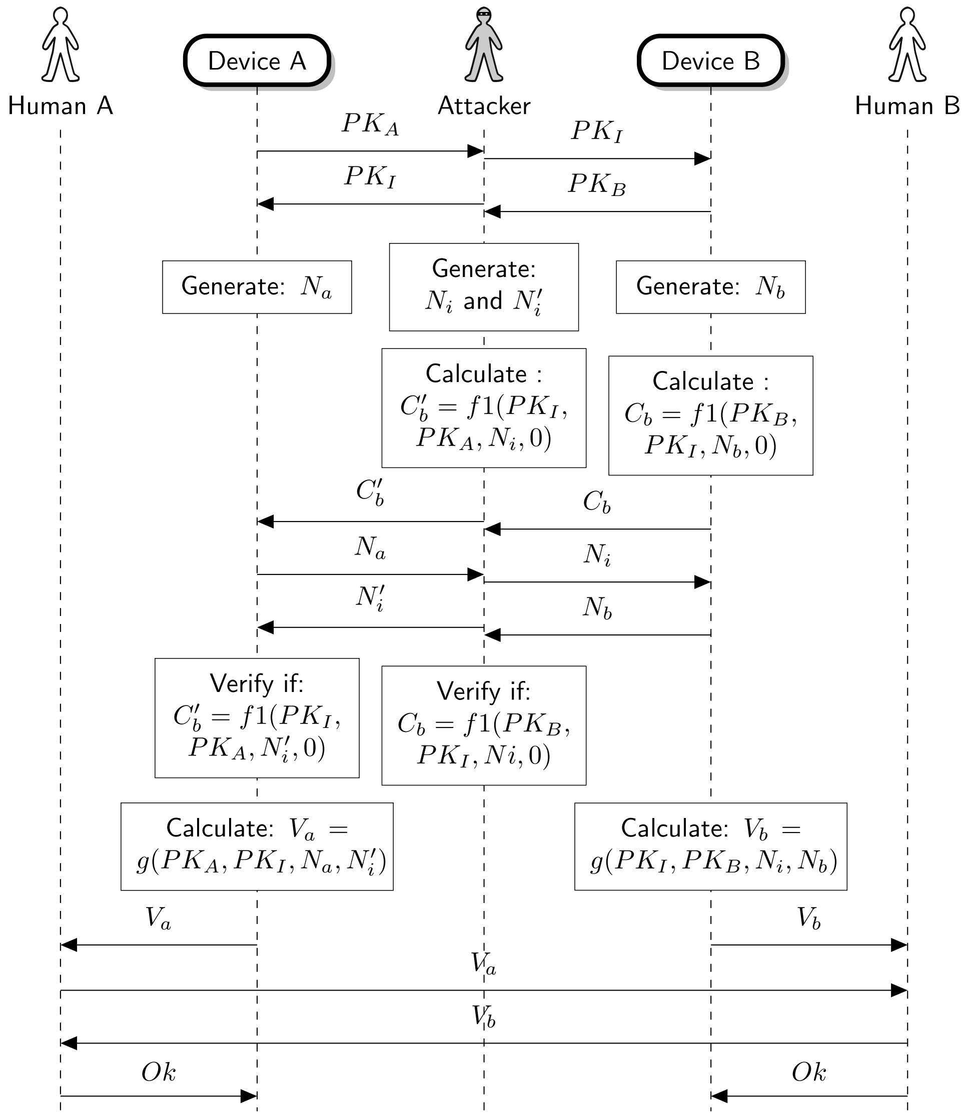
We modelled authentication using correspondence assertions which are used to represent a sequence of events in the form “if an event x has been executed, then event x' has been previously executed” [20]. In ProVerif we use special *begin* and *end* events that are parameterised by protocol terms. To model the authentication of A to B, we modified A’s process to include a *begin* event using B’s parameters as soon as A starts a pairing session with B (after the initial public key exchange). We also modify B’s process to include an *end* event using B’s parameters once B successfully completes a pairing session. Similarly, we model authentication from B to A by making the same modifications, but using A’s parameters and including the *begin* event in B’s process and *end* event in A’s process. Thus, in ProVerif, we modelled the authentication properties in the form: if the event end(X’s parameters) is executed, then the event begin(X’s parameters) should have been previously executed. If a state where the event end(X’s parameters) is executed without a previous begin(X’s parameters) event, the authentication property is violated. This condition, means that an attacker could have started a communication with an agent X and the execution reached an state where the event end(X’s parameters) was executed without the event begin (X’s parameters) being executed before. Such an implementation is similar to the implementation described in [34].

ProVerif supports two types of correspondence assertion types: correspondence (non-injective) and injective correspondence. The difference between them is that the injective correspondence captures a one-to-one relationship between the number of protocol runs performed, while the first does not. For example, injective correspondence is useful to capture a setting where a server should only complete a single transaction per transaction started by the client (e.g. financial transactions). In other words, injective correspondence implies that for every end(params) event, there should be a matching begin(params) event, while for the (non-injective) correspondence, implies that if end(params) is output, then at least one begin(params) event was previously outputted.

In our analysis, we followed a similar authentication implementation that was made by Chang and Shmatikov [34] for the protocol version of SSP. Therefore, we model both injective and non-injective versions of two types of authentication properties. The first type is modelled by using the agents' identities (public keys) as parameters of begin and end events. In this first type, if the authentication property holds, then we can assume that each agent is certain about the identity of the other agent in the ceremony run. However, other parameters, such as nonces and the other agents' identities involved do not necessarily match. Therefore, we model a second authentication type, where the events are parameterised by the nonces generated by A and B, and their respective identities, which are the parameters used in the challenge-response performed in the ceremony. Finally, we modelled the authentication properties from both sides, that is from A to B and B to A authentication.

We modelled each one of the scenarios presented in Section 5.4.2 in ProVerif. Similarly to our findings from the informal analysis, in scenarios i and ii, no attack traces were found.

However, as described in our informal analysis, it is often the case where one or both users may not check the values adequately (due to several factors) and just proceed with the pairing. Scenarios iii, iv and v reflect these cases, varying from one human agent to both agents not behaving as expected. The results of formal analysis are very similar in the three cases. When only one human agent does not behave as expected and always confirms the pairing, the authentication property does not hold for only the side that misbehaves. For the the other side, the authentication holds. As mentioned in Section 5.4.2, the harm caused by this attack may vary, since the side that behaved properly can inform the other side about the failure. However, in cases such as a pairing directly followed by file sharing, for example, it may be too late and the file could have been already received by the attacker. When both sides misbehave (scenario v), the attacker successfully deploys a MITM attack. The sequence of steps found in the attack trace generated by ProVerif in this scenario is presented in Figure 5.9. Note that in the case where both sides misbehave, the messages exchanged between the users are not likely to happen in reality, since knowing the other side's value is not relevant in such settings. Finally, by analysing Figure 5.9, we can easily see how the attack works on the other cases (iii and iv). The only difference would be in the last message, where the side behaving as expected would send an *Abort* message (since V_a and V_b are different) rather than *Ok*.



Human A
Device A
Attacker
Device B
Human B
PK_A
PK_I
PK_I
PK_B
Generate: N_a
Generate: N_i and N_i'
Generate: N_b
Calculate : $C_b' = f1(PK_I, PK_A, N_i, 0)$
Calculate : $C_b = f1(PK_B, PK_I, N_b, 0)$
C_b'
C_b
N_a
N_i
N_i'
N_b
Verify if: $C_b' = f1(PK_I, PK_A, N_i', 0)$
Verify if: $C_b = f1(PK_B, PK_I, Ni, 0)$
Calculate: $V_a = g(PK_A, PK_I, N_a, N_i')$
Calculate: $V_b = g(PK_I, PK_B, N_i, N_b)$
V_a
V_b
V_a
V_b
Ok
Ok

Figure 5.9: Sequence diagram for an MITM attack in the SSP in NC mode when A and B always confirm the pairing

For the last scenario, vi, where we assume a *DY* attacker on the *HD* channel between A and U_A, two different attacks were found. The first, where the authentication from B to A is violated, is very straightforward. Since the attacker controls the channel between A and U_A, the trace of the attack indicates that A communicates with the attacker I as a normal pairing session until A sends the verification value V_a to U_A. At this point, the attacker simply replies to A on the HD channel with an Ok message, concluding the pairing session. In practice, this attack can be interpreted in different ways. We could just assume that it happens exactly as the trace shows. That is, even though the number displayed to U_A differs to the one presented to U_B, the attacker (in this case, probably malware installed on device A) sends Ok directly back to A, without prompting U_A for confirmation. Another possibility would be that the behaviour of the buttons “Ok” and “Abort” on the display are modified by the attacker. In this case, even if U_A chooses to abort the connection, the attacker would replace the message by Ok when sending it to A. In both cases, a user that knows how the ceremony works would possibly detect the different behaviour of the ceremony or be notified by U_B that the values do not match. However, the user would have to actively interrupt the pairing (e.g. closing the application), rather than being protected by the ceremony run itself.

A more interesting attack is shown in the attack trace where A's authentication to B is violated. In fact, this attack violates authentication of both sides and it would be undetectable by the users since the ceremony runs as normal. Figure 5.10 presents the attack trace found in ProVerif. The attacker again starts two parallel sessions with A and B, acting as a man-in-the-middle. The ceremony runs normally for both sides until right before the moment where the verification values V_a and V_b are about to be displayed. At this point, the attacker intercepts V_a that is sent on the HD channel between A and U_A and replaces its value by V_i, which is the same V_i generated in the session he runs with B. By doing that, the values displays for both U_A and U_B will be the same. After that, the ceremony runs as normal, U_A and U_B exchange the values shown on their displays (which will match) and confirm the pairing, allowing the attacker to successfully deploy a MITM attack.

5.4.4 Fixing the Simple Secure Pairing Ceremony

Although the authentication mechanism used in SSP in NC mode, where the users are asked to compare two short values, is relatively easy and simple to be used by humans [103, 104, 58], it presents security problems. As discussed in Section 5.4.2, this method is susceptible to a high error rate, going up to a 20% error rate in some scenarios, possibly even more. These errors happen when users fail to compare the values, or even in the case where they do not compare the numbers at all. Additionally, this

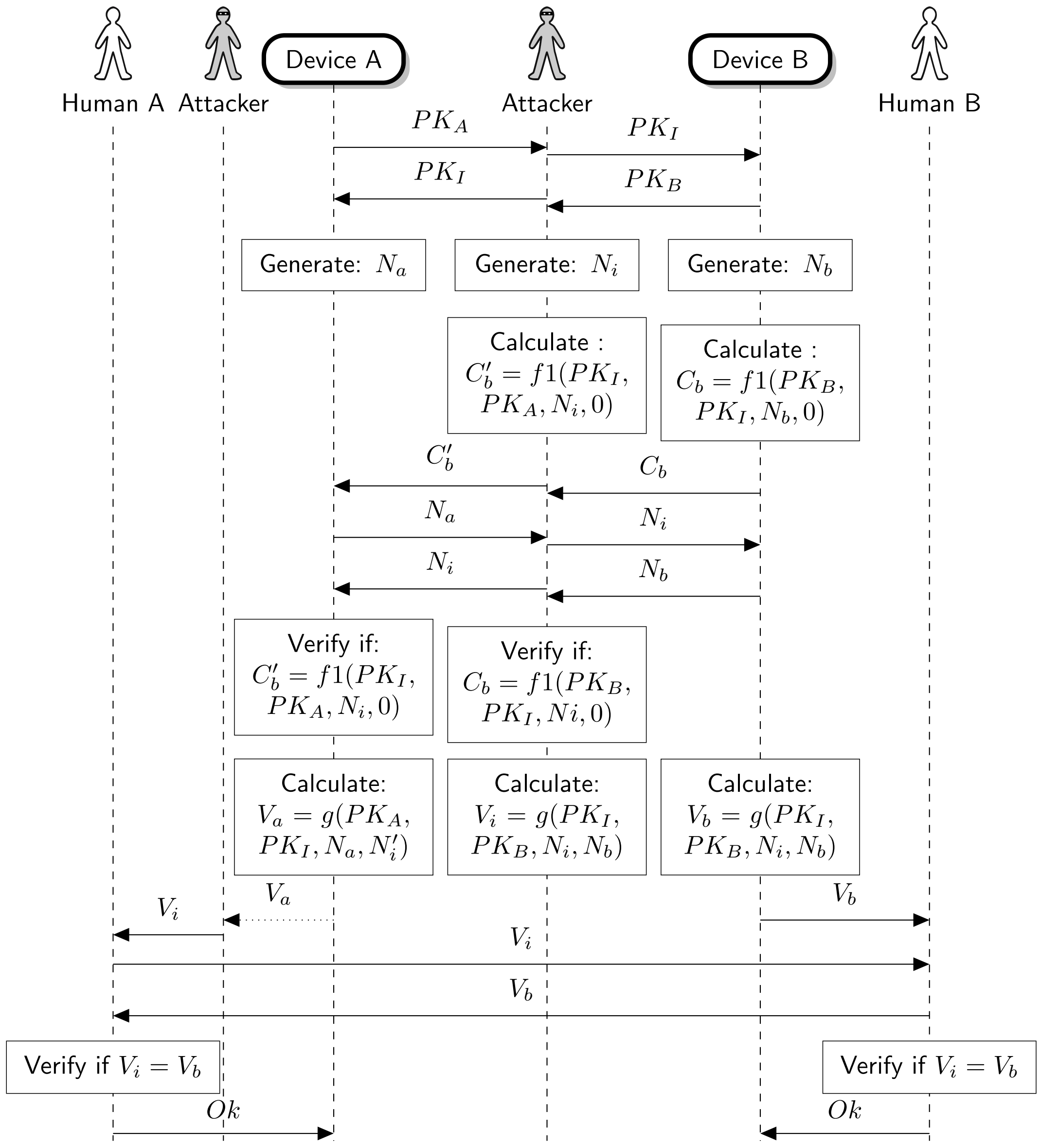

Human A
Attacker
Device A
Attacker
Device B
Human B
PK_A
PK_I
PK_I
PK_B
Generate: N_a
Generate: N_i
Generate: N_b
Calculate : $C'_b = f1(PK_I, PK_A, N_i, 0)$
Calculate : $C_b = f1(PK_B, PK_I, N_b, 0)$
C'_b
C_b
N_a
N_i
N_i
N_b
Verify if: $C'_b = f1(PK_I, PK_A, N_i, 0)$
Verify if: $C_b = f1(PK_B, PK_I, Ni, 0)$
Calculate: $V_a = g(PK_A, PK_I, N_a, N'_i)$
Calculate: $V_i = g(PK_I, PK_B, N_i, N_b)$
Calculate: $V_b = g(PK_I, PK_B, N_i, N_b)$
V_a
V_i
V_b
V_i
V_b
Verify if $V_i = V_b$
Verify if $V_i = V_b$
Ok
Ok

Figure 5.10: Sequence diagram for an MITM attack in the SSP in NC mode when HD_A is under a DY attacker

user interaction is likely to be susceptible to user-conditioning, lack of knowledge of security threats, inaccurate mental models and the decision making influencing factors. Therefore, we propose a different approach to solve the problems we found.

Our amended version for the SSP under the NC mode changes the authentication task given to the user. Instead of asking the users to compare numbers, we make use of a forcing function that prevents the user from proceeding with the pairing without completing the security task (authentication) and that brings the focus on the security activity to the main workflow. Additionally, we focus on making the minimum changes possible to the specification, making the impact of such a change very small and therefore more likely to be implemented in practice.

The new version of the ceremony, shown in Figure 5.11 runs normally until the moment that V_a and V_b are calculated. At this point, we split the values calculated into two parts, producing V_{a1} and V_{a2} for V_a, and V_{b1} and V_{b2} for V_b. Next, rather than sending the complete value to the human via HD channel, each device sends just one of the halves calculated. The initiating device (A) sends the first half of its V_a and the non-initiating device (B) sends the second half of its V_b.

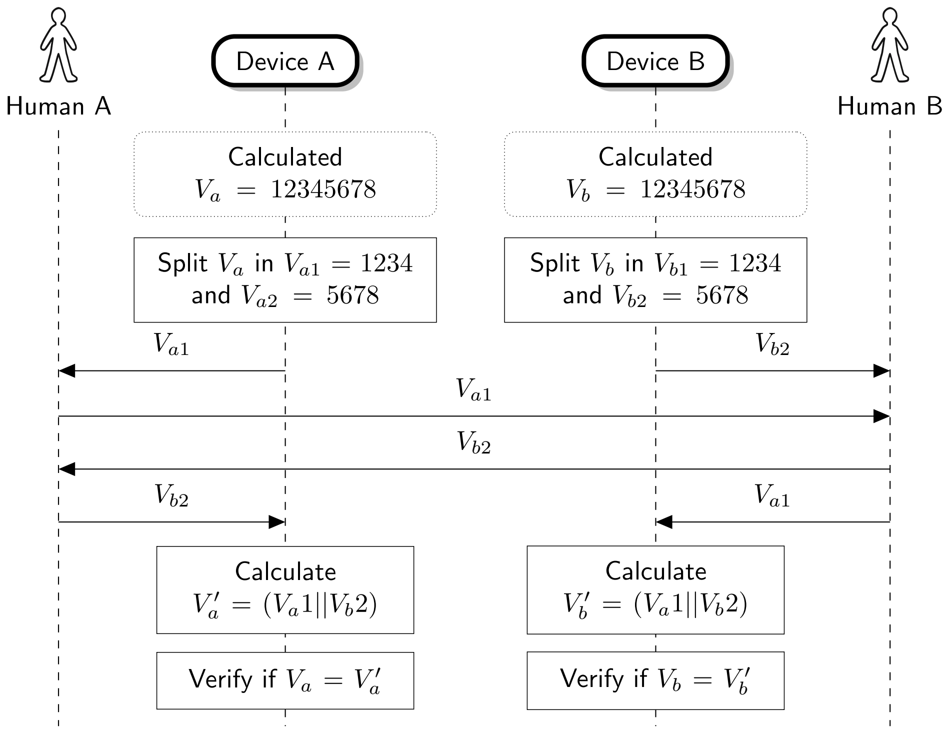

Human A
Device A
Device B
Human B
Calculated $V_a = 12345678$
Calculated $V_b = 12345678$
Split V_a in $V_{a1} = 1234$ and $V_{a2} = 5678$
Split V_b in $V_{b1} = 1234$ and $V_{b2} = 5678$
V_{a1}
V_{b2}
V_{a1}
V_{b2}
V_{b2}
V_{a1}
Calculate $V_a' = (V_a1||V_b2)$
Calculate $V_b' = (V_a1||V_b2)$
Verify if $V_a = V_a'$
Verify if $V_b = V_b'$

Figure 5.11: Sequence diagram for our fix to the SSP in NC mode

After receiving the values from their respective devices, humans U_A and U_B then exchange the values they received via the HH channel. The following step is where our forcing function takes place. Now, rather than just choosing between *Ok* and *Abort*, the user is required to type the value received from the other human peer into their device. This prevents the user from not checking the value and shifts the comparison task from the human to the device. Therefore, the human peers now send the values they receive from the other human to their devices via the HD channel. Each device, when receiving the values, merge the half displayed with the half received and compare with the full value calculated earlier. If the values match, the pairing succeeds, otherwise it fails.

