The Australian National University
2600 ACT | Canberra | Australia

Australian National University

School of Computing

College of Engineering, Computing and Cybernetics (CECC)

Tamgram-based Formalization of Bluetooth

— 12 pt research project (S1/S2 2023)

A report submitted for the course
COMP4560, Advanced Computing Project

By:
Changling Wang

Supervisors:
Assoc. Prof. Alwen Tiu
Darren Li

October 2023

Declaration:

I declare that this work:

- upholds the principles of academic integrity, as defined in the University Academic Misconduct Rules;
- is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the class summary and/or Wattle site;
- is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

October, Changling Wang

Acknowledgements

Thank God for granting me the courage to make this report that lengthy,
the serenity to accept that I cannot make it even lengthier,
and the wisdom to know when to stop.

Also thank Alwen for tolerating my slow progress on the report [1], and Darren for typesetting the draft of this report in LaTeXfor me.

[1] as well as on the entire project, maybe

Table of Contents

Chapter 1

Introduction

Bluetooth wireless technology (**Bluetooth**) is a pervasive short-range communications system used by billions of devices. It is specified in an open standard maintained by the *Bluetooth special interest group* (**SIG**), and its latest version is 5.3. There are two forms of Bluetooth systems: *Basic Rate/Enhanced Data Rate* (**BR/EDR**) and *Low Energy* (**LE** or **BLE**). BR/EDR is best suited for connection-oriented and high-throughput use cases, such as streaming audio. BLE is optimized for connection-less and very-low-power use cases such as fitness tracking. [1, 2]

In Bluetooth, *pairing* is the process for creating one or more secret keys [1] shared between the pairing devices. Over time, Bluetooth has adopted several different security mechanisms for its pairing processes, and the latest one (up to Bluetooth v5.3) is *Secure Connections* (**SC**), with its variants for both the BR/EDR system and the BLE system. In SC pairing, the most essential shared keys (root of trust) to establish are *Link Key* (**LK**) and *Long Term Key* (**LTK**), respectively for BR/EDR an BLE, and the establishment procedures are similar. LK/LTK is the bare minimum for establishing an encrypted session for communication, and various other keys may be distributed during the *Transport Specific Key Distribution* phase (**TSKD**) towards the end of the pairing process. These keys serve needs for extra services (such as data signing) or security properties of communication (such as privacy).

The *Identity Resolving Key* (**IRK**) is one of keys that may be distributed during TSKD, and it is used for private device address generation and resolution, which is part of a BLE feature providing privacy by changing the device address regularly to reduce the ability to track a device over time [1]. Conversely, the leakage of IRK would enable an adversary to resolve and identify the device address and therefore detect the presence of a particular device in vicinity, compromising the privacy of victim device.

It was observed that in certain implementations of Bluetooth, an adversary could steal the IRK of a victim device silently [3]. At the first glance, the leakage of IRK may not

seem to be a severe problem, due to the constraint that the attacker must be physically in proximity to the victim in order to detect the victim's presence. However, due to the adoption of the emerging crowd-sourced offline tracking devices, such as Apple's AirTag [4] and Samsung's Galaxy SmartTag [5], which use Bluetooth (as well as various other wireless technologies) to advertise their presence to their nearby devices to help with reporting their location to the device owners, the leakage of identity information like IRK could lead to more complicated privacy issues. In fact, it was already shown in [6] that Samsung's offline device tracking system exposes various privacy issues related to the leakage of IRK. Besides, similar issues were also observed in Australia's contact tracing app used during the Covid-19 pandemic [7].

In light of the aforementioned findings, we want to more systematically study the model of IRK in the Bluetooth Core Specification, and decided to conduct a more formal analysis of it. On the other hand, because of the massive size of Bluetooth Core Specification[1], it would be time-consuming and error-prone to conduct the analysis manually. Hence, we decided to resort to computer-aided protocol analysis by formalizing the entire pairing process of Bluetooth with Tamgram [8]. As it turned out, we do find that there exists a certain level of ambiguity in the specification on the privacy features of Bluetooth alongside our formalization, which may have led to the aforementioned privacy issues.