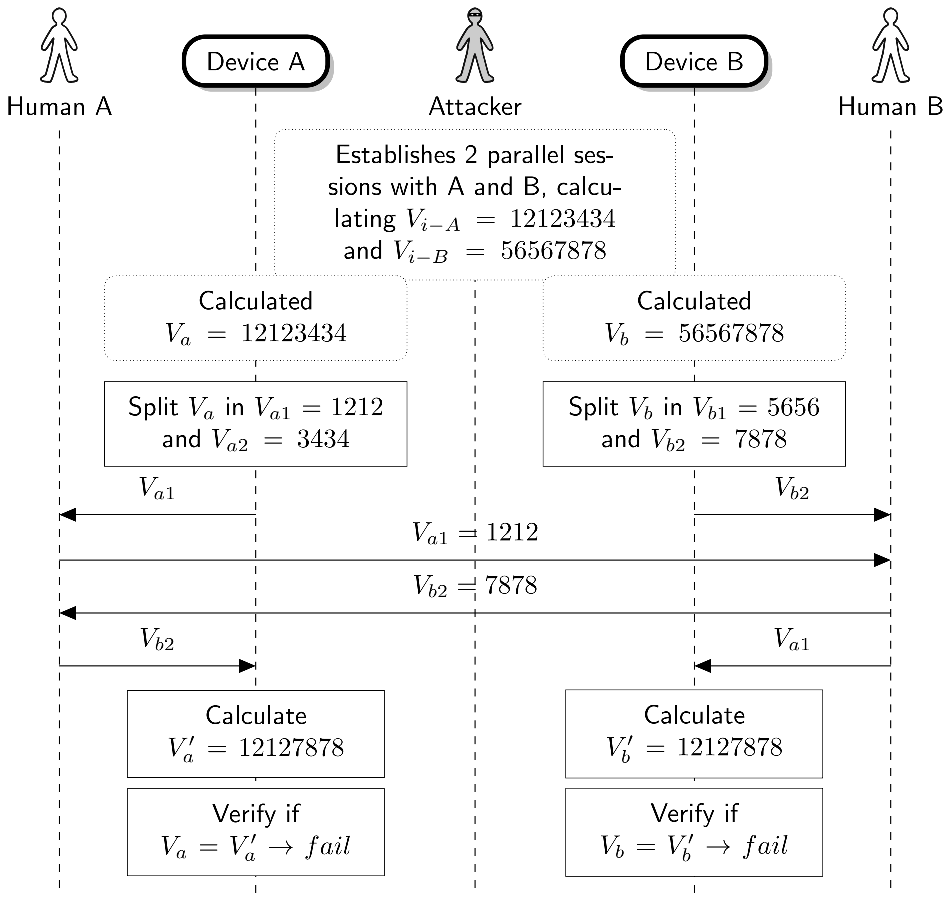

Human A
Device A
Attacker
Device B
Human B
Establishes 2 parallel sessions with A and B, calculating $V_{i-A} = 12123434$ and $V_{i-B} = 56567878$
Calculated $V_a = 12123434$
Calculated $V_b = 56567878$
Split V_a in $V_{a1} = 1212$ and $V_{a2} = 3434$
Split V_b in $V_{b1} = 5656$ and $V_{b2} = 7878$
V_{a1}
V_{b2}
$V_{a1} = 1212$
$V_{b2} = 7878$
V_{b2}
V_{a1}
Calculate $V'_a = 12127878$
Calculate $V'_b = 12127878$
Verify if $V_a = V'_a \to fail$
Verify if $V_b = V'_b \to fail$

Figure 5.12: Sequence diagram of a MITM attempt against our fix to the SSP in NC mode

Figure 5.12 shows a MITM attempt against our variant of SSP. The main difference now is that the authentication property of the ceremony does not depend on correct user behaviour. Now, the user cannot make a mistake that impacts on the security properties of the ceremony. If the user inputs an invalid half of V into his device, the pairing will fail. In the case of an attempted MITM attack, as shown in Figure 5.12, the values exchanged via the HH channel will be sent on the HD channel afterwards. Since the attacker does not control this channel (he can only eavesdrop), he cannot stop this authentication step and prevent the MITM attempt from failing. The only possibility is in the case of the attacker having DY on one of both HD channels, in this case, he could modify the value displayed to the users and inserted by the users, and therefore perform an attack similar to the one presented earlier, in Figure 5.10. Although possible, this attack is more complex than the one presented for SSP 'as-is'.

The attacker would have to display one value to the user, so the correct value needed by the attacker would be sent via the HH channel. In addition, the attacker would have to modify the value entered by the user, replacing it by a value that matches the expected one. Thus, our variation has inherent resistance against user security mistakes, as long as the devices are not compromised.

We updated the formal implementation in ProVerif for scenarios i and ii and similarly to the previous implementation, no attack traces were found. For the scenarios that involved user mistake (iii, iv and v), there is no possible formal implementation since the user cannot perform the authentication task incorrectly, otherwise the pairing ceremony will fail. For vi, where there is a DY attacker on the HD channel between A and U_A, the authentication of B to A fails. This failure happens because the attacker knows the complete value of V_a and therefore sends the second half of V_a through the HD channel between A and U_A. This action can be interpreted as, in practice, the attacker could block what U_A sends to A and replace it by V_{a2}. Similarly, the authentication of A to B fails because the attacker may display the value V_{b1} on A's display. That is, the attacker replaces V_{a1} by V_{b1} on the HD channel between A and U_A. Then, U_A sends V_{b1} to U_B via HH channel, who will then send the value modified by the attacker to B and complete the MITM attack successfully.

The method we proposed here is inherently secure against incorrect user interaction. However, it requires more effort from the users involved. Both users would have to read and type values in to their devices. In usability tests performed in [103, 104, 58] users found, as expected, the default method used in SSP's NC mode easier and faster to use when compared to another method that presents similar interaction to the method we propose here. However, as described by Kainda et al. [58], although the actions of reading and typing values in to devices are more difficult to perform in terms of usability, the mechanism we proposed cannot be incorrectly operated by users. Additionally, the popularity of Short Message Service (SMS) and similar services for mobile devices means that more and more users tend to become familiar with the task of typing text in to devices. Furthermore, in the study performed by Uzun et al. [103] and another by Valkonen et al. [104], a similar method to the one we presented here was perceived as more difficult to use than the default SSP, but also more professional. Among users in the experiment performed by Uzun et al., most users mentioned that this method was the most preferred personal choice and they would like to have it available on their devices.

Looking back to the taxonomy we presented in Chapter 3, we find that the activities involved in the task we added to our ceremony follows the recommendations we proposed. It respects user faculties, since it only asks users to read short strings and input them into a device. More importantly, it removes an authentication task from the user and gives it back to the device, following the recommendation of not relying on user authentication capabilities. It also integrates the security into the main workflow by including the security-related user interaction as an active part of the ceremony. Finally, the use of the forcing function we proposed prevents the user from performing an inappropriate interaction.

5.5 WhatsApp Registration Ceremony

The WhatsApp registration ceremony is simple and relies on the properties that an SSL/TLS connection with the registration server provides. The registration process is straightforward. It uses the user's mobile phone number as the basis for identification, which is sent to the registration server. In response, the server returns an acknowledgement message. The server then sends a PIN[1] code via SMS to the phone number that requested the registration. As we can see, the message that contains the PIN code is sent via an alternative channel. Finally, the registration is confirmed when the PIN code is sent back to the server, now using a regular network channel. This message binds the phone number to the device, and allows the server to complete the registration. As a response, the server sends a confirmation message to the device. Note that all the messages exchanged between the device and the registration server are sent using an HTTPS connection, meaning that there is server authentication and the communication is encrypted.

The whole registration ceremony using protocol notation is presented in Figure 5.13. As we can see, we have four agents and four communication channels in place. First, we have agent S which is the registration server. Second, there is U_A which represents the user operating the device A. In A, we have two subdivisions in our ceremony. Rather than specifying just a device A, we consider two different agents. The agent A_W represents the WhatsApp application installed on the device, while A_M is an SMS application. In most cases, both applications will be installed on the same device, but in fact, it could also be the case where they are on two different devices. The main reason for this separation is because of the nature of the protocol. Although the SMS is sent to device A, in most cases (except for Android OS devices), A_W cannot directly read the contents. Therefore, the user will have to read the contents of the SMS and then type it into the application. This causes the need for additional messages 5 and 6 that deal with the actions of receiving the PIN and sending it to the WhatsApp

[1]Although we disagree with the name "PIN" to represent the value sent by the server, all the references to the WhatsApp protocol call this value by "PIN". In reality, this value acts as a one-time-password rather than a PIN.

application. Once again, this works in the same way, whether the applications are installed on the same device or not.

The four channels we use are composed of:

- the network channel DD between A_W and S (assuming a HTTPS connection);
- a secondary and restricted device-device channel, DD_{SMS} which is specifically used to the deliver the SMS message;
- a HD_{AW} channel between the user U_A and the agent A_W;
- and finally, another HD_{AM} channel between the user U_A and the agent A_M.

$$\begin{array}{lccccc}
1. & U_A & \xrightarrow[HD_{AW}]{} & A_W & : & PhoneNumber \\
2. & A_W & \xrightarrow[DD]{} & S & : & PhoneNumber \\
3. & S & \xrightarrow[DD]{} & A_W & : & NumberReceived \\
4. & S & \xrightarrow[DD_{SMS}]{} & A_M & : & PIN \\
5. & A_M & \xrightarrow[HD_{AM}]{} & U_A & : & PIN \\
6. & U_A & \xrightarrow[HD_{AW}]{} & A_W & : & PIN \\
7. & A_W & \xrightarrow[DD]{} & S & : & PIN \\
8. & S & \xrightarrow[DD]{} & A_W & : & RegistrationConfirmation
\end{array}$$

Figure 5.13: Ceremony description for WhatsApp registration

A more detailed sequence of actions and messages is presented in Figure 5.14. We identified the SMS channel using a larger arrow and presented the application as different agents. The sequence diagram presents the registration ceremony using the *sms* method.

In the $self$ method, presented in Figure 5.15, the larger arrow does not exist. Instead, there is a change in the $NumberReceived$ message contents, which now includes the PIN code. The SMS message containing the PIN code would be sent from the $WhatsApp\ App$ to the $SMS\ App$. The $SMS\ App$ then sends an SMS to the device's own number (using the short message service center - SMSC), 'validating' the number.

Note that the DD_{SMS} channel used in the *sms* method does not exist in the $self$ method. There is a new DD channel, called DD_{SMSC}, that is responsible for the message exchange between $SMS\ App$ and $SMS\ Center$. There is also a HD_{WM} channel, which is an additional device-device channel that exists between the two applications involved in the ceremony: WhatsApp and SMS app.

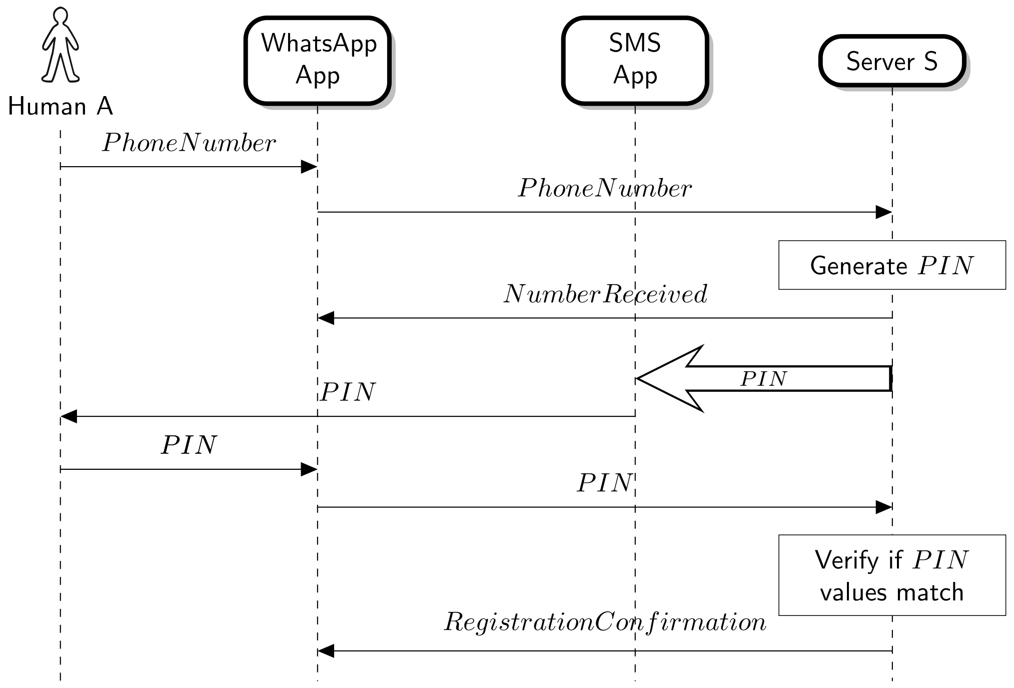


Figure 5.14: Sequence diagram of the WhatsApp registration ceremony using the *sms* method

As presented in Section 2.4.2, in older versions of the registration ceremony, the PIN used to be generated in the device and then sent to the server which caused several security problems. In our analysis we consider only the most recent version, where the PIN is generated on the server side and the methods $self$ and sms are available.

Finally, for analytical purposes, we will also discuss a naive version of the registration protocol, where an SMS message is not used to authenticate the ownership of the number. The ceremony starts in two possible ways. In the first, the user would insert the desired phone number in to the device (using the WhatsApp application), while in the second the application detects the phone number automatically. After obtaining the device's phone number, the application then sends it to the registration server, which will confirm the registration back to the application, as shown in Figure 5.16.

5.5.1 Threat Model

The threat model for the WhatsApp ceremony is difficult to define. Where a HTTPS connection is used, there are several assumptions on the DD channel that are very difficult to model and analyse. There are also different threat models for the DD_{sms} channel, HD_{WM} channel and HD_{AM} channel. For all registration methods, we would

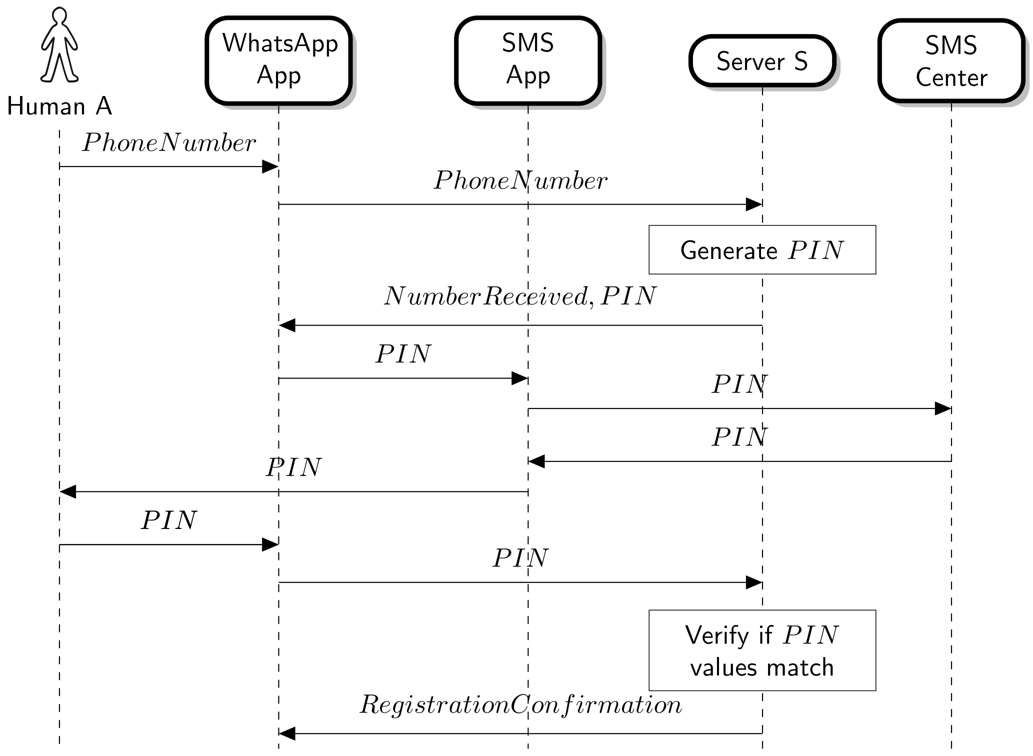


Figure 5.15: Sequence diagram of the WhatsApp registration ceremony using the $self$ method

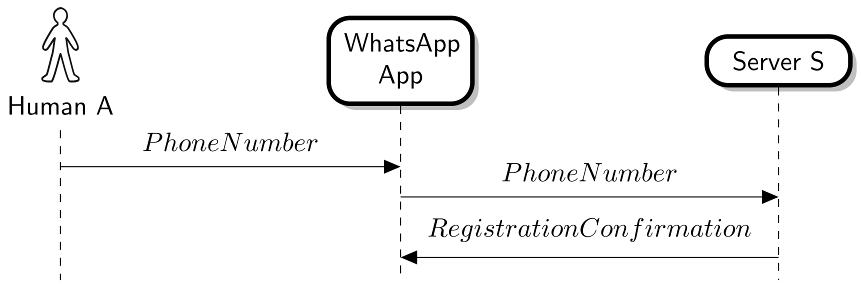


Figure 5.16: Sequence diagram of the WhatsApp registration ceremony without an alternative channel verification

typically assume a DY attacker on the DD channel, however, the current protocol uses the TLS/SSL protocol, which provides server authentication and confidentiality. On the other hand, assuming no attacker on the DD channel may hide some subtleties of the protocol and potential attacks. Therefore, we will approach this scenario would by analysing the ceremony against different variants of the threat model and verifying the outcomes. Thus, for the DD channel, we will consider different variants of the threat model in our analysis. We will discuss a setting where there is no attacker (N), which is likely to be the threat model assumed by the registration protocol designers; another setting that gives DY capabilities to the attacker; and finally, an I attacker, which is capable of initiating a new communication with the server using the knowledge he possesses.

For the restricted DD_{sms} channel, we assume no attackers (N). Finally, for the two channels that involve human interaction (HD), we need to consider variations going from N to E, since the registration may be run in a public location. A DY attacker for the HD channels in this ceremony are very unlikely to exist in practice as long as the device is not corrupted (although we can still analyse the outcome this a scenario in our ceremony).

Note that for the $self$ method, there is an additional device-device channel which exists between the two running applications in the ceremony: WhatsApp and SMS app. For this channel, we assume an N attacker in our analysis. We assume the same attacker capabilities (N) for the DD_{SMSC} as well. All of these different registration methods and channels give us different scenarios where we can analyse the ceremony outcomes and the relevance of the SMS mechanism used.

5.5.2 Informal Analysis

In our informal analysis of the WhatsApp registration ceremony, we consider the three registration methods discussed earlier. This gives us four different scenarios given some variations on the threat model:

i) In this setting, we assume a naive implementation of the registration of the protocol, where no SMS verification is used. That is, the WhatsApp software sends the phone number (received from the user or automatically detected) to the server which confirms the registration. Therefore, there is a maximum of two channels in use in this ceremony. In the case of the user entering the phone number, we have a HD channel where the attacker may be capable of eavesdropping (E). If the phone number is automatically detected, the HD channel does not exist. For the DD channel we assume a DY and an I attacker.

ii) This scenario assumes the use of the $self$ method. In the first setting, we assume a DY attacker for the DD channel, and no attackers (N) for all the remaining communication channels.

iii) In this scenario we change the method to sms. In this setting, we assume a DY attacker for the DD channel, N attacker for the DD_{SMS} channel, and N for both HD channels.

iv) Next, we assume a similar setting to the one we have in iii, but we change the attacker on both HD channels to E.

The ceremony for the scenario i, is obviously flawed. Anyone who knows the victim's phone number can perform the registration impersonating the victim. If the user is asked to insert the phone number, the impersonation attack is very straightforward. There is no need to modify any channel or agent's behaviour. The attacker would simply initiate communication with the server by using his I capability and send the victim's phone number. In practice, an attacker would simply enter the victim's phone number in to the device he possesses and would successfully register an account. However, if the number is detected automatically by the operating system, which is possible in Android but not in iOS, the security would be reduced to operating system (OS) security. This second variation is more reliable than the first, but it is still flawed. To succeed, the attacker would need to corrupt the OS or the application to give himself the I capability and send the phone number he desires instead of the correct one. The attacker does not need to attack the victim's device, he only needs to corrupt any device he possesses.

The next scenario, ii, consists of a more realistic setting. In this case, SMS messages are used to validate the ownership of the phone number using the $self$ method, as shown in Figure 5.15. This method is relatively similar to the *sms* method presented earlier. The main change is that now the PIN is sent back to the device along with the $NumberReceived$ message. Additionally, the SMS containing the PIN is no longer sent by the server. Instead, the device sends an SMS to its own number in order to validate the ownership of the phone number. The method is inherently flawed since the PIN is sent back to the client without any validation of the ownership of the number. The authentication method via PIN using the SMS message is of no use since the client already knows the PIN value.

The attack we found in the scenario ii is very straightforward. The attacker simply initiates a communication with the server using the DD channel sending the victim's phone number. The server replies with the confirmation message and the PIN. At this point, the attacker has all the information necessary to perform the attack. Despite the fact that SSL/TLS is used, the attacker initiates the SSL/TLS session with the server, meaning that all the messages are exchanged between the attacker and the server. In this case, the use of SSL/TLS would only be of benefit if the attacker had only E or N capabilities on the channel, which is not realistic. In this case, the attacker would not be able to read the PIN value, since it would be encrypted due to the SSL/TLS session between the client and the server. However, the attack we presented does not require the user to be engaged in a registration session. The attacker could perform the whole registration ceremony by himself (similarly to the attack discussed in Section 2.4.2.1). Note that this attack would work even considering weaker variations of the attacker model. An I attacker on the DD channel would already be powerful enough to be able to deploy the same attack. Thus, it is clear that the $self$ method should be avoided by all means (although it is still available in the latest version) since it leaks the PIN prior to the verification of the ownership of the number.

In the Figure 5.15, we presented the case where the user has to type the PIN in to the WhatsApp application. However, there is another possible sequence of actions. The application could monitor the SMS application for incoming messages and automatically read the contents of the SMS when it is received. It is not possible in iOS, but it is possible in Android. However, the attack discussed above would still work, since the WhatsApp application would be installed on the attacker's device and the sequence of sending/receiving the SMS is not relevant because the attacker already knows the PIN at this point.

For the sms method, the WhatsApp developers/designers probably assumed that the use of SSL/TLS would imply an N threat model for the DD channels. The phone number is considered public, hence the need for an SMS to verify the ownership of the number. As opposed to the $self$ method, this method would require the attacker to obtain the PIN code that would be sent to the victim's device. As long as the attacker does not control the DD_{SMS} channel, there is no other way to obtain the PIN code apart from obtaining it from the victim's device. Thus, when changing the authentication method to sms, an important change in the ceremony happens. Now, assuming a SSL/TLS connection between the victim's device and the server, the only point where the attacker can learn the PIN value is by eavesdropping one of the HD channels between the user and the SMS application or between the user and the WhatsApp application.

In scenario iii, the attacker, despite having full control of the DD channel, does not have any capability on the other channels and therefore cannot learn the PIN at any point, being unable to confirm his registration. Note that even though the attacker has full control over the DD channel, he will not be able to read the PIN sent by the WhatsApp application to the server on this channel because it is encrypted using an SSL/TLS connection. Additionally, the attacker cannot impersonate the server to the WhatsApp application and act as a man-in-the-middle, since SSL/TLS provides server authentication.

However, in scenario iv, the attacker has eavesdropping capabilities on the HD channels. In practice, this could happen in the case where the attacker is near the victim's device, being able to see its screen. A realistic example could be the case where the victim's device is left on a desk during a meeting or a class without its owner paying much attention to it. The attacker could then launch the attack by engaging in a registration process using the victim's phone number. When the SMS is sent to the user's device, the attacker would just need to read the PIN code that would appear on the victim's display. By using this code, the attacker would be able to register an account impersonating the victim. In this attack, the legitimate user could notice an unexpected SMS being received, but cannot be sure whether this is used as part of an attack or not. In this case, the use of an SSL/TLS connection on the DD channel does not stop the attack from happening. The SSL/TLS connection would be established using the attacker's device. The WhatsApp application on the victim's device is not used during this attack since the attacker initiates the registration process.

5.5.3 Formal Analysis

In our formal model of the WhatsApp registration ceremonies we implemented similar scenarios to those we have seen in the informal analysis. Our implementation of these scenarios used similar mechanisms to those we discussed in Sections 5.3.3 and 5.4.3, with the main focus on modelling authentication, which is the most relevant part of this ceremony.

We modelled in ProVerif each one of the scenarios presented in Section 5.5.2. There is a limitation however. ProVerif does not support multiple variations of the attacker model, which does not allow us to model an I-only attacker. In fact, as we presented in Section 5.3.3, we had to use an implementation trick to be able to use DY, E and N in the same ceremony implementation. The results, however, were not affected by this limitation, as we will see in this section.

In the *sms* method implementation, we had to change an additional configuration setting. We changed the directive **ignoreTypes** to *false*, to force the ceremony to respect the type system. The reason for this change is that the verification had problems because the types defined for PIN and for phone numbers were not being respected, although they were different. When that happened, termination problems occurred because an attacker could send a phone number as a PIN and vice-versa, which does not happen in practice. The side-effect of this change, is that the analysis can possibly find fewer attacks. To prevent this from happening, we carefully analysed the trace before making the change. The previous trace only pointed to the problem we described, so the impact of this change in this scenario is not likely to be a problem.

For scenario i, the attack trace presents an attacker sending a phone number to the server, and receiving the confirmation back. This is similar to the attack we informally discussed in Section 5.5.2 for the same scenario.

In scenario ii, the same attack presented in Section 5.5.2 was found. The attack trace presented in ProVerif in this case is very simple. It describes the attacker sending a phone number to the server using the DD channel and receiving the message containing $NumberReceived$ and PIN back. By having the PIN, the attacker then sends it to the server using the DD channel and the attack is completed. As the attack trace shows, only the attacker, the server S and the DD channel are used for this attack to succeed. The SMS and the other agents in the protocol are not even used in the attack. Therefore, all the other agents and channels in the $self$ method just add more complexity to the ceremony without bringing any additional features or gains.

Moving to the *sms* method implementation, similar results to those discussed in Section 5.5.2 were found. First, for scenario iii, no attacks were found, since the attacker controls the DD channel but cannot obtain the PIN to perform the impersonation attack because the PIN value is protected by the SSL/TLS connection. For the scenario iv, we added eavesdropping capabilities on the HD channels for the attacker to simulate the case where the attacker is near the victim's device, being able to see its display. The attack trace is presented in Figure 5.17. In the attack found by ProVerif, similarly to the attack we described in Section 5.5.2, the attacker starts a registration process by sending a phone number on the DD channel to the server S. The server sends the PIN to the victim's device using the DD_{SMS} channel. The $SMSApp$ then sends (displays) the PIN to the human using the HD_{AM} channel. Since the attacker is able to eavesdrop the HD_{AM} channel, which is represented in the trace by the replication of the message sent on the eavesdrop channel we implemented, the attacker learns the PIN. With the possession of the PIN value, the attacker completes the registration, and therefore the impersonation attack, by sending the PIN he eavesdropped, in to the server S.

Implementing a ceremony that models the properties provided by the SSL/TLS protocol was tricky. If we do not assume an SSL/TLS connection, a MITM attack is possible. The attacker would simply intercept all the messages between the victim and the server, and forward the same messages to the server. In the end, the attacker would complete the registration with the server using the victim's details, and therefore perform an impersonation attack. Since an SSL/TLS connection prevents this type of attack from happening, and provides confidentiality of the messages exchanged, we modelled authentication in a form that we can verify if a PIN received by the server (S) (modelled as an event in Proverif) was preceded by a PIN sent by the WhatsApp application A_W. By doing this, we can still detect attacks where the attacker initiates a registration process with the server (as the attack found in scenario iv), but we do not 'allow' the attacker to perform a MITM attack or read the PIN sent by the victim

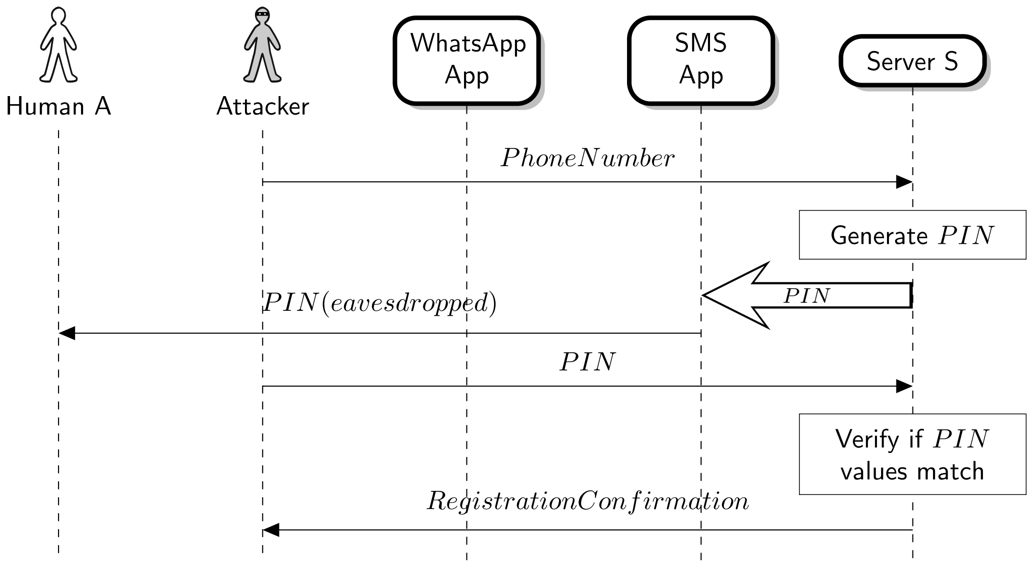


Figure 5.17: Sequence diagram of the attack on the WhatsApp registration ceremony using the *sms* method

through the DD channel.

5.6 Gains by Analysing Security Ceremonies

As we could see in the analysis we performed in this Chapter, as well as in the contents discussed throughout this thesis, the most important gains we obtain by extending protocol design and analysis to ceremonies are:

Realistic analysis – Ceremony analysis implies considering the security problem from practical point of view and in its context of use. Although every change in context implies different ceremonies and additional analysis, the level of accuracy of the results is very high, as the description of the actions are significantly closer to the practical use when compared to traditional protocol descriptions.

Better usability – By having more details of practical aspects, such as user interaction, messages sent and displayed to users and/or exchanged among users, it is easier to perceive how a ceremony would work in practice and consequently how usable it is. By not overstating the attacker's capabilities by assuming an excessively-pessimistic threat model, we prevent the protocol/ceremony designer from adding (unnecessary) features that could imply non-plausible assumptions and/or degraded usability.

Earlier problem detection – Ceremonies are likely to improve the detection of security problems during design phases. Since the level of detail is higher, and the assumptions are more accurate and less numerous, it is likely to be easier to detect security problems during the design process. A good example is the PIN issues we discussed in Bluetooth's legacy pairing and with the $self$ method in the WhatsApp registration.

Better assumptions – Assumptions, which previously provided relatively static input to protocol design, are now a more explicit part of the model. This allows a more detailed analysis of their influence on the ceremony's security goals. As an example, in the Bluetooth SSP we moved user interaction from an assumption to a more explicit exchange of messages with smaller assumptions. Therefore, we were able analyse different scenarios that covered variations of the user behaviour and examine the outcomes. Such differences allow for a better understanding of the impacts in case an assumption does not hold, and may also (although it is very subjective) help the designer to further rationalise about the assumptions made regarding the human peer.

Prediction of the impacts of user misbehaviour – Due to the explicit inclusion of user agents, we can now verify what happens in a ceremony if a user misbehaves, as we did in the Bluetooth SSP analysis. We were able to analyse the impacts of one peer misbehaving in a ceremony as well as the case where two or more peers do not perform their tasks as expected. The changes to the threat model we propose allows a systematic variation of the agent's behaviour and a proper measurement of its impact.

Prediction of the impacts of any agent's misbehaviour – In addition to the possibility of analysing user's misbehaviour, we can verify the impacts, in a ceremony, of any agent's misbehaviour. In the SSP ceremony, we demonstrated this feature by modifying the threat model of the HD channels during the pairing process.

Training users to recognise threats – In some scenarios, adaptive threat models may lead users to be able to detect different threat models for those situations. By executing similar ceremonies which are adapted to run under different (realistic) threat models, users may start to notice the different surrounding threats they are subject to. For example, in Bluetooth SSP pairing, an automatic transition between pairing modes according to the surrounding environment could be used as part of the ceremony. The ceremony could enforce the pairing mode to be used by taking into account, for example, whether there are more Bluetooth enabled devices around before allowing pairing under the just works mode. If there is more than one, the just works mode should not be available since the weakened threat model requires that. If only one device is found, the just works mode could be used securely, since it respects the threat model specified for its use.

Reducing the gap between protocol and HCI communities - By including user interaction in the specification, and analysing such an interaction considering inherent human characteristics and capabilities, we essentially make use of existing knowledge, methods and techniques of two different research areas to achieve a common goal. The findings of such an approach can potentially provide cues for taking research in these areas even further.

5.7 Summary

The findings we obtained from analysing Bluetooth's legacy pairing ceremony are relevant for several reasons. They demonstrate an inaccurate assumption that was made at the design phase of the protocol. Assuming that threats were only existent on the device-device channel is not consistent with the scenarios where the protocol would take place in practice. This may have happened due to the inherent idea of threat modelling in protocols where the threats are only assumed to exist on the network channel. However, protocols such as legacy pairing involve more than just networked communication. Attacks found on the protocol corroborates this. In fact, if the possibility of an attacker eavesdropping the human-device or human-human channel was considered at the design phase, it would be straightforward to notice that the whole security of the protocol relies on the PIN in the first place. As a result, attacks similar to those we discussed could have been found and solved at the design or analysis phase.

Similarly, in the analysis of Bluetooth's SSP ceremony, we found that although the protocol can be considered secure, in practice it is likely to be susceptible to attacks. The assumptions made regarding the human agent and its behaviour are not realistic and do not consider several of the human characteristics we discussed in Chapter 3. We have shown that if the user behaves according to what was assumed, the ceremony is safe. However, there is a high probability that the users do not behave in the expected way, and eventually make errors. Such a probability is even likely to increase over time. To fix this problem we proposed a change in the pairing mechanism which uses a forcing function and maintains the safety of the ceremony even in the case of a user making an inappropriate interaction. Our amended version respects the taxonomy presented in Chapter 3 and follows the proposed recommendations.

For the WhatsApp registration ceremony, we analysed several variants of the authentication mechanisms and threat models. Similarly to the Bluetooth SSP ceremony, where an authentication message is exchanged via an alternative channel, the WhatsApp registration ceremony also makes use of alternative channels for authentication. One relevant difference is that the SSP ceremony achieves the claimed properties even in the case of an attacker being able to eavesdrop the HD channels, while the WhatsApp ceremony does not. In order to achieve the claimed properties, the WhatsApp ceremony limits the attacker capabilities on the DD channel by using an SSL/TLS connection and assumes that the attacker is not capable of performing any action on all remaining communication channels. However, this change in the DD channel demonstrates how difficult composability of security protocols may be. By assuming that the channel was authenticated and confidential (properties provided by SSL/TLS), the designers made several mistakes, such as sending the PIN on this channel right after receiving a phone number, without verifying the ownership of the number first. This allows strong impersonation attacks even in the case where the victim is not actively engaged in registering for the service. For the HD channels, the assumption that the attacker cannot eavesdrop this channel has also proven to be inaccurate in practice, as we demonstrated in our analysis.

Another interesting point raised by the ceremony analysis we performed throughout this chapter was the systematic variation and analysis of scenarios. This gave us interesting insights about the attacks we knew and/or found. Such an approach is interesting for ceremonies, since they allow us to fine-tune the threat model according to the results of the analysis and confirm whether the attack found is realistic or not.

Finally, it was interesting to note the importance of the taxonomy and design recommendations we presented in Chapter 3. We found that even when no attacks were initially found in our analysis, the assumptions regarding the user behaviour need to be carefully analysed. As we demonstrated in the analysis, inaccurate assumptions may lead to security flaws in the ceremony.

Chapter 6

Conclusions and Future Work

In this thesis we investigated and studied the benefits of designing and analysing security protocols from an extended point of view. We discussed the topic of security ceremonies and their components from different perspectives and presented examples of how a multidisciplinary approach could be beneficial to minimising security problems we frequently experience when protocols are used in real-world scenarios.

Security protocols have been actively researched over the last few decades. Methods and techniques to verify whether the claimed security properties are achieved have also been the focus of intensive research. Similarly, but for a slightly shorter period, human-computer interaction has also been the focus of active research. Despite their different nature, they are both very relevant to real word scenarios where secure communication is required. The design and analysis of security ceremonies can benefit from the existing knowledge, methods and tools from both fields. The existing gap between these two relevant research areas is narrowed when we use security ceremonies.

As we discussed throughout this thesis, knowing human capabilities and limitations, as we presented in Chapter 3, is important in order to not overestimate the expectations around the human peer in a ceremony. This also helps us to understand why some protocols fail in practice and teaches us how to avoid such problems in the future. Simply stating that the human is the weakest link does not improve security or make existing security problems less relevant. Understanding the reasons why such user interactions may have caused a security flaw and developing mechanisms that are usable, feasible and respect inherent human characteristics is challenging, but must always be done. The design of a human interaction should be revised if it violates the recommendations we proposed, or does not consider the components we presented.

Ceremonies, as well as protocols, are developed to support human demands for security solutions. An important difference, however, is that security ceremonies explicitly include the human components at the design and analysis phases. The framework we presented in Chapter 4 allows such a description and discusses how the definition of a realistic threat model is important to produce a ceremony that achieves its security goals and is, at the same time, usable. The framework presented makes use of the existing models for security protocols and extends them to encompass ceremony needs.

The examples we analysed in Chapter 5 are very interesting for several reasons. They clearly show the difference between ceremony and protocol analysis. From the granularity of the assumptions involved to the additional number of agents, channels and messages exchanged, we were able to discuss in detail the gains of analysing ceremonies. We presented scenarios where practical security problems could have been found at the design phase. We also demonstrated how reasoning about a realistic threat model can be performed and how the analysis of the possible outcomes is relevant. Not all of the attacks found were plausible in practice, but just the fact that we could identify a possible breach is enough to provide clues to how a ceremony could be improved if necessary.