1.1 Contributions

The contributions of this project can be summarized as follows:

- We conducted a specification-based analysis of the cause of IRK leakage. We provide a Tamgram-based formalization of the IRK distribution procedure, which comprehensively analyzes all the potential pairing feature configurations of target devices and uncovers the scenarios where IRK leakage could occur. To the best of our knowledge, this is the first formalized analysis of the privacy issues of the pairing process of Bluetooth.
- We provide a Tamgram-based formalization of SC pairing of Bluetooth, which complements the previous formalized analysis of Bluetooth [9, 10] by integrating the *IO Capabilities* (**IO Caps**) exchange into the formalization.
- We identified several inconsistencies in the Bluetooth Core Specification, which lead to asymmetrical behaviors during the SC pairing over BR/EDR and BLE. We then show that such asymmetries may expose certain vulnerabilities that previous researches missed.

[1] v5.3 consists of 3085 pages in total

Chapter 2

Background

2.1 Bluetooth Pairing

"Pairing is a three-phase process" (pp. 1556 of [1]). In the case of SC over BLE, these 3 phases are (1) Pairing Feature Exchange, (2) LTK Generation and (3) TSKD.

During phase 1, the pairing devices exchange various information to negotiate the pairing protocol for LTK generation, such as the MITM flag which indicates the need for *man-in-the-middle* attack (**MITM**) protection, and the IO Caps which allow them to work out a more secure (and convenient) pairing method when possible. Based on these exchanged features, a pairing protocol will be picked as specified by [1], which will then be invoked during phase 2 to generate the LTK. The first two phases of the pairing process of SC over BR/EDR is basically the same, except for some asymmetries which will be discussed in more details later.

When pairing over BLE, if the features exchanged in phase 1 also indicate that some other keys, such as the Initiator's IRK, shall be distributed, then they will be distributed during phase 3, after encrypted communication is enabled with the LTK generated during phase 2. These keys can also be distributed after pairing over BR/EDR, but the corresponding feature exchange happens after the generation of LK. This is because the keys exchanged during TSKD are BLE-specific (which is why phase 3 is called *Transport Specific*), and they were introduced as part of the introduction of *Cross-Transport Key Derivation* (**CTKD**), which aims to simplify the pairing procedure of devices that want to pair over both transports, by allowing them to derive LK from LTK without pairing again over BR/EDR or vice versa.

During phase 2, SC has four *association models* (**pairing protocols**), namely *Numeric Comparison* (**NC**), *Passkey Entry* (**PE**), *Just Works* (**JW**) and *Out of Band* (**OOB**).

NC and PE are designed to provide authenticity by involving user interactions to assist

the pairing devices to authenticate each other. The specification specifies how to determine which pairing protocol to use, depending on the IO capabilities of the pairing devices. If authenticity is not required or the IO capabilities of the devices do not suffice for authenticated pairing, JW could be used for an unauthenticated but user-interaction-free pairing. Lastly, by exchanging the authentication information over another communication channel (out of band), such as via NFC, OOB enables authenticated pairing "without" user interactions.

In this project, all association models but OOB are considered. OOB is excluded because of its reliance on the out-of-band channel. To the best of our knowledge, in all the related researches that looked into OOB [9, 10], the out-of-band channel has always been modeled as a private channel that an attacker (in the Dolev-Yao Model) has neither access to nor control over. With such an ideal channel, none of those research has found any vulnerability in the OOB protocol of SC. On the other hand, since the focus of this project is not on any specific implementation of the OOB channel, it is hard to come up with any sensible substitute for the well-studied, perfectly private out-of-band channel. As a result, OOB is simply excluded from the scope of this project.

2.2 Tamarin

Tamarin is a tool for the symbolic modeling and analysis of security protocols [11]. It represents the messages exchanged and computations as algebraic terms [10]. To formalize a security protocol, the user needs to represent different stages of the protocol as states consisting of these algebraic terms, and then specify the process of the protocol as transitions between the states. As for analysis of a protocol, the user needs to specify the properties of interest in the format of logical expressions, which Tamarin will then try to either verify or falsify automatically.

In this project, we never directly do the formalization in Tamarin's language, instead, the formalization is written in Tamgram's language, which Tamgram then translates into Tamarin's language before being submitted to Tamarin to get analyzed.