We demonstrated that existing protocol verification tools and methods can be adapted to mechanically analyse security ceremonies. Another interesting contribution was the systematic variation of the threat model and its impact on the analysis of the ceremonies presented. Such an approach can be used for any ceremony and allows a ceremony designer to fine-tune the threat model. By checking the analysis results and verifying whether the attack found is realistic or not, a more accurate threat model can be defined for a ceremony while the threat model stays realistic. Finally, it was interesting to note how the application of the ceremony design and analysis cycle presented in Section 5.2 is important. Some ceremonies that were initially considered secure can have security flaws in practice because they would not accurately consider the capabilities of the human node.

6.1 Future Work

There are several avenues of research that we believe would be interesting to investigate further. The first, is to increase the number of analysed ceremonies. New scenarios, new interaction types, and analysis of ceremonies that achieve different security goals will surely enrich the coverage we currently have. This would allow further evaluation and refinements of the proposed framework and the human-protocol interaction taxonomy and recommendations.

An interesting type of ceremony to be analysed is a ceremony that includes humans making use of constrained devices, such as one-time-password generators. These devices are widely used in several real-world scenarios, ranging from ATM systems to web authentication. They cannot be modelled as computers or human agents. They are different types of agents in a protocol and therefore are likely to be exposed to a different threat model as well. Another characteristic that is likely to be interesting from a ceremony perspective is modelling the possession of such a device.

Other ceremonies include SSL/TLS implementation in web browsers. One ceremony that still lacks good usability and user interaction is the SSL/TLS handshake ceremony in web browsers. The dynamic acceptance of new and/or unknown digital certificates presented by the server in the handshake is clearly a burden for the users. Assuming that users can handle this task is certainly a flaw and deserves more attention.

A good starting point to approach the problem from a ceremony-based perspective is presented by Gajek [49, 50] et al. and further investigated by Radke et al. [81]. It would be interesting to further analyse the suggestions made by Radke et al. as well as trying different human-server authentication mechanisms, possibly using alternative (authenticated) channels.

All of these different scenarios, along with some user-based experiments are likely to provide interesting outcomes and would help us to further evaluate and refine our framework, as well as verify and improve the human-protocol interaction taxonomy and the set of design recommendations.

It is also worth investigating and elaborating a set of human strengths that can be exploited by the ceremony designers. We briefly mentioned that users are good at authenticating people they know. Other human strengths may include unstructured problem-solving (i.e. solving for problems in which there are no existing rules to solve), acquiring and processing new information (e.g. deciding what is relevant in a new discover), among others. Such a list of strengths could be of great use when design a user interaction.

Another point that can be investigated further is the specification and implementation of the adaptive threat model we proposed in Chapter 4 in the existing verification methods and tools for security protocols. This would allow these existing methods and tools, such as ProVerif, to provide better and extended support for security ceremonies. In Chapter 5, we demonstrated and implemented mechanisms that allowed us to tailor the set of attacker capabilities for each ceremony and adjust them to be more realistic. However, in our analysis, we were limited to a small set of capabilities (as we discussed in Section 5.5.3) that we were able to implement using the tools as they currently are. Additional attacker variations to support a wider range of capabilities are worth studying and implementing. These implementations would definitely allow for better automated testing and provide support for designing a wider range of security ceremonies.

The contextual coverage that ceremonies bring to security protocols is worth investigating further. This can give us better insights into the problem of protocol composability. The composability problem normally happens because of clashes among assumptions that are embedded in protocols. By being able to model more details of the environment, we may now have more tools to predict what happens when two protocols, which are designed focusing on their own respective environments, are put to work together. As we could see in the WhatsApp ceremonies we analysed in Chapter 5.5, just putting two protocols together does not necessarily mean that the properties that each protocol individually provides will be provided when the protocols are used together. The assumption that an attacker could not eavesdrop the PIN code sent by the server to the user's device in the WhatsApp ceremony demonstrates that. Although the SSL/TLS protocol provides confidentiality, when the protocol was put together with another, the confidentiality property of the information sent through the SSL/TLS channel was no longer achieved.

Finally, it would be interesting to investigate how ceremony analysis copes with social engineering attacks. By being able to model human nodes and the exchange of messages between humans, and between humans and devices, we may be able to analyse the impacts of an attacker manipulating a human peer and investigate possible solutions.

Bibliography

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: the spi calculus. In *Proceedings of the 4th ACM conference on Computer and communications security*, CCS '97, pages 36–47, New York, NY, USA, 1997. ACM.

[2] M. Abadi and A. D. Gordon. Reasoning about cryptographic protocols in the spi calculus. In *Proceedings of the 8th International Conference on Concurrency Theory*, CONCUR '97, pages 59–73, London, UK, UK, 1997. Springer-Verlag.

[3] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics*, TCS '00, pages 3–22, London, UK, UK, 2000. Springer-Verlag.

[4] A. Adams and M. A. Sasse. Users are not the enemy. *Communications of the ACM*, 42:40–46, Dec. 1999.

[5] R. D. Alexander. The evolution of social behavior. *Annual Review of Ecology and Systematics*, 5:325–383, 1974.

[6] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Publishing, 2 edition, 2008.

[7] R. Anderson and R. Needham. Programming satan's computer. In J. Leeuwen, editor, *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–440. Springer Berlin Heidelberg, 1995.

[8] W. Arsac, G. Bella, X. Chantry, and L. Compagna. Validating security protocols under the general attacker. In P. Degano and L. Viganò, editors, *Foundations and Applications of Security Analysis*, volume 5511 of *Lecture Notes in Computer Science*, pages 34–51. Springer Berlin / Heidelberg, 2009.

[9] W. Arsac, G. Bella, X. Chantry, and L. Compagna. Multi-attacker protocol validation. *Journal of Automated Reasoning*, 46(3-4):353–388, Apr. 2011.

[10] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *Security and Privacy, 2005 IEEE Symposium on*, pages 171–182, 2005.

[11] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of the 2002 Network and Distributed Systems Security Symposium*, NDSS'02, San Diego, CA, feb 2002.

[12] G. Bella. *Formal Correctness of Security Protocols*. Information Security and Cryptography. Springer Berlin Heidelberg, New York, 2007.

[13] G. Bella. The rational attacker. Invited talk at SAP Research France, Sophia Antipolis (http://www.dmi.unict.it/ giamp/Seminars/rationalattackerSAP08.pdf), 2008.

[14] G. Bella, S. Bistarelli, and F. Massacci. Retaliation: Can we live with flaws? In J. Thomas and M. Essaaidi, editors, *Information Assurance and Computer Security*, volume 6 of *NATO Security through Science*, chapter 1, page 266. IOS Press, Nov. 2006.

[15] G. Bella and L. Coles-Kemp. Layered analysis of security ceremonies. In D. Gritzalis, S. Furnell, and M. Theoharidou, editors, *Information Security and Privacy Research*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 273–286. Springer Berlin Heidelberg, 2012.

[16] G. Bella, C. Longo, and L. C. Paulson. Is the verification problem for cryptographic protocols solved? In *Proceedings of the 11th international conference on Security Protocols*, pages 183–189, Berlin, Heidelberg, 2005. Springer-Verlag.

[17] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Advances in Cryptology – CRYPTO' 93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer Berlin / Heidelberg, 1994.

[18] S. M. Bellovin and M. Merritt. Limitations of the kerberos authentication system. *ACM SIGCOMM Computer Communication Review*, 20:119–132, Oct. 1990.

[19] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE workshop on Computer Security Foundations*, number 82–96 in CSFW '01, Washington, DC, USA, 2001. IEEE Computer Society.

[20] B. Blanchet, B. Smyth, and V. Cheval. *ProVerif 1.88: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial.* INRIA Paris-Rocquencourt, LSV, ENS Cachan & CNRS & INRIA Saclay IIle-de-France, Paris, France, Aug. 2013.

[21] Bluetooth Special Interest Group. Specification of bluetooth system version 1.0a, Dec. 1999.

[22] Bluetooth Special Interest Group. Specification of bluetooth system version 1.2, Nov. 2003.

[23] Bluetooth Special Interest Group. Specification of bluetooth system version 2.0 + EDR, Nov. 2004.

[24] Bluetooth Special Interest Group. Simple pairing whitepaper v10r00. Technical report, Bluetooth Special Interest Group, Aug. 2006.

[25] Bluetooth Special Interest Group. Bluetooth specification version 2.1 + EDR, July 2007.

[26] S. Brostoff and M. A. Sasse. "ten strikes and you're out": Increasing the number of login attempts can improve password usability. In *Proceedings of CHI 2003 Workshop on HCI and Security Systems.* John Wiley, Apr. 2003.

[27] V. Bruce, Z. Henderson, K. Greenwood, P. J. B. Hancock, A. M. Burton, and P. Miller. Verification of face identities from images captured on video. *Journal of Experimental Psychology: Applied*, 5(4):339 – 360, 1999.

[28] J. C. Brustoloni and R. Villamarín-Salomón. Improving security decisions with polymorphic and audited dialogs. In *Proceedings of the 3rd symposium on Usable privacy and security*, SOUPS '07, pages 76–85, New York, NY, USA, 2007. ACM.

[29] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proc of the Royal Society of London*, 426(1871):233–271, 1989.

[30] M. Ĉagalj, S. Ĉapkun, and J.-P. Hubaux. Key agreement in peer-to-peer wireless networks. *Proceedings of the IEEE*, 94(2):467–478, 2006.

[31] M. C. Carlos, J. Martina, G. Price, and R. F. Custodio. An updated threat model for security ceremonies. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1836–1843, New York, NY, USA, Mar. 2013. ACM.

[32] M. C. Carlos, J. E. Martina, G. Price, and R. F. Custodio. A proposed framework for analysing security ceremonies. In P. Samarati, W. Lou, and J. Zhou, editors, *Proceedings of the 7th International Conference on Security and Cryptography*, SECRYPT 12, pages 440–445. SciTePress, July 2012.

[33] M. C. Carlos and G. Price. Understanding the weaknesses of human-protocol interaction. In *Proceedings of the 16th international conference on Financial Cryptography and Data Security*, FC'12, pages 13–26, Berlin, Heidelberg, Mar. 2012. Springer-Verlag.

[34] R. Chang and V. Shmatikov. Formal analysis of authentication in bluetooth device pairing. In P. Degano, R. Kusters, L. Vigano, and S. Zdancewic, editors, *Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis*, FCS-ARSPA'07, July 2007.

[35] L. F. Cranor. A framework for reasoning about the human in the loop. In *Proceedings of the 1st Conference on Usability, Psychology, and Security*, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association.

[36] S. Creese, M. Goldsmith, A. W. Roscoe, and I.Zakiuddin. The attacker in ubiquitous computing environments: formalising the threat model. In *Proceedings of FAST 2003, Pisa*, 2003.

[37] C. J. F. Cremers. *Scyther - Semantics and Verification of Security Protocols.* Ph.D. dissertation, Eindhoven University of Technology, 2006.

[38] R. Dhamija and A. Perrig. Déjà vu: a user study using images for authentication. In *Proceedings of the 9th conference on USENIX Security Symposium - Volume 9*, SSYM'00, pages 4–4, Berkeley, CA, USA, 2000. USENIX Association.

[39] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the 2005 Symposium on Usable Privacy and Security*, SOUPS '05, pages 77–88, New York, NY, USA, 2005. ACM.

[40] R. Dhamija and J. D. Tygar. Phish and hips: Human interactive proofs to detect phishing attacks. In H. Baird and D. Lopresti, editors, *Human Interactive Proofs*, volume 3517 of *Lecture Notes in Computer Science*, pages 69–83. Springer Berlin / Heidelberg, 2005.

[41] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 581–590, New York, NY, USA, 2006. ACM.

[42] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. RFC 5246 (Standards Track), Aug. 2008.

[43] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, Mar. 1983.

[44] J. S. Downs, M. B. Holbrook, and L. F. Cranor. Decision strategies and susceptibility to phishing. In *Proceedings of the second symposium on Usable privacy and security*, SOUPS '06, pages 79–90, New York, NY, USA, 2006. ACM.

[45] S. Egelman, L. F. Cranor, and J. Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 1065–1074, New York, NY, USA, 2008. ACM.

[46] C. Ellison. UPnP security ceremonies design document. Technical report, UPnP Forum, Oct. 2003.

[47] C. Ellison. Ceremony Design and Analysis. Cryptology ePrint Archive, Report 2007/399, Oct. 2007.

[48] M. L. Finucane, A. Alhakami, P. Slovic, and S. M. Johnson. The affect heuristic in judgments of risks and benefits. *Journal of Behavioral Decision Making*, 13(1):1–17, 2000.

[49] S. Gajek, M. Manulis, A.-R. Sadeghi, and J. Schwenk. Provably secure browser-based user-aware mutual authentication over tls. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, ASIACCS '08, pages 300–311, New York, NY, USA, 2008. ACM.

[50] S. Gajek, M. Manulis, and J. Schwenk. User-aware provably secure protocols for browser-based mutual authentication. *Int. J. Appl. Cryptol.*, 1(4):290–308, Aug. 2009.

[51] K. Haataja and P. Toivanen. Practical man-in-the-middle attacks against bluetooth secure simple pairing. In *4th International Conference on Wireless Communications, Networking and Mobile Computing*, WiCOM '08, pages 1–5, oct 2008.

[52] L. D. Harmon and B. Julesz. Masking in visual recognition: Effects of two-dimensional filtered noise. *Science*, 180(4091):1194–1197, 1973.

[53] C. Herley. So long, and no thanks for the externalities: the rational rejection of security advice by users. In *Proceedings of the 2009 workshop on New security paradigms workshop*, NSPW '09, pages 133–144, New York, NY, USA, 2009. ACM.

[54] J.-H. Hoepman. The ephemeral pairing problem. In A. Juels, editor, *Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*, pages 212–226. Springer Berlin Heidelberg, 2004.

[55] P. G. Inglesant and M. A. Sasse. The true cost of unusable password policies: password use in the wild. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 383–392, New York, NY, USA, 2010. ACM.

[56] M. Jakobsson. The human factor in phishing. In *Privacy & Security of Consumer Information '07*, 2007.

[57] M. Jakobsson and S. Wetzel. Security weaknesses in bluetooth. In D. Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 176–191. Springer Berlin / Heidelberg, 2001.

[58] R. Kainda, I. Flechais, and A. W. Roscoe. Usability and security of out-of-band channels in secure device pairing protocols. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS '09, pages 11:1–11:12, New York, NY, USA, 2009. ACM.

[59] C. Karlof, J. Tygar, and D. Wagner. Conditioned-safe ceremonies and a user study of an application to web authentication. In *Sixteenth Annual Network and Distributed Systems Security Symposium*, NDSS 2009, Feb. 2009.

[60] P. Kumaraguru, J. Cranshaw, A. Acquisti, L. Cranor, J. Hong, M. A. Blair, and T. Pham. School of phish: a real-world evaluation of anti-phishing training. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS '09, pages 3:1–3:12, New York, NY, USA, 2009. ACM.

[61] A. Kurtz. Shooting the messenger. World Wide Web electronic publication. Retrieved August 25, 2013 from: `http://www.andreas-kurtz.de/2011/09/shooting-messenger.html`, Sept. 2011.

[62] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In *Proceedings of the Second International Workshop on Tools and Algorithms*

for Construction and Analysis of Systems, TACAs '96, pages 147–166, London, UK, UK, 1996. Springer-Verlag.

[63] M. Marlinspik. More tricks for defeating SSL in practice. In *BlackHat USA Briefings*, Las Vegas, NV, July 2009.

[64] M. Marlinspik. New tricks for defeating SSL in practice. In *BlackHat Europe Briefings*, Amsterdam, Apr. 2009.

[65] M. Marlinspike. SSLSniff version 0.8. http://www.thoughtcrime.org/software/sslsniff/, July 2011.

[66] J. E. Martina and M. C. Carlos. Why should we analyse security ceremonies? First CryptoForma workshop, May 2010.

[67] J. L. Massey, G. H. Khachatrian, and M. K. Kuregian. Safer+. In *Proceedings of First Advanced Encryption Standard Candidate Conference*. National Institute of Standards and Technology (NIST), 1998.

[68] C. Meadows. Language generation and verification in the nrl protocol analyzer. In *Proceedings of the 9th IEEE workshop on Computer Security Foundations*, CSFW '96, pages 48–, Washington, DC, USA, 1996. IEEE Computer Society.

[69] C. Meadows. Formal methods for cryptographic protocol analysis: emerging issues and trends. *Selected Areas in Communications, IEEE Journal on*, 21(1):44–54, 2003.

[70] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and computation*, 100(1):1–40, Sept. 1992.

[71] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of ACM*, 21(12):993–999, Dec. 1978.

[72] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 4120 (Standards Track), July 2005.

[73] D. A. Norman. Design rules based on analyses of human error. *Commun. ACM*, 26:254–258, Apr. 1983.

[74] D. A. Norman. *The design of everyday things*. Basic Books, New York, NY, USA, Aug. 2002.

[75] N. Olivarez-Giles. Whatsapp adds voice messaging as it hits 300 million monthly active users. World Wide Web electronic publication. Retrieved August 25, 2013 from: http://www.theverge.com/2013/8/6/4595496/whatsapp-300-million-active-users-voice-messaging-update, Aug. 2013.

[76] R. Oppliger and S. Gajek. Effective protection against phishing and web spoofing. In J. Dittmann, S. Katzenbeisser, and A. Uhl, editors, *Communications and Multimedia Security*, volume 3677 of *Lecture Notes in Computer Science*, pages 32–41. Springer Berlin / Heidelberg, 2005.

[77] G. Parker. Assessment strategy and the evolution of fighting behaviour. *Journal of Theoretical Biology*, 47(1):223–243, 1974.

[78] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal Computer Security*, 6(1-2):85–128, Jan. 1998.

[79] T. R. Peltier. Social engineering: Concepts and solutions. *Information Systems Security*, 15(5):13–21, 2006.

[80] K. Radke, C. Boyd, J. Gonzalez Nieto, and M. Brereton. Ceremony analysis: Strengths and weaknesses. In J. Camenisch, S. Fischer-Hübner, Y. Murayama, A. Portmann, and C. Rieder, editors, *Future Challenges in Security and Privacy for Academia and Industry*, volume 354 of *IFIP Advances in Information and Communication Technology*, pages 104–115. Springer Berlin Heidelberg, 2011.

[81] K. Radke, C. Boyd, J. M. G. Nieto, and M. Brereton. Towards a secure human-and-computer mutual authentication protocol. In *Proceedings of the Tenth Australasian Information Security Conference (AISC 2012)*, pages 39–46. Australian Computer Society Inc, 2012.

[82] J. Reason. Understanding adverse events: human factors. *Quality in Health Care*, 4(2):80–89, 1995.

[83] R. Ruksenas, P. Curzon, and A. Blandford. Detecting cognitive causes of confidentiality leaks. *Electronic Notes in Theoretical Computer Science*, 183:21–38, 2007. Proceedings of the First International Workshop on Formal Methods for Interactive Systems (FMIS 2006).

[84] R. Ruksenas, P. Curzon, and A. Blandford. Modelling and analysing cognitive causes of security breaches. *Innovations in Systems and Software Engineering*, 4:143–160, 2008.

[85] P. Ryan and S. Schneider. *Modelling and analysis of security protocols.* Addison Wesley, 1 edition, 2001.

[86] J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[87] M. A. Sasse, S. Brostoff, and D. Weirich. Transforming the 'weakest link' - a human/computer interaction approach to usable and effective security. *BT Technology Journal*, 19:122–131, July 2001.

[88] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer. Emperor's new security indicators: An evaluation of website authentication and the effect of role playing on usability studies. In *In Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 51–65. IEEE, May 2007.

[89] S. Schneider. Verifying authentication protocols in csp. *Software Engineering, IEEE Transactions on*, 24(9):741–758, 1998.

[90] B. Schneier. Two-factor authentication: too little, too late. *Commun. ACM*, 48(4):136, Apr. 2005.

[91] S. Schrittwieser, P. Fruehwirt, P. Kieseberg, M. Leithner, M. Mulazzani, M. Huber, and E. Weippl. Guess who is texting you? evaluating the security of smartphone messaging applications. In *Network and Distributed System Security Symposium (NDSS 2012)*, 2 2012.

[92] Y. Shaked and A. Wool. Cracking the bluetooth pin. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, MobiSys '05, pages 39–50, New York, NY, USA, 2005. ACM.

[93] P. Sinha, B. Balas, Y. Ostrovsky, and R. Russell. Face recognition by humans: Nineteen results all computer vision researchers should know about. *Proceedings of the IEEE*, 94(11):1948–1962, 2006.

[94] S. W. Smith. Humans in the loop: Human-computer interaction and security. *IEEE Security and Privacy*, 1(3):75–79, May 2003.

[95] F. Stajano. The resurrecting duckling — what next? In B. Christianson, J. Malcolm, B. Crispo, and M. Roe, editors, *Security Protocols*, volume 2133 of *Lecture Notes in Computer Science*, pages 204–214. Springer Berlin Heidelberg, 2001.

[96] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Proceedings of the 7th International Workshop on Security*

Protocols, number 1796 in Lecture Notes in Computer Science, pages 172–194, Berlin, Heidelberg, 1999. Springer-Verlag.

[97] F. Stajano and R. Anderson. The resurrecting duckling: security issues for ubiquitous computing. *Computer*, 35(4):22–26, Apr. 2002.

[98] F. Stajano and P. Wilson. Understanding scam victims: seven principles for systems security. Technical Report 754, University of Cambridge, Aug. 2009.

[99] F. Stajano and P. Wilson. Understanding scam victims: seven principles for systems security. *Communications of the ACM*, 54(3):70–75, Mar. 2011.

[100] L. Standing, J. Conezio, and R. N. Haber. Perception and memory for pictures: Single-trial learning of 2500 visual stimuli. *Psychonomic Science*, 19:73–74, 1970.

[101] J. M. Stanton, K. R. Stam, P. Mastrangelo, and J. Jolton. Analysis of end user security behaviors. *Computers & Security*, 24(2):124 – 133, 2005.

[102] Team Venomous. WhatsAPI. https://github.com/venomous0x/WhatsAPI/blob/master/README.md, July 2013.

[103] E. Uzun, K. Karvonen, and N. Asokan. Usability analysis of secure pairing methods. In *Proceedings of the 11th International Conference on Financial cryptography and 1st International conference on Usable Security*, FC'07/USEC'07, pages 307–324, Berlin, Heidelberg, 2007. Springer-Verlag.

[104] J. Valkonen, A. Toivonen, and K. Karvonen. Usability testing for secure device pairing in home networks. In A. Bajart, H. Muller, and T. Strang, editors, *Proceedings of UbiComp 2007 Workshop*, Sept. 2007.

[105] M. Vanhoef. Whatsapp considered insecure. World Wide Web electronic publication. Retrieved August 25, 2013 from: http://www.mathyvanhoef.com/2012/05/whatsapp-considered-insecure.html, May 2012.

[106] S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 309–326. Springer Berlin Heidelberg, 2005.

[107] R. West. The psychology of security. *Communications of the ACM*, 51:34–40, Apr. 2008.

[108] WhatsApp Inc. new daily record: 4b inbound, 6b outbound = 10b total messages a day! World Wide Web electronic publication. Retrieved August 25, 2013 from: https://twitter.com/WhatsApp/statuses/238680463139565568, Aug. 2012.

[109] WhatsApp Inc. new daily record: 10b+ msgs sent (inbound) and 17b+ msgs received (outbound) by our users = 27 billion msgs handled in just 24 hours! World Wide Web electronic publication. Retrieved August 25, 2013 from: https://twitter.com/WhatsApp/status/344966710241161216, June 2013.

[110] F. L. Wong and F. Stajano. Multichannel security protocols. *IEEE Pervasive Computing*, 6(4):31–39, Oct. 2007.

[111] M. Wu, R. C. Miller, and S. L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 601–610, New York, NY, USA, 2006. ACM.

[112] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: empirical results. *IEEE Security and Privacy*, 2(5):25–31, Sept. 2004.

[113] K.-P. Yee. Aligning security and usability. *IEEE Security and Privacy*, 2:48–55, Sept. 2004.

Appendix A

Proverif Source Code of the Bluetooth Legacy Pairing Ceremonies

A.1 Bluetooth Legacy Pairing - scenario i

```
(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)

(* Types *)
type key.
type nonce. (* for nonces *)
type ui_input. (* for user input messages via UI *)
type ui_message. (* for messages displayed via UI  *)

(* Symetric Encryption *)
fun senc(bitstring, key): bitstring.
reduc forall m: bitstring , k : key; sdec(senc(m,k), k) = m.

(* Ceremony/Protocol Specific Functions *)
fun e22(bitstring,bitstring,nonce) : key.
fun e1(key,bitstring,bitstring) : bitstring.

(* Public information *)
free BD_ADDR_A : bitstring. (* A's Bluetooth address *)
free BD_ADDR_B : bitstring. (* B's Bluetooth address *)

(* Private information*)
free pin:bitstring[private].

(* Queries / Tests *)
weaksecret pin.

(* User A *)
let processUA =
  in (hdA, pin_reqX : ui_message);    (* 3. A -hd-> UA : PIN REQ
     *)
  out (hh, pin);    (* 5. UA -hh-> UB : PIN *)
  out (hdA, pin).   (* 6. UA -hh-> A : PIN *)

(* User B *)
let processUB =
  in (hdB, pin_reqX : ui_message);    (* 3. A -hd-> UA : PIN REQ
     *)
  in (hh, pinX : bitstring);    (* 5. UA -hh-> UB : PIN REQ *)
  out (hdB, pinX).    (* 7. UB -hh-> B : PIN *)

(* Device A *)
let processA =
  new in_rand : nonce;
  out(dd, in_rand);   (* 1. A -> B : IN_RAND *)
  in(dd, accept: ui_input);     (* 2. B -> A : ACCEPT *)
  new pin_reqa : ui_message;
  out(hdA, pin_reqa);   (* 3. A -hd-> UA : PIN REQ *)
  in (hdA, pinX : bitstring);   (* 6. UA -hd-> A : PIN *)
  let kinit_a = e22(BD_ADDR_B, pinX, in_rand) in    (* A
     generates Kinit *)
  new au_rand_a : bitstring;
  out(dd, au_rand_a);   (* 3. A -> B : AU_RANDa *)
  in(dd, sres1X:bitstring); (* 4. B -> A : SRES1 *)
  if sres1X = e1(kinit_a, BD_ADDR_B,au_rand_a) then
  in(dd, au_rand_bX:bitstring); (* 5. B -> A : AU_RANDb *)
  let sres2 = e1(kinit_a, BD_ADDR_A, au_rand_bX) in
  out(dd, sres2).     (* 6. A -> B : SRES2 *)

(* Device B *)
let processB =
  in(dd, in_randX : nonce); (* 1. A -> B : IN_RAND *)
  new accept:ui_input;
  out(dd, accept);      (* 2. B -> A : ACCEPT *)
  new pin_reqb : ui_message;
  out(hdB, pin_reqb);   (* 3. A -hd-> UA : PIN REQ *)
  in (hdB, pinX : bitstring);   (* 7. UB -hd-> B : PIN *)
  let kinit_b = e22(BD_ADDR_B, pinX, in_randX) in   (* B
     generates Kinit *)
  in(dd, au_rand_aX:bitstring);   (* 3. A -> B : AU_RANDa *)
  let sres1 = e1(kinit_b, BD_ADDR_B,au_rand_aX) in  (* calculate
     SRES1 *)
  out(dd, sres1);       (* 4. B -> A : SRES1 *)
  new au_rand_b : bitstring;
  out(dd, au_rand_b);     (* 5. B -> A : AU_RANDb*)
  in(dd, sres2X:bitstring); (* 6. A -> B : SRES2 4 *)
  if sres2X = e1(kinit_b, BD_ADDR_B,au_rand_b) then
  0.


process
  (
    (!processA) | (!processB) | (!processUA) | (!processUB)
  )
```

A.2 Bluetooth Legacy Pairing - scenario ii

```
(*------------------------------------------------
 Protocol specification
-----------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)

(* Types *)
type key.
type nonce.
type ui_input.
type ui_message.

(* Symetric Encryption *)
fun senc(bitstring, key): bitstring.
reduc forall m: bitstring , k : key; sdec(senc(m,k), k) = m.

(* Ceremony/Protocol Specific Functions *)
fun e22(bitstring,bitstring,nonce) : key.
fun e1(key,bitstring,bitstring) : bitstring.

(* Public information *)
free BD_ADDR_A : bitstring. (* A's Bluetooth address *)
free BD_ADDR_B : bitstring. (* B's Bluetooth address *)

(* Private information*)
free pin:bitstring[private].

(* Queries / Tests *)
free s:bitstring[private]. (* secret that the attacker should not
    be able to obtain *)
query attacker(s).
```

```
(* User A *)
let processUA =
  in (hdA, pin_reqX : ui_message);    (* 3. A -hd-> UA : PIN REQ
     *)
  out (hh, pin);    (* 5. UA -hh-> UB : PIN *)
  out (hdA, pin).   (* 6. UA -hh-> A : PIN *)

(* User B *)
let processUB =
  in (hdB, pin_reqX : ui_message);    (* 3. A -hd-> UA : PIN REQ
     *)
  in (hh, pinX : bitstring);    (* 5. UA -hh-> UB : PIN REQ *)
  out (hdB, pinX).    (* 7. UB -hh-> B : PIN *)

(* Device A *)
let processA =
  new in_rand : nonce;
  out(dd, in_rand);   (* 1. A -> B : IN_RAND *)
  in(dd, accept: ui_input);     (* 2. B -> A : ACCEPT *)
  new pin_reqa : ui_message;
  out(hdA, pin_reqa);   (* 3. A -hd-> UA : PIN REQ *)
  in (hdA, pinX : bitstring);   (* 6. UA -hd-> A : PIN *)
  let kinit_a = e22(BD_ADDR_B, pinX, in_rand) in    (* A
     generates Kinit *)
  new au_rand_a : bitstring;
  out(dd, au_rand_a);   (* 3. A -> B : AU_RANDa *)
  in(dd, sres1X:bitstring); (* 4. B -> A : SRES1 *)
  if sres1X = e1(kinit_a, BD_ADDR_B,au_rand_a) then
  in(dd, au_rand_bX:bitstring); (* 5. B -> A : AU_RANDb *)
  let sres2 = e1(kinit_a, BD_ADDR_A, au_rand_bX) in
  out(dd, sres2);     (* 6. A -> B : SRES2 *)
  out(dd, senc(s,kinit_a)).

(* Device B *)
let processB =
  in(dd, in_randX : nonce); (* 1. A -> B : IN_RAND *)
  new accept:ui_input;
  out(dd, accept);      (* 2. B -> A : ACCEPT *)
  new pin_reqb : ui_message;
  out(hdB, pin_reqb);   (* 3. A -hd-> UA : PIN REQ *)
  in (hdB, pinX : bitstring);   (* 7. UB -hd-> B : PIN *)
  let kinit_b = e22(BD_ADDR_B, pinX, in_randX) in   (* B
     generates Kinit *)
  in(dd, au_rand_aX:bitstring);   (* 3. A -> B : AU_RANDa *)
  let sres1 = e1(kinit_b, BD_ADDR_B,au_rand_aX) in  (* calculate
     SRES1 *)
  out(dd, sres1);       (* 4. B -> A : SRES1 *)
  new au_rand_b : bitstring;
  out(dd, au_rand_b);     (* 5. B -> A : AU_RANDb*)
  in(dd, sres2X:bitstring); (* 6. A -> B : SRES2 4 *)
  if sres2X = e1(kinit_b, BD_ADDR_B,au_rand_b) then
  in(dd,x:bitstring);
  let z = sdec(x,kinit_b) in
  if z = s then
  0.


process
  (
    (!processA) | (!processB) | (!processUA) | (!processUB)
  )
```

A.3 Bluetooth Legacy Pairing - scenario iii

```
(*---------------------------------------------------
 Protocol specification
---------------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdAe : channel. (* human-device channel (eavesdrop) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hdBe : channel. (* human-device channel (eavesdrop) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)
free hhe : channel. (* human-human channel (eavesdrop) *)

(* Types *)
type key.
type nonce.
type ui_input.
type ui_message.

(* Symetric Encryption *)
fun senc(bitstring, key): bitstring.
reduc forall m: bitstring , k : key; sdec(senc(m,k), k) = m.

(* Ceremony/Protocol Specific Functions *)
fun e22(bitstring,bitstring,nonce) : key.
fun e1(key,bitstring,bitstring) : bitstring.

(* Public information *)
free BD_ADDR_A : bitstring. (* A's Bluetooth address *)
free BD_ADDR_B : bitstring. (* B's Bluetooth address *)

(* Private information*)
free pin:bitstring[private].

(* Queries / Tests *)
free s:bitstring[private]. (* secret that the attacker should not
    be able to obtain *)
```

```
query attacker(s).


(* User A *)
let processUA =
  in (hdA, pin_reqX : ui_message);    (* 3. A -hd-> UA : PIN REQ
     *)
  out (hh, pin);    (* 5. UA -hh-> UB : PIN *)
  out (hdA, pin).   (* 6. UA -hh-> A : PIN *)

(* User B *)
let processUB =
  in (hdB, pin_reqX : ui_message);    (* 3. A -hd-> UA : PIN REQ
     *)
  in (hh, pinX : bitstring);    (* 5. UA -hh-> UB : PIN REQ *)
  out (hdB, pinX).    (* 7. UB -hh-> B : PIN *)

(* Device A *)
let processA =
  new in_rand : nonce;
  out(dd, in_rand);   (* 1. A -> B : IN_RAND *)
  in(dd, accept: ui_input);     (* 2. B -> A : ACCEPT *)
  new pin_reqa : ui_message;
  out(hdA, pin_reqa);   (* 3. A -hd-> UA : PIN REQ *)
  in (hdA, pinX : bitstring);   (* 6. UA -hd-> A : PIN *)
  let kinit_a = e22(BD_ADDR_B, pinX, in_rand) in    (* A
     generates Kinit *)
  new au_rand_a : bitstring;
  out(dd, au_rand_a);   (* 3. A -> B : AU_RANDa *)
  in(dd, sres1X:bitstring); (* 4. B -> A : SRES1 *)
  if sres1X = e1(kinit_a, BD_ADDR_B,au_rand_a) then
  in(dd, au_rand_bX:bitstring); (* 5. B -> A : AU_RANDb *)
  let sres2 = e1(kinit_a, BD_ADDR_A, au_rand_bX) in
  out(dd, sres2);     (* 6. A -> B : SRES2 *)
  out(dd, senc(s,kinit_a)).

(* Device B *)
let processB =
  in(dd, in_randX : nonce); (* 1. A -> B : IN_RAND *)
  new accept:ui_input;
  out(dd, accept);      (* 2. B -> A : ACCEPT *)
  new pin_reqb : ui_message;
  out(hdB, pin_reqb);   (* 3. A -hd-> UA : PIN REQ *)
  in (hdB, pinX : bitstring);   (* 7. UB -hd-> B : PIN *)
  let kinit_b = e22(BD_ADDR_B, pinX, in_randX) in   (* B
     generates Kinit *)
  in(dd, au_rand_aX:bitstring);   (* 3. A -> B : AU_RANDa *)
  let sres1 = e1(kinit_b, BD_ADDR_B,au_rand_aX) in  (* calculate
     SRES1 *)
  out(dd, sres1);       (* 4. B -> A : SRES1 *)
  new au_rand_b : bitstring;
  out(dd, au_rand_b);     (* 5. B -> A : AU_RANDb*)
  in(dd, sres2X:bitstring); (* 6. A -> B : SRES2 4 *)
  if sres2X = e1(kinit_b, BD_ADDR_B,au_rand_b) then
  in(dd,x:bitstring);
  let z = sdec(x,kinit_b) in
  if z = s then
  0.


process
  (
    (!processA) | (!processB) | (!processUA) | (!processUB)
    | (!in(hdA,x:bitstring); out(hdAe,x)) (* this replicates
       everything in hdA to hdAe to reproduce a passive attacker
       (simulates eavesdrop on hdA) *)
    | (!in(hdB,x:bitstring); out(hdBe,x)) (* this replicates
       everything in hdB to hdBe to reproduce a passive attacker
       (simulates eavesdrop on hdB) *)
    | (!in(hh,x:bitstring); out(hhe,x)) (* this replicates
       everything in hh to hh to reproduce a passive attacker (
       simulates eavesdrop on hh) *)
  )
```

A.4 Bluetooth Legacy Pairing - scenario iv

```
(*-----------------------------------------------
 Protocol specification
-----------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdAe : channel. (* human-device channel (eavesdrop) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hdBe : channel. (* human-device channel (eavesdrop) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)
free hhe : channel. (* human-human channel (eavesdrop) *)

(* Types *)
type key.
type nonce.
type ui_input.
type ui_message.

(* Symetric Encryption *)
fun senc(bitstring, key): bitstring.
reduc forall m: bitstring , k : key; sdec(senc(m,k), k) = m.

(* Ceremony/Protocol Specific Functions *)
fun e22(bitstring,bitstring,nonce) : key.
fun e1(key,bitstring,bitstring) : bitstring.

(* Public information *)
free BD_ADDR_A : bitstring. (* A's Bluetooth address *)
free BD_ADDR_B : bitstring. (* B's Bluetooth address *)

(* Private information*)
free pin:bitstring[private].
weaksecret pin.

(* Queries / Tests *)
free s:bitstring[private]. (* secret that the attacker should not
    be able to obtain *)
query attacker(s).


(* User A *)
let processUA =
  in (hdA, pin_reqX : ui_message);    (* 3. A -hd-> UA : PIN REQ
     *)
  out (hh, pin);    (* 5. UA -hh-> UB : PIN *)
  out (hdA, pin).   (* 6. UA -hh-> A : PIN *)

(* User B *)
let processUB =
  in (hdB, pin_reqX : ui_message);    (* 3. A -hd-> UA : PIN REQ
     *)
  in (hh, pinX : bitstring);    (* 5. UA -hh-> UB : PIN REQ *)
  out (hdB, pinX).    (* 7. UB -hh-> B : PIN *)

(* Device A *)
let processA =
  new in_rand : nonce;
  out(dd, in_rand);   (* 1. A -> B : IN_RAND *)
  in(dd, accept: ui_input);     (* 2. B -> A : ACCEPT *)
  new pin_reqa : ui_message;
  out(hdA, pin_reqa);   (* 3. A -hd-> UA : PIN REQ *)
  in (hdA, pinX : bitstring);   (* 6. UA -hd-> A : PIN *)
  let kinit_a = e22(BD_ADDR_B, pinX, in_rand) in    (* A
     generates Kinit *)
  new au_rand_a : bitstring;
  out(dd, au_rand_a);   (* 3. A -> B : AU_RANDa *)
  in(dd, sres1X:bitstring); (* 4. B -> A : SRES1 *)
  if sres1X = e1(kinit_a, BD_ADDR_B,au_rand_a) then
  in(dd, au_rand_bX:bitstring); (* 5. B -> A : AU_RANDb *)
  let sres2 = e1(kinit_a, BD_ADDR_A, au_rand_bX) in
  out(dd, sres2);     (* 6. A -> B : SRES2 *)
  out(dd, senc(s,kinit_a)).

(* Device B *)
let processB =
  in(dd, in_randX : nonce); (* 1. A -> B : IN_RAND *)
  new accept:ui_input;
  out(dd, accept);      (* 2. B -> A : ACCEPT *)
  new pin_reqb : ui_message;
  out(hdB, pin_reqb);   (* 3. A -hd-> UA : PIN REQ *)
  in (hdB, pinX : bitstring);   (* 7. UB -hd-> B : PIN *)
  let kinit_b = e22(BD_ADDR_B, pinX, in_randX) in   (* B
     generates Kinit *)
  in(dd, au_rand_aX:bitstring);   (* 3. A -> B : AU_RANDa *)
  let sres1 = e1(kinit_b, BD_ADDR_B,au_rand_aX) in  (* calculate
     SRES1 *)
  out(dd, sres1);       (* 4. B -> A : SRES1 *)
  new au_rand_b : bitstring;
  out(dd, au_rand_b);     (* 5. B -> A : AU_RANDb*)
  in(dd, sres2X:bitstring); (* 6. A -> B : SRES2 4 *)
  if sres2X = e1(kinit_b, BD_ADDR_B,au_rand_b) then
  in(dd,x:bitstring);
  let z = sdec(x,kinit_b) in
  if z = s then
  0.


process
  (
    (!processA) | (!processB) | (!processUA) | (!processUB)
    | (!in(hdA,x:bitstring); out(hdAe,x)) (* this replicates
       everything in hdA to hdAe to reproduce a passive attacker
       (simulates eavesdrop on hdA) *)
    | (!in(hdB,x:bitstring); out(hdBe,x)) (* this replicates
       everything in hdB to hdBe to reproduce a passive attacker
       (simulates eavesdrop on hdB) *)
    | (!in(hh,x:bitstring); out(hhe,x)) (* this replicates
       everything in hh to hh to reproduce a passive attacker (
       simulates eavesdrop on hh) *)
  )
```