2.3 Tamgram

As summarized on [8], Tamgram is "A Frontend for Large-scale Protocol Modeling in Tamarin". It aims at addressing the usability issues of Tamarin by introducing various features to facilitate the translation from specification to formalization. For example, the sequential execution of a program can be easily modeled with a *Process*, which enforces the sequential order of the state transitions within its scope. A Tamgram Process also comes with an easy-to-use process-local memory manipulation, and *cell*s can be declared and used to mimic variables in normal programming languages. Furthermore, Tamgram facilitates the modeling of programs' control flows with its *if-then-else* clauses, conditional and unconditional loops, as well as the non-deterministic *choice* clauses.

Another feature of Tamgram that got extensively used in our model is the *Process Macros*, which helped keep the formalization modularized and maximize code reuse.[12]

Instead of giving examples here to showcase the basic operations and features of Tamgram, we will introduce them directly in the context of this project as we walk through the formalization in Chapter 3.

2.4 Related Work

[9] and [10] are the latest works on formalized analysis of Bluetooth's pairing protocols, respectively using Proverif and Tamgram. However, both of them only focused on the security properties of the generated LK/LTK, which are completely determined by what happens during phase 2.

As research on phase 3, [2] demonstrates a spoofing attack which exploits CTKD to overwrite the victim device's established shared keys with the keys generated from pairing with an adversarial device. There are also spoofing attacks on the reconnection phase of BLE [13]. However, [1] seems to have adopted appropriate fixes in its specification in response to the identification of these attacks.

In this project, we paid more attention to phase 1 and identified several issues that could be hard to find otherwise.

Chapter 3

Formalization with Tamgram

3.1 Scope of Specification Focused on

The latest version of Bluetooth Core Specification [1] comprises 7 volumes, which add up to over 3000 pages in length. It is neither feasible nor necessary for this project to go through all the details, and we only focused on the sections related to the pairing procedures, which are listed as follows:

- Vol. 1 Part A 5 Security Overview
- Vol. 2 Part H 7 Secure Simple Pairing (Pairing over BR/EDR)
- Vol. 3 Part H 2 Security Manager (Pairing over BLE)
- Vol. 3 Part H 3.6.1 Security in Bluetooth Low Energy (TSKD)

In the following sections, we will walk through the key components of the formalization of the pairing process of SC over BR/EDR, as well as that of the simplified model tailored for the analysis of IRK leakage. Both of them have their respective counterparts over BLE, but the walk-through of those formalization is omitted due to the high degree of similarity in their general structure.

3.2 Overview

In this project, the entire pairing process, from Feature Exchange to TSKD/CTKD, is modeled (in 2 separate models). In contrast, other researches [9, 10] usually skip Feature Exchange and start the formalization from the key generation phase. It is completely justified to skip Feature Exchange if the focus is on the security properties of LK/LTK generation procedure. However, since the distribution (leakage) of IRK takes place only after the generation of LK/LTK (during TSKD), and the trigger condition for

leakage involves certain flag configuration exchanged before (in the case LTK pairing) the generation phase (during Feature Exchange), it is necessary to consider the entire pairing process when analyzing the IRK leakage.

Originally, the plan was to build a complete model that captures all details of pairing, which can be used to analyze not only IRK leakage, but also other aspects of pairing, including those that were looked into by other researches. However, after we partially finished the implementation of such a formalization, we found it took an unbearably long time[1] to analyze even a single property (Confidentiality of generated LTK/LK over BR/EDR) for a single combination of features of pairing devices. As a result, we built another simplified model tailored for the analysis of IRK leakage.

Alongside the formalization, we also found that, although the association models of SC over BR/EDR (introduced in version 4.1) and SC over LE (introduced in version 4.2) are described as "functionally equivalent" (Vol1 PartA 5.4.1 of [1]), there exists some asymmetries between the specification of the 2 systems. As a result, certain pairs of devices might end up triggering different pairing protocols while pairing over different transports, and certain pairing protocol might expose some security vulnerabilities that are not present in its counterpart over the other mode.

3.3 Pairing Feature Exchange

The pairing process always starts from pairing devices exchanging their respective pairing features, which include their IO Caps, Authentication Requirements (MITM protection and bonding requirements), availability of support for SC, maximum encryption key sizes etc. Due to the scope of this project, the relevant features we need to consider are reduced to IO Caps, MITM protection requirements and the IRK distribution requirements. As for the analysis of LK/LTK generation, we have to presume MITM protection is required as otherwise it would be trivial for the adversary to compromise the pairing process, and the IRK distribution is also irrelevant, hence only IO Caps are considered in our formalization of pairing.