Appendix B

Proverif Source Code of Bluetooth SSP Ceremonies

B.1 Bluetooth SSP - scenario i

```
(*-----------------------------------------------
 Protocol specification
-----------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)

(* Defining type nonce *)
type nonce.

(* Types and functions for public key cryptography *)
type skey.
type pkey.
fun pk(skey):pkey.

(* Commitment function *)
fun f1(pkey,pkey,nonce,bitstring):bitstring.
```

```
(* Confirmation value *)
fun g(pkey,pkey,nonce,nonce):bitstring.

(* zero *)
const zero:bitstring.

(* Queries and events *)
event beginAparam(pkey).
event beginBparam(pkey).
event endAparam(pkey).
event endBparam(pkey).
event beginAConfirm(pkey,pkey,nonce,nonce).
event beginBConfirm(pkey,pkey,nonce,nonce).
event endAConfirm(pkey,pkey,nonce,nonce).
event endBConfirm(pkey,pkey,nonce,nonce).

query pkeyX:pkey; event (endAparam(pkeyX)) ==> event (beginAparam
   (pkeyX)).
query pkeyX:pkey; event (endBparam(pkeyX)) ==> event (beginBparam
   (pkeyX)).
query pkeyX:pkey; inj-event (endAparam(pkeyX)) ==> inj-event (
   beginAparam(pkeyX)).
query pkeyX:pkey; inj-event (endBparam(pkeyX)) ==> inj-event (
   beginBparam(pkeyX)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event (
   endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event (endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event(
   endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event(endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).

(* Device A *)
let processA(pkA:pkey, skA:skey) =
  out(dd, pkA); (* msg 1 ... A  -DD->  B : PKa *)
  in(dd, pkBX :pkey); (* msg 2 ... B -DD-> A : PKb *)
  event beginBparam(pkBX);
  new Na :nonce;
  in(dd, CbX :bitstring); (* msg 3 ... B -DD-> A : Cb = f1(PKb,
     PKa',Nb,0) *)
  out(dd, Na); (* msg 4 ... A -DD-> B : Na *)
  in(dd, NbX :nonce); (* msg 5 ... B -DD-> A : Nb *)
  event beginBConfirm(pkA, pkBX, Na, NbX);
  if CbX = f1(pkBX, pkA, NbX, zero) then (* checking commitment
     value *)
  let Va = g(pkA, pkBX, Na, NbX) in
  new okA : nonce;
  out(hdA, (Va,okA)); (* msg 6 ... A  -HD->  UA : Va *)
  in(hdA,=okA); (* msg 10 ... UA -HD-> A : ok *)
  event endAparam(pkA);
  event endAConfirm(pkA,pkBX,Na,NbX);
  0.

(* Device B *)
let processB(pkB:pkey, skB:skey) =
  in(dd, pkAX :pkey); (* msg 1 ... A  -DD->  B : PKa *)
  event beginAparam(pkAX);
  out(dd, pkB); (* msg 2 ... B  -DD->  A : PKb *)
  new Nb : nonce;
  let Cb=f1(pkB, pkAX, Nb, zero) in
  out(dd, Cb); (* msg 3 ... B  -DD->  A : Cb = f1(PKb,PKa',Nb,0)
     *)
  in(dd, NaX :nonce); (* msg 4 ... A  -DD->  B : Na *)
  event beginAConfirm(pkAX, pkB, NaX, Nb);
  out(dd,Nb); (* msg 5 ... B  -DD->  A : Nb *)
  let Vb = g(pkAX, pkB, NaX, Nb) in
  new okB : nonce;
  out(hdB, (Vb,okB)); (* msg 7 ... B  -HD->  UB : Vb *)
  in(hdB,=okB); (* msg 11 ... UB -HD-> B : ok *)
  event endBparam(pkB);
  event endBConfirm(pkAX,pkB,NaX,Nb);
  0.

(* User A *)
let processUA =
  in(hdA, (VaX : bitstring, okAX : nonce));
  out(hh, (VaX,okAX));
  in(hh, (VbX: bitstring,=okAX,okBX:nonce));
  out(hh,okBX);
  if VbX = VaX then
  out(hdA, okAX);
  0.

(* User B *)
let processUB =
  in(hdB, (VbX : bitstring, okBX : nonce));
  in(hh, (VaX :bitstring, okAX :nonce));
  out(hh, (VbX,okAX,okBX));
  in(hh,=okBX);
  if VaX = VbX then
  out(hdB, okBX);
  0.

process
  new skA:skey;
  new skB:skey;
  ( (!processA(pk(skA),skA)) | (!processB(pk(skB),skB)) |
  (!processUA) | (!processUB) )
```

B.2 Bluetooth SSP - scenario ii

```
(*---------------------------------------------
 Protocol specification
---------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdAe : channel. (* human-device channel (eavesdrop) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hdBe : channel. (* human-device channel (eavesdrop) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)
free hhe : channel. (* human-human channel (eavesdrop) *)


(* Defining type nonce *)
type nonce.

(* Types and functions for public key cryptography *)
type skey.
type pkey.
fun pk(skey):pkey.

(* Commitment function *)
fun f1(pkey,pkey,nonce,bitstring):bitstring.

(* Confirmation value *)
fun g(pkey,pkey,nonce,nonce):bitstring.

(* zero *)
const zero:bitstring.

(* Queries and events *)
event beginAparam(pkey).
event beginBparam(pkey).
event endAparam(pkey).
event endBparam(pkey).
event beginAConfirm(pkey,pkey,nonce,nonce).
event beginBConfirm(pkey,pkey,nonce,nonce).
event endAConfirm(pkey,pkey,nonce,nonce).
event endBConfirm(pkey,pkey,nonce,nonce).

query pkeyX:pkey; event (endAparam(pkeyX)) ==> event (beginAparam
    (pkeyX)).
query pkeyX:pkey; event (endBparam(pkeyX)) ==> event (beginBparam
    (pkeyX)).
query pkeyX:pkey; inj-event (endAparam(pkeyX)) ==> inj-event (
    beginAparam(pkeyX)).
query pkeyX:pkey; inj-event (endBparam(pkeyX)) ==> inj-event (
    beginBparam(pkeyX)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event (
    endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
    beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
    event (endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
    beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event(
    endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
    beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
    event(endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
    beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).

(* Device A *)
let processA(pkA:pkey, skA:skey) =
  out(dd, pkA); (* msg 1 ... A  -DD->  B : PKa *)
  in(dd, pkBX :pkey); (* msg 2 ... B -DD-> A : PKb *)
  event beginBparam(pkBX);
  new Na :nonce;
  in(dd, CbX :bitstring); (* msg 3 ... B -DD-> A : Cb = f1(PKb,
     PKa',Nb,0) *)
  out(dd, Na); (* msg 4 ... A -DD-> B : Na *)
  in(dd, NbX :nonce); (* msg 5 ... B -DD-> A : Nb *)
  event beginBConfirm(pkA, pkBX, Na, NbX);
  if CbX = f1(pkBX, pkA, NbX, zero) then (* checking commitment
     value *)
  let Va = g(pkA, pkBX, Na, NbX) in
  new okA : nonce;
  out(hdA, (Va,okA)); (* msg 6 ... A  -HD->  UA : Va *)
  in(hdA,=okA); (* msg 10 ... UA -HD-> A : ok *)
  event endAparam(pkA);
  event endAConfirm(pkA,pkBX,Na,NbX);
  0.

(* Device B *)
let processB(pkB:pkey, skB:skey) =
  in(dd, pkAX :pkey); (* msg 1 ... A  -DD->  B : PKa *)
  event beginAparam(pkAX);
  out(dd, pkB); (* msg 2 ... B  -DD->  A : PKb *)
  new Nb : nonce;
  let Cb=f1(pkB, pkAX, Nb, zero) in
  out(dd, Cb); (* msg 3 ... B  -DD->  A : Cb = f1(PKb,PKa',Nb,0)
     *)
  in(dd, NaX :nonce); (* msg 4 ... A  -DD->  B : Na *)
  event beginAConfirm(pkAX, pkB, NaX, Nb);
  out(dd,Nb); (* msg 5 ... B  -DD->  A : Nb *)
  let Vb = g(pkAX, pkB, NaX, Nb) in
  new okB : nonce;
  out(hdB, (Vb,okB)); (* msg 7 ... B  -HD->  UB : Vb *)
  in(hdB,=okB); (* msg 11 ... UB -HD-> B : ok *)
  event endBparam(pkB);
  event endBConfirm(pkAX,pkB,NaX,Nb);
  0.

(* User A *)
let processUA =
  in(hdA, (VaX : bitstring, okAX : nonce));
  out(hh, (VaX,okAX));
  in(hh, (VbX: bitstring,=okAX,okBX:nonce));
  out(hh,okBX);
  if VbX = VaX then
  out(hdA, okAX);
  0.

(* User B *)
let processUB =
  in(hdB, (VbX : bitstring, okBX : nonce));
  in(hh, (VaX :bitstring, okAX :nonce));
  out(hh, (VbX,okAX,okBX));
  in(hh,=okBX);
  if VaX = VbX then
  out(hdB, okBX);
  0.

process
  new skA:skey;
  new skB:skey;
  (
    (!processA(pk(skA),skA)) |
    (!processB(pk(skB),skB)) |
    (!processUA) |
    (!processUB) |
    (!in(hdA,x:bitstring); out(hdAe,x)) | (* this replicates
       everything in hdA to hdAe to reproduce a passive attacker
       *)
    (!in(hdB,x:bitstring); out(hdBe,x)) | (* this replicates
       everything in hdB to hdBe to reproduce a passive attacker
       *)
    (!in(hh,x:bitstring); out(hhe,x))  (* this replicates
       everything in hh to hh to reproduce a passive attacker *)
  )
```

B.3 Bluetooth SSP - scenario iii

```
(*-----------------------------------------------
 Protocol specification
-----------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdAe : channel. (* human-device channel (eavesdrop) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hdBe : channel. (* human-device channel (eavesdrop) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)
free hhe : channel. (* human-human channel (eavesdrop) *)


(* Defining type nonce *)
type nonce.

(* Types and functions for public key cryptography *)
type skey.
type pkey.
fun pk(skey):pkey.

(* Commitment function *)
fun f1(pkey,pkey,nonce,bitstring):bitstring.

(* Confirmation value *)
fun g(pkey,pkey,nonce,nonce):bitstring.

(* zero *)
const zero:bitstring.

(* Queries and events *)
event beginAparam(pkey).
event beginBparam(pkey).
event endAparam(pkey).
event endBparam(pkey).
event beginAConfirm(pkey,pkey,nonce,nonce).
event beginBConfirm(pkey,pkey,nonce,nonce).
event endAConfirm(pkey,pkey,nonce,nonce).
event endBConfirm(pkey,pkey,nonce,nonce).

query pkeyX:pkey; event (endAparam(pkeyX)) ==> event (beginAparam
   (pkeyX)).
query pkeyX:pkey; event (endBparam(pkeyX)) ==> event (beginBparam
   (pkeyX)).
query pkeyX:pkey; inj-event (endAparam(pkeyX)) ==> inj-event (
   beginAparam(pkeyX)).
query pkeyX:pkey; inj-event (endBparam(pkeyX)) ==> inj-event (
   beginBparam(pkeyX)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event (
   endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event (endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event(
   endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event(endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).

(* Device A *)
let processA(pkA:pkey, skA:skey) =
  out(dd, pkA); (* msg 1 ... A  -DD->  B : PKa *)
  in(dd, pkBX :pkey); (* msg 2 ... B -DD-> A : PKb *)
  event beginBparam(pkBX);
  new Na :nonce;
  in(dd, CbX :bitstring); (* msg 3 ... B -DD-> A : Cb = f1(PKb,
     PKa',Nb,0) *)
  out(dd, Na); (* msg 4 ... A -DD-> B : Na *)
  in(dd, NbX :nonce); (* msg 5 ... B -DD-> A : Nb *)
  event beginBConfirm(pkA, pkBX, Na, NbX);
  if CbX = f1(pkBX, pkA, NbX, zero) then (* checking commitment
     value *)
  let Va = g(pkA, pkBX, Na, NbX) in
  new okA : nonce;
  out(hdA, (Va,okA)); (* msg 6 ... A  -HD->  UA : Va *)
  in(hdA,=okA); (* msg 10 ... UA -HD-> A : ok *)
  event endAparam(pkA);
  event endAConfirm(pkA,pkBX,Na,NbX);
  0.

(* Device B *)
let processB(pkB:pkey, skB:skey) =
  in(dd, pkAX :pkey); (* msg 1 ... A  -DD->  B : PKa *)
  event beginAparam(pkAX);
  out(dd, pkB); (* msg 2 ... B  -DD->  A : PKb *)
  new Nb : nonce;
  let Cb=f1(pkB, pkAX, Nb, zero) in
  out(dd, Cb); (* msg 3 ... B  -DD->  A : Cb = f1(PKb,PKa',Nb,0)
     *)
  in(dd, NaX :nonce); (* msg 4 ... A  -DD->  B : Na *)
  event beginAConfirm(pkAX, pkB, NaX, Nb);
  out(dd,Nb); (* msg 5 ... B  -DD->  A : Nb *)
  let Vb = g(pkAX, pkB, NaX, Nb) in
  new okB : nonce;
  out(hdB, (Vb,okB)); (* msg 7 ... B  -HD->  UB : Vb *)
  in(hdB,=okB); (* msg 11 ... UB -HD-> B : ok *)
  event endBparam(pkB);
  event endBConfirm(pkAX,pkB,NaX,Nb);
  0.

(* User A *)
let processUA =
  in(hdA, (VaX : bitstring, okAX : nonce));
  out(hh, (VaX,okAX));
  in(hh, (VbX: bitstring,=okAX,okBX:nonce));
  out(hh,okBX);
  (* if VbX = VaX then *) (* User now doesn't compare the numbers
      and send Ok anyway *)
  out(hdA, okAX);
  0.

(* User B *)
let processUB =
  in(hdB, (VbX : bitstring, okBX : nonce));
  in(hh, (VaX :bitstring, okAX :nonce));
  out(hh, (VbX,okAX,okBX));
  in(hh,=okBX);
  if VaX = VbX then
  out(hdB, okBX);
  0.

process
  new skA:skey;
  new skB:skey;
  (
    (!processA(pk(skA),skA)) |
    (!processB(pk(skB),skB)) |
    (!processUA) |
    (!processUB) |
    (!in(hdA,x:bitstring); out(hdAe,x)) | (* this replicates
       everything in hdA to hdAe to reproduce a passive attacker
       *)
    (!in(hdB,x:bitstring); out(hdBe,x)) | (* this replicates
       everything in hdB to hdBe to reproduce a passive attacker
       *)
    (!in(hh,x:bitstring); out(hhe,x))  (* this replicates
       everything in hh to hh to reproduce a passive attacker *)
  )
```