In BR/EDR, 4 values are defined for IO Caps, which are *DisplayOnly*, *DisplayYesNo*, *KeyboardOnly* and *NoInputNoOutput* (Table 3.1).

Since Tamgram/Tamarin does not have a type system, these IO Caps are simply defined as constant strings in the model:

```
fun DisplayOnly() = "DisplayOnly"
fun DisplayYesNo() = "DisplayYesNo"
fun KeyboardOnly() = "KeyboardOnly"
fun NoInputNoOutput() = "NoInputNoOutput"
```

After the exchange of IO Caps, the devices will need to decide the pairing protocol based on the IO Caps, and the decision policy for SC over BR/EDR is mandated by the

[1] didn't terminate after 2h of execution on WSL Ubuntu with Intel i7-11800H

Table 3.1: Definition of IO Caps over BR/EDR on pp. 700 of [1]

Name	Length (bytes)	Type	Detailed
..	..	..	..
IO_Capabilities	1	uint8	0: Display only 1: Display YesNo 2: KeyboardOnly 3: NoInputNoOutput 4-255: reserved for future use
..	..	..	..

specification (Figure 3.1).

The lookup of this mapping will turn out to show up quite frequently throughout the formalization, so it is implemented in a dedicated process macro, which reads in the IO Caps of the pairing devices and stores the resultant pairing protocol in the given cell named **'protocol**:

```
process determine_stage_1_protocol (
   'iocap_a,
   'iocap_b,
   rw 'protocol
) =
```

As mentioned earlier, a *cell* in Tamgram can be generally thought of as a variable in normal programming languages, and it is characterized by the **single quote** prepended to its name.

Under the hood, the mapping look-up is implemented with 2 levels of nested *choice* statements, combined with the pattern matching operator **cas**:

```
choice {
  { // row 1
    [ 'iocap_b cas DisplayOnly() ] --> [];

    choice {
      { // col1
        [ 'iocap_a cas DisplayOnly() ]
          -->
          [ 'protocol := Just_Works() ]
      };
      { // col2
        [ 'iocap_a cas DisplayYesNo() ]
          -->
          [ 'protocol := Just_Works_AC() ]
      };
      // Other columns..
    }
```

```
  };
  // Other rows..
}
```

3.3.1 Inter-Process Communication

In Tamarin and Tamgram, exchanging pairing features, as well as most of the subsequent communications between the involved entities in the formalization, is modeled as sending messages through a public and unprotected network, which the adversary is modeled to have full access to and control over. Specifically, the adversary is modeled to be able to arbitrarily block messages sent and send forged messages to others. As a result, it is pointless to specify a target when sending out a message, because there is no guarantee that it will be sent to that target anyway. Hence, all the messages sent in Tamgram and Tamarin are simply *targetless*, and whenever an entity wants to send a message, it just publishes the message onto the network, using the operator **Out**:

```
process Initiator =
   // Start of IO cap exchange
   [ IO_Cap_A(addr, iocap) ]
      -->
      [ Out(<addr, iocap>) ];
```

Similarly, the target receiver does not need to specify the sender whom the message should be sent from. However, the receiver does have the ability to pattern-match the format of the message the he/she is waiting for, and will ignore all the messages in the wrong format. In Tamgram and Tamarin, an entity uses the operator **In** to receives a message from the network:

```
[ In(<addr, iocap>) ]
   -->
   [ 'BTaddrA := addr, 'IOcapA := iocap ];
```

It is also possible to model messages sent through a private channel, which the adversary has neither access to nor control over, but these messages are simply represented as facts in state transitions. For example, the initialization of a pairing device can be thought of as the user sending a private message to the device, telling it what its IO Caps and device address are:

```
process Responder =
   // Start of IO cap exchange
   [ IO_Cap_B(addr, iocap) ]
      -->
      [ 'BTaddrB := addr, 'IOcapB := iocap ];
```

3.4 Pairing Devices & Users

In this model, there are 3 Tamgram processes involved, corresponding respectively to the *Initiator/Central* (the device that initiates the pairing procedure), the *Respon-*

		Device A (Initiator