B.4 Bluetooth SSP - scenario iv

```
(*------------------------------------------------
 Protocol specification
------------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdAe : channel. (* human-device channel (eavesdrop) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hdBe : channel. (* human-device channel (eavesdrop) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)
free hhe : channel. (* human-human channel (eavesdrop *)


(* Defining type nonce *)
type nonce.

(* Types and functions for public key cryptography *)
type skey.
type pkey.
fun pk(skey):pkey.

(* Commitment function *)
fun f1(pkey,pkey,nonce,bitstring):bitstring.

(* Confirmation value *)
fun g(pkey,pkey,nonce,nonce):bitstring.

(* zero *)
const zero:bitstring.

(* Queries and events *)
event beginAparam(pkey).
event beginBparam(pkey).
event endAparam(pkey).
event endBparam(pkey).
event beginAConfirm(pkey,pkey,nonce,nonce).
event beginBConfirm(pkey,pkey,nonce,nonce).
event endAConfirm(pkey,pkey,nonce,nonce).
event endBConfirm(pkey,pkey,nonce,nonce).

query pkeyX:pkey; event (endAparam(pkeyX)) ==> event (beginAparam
   (pkeyX)).
query pkeyX:pkey; event (endBparam(pkeyX)) ==> event (beginBparam
   (pkeyX)).
query pkeyX:pkey; inj-event (endAparam(pkeyX)) ==> inj-event (
   beginAparam(pkeyX)).
query pkeyX:pkey; inj-event (endBparam(pkeyX)) ==> inj-event (
   beginBparam(pkeyX)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event (
   endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event (endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event(
   endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event(endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).

(* Device A *)
let processA(pkA:pkey, skA:skey) =
  out(dd, pkA); (* msg 1 ... A  -DD->  B : PKa *)
  in(dd, pkBX :pkey); (* msg 2 ... B -DD-> A : PKb *)
  event beginBparam(pkBX);
  new Na :nonce;
  in(dd, CbX :bitstring); (* msg 3 ... B -DD-> A : Cb = f1(PKb,
     PKa',Nb,0) *)
  out(dd, Na); (* msg 4 ... A -DD-> B : Na *)
  in(dd, NbX :nonce); (* msg 5 ... B -DD-> A : Nb *)
  event beginBConfirm(pkA, pkBX, Na, NbX);
  if CbX = f1(pkBX, pkA, NbX, zero) then (* checking commitment
     value *)
  let Va = g(pkA, pkBX, Na, NbX) in
  new okA : nonce;
  out(hdA, (Va,okA)); (* msg 6 ... A  -HD->  UA : Va *)
  in(hdA,=okA); (* msg 10 ... UA -HD-> A : ok *)
  event endAparam(pkA);
  event endAConfirm(pkA,pkBX,Na,NbX);
  0.

(* Device B *)
let processB(pkB:pkey, skB:skey) =
  in(dd, pkAX :pkey); (* msg 1 ... A  -DD->  B : PKa *)
  event beginAparam(pkAX);
  out(dd, pkB); (* msg 2 ... B  -DD->  A : PKb *)
  new Nb : nonce;
  let Cb=f1(pkB, pkAX, Nb, zero) in
  out(dd, Cb); (* msg 3 ... B  -DD->  A : Cb = f1(PKb,PKa',Nb,0)
     *)
  in(dd, NaX :nonce); (* msg 4 ... A  -DD->  B : Na *)
  event beginAConfirm(pkAX, pkB, NaX, Nb);
  out(dd,Nb); (* msg 5 ... B  -DD->  A : Nb *)
  let Vb = g(pkAX, pkB, NaX, Nb) in
  new okB : nonce;
  out(hdB, (Vb,okB)); (* msg 7 ... B  -HD->  UB : Vb *)
  in(hdB,=okB); (* msg 11 ... UB -HD-> B : ok *)
  event endBparam(pkB);
  event endBConfirm(pkAX,pkB,NaX,Nb);
  0.

(* User A *)
let processUA =
  in(hdA, (VaX : bitstring, okAX : nonce));
  out(hh, (VaX,okAX));
  in(hh, (VbX: bitstring,=okAX,okBX:nonce));
  out(hh,okBX);
  if VbX = VaX then
  out(hdA, okAX);
  0.

(* User B *)
let processUB =
  in(hdB, (VbX : bitstring, okBX : nonce));
  in(hh, (VaX :bitstring, okAX :nonce));
  out(hh, (VbX,okAX,okBX));
  in(hh,=okBX);
  (* if VaX = VbX then *) (* User now doesn't compare the numbers
      and send Ok anyway *)
  out(hdB, okBX);
  0.

process
  new skA:skey;
  new skB:skey;
  (
    (!processA(pk(skA),skA)) |
    (!processB(pk(skB),skB)) |
    (!processUA) |
    (!processUB) |
    (!in(hdA,x:bitstring); out(hdAe,x)) | (* this replicates
       everything in hdA to hdAe to reproduce a passive attacker
       *)
    (!in(hdB,x:bitstring); out(hdBe,x)) | (* this replicates
       everything in hdB to hdBe to reproduce a passive attacker
       *)
    (!in(hh,x:bitstring); out(hhe,x))  (* this replicates
       everything in hh to hh to reproduce a passive attacker *)
  )
```

B.5 Bluetooth SSP - scenario iv

```
(*-------------------------------------------------
 Protocol specification
-----------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdAe : channel. (* human-device channel (eavesdrop) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hdBe : channel. (* human-device channel (eavesdrop) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)
free hhe : channel. (* human-human channel (eavesdrop *)


(* Defining type nonce *)
type nonce.

(* Types and functions for public key cryptography *)
type skey.
type pkey.
fun pk(skey):pkey.

(* Commitment function *)
fun f1(pkey,pkey,nonce,bitstring):bitstring.

(* Confirmation value *)
fun g(pkey,pkey,nonce,nonce):bitstring.

(* zero *)
const zero:bitstring.

(* Queries and events *)
event beginAparam(pkey).
event beginBparam(pkey).
event endAparam(pkey).
event endBparam(pkey).
event beginAConfirm(pkey,pkey,nonce,nonce).
event beginBConfirm(pkey,pkey,nonce,nonce).
event endAConfirm(pkey,pkey,nonce,nonce).
event endBConfirm(pkey,pkey,nonce,nonce).

query pkeyX:pkey; event (endAparam(pkeyX)) ==> event (beginAparam
   (pkeyX)).
query pkeyX:pkey; event (endBparam(pkeyX)) ==> event (beginBparam
   (pkeyX)).
query pkeyX:pkey; inj-event (endAparam(pkeyX)) ==> inj-event (
   beginAparam(pkeyX)).
query pkeyX:pkey; inj-event (endBparam(pkeyX)) ==> inj-event (
   beginBparam(pkeyX)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event (
   endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event (endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event(
   endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event(endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).

(* Device A *)
let processA(pkA:pkey, skA:skey) =
  out(dd, pkA); (* msg 1 ... A  -DD->  B : PKa *)
  in(dd, pkBX :pkey); (* msg 2 ... B -DD-> A : PKb *)
  event beginBparam(pkBX);
  new Na :nonce;
  in(dd, CbX :bitstring); (* msg 3 ... B -DD-> A : Cb = f1(PKb,
     PKa',Nb,0) *)
  out(dd, Na); (* msg 4 ... A -DD-> B : Na *)
  in(dd, NbX :nonce); (* msg 5 ... B -DD-> A : Nb *)
  event beginBConfirm(pkA, pkBX, Na, NbX);
  if CbX = f1(pkBX, pkA, NbX, zero) then (* checking commitment
     value *)
  let Va = g(pkA, pkBX, Na, NbX) in
  new okA : nonce;
  out(hdA, (Va,okA)); (* msg 6 ... A  -HD->  UA : Va *)
  in(hdA,=okA); (* msg 10 ... UA -HD-> A : ok *)
  event endAparam(pkA);
  event endAConfirm(pkA,pkBX,Na,NbX);
  0.

(* Device B *)
let processB(pkB:pkey, skB:skey) =
  in(dd, pkAX :pkey); (* msg 1 ... A  -DD->  B : PKa *)
  event beginAparam(pkAX);
  out(dd, pkB); (* msg 2 ... B  -DD->  A : PKb *)
  new Nb : nonce;
  let Cb=f1(pkB, pkAX, Nb, zero) in
  out(dd, Cb); (* msg 3 ... B  -DD->  A : Cb = f1(PKb,PKa',Nb,0)
     *)
  in(dd, NaX :nonce); (* msg 4 ... A  -DD->  B : Na *)
  event beginAConfirm(pkAX, pkB, NaX, Nb);
  out(dd,Nb); (* msg 5 ... B  -DD->  A : Nb *)
  let Vb = g(pkAX, pkB, NaX, Nb) in
  new okB : nonce;
  out(hdB, (Vb,okB)); (* msg 7 ... B  -HD->  UB : Vb *)
  in(hdB,=okB); (* msg 11 ... UB -HD-> B : ok *)
  event endBparam(pkB);
  event endBConfirm(pkAX,pkB,NaX,Nb);
  0.

(* User A *)
let processUA =
  in(hdA, (VaX : bitstring, okAX : nonce));
  out(hh, (VaX,okAX));
  in(hh, (VbX: bitstring,=okAX,okBX:nonce));
  out(hh,okBX);
  (* if VbX = VaX then *) (* User now doesn't compare the numbers
      and send Ok anyway *)
  out(hdA, okAX);
  0.

(* User B *)
let processUB =
  in(hdB, (VbX : bitstring, okBX : nonce));
  in(hh, (VaX :bitstring, okAX :nonce));
  out(hh, (VbX,okAX,okBX));
  in(hh,=okBX);
  (* if VaX = VbX then *) (* User now doesn't compare the numbers
      and send Ok anyway *)
  out(hdB, okBX);
  0.

process
  new skA:skey;
  new skB:skey;
  (
    (!processA(pk(skA),skA)) |
    (!processB(pk(skB),skB)) |
    (!processUA) |
    (!processUB) |
    (!in(hdA,x:bitstring); out(hdAe,x)) | (* this replicates
       everything in hdA to hdAe to reproduce a passive attacker
       *)
    (!in(hdB,x:bitstring); out(hdBe,x)) | (* this replicates
       everything in hdB to hdBe to reproduce a passive attacker
       *)
    (!in(hh,x:bitstring); out(hhe,x))  (* this replicates
       everything in hh to hh to reproduce a passive attacker *)
  )
```

B.6 Bluetooth SSP - scenario vi

```
(*-----------------------------------------------
 Protocol specification
-----------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel. (* human-device channel (no attackers) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hdBe : channel. (* human-device channel (eavesdrop) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)
free hhe : channel. (* human-human channel (eavesdrop) *)


(* Defining type nonce *)
type nonce.

(* Types and functions for public key cryptography *)
type skey.
type pkey.
fun pk(skey):pkey.

(* Commitment function *)
fun f1(pkey,pkey,nonce,bitstring):bitstring.

(* Confirmation value *)
fun g(pkey,pkey,nonce,nonce):bitstring.

(* zero *)
const zero:bitstring.

(* Queries and events *)
event beginAparam(pkey).
event beginBparam(pkey).
event endAparam(pkey).
event endBparam(pkey).
event beginAConfirm(pkey,pkey,nonce,nonce).
event beginBConfirm(pkey,pkey,nonce,nonce).
event endAConfirm(pkey,pkey,nonce,nonce).
event endBConfirm(pkey,pkey,nonce,nonce).

query pkeyX:pkey; event (endAparam(pkeyX)) ==> event (beginAparam
   (pkeyX)).
query pkeyX:pkey; event (endBparam(pkeyX)) ==> event (beginBparam
   (pkeyX)).
query pkeyX:pkey; inj-event (endAparam(pkeyX)) ==> inj-event (
   beginAparam(pkeyX)).
query pkeyX:pkey; inj-event (endBparam(pkeyX)) ==> inj-event (
   beginBparam(pkeyX)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event (
   endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event (endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event(
   endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event(endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).

(* Device A *)
let processA(pkA:pkey, skA:skey) =
  out(dd, pkA); (* msg 1 ... A  -DD->  B : PKa *)
  in(dd, pkBX :pkey); (* msg 2 ... B -DD-> A : PKb *)
  event beginBparam(pkBX);
  new Na :nonce;
  in(dd, CbX :bitstring); (* msg 3 ... B -DD-> A : Cb = f1(PKb,
     PKa',Nb,0) *)
  out(dd, Na); (* msg 4 ... A -DD-> B : Na *)
  in(dd, NbX :nonce); (* msg 5 ... B -DD-> A : Nb *)
  event beginBConfirm(pkA, pkBX, Na, NbX);
  if CbX = f1(pkBX, pkA, NbX, zero) then (* checking commitment
     value *)
  let Va = g(pkA, pkBX, Na, NbX) in
  new okA : nonce;
  out(hdA, (Va,okA)); (* msg 6 ... A  -HD->  UA : Va *)
  in(hdA,=okA); (* msg 10 ... UA -HD-> A : ok *)
  event endAparam(pkA);
  event endAConfirm(pkA,pkBX,Na,NbX);
  0.

(* Device B *)
let processB(pkB:pkey, skB:skey) =
  in(dd, pkAX :pkey); (* msg 1 ... A  -DD->  B : PKa *)
  event beginAparam(pkAX);
  out(dd, pkB); (* msg 2 ... B  -DD->  A : PKb *)
  new Nb : nonce;
  let Cb=f1(pkB, pkAX, Nb, zero) in
  out(dd, Cb); (* msg 3 ... B  -DD->  A : Cb = f1(PKb,PKa',Nb,0)
     *)
  in(dd, NaX :nonce); (* msg 4 ... A  -DD->  B : Na *)
  event beginAConfirm(pkAX, pkB, NaX, Nb);
  out(dd,Nb); (* msg 5 ... B  -DD->  A : Nb *)
  let Vb = g(pkAX, pkB, NaX, Nb) in
  new okB : nonce;
  out(hdB, (Vb,okB)); (* msg 7 ... B  -HD->  UB : Vb *)
  in(hdB,=okB); (* msg 11 ... UB -HD-> B : ok *)
  event endBparam(pkB);
  event endBConfirm(pkAX,pkB,NaX,Nb);
  0.

(* User A *)
let processUA =
  in(hdA, (VaX : bitstring, okAX : nonce));
  out(hh, (VaX,okAX));
  in(hh, (VbX: bitstring,=okAX,okBX:nonce));
  out(hh,okBX);
  if VbX = VaX then
  out(hdA, okAX);
  0.

(* User B *)
let processUB =
  in(hdB, (VbX : bitstring, okBX : nonce));
  in(hh, (VaX :bitstring, okAX :nonce));
  out(hh, (VbX,okAX,okBX));
  in(hh,=okBX);
  if VaX = VbX then
  out(hdB, okBX);
  0.
```

```
process
  new skA:skey;
  new skB:skey;
  (
    (!processA(pk(skA),skA)) |
    (!processB(pk(skB),skB)) |
    (!processUA) |
    (!processUB) |
    (!in(hdB,x:bitstring); out(hdBe,x)) | (* this replicates
       everything in hdB to hdBe to reproduce a passive attacker
       *)
    (!in(hh,x:bitstring); out(hhe,x))  (* this replicates
       everything in hh to hh to reproduce a passive attacker *)
  )
```

Appendix C

Proverif Source Code of Bluetooth SSP Ceremonies - Amended Version

C.1 Bluetooth SSP amended version - scenario i

```
(*------------------------------------------------
 Protocol specification
------------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)

(* Defining type nonce *)
type nonce.

(* Types and functions for public key cryptography *)
type skey.
type pkey.
fun pk(skey):pkey.
```

```
(* Commitment function *)
fun f1(pkey,pkey,nonce,bitstring):bitstring.

(* Confirmation value *)
fun g(pkey,pkey,nonce,nonce):bitstring.

(* Functions for truncating Vs *)
fun splitVsP1(bitstring) : bitstring.
fun splitVsP2(bitstring) : bitstring.
reduc forall x: bitstring; mergeVs(splitVsP1(x),splitVsP2(x)) = x
    .

(* zero *)
const zero:bitstring.

(* Queries and events *)
event beginAparam(pkey).
event beginBparam(pkey).
event endAparam(pkey).
event endBparam(pkey).
event beginAConfirm(pkey,pkey,nonce,nonce).
event beginBConfirm(pkey,pkey,nonce,nonce).
event endAConfirm(pkey,pkey,nonce,nonce).
event endBConfirm(pkey,pkey,nonce,nonce).

query pkeyX:pkey; event (endAparam(pkeyX)) ==> event (beginAparam
    (pkeyX)).
query pkeyX:pkey; event (endBparam(pkeyX)) ==> event (beginBparam
    (pkeyX)).
query pkeyX:pkey; inj-event (endAparam(pkeyX)) ==> inj-event (
    beginAparam(pkeyX)).
query pkeyX:pkey; inj-event (endBparam(pkeyX)) ==> inj-event (
    beginBparam(pkeyX)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event (
    endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
    beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
    event (endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
    beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event(
    endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
    beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
```

```
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
    event(endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
    beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).

(* Device A *)
let processA(pkA:pkey, skA:skey) =
  out(dd, pkA); (* msg 1 ... A  -DD->  B : PKa *)
  in(dd, pkBX :pkey); (* msg 2 ... B -DD-> A : PKb *)
  event beginBparam(pkBX);
  new Na :nonce;
  in(dd, CbX :bitstring); (* msg 3 ... B -DD-> A : Cb = f1(PKb,
     PKa',Nb,0) *)
  out(dd, Na); (* msg 4 ... A -DD-> B : Na *)
  in(dd, NbX :nonce); (* msg 5 ... B -DD-> A : Nb *)
  event beginBConfirm(pkA, pkBX, Na, NbX);
  if CbX = f1(pkBX, pkA, NbX, zero) then (* checking commitment
     value *)
  let Va = g(pkA, pkBX, Na, NbX) in
  let part1Va = splitVsP1(Va) in
  out(hdA, part1Va);
  in(hdA, part2VbX :bitstring);
  if Va = mergeVs(part1Va,part2VbX) then
  event endAparam(pkA);
  event endAConfirm(pkA,pkBX,Na,NbX);
  0.

(* Device B *)
let processB(pkB:pkey, skB:skey) =
  in(dd, pkAX :pkey); (* msg 1 ... A  -DD->  B : PKa *)
  event beginAparam(pkAX);
  out(dd, pkB); (* msg 2 ... B  -DD->  A : PKb *)
  new Nb : nonce;
  let Cb=f1(pkB, pkAX, Nb, zero) in
  out(dd, Cb); (* msg 3 ... B  -DD->  A : Cb = f1(PKb,PKa',Nb,0)
     *)
  in(dd, NaX :nonce); (* msg 4 ... A  -DD->  B : Na *)
  event beginAConfirm(pkAX, pkB, NaX, Nb);
  out(dd,Nb); (* msg 5 ... B  -DD->  A : Nb *)
  let Vb = g(pkAX, pkB, NaX, Nb) in
  let part2Vb = splitVsP2(Vb) in
  out(hdB, part2Vb);
  in(hdB, part1VaX :bitstring);
  if Vb = mergeVs(part1VaX,part2Vb) then
  event endBparam(pkB);
  event endBConfirm(pkAX,pkB,NaX,Nb);
  0.

(* User A *)
let processUA =
  in(hdA, (trVa1 : bitstring));
  out(hh, trVa1);
  in(hh, trVb2 :bitstring);
  out(hdA, trVb2);
  0.

(* User B *)
let processUB =
  in(hdB, (trVb2 : bitstring));
  out(hh, trVb2);
  in(hh, trVa1 : bitstring);
  out(hdB, trVa1);
  0.

process
  new skA:skey;
  new skB:skey;
  (
    (!processA(pk(skA),skA)) |
    (!processB(pk(skB),skB)) |
    (!processUA) |
    (!processUB)
  )
```

C.2 Bluetooth SSP amended version - scenario ii

```
(*-----------------------------------------------
 Protocol specification
-----------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdAe : channel. (* human-device channel (eavesdrop) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hdBe : channel. (* human-device channel (eavesdrop) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)
free hhe : channel. (* human-human channel (eavesdrop) *)


(* Defining type nonce *)
type nonce.

(* Types and functions for public key cryptography *)
type skey.
type pkey.
fun pk(skey):pkey.

(* Commitment function *)
fun f1(pkey,pkey,nonce,bitstring):bitstring.

(* Confirmation value *)
fun g(pkey,pkey,nonce,nonce):bitstring.

(* Functions for truncating Vs *)
fun splitVsP1(bitstring) : bitstring.
fun splitVsP2(bitstring) : bitstring.
reduc forall x: bitstring; mergeVs(splitVsP1(x),splitVsP2(x)) = x
   .

(* zero *)
const zero:bitstring.
```

```
(* Queries and events *)
event beginAparam(pkey).
event beginBparam(pkey).
event endAparam(pkey).
event endBparam(pkey).
event beginAConfirm(pkey,pkey,nonce,nonce).
event beginBConfirm(pkey,pkey,nonce,nonce).
event endAConfirm(pkey,pkey,nonce,nonce).
event endBConfirm(pkey,pkey,nonce,nonce).

query pkeyX:pkey; event (endAparam(pkeyX)) ==> event (beginAparam
   (pkeyX)).
query pkeyX:pkey; event (endBparam(pkeyX)) ==> event (beginBparam
   (pkeyX)).
query pkeyX:pkey; inj-event (endAparam(pkeyX)) ==> inj-event (
   beginAparam(pkeyX)).
query pkeyX:pkey; inj-event (endBparam(pkeyX)) ==> inj-event (
   beginBparam(pkeyX)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event (
   endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event (endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event(
   endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event(endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).

(* Device A *)
let processA(pkA:pkey, skA:skey) =
  out(dd, pkA); (* msg 1 ... A  -DD->  B : PKa *)
  in(dd, pkBX :pkey); (* msg 2 ... B -DD-> A : PKb *)
  event beginBparam(pkBX);
  new Na :nonce;
  in(dd, CbX :bitstring); (* msg 3 ... B -DD-> A : Cb = f1(PKb,
     PKa',Nb,0) *)
  out(dd, Na); (* msg 4 ... A -DD-> B : Na *)
  in(dd, NbX :nonce); (* msg 5 ... B -DD-> A : Nb *)
  event beginBConfirm(pkA, pkBX, Na, NbX);
  if CbX = f1(pkBX, pkA, NbX, zero) then (* checking commitment
     value *)
  let Va = g(pkA, pkBX, Na, NbX) in
  let part1Va = splitVsP1(Va) in
  out(hdA, part1Va);
  in(hdA, part2VbX :bitstring);
  if Va = mergeVs(part1Va,part2VbX) then
  event endAparam(pkA);
  event endAConfirm(pkA,pkBX,Na,NbX);
  0.

(* Device B *)
let processB(pkB:pkey, skB:skey) =
  in(dd, pkAX :pkey); (* msg 1 ... A  -DD->  B : PKa *)
  event beginAparam(pkAX);
  out(dd, pkB); (* msg 2 ... B  -DD->  A : PKb *)
  new Nb : nonce;
  let Cb=f1(pkB, pkAX, Nb, zero) in
  out(dd, Cb); (* msg 3 ... B  -DD->  A : Cb = f1(PKb,PKa',Nb,0)
     *)
  in(dd, NaX :nonce); (* msg 4 ... A  -DD->  B : Na *)
  event beginAConfirm(pkAX, pkB, NaX, Nb);
  out(dd,Nb); (* msg 5 ... B  -DD->  A : Nb *)
  let Vb = g(pkAX, pkB, NaX, Nb) in
  let part2Vb = splitVsP2(Vb) in
  out(hdB, part2Vb);
  in(hdB, part1VaX :bitstring);
  if Vb = mergeVs(part1VaX,part2Vb) then
  event endBparam(pkB);
  event endBConfirm(pkAX,pkB,NaX,Nb);
  0.

(* User A *)
let processUA =
  in(hdA, (trVa1 : bitstring));
  out(hh, trVa1);
  in(hh, trVb2 :bitstring);
  out(hdA, trVb2);
  0.

(* User B *)
let processUB =
  in(hdB, (trVb2 : bitstring));
  out(hh, trVb2);
  in(hh, trVa1 : bitstring);
  out(hdB, trVa1);
  0.

process
  new skA:skey;
  new skB:skey;
  (
    (!processA(pk(skA),skA)) |
    (!processB(pk(skB),skB)) |
    (!processUA) |
    (!processUB) |
    (!in(hdA,x:bitstring); out(hdAe,x)) | (* this replicates
       everything in hdA to hdAe to reproduce a passive attacker
       *)
    (!in(hdB,x:bitstring); out(hdBe,x)) | (* this replicates
       everything in hdB to hdBe to reproduce a passive attacker
       *)
    (!in(hh,x:bitstring); out(hhe,x))  (* this replicates
       everything in hh to hh to reproduce a passive attacker *)
  )
```

C.3 Bluetooth SSP amended version - scenario vi

```
(*-----------------------------------------------
 Protocol specification
-----------------------------------------------*)

(* Channels *)
free dd : channel. (* device-device channel *)
free hdA : channel. (* human-device channel (no attackers) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hdBe : channel. (* human-device channel (eavesdrop) *)
free hh : channel [private]. (* human-human channel (no attackers
   ) *)
free hhe : channel. (* human-human channel (eavesdrop) *)

(* Defining type nonce *)
type nonce.

(* Types and functions for public key cryptography *)
type skey.
type pkey.
fun pk(skey):pkey.

(* Commitment function *)
fun f1(pkey,pkey,nonce,bitstring):bitstring.

(* Confirmation value *)
fun g(pkey,pkey,nonce,nonce):bitstring.

(* Functions for truncating Vs *)
fun splitVsP1(bitstring) : bitstring.
fun splitVsP2(bitstring) : bitstring.
reduc forall x: bitstring; mergeVs(splitVsP1(x),splitVsP2(x)) = x
   .

(* zero *)
const zero:bitstring.

(* Queries and events *)
event beginAparam(pkey).
event beginBparam(pkey).
event endAparam(pkey).
event endBparam(pkey).
event beginAConfirm(pkey,pkey,nonce,nonce).
event beginBConfirm(pkey,pkey,nonce,nonce).
event endAConfirm(pkey,pkey,nonce,nonce).
event endBConfirm(pkey,pkey,nonce,nonce).

query pkeyX:pkey; event (endAparam(pkeyX)) ==> event (beginAparam
   (pkeyX)).
query pkeyX:pkey; event (endBparam(pkeyX)) ==> event (beginBparam
   (pkeyX)).
query pkeyX:pkey; inj-event (endAparam(pkeyX)) ==> inj-event (
   beginAparam(pkeyX)).
query pkeyX:pkey; inj-event (endBparam(pkeyX)) ==> inj-event (
   beginBparam(pkeyX)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event (
   endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event (endAConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginAConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; event(
   endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).
query pkeyW:pkey, pkeyX:pkey, nonceY:nonce, nonceZ:nonce; inj-
   event(endBConfirm(pkeyW,pkeyX,nonceY,nonceZ)) ==> inj-event(
   beginBConfirm(pkeyW,pkeyX,nonceY,nonceZ)).

(* Device A *)
let processA(pkA:pkey, skA:skey) =
  out(dd, pkA); (* msg 1 ... A  -DD->  B : PKa *)
  in(dd, pkBX :pkey); (* msg 2 ... B -DD-> A : PKb *)
  event beginBparam(pkBX);
  new Na :nonce;
  in(dd, CbX :bitstring); (* msg 3 ... B -DD-> A : Cb = f1(PKb,
     PKa',Nb,0) *)
  out(dd, Na); (* msg 4 ... A -DD-> B : Na *)
  in(dd, NbX :nonce); (* msg 5 ... B -DD-> A : Nb *)
  event beginBConfirm(pkA, pkBX, Na, NbX);
  if CbX = f1(pkBX, pkA, NbX, zero) then (* checking commitment
     value *)
  let Va = g(pkA, pkBX, Na, NbX) in
  let part1Va = splitVsP1(Va) in
  out(hdA, part1Va);
  in(hdA, part2VbX :bitstring);
  if Va = mergeVs(part1Va,part2VbX) then
  event endAparam(pkA);
  event endAConfirm(pkA,pkBX,Na,NbX);
  0.

(* Device B *)
let processB(pkB:pkey, skB:skey) =
  in(dd, pkAX :pkey); (* msg 1 ... A  -DD->  B : PKa *)
  event beginAparam(pkAX);
  out(dd, pkB); (* msg 2 ... B  -DD->  A : PKb *)
  new Nb : nonce;
  let Cb=f1(pkB, pkAX, Nb, zero) in
  out(dd, Cb); (* msg 3 ... B  -DD->  A : Cb = f1(PKb,PKa',Nb,0)
     *)
  in(dd, NaX :nonce); (* msg 4 ... A  -DD->  B : Na *)
  event beginAConfirm(pkAX, pkB, NaX, Nb);
  out(dd,Nb); (* msg 5 ... B  -DD->  A : Nb *)
  let Vb = g(pkAX, pkB, NaX, Nb) in
  let part2Vb = splitVsP2(Vb) in
  out(hdB, part2Vb);
  in(hdB, part1VaX :bitstring);
  if Vb = mergeVs(part1VaX,part2Vb) then
  event endBparam(pkB);
  event endBConfirm(pkAX,pkB,NaX,Nb);
  0.

(* User A *)
let processUA =
  in(hdA, (trVa1 : bitstring));
  out(hh, trVa1);
  in(hh, trVb2 :bitstring);
  out(hdA, trVb2);
  0.

(* User B *)
let processUB =
  in(hdB, (trVb2 : bitstring));
  out(hh, trVb2);
  in(hh, trVa1 : bitstring);
  out(hdB, trVa1);
  0.

process
  new skA:skey;
  new skB:skey;
  (
    (!processA(pk(skA),skA)) |
    (!processB(pk(skB),skB)) |
    (!processUA) |
    (!processUB) |
    (!in(hdB,x:bitstring); out(hdBe,x)) | (* this replicates
       everything in hdB to hdBe to reproduce a passive attacker
       *)
    (!in(hh,x:bitstring); out(hhe,x))  (* this replicates
       everything in hh to hh to reproduce a passive attacker *)
  )
```

Appendix D

Proverif Source Code of WhatsApp Ceremonies

D.1 WhatsApp Registration Ceremony - scenario i

```
(*-----------------------------------------------
 Protocol specification
-----------------------------------------------*)
(* Channels *)
free dd  : channel. (* device-device channel *)
free hd : channel [private]. (* human-device channel (no
   attackers) *)
free hde : channel. (* human-device channel (no attackers) *)

(* Defining type nonce *)
type nonce.

(* Public Phone Number*)
free Num : bitstring.

(* Queries and events *)
event beginSparam(bitstring).
event endSparam(bitstring).

query NumX : bitstring; event (endSparam(NumX)) ==> event (
   beginSparam(NumX)).


(* User A *)
```

```
let processUA =
    out(hd, Num); (* msg 1 ... Ua -HD-> A : Num *)
    0.

(* Agent A - WhatsApp *)
let processA =
    in(hd, NumX:bitstring); (* msg 1 ... Ua -HD-> A : Num *)
    out(dd, NumX); (* msg 2 ... A -DD-> S : Num *)
    event beginSparam(NumX);
    in(dd, RegConf:bitstring); (* msg 3 ... S -DD-> A : RegConf
       *)
    0.

(* Server S *)
let processS =
    in(dd, NumX:bitstring); (* msg 2 ... A -DD-> S : Num *)
    event endSparam(NumX);
    new RegConf:bitstring;
    out(dd,RegConf); (* msg 3 ... S -DD-> A : RegConf *)
  0.

process
  (
    (!processUA) |
    (!processA) |
    (!processS) |
        (!in(hd,x:bitstring); out(hde,x)) (* this replicates
           everything in hdA to hdAe to reproduce a passive
           attacker *)
  )
```

D.2 WhatsApp Registration Ceremony - scenario ii

```
(*----------------------------------------------
 Protocol specification
----------------------------------------------*)
(* Channels *)
free dd  : channel. (* device-device channel *)
free dd2 : channel [private]. (* device-device authenticated
   channel *)
free dd3 : channel [private]. (* device-device channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)

(* Defining type nonce *)
type nonce.

(* Public Phone Number*)
free Num : bitstring.

(* Queries and events *)
event beginSparam(nonce).
event endSparam(nonce).

query PINx : nonce; event (endSparam(PINx)) ==> event (
   beginSparam(PINx)).


(* User A *)
let processUA =
    out(hdA, Num); (* msg 1 ... Ua -HD-> A : Num *)
    in(hdB, PINx:nonce); (* msg 7 ... B -HD-> UA : PIN *)
    out(hdA, PINx); (* msg 8 ... Ua -HD-> A : Num *)
    0.

(* Agent A - WhatsApp *)
let processA =
    in(hdA, NumX:bitstring); (* msg 1 ... Ua -HD-> A : Num *)
    out(dd, NumX); (* msg 2 ... A -DD-> S : Num *)
    in(dd, (NumRec:bitstring,PINx:nonce)); (* msg 3 ... S -DD-> A
        : NumRec, PIN *)
    event beginSparam(PINx);
    out(dd3, PINx); (* msg 4 ... A -DD3-> B : PIN *)
    in(hdA, PINb:nonce); (* msg 8 ... Ua -HD-> A : PIN *)
    out(dd, PINb); (* msg 9 ... A -DD-> S : PIN *)
    in(dd, RegConf:bitstring); (* msg 10 ... S -DD-> A : RegConf
       *)
    0.

(* Agent B - SMS App *)
let processB =
    in(dd3, PINx:nonce); (* msg 4 ... A -DD3-> B : PIN *)
    out(dd2, PINx); (* msg 5 ... B  -DD2-> SC : PIN *)
    in(dd2, PINsc:nonce); (* msg 6 ... SC -DD2-> B : PIN *)
    out(hdB, PINsc); (* msg 7 ... B  -HD->  UA : PIN *)
    0.

(* Server SMSC *)
let processSMSC =
    in(dd2, PINx:nonce); (* msg 5 ... B  -DD2-> SC : PIN *)
    out(dd2, PINx); (* msg 6 ... SC -DD2-> B : PIN *)
    0.

(* Server S *)
let processS =
    in(dd, NumX:bitstring); (* msg 2 ... A -DD-> S : Num *)
    new NumRec:bitstring;
    new PIN:nonce;
    out(dd, (NumRec,PIN)); (* msg 3 ... S -DD-> A : NumRec *)
    in(dd, PINx:nonce); (* msg 9 ... A -DD-> S : PINx *)
    if PINx = PIN then
    event endSparam(PIN);
    new RegConf:bitstring;
    out(dd,RegConf); (* msg 10 ... S -DD-> A : RegConf *)
  0.

process
  (
    (!processUA) |
    (!processA) |
    (!processB) |
    (!processS)
    )
```

D.3 WhatsApp Registration Ceremony - scenario iv

```
(*---------------------------------------------
 Protocol specification
---------------------------------------------*)
(* Channels *)
free dd  : channel. (* device-device channel *)
free dd2 : channel [private]. (* device-device authenticated
   channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)

(* Defining type nonce *)
type nonce.

(* Public Phone Number*)
free Num : bitstring.

(* Queries and events *)
event beginSparam(nonce).
event endSparam(nonce).

query PINx : nonce; event (endSparam(PINx)) ==> event (
   beginSparam(PINx)).
query PINx : nonce; inj-event (endSparam(PINx)) ==> inj-event (
   beginSparam(PINx)).

(*
With the setting (set ignoreTypes = false), the protocol respects
    the type system.
However, it finds fewer attacks.
In this specific case, respecting types is relevant since the
   human-interaction
here cannot ignore types, but it should be carefully used for
   other interactions.
I recommend test without it first on every scenario and check
   whether the termination
problem is really related to it.
*)
set ignoreTypes = false.
```

```
(* User A *)
let processUA =
    out(hdA, Num); (* msg 1 ... Ua -HD-> A : Num *)
    in(hdB, PINx:nonce); (* msg 5 ... B -HD-> UA : PIN *)
    out(hdA, PINx); (* msg 6 ... Ua -HD-> A : Num *)
    0.

(* Agent A - WhatsApp *)
let processA =
    in(hdA, NumX:bitstring); (* msg 1 ... Ua -HD-> A : Num *)
    out(dd, NumX); (* msg 2 ... A -DD-> S : Num *)
    in(dd, NumRec:bitstring); (* msg 3 ... S -DD-> A : NumRec *)
    in(hdA, PINx:nonce); (* msg 6 ... Ua -HD-> A : PIN *)
    (* colocando event beginSparam(PINx) aqui antes, eu forco o
       caso de que A sempre envia o PINx antes - free from MITM
       attacks due to SSL *)
    event beginSparam(PINx);
    out(dd, PINx); (* msg 7 ... A -DD-> S : PIN *)
    (* colocando aqui depois, eu mostro o caso onde onde pode
       existir MITM *)
    in(dd, RegConf:bitstring); (* msg 8 ... S -DD-> A : RegConf
       *)
    0.

(* Agent B - SMS App *)
let processB =
    in(dd2, PINx:nonce); (* msg 4 ... S -DDs-> B : PIN *)
    out(hdB, PINx); (* msg 5 ... B -HD-> UA : PIN *)
    0.

(* Server S *)
let processS =
    in(dd, NumX:bitstring); (* msg 2 ... A -DD-> S : Num *)
    new NumRec:bitstring;
    out(dd, NumRec); (* msg 3 ... S -DD-> A : NumRec *)
    new PIN:nonce;
    out(dd2, PIN); (* msg 4 ... S -DDs-> B : PIN *)
    in(dd, PINx:nonce); (* msg 7 ... A -DD-> S : PINx *)
    if PINx = PIN then
    new RegConf:bitstring;
    out(dd,RegConf); (* msg 8 ... S -DD-> A : RegConf *)
    event endSparam(PIN);
  0.

process
  (
    (!processUA) |
    (!processA) |
    (!processB) |
    (!processS)
  )
```

D.4 WhatsApp Registration Ceremony - scenario i

```
(*----------------------------------------------
 Protocol specification
----------------------------------------------*)
(* Channels *)
free dd  : channel. (* device-device channel *)
free dd2 : channel [private]. (* device-device authenticated
   channel *)
free hdA : channel [private]. (* human-device channel (no
   attackers) *)
free hdAe : channel. (* human-device channel (no attackers) *)
free hdB : channel [private]. (* human-device channel (no
   attackers) *)
free hdBe : channel. (* human-device channel (no attackers) *)

(* Defining type nonce *)
type nonce.

(* Public Phone Number*)
free Num : bitstring.

(* Queries and events *)
event beginSparam(nonce).
event endSparam(nonce).

query PINx : nonce; event (endSparam(PINx)) ==> event (
   beginSparam(PINx)).
query PINx : nonce; inj-event (endSparam(PINx)) ==> inj-event (
   beginSparam(PINx)).


(* User A *)
let processUA =
    out(hdA, Num); (* msg 1 ... Ua -HD-> A : Num *)
    in(hdB, PINx:nonce); (* msg 5 ... B -HD-> UA : PIN *)
    out(hdA, PINx); (* msg 6 ... Ua -HD-> A : Num *)
    0.

(* Agent A - WhatsApp *)
let processA =
    in(hdA, NumX:bitstring); (* msg 1 ... Ua -HD-> A : Num *)
    out(dd, NumX); (* msg 2 ... A -DD-> S : Num *)
    in(dd, NumRec:bitstring); (* msg 3 ... S -DD-> A : NumRec *)
    in(hdA, PINx:nonce); (* msg 6 ... Ua -HD-> A : PIN *)
    event beginSparam(PINx);
    out(dd, PINx); (* msg 7 ... A -DD-> S : PIN *)
    in(dd, RegConf:bitstring); (* msg 8 ... S -DD-> A : RegConf
       *)
    0.

(* Agent B - SMS App *)
let processB =
    in(dd2, PINx:nonce); (* msg 4 ... S -DDs-> B : PIN *)
    out(hdB, PINx); (* msg 5 ... B -HD-> UA : PIN *)
    0.

(* Server S *)
let processS =
    in(dd, NumX:bitstring); (* msg 2 ... A -DD-> S : Num *)
    new NumRec:bitstring;
    out(dd, NumRec); (* msg 3 ... S -DD-> A : NumRec *)
    new PIN:nonce;
    out(dd2, PIN); (* msg 4 ... S -DDs-> B : PIN *)
    in(dd, PINx:nonce); (* msg 7 ... A -DD-> S : PINx *)
    if PINx = PIN then
    new RegConf:bitstring;
    out(dd,RegConf); (* msg 8 ... S -DD-> A : RegConf *)
    event endSparam(PIN);
  0.

process
  (
    (!processUA) |
    (!processA) |
    (!processB) |
    (!processS) |
        (!in(hdA,x:bitstring); out(hdAe,x)) | (* this replicates
           everything in hdA to hdAe to reproduce a passive
           attacker *)
    (!in(hdB,x:bitstring); out(hdBe,x))  (* this replicates
       everything in hdB to hdBe to reproduce a passive attacker
       *)
  )
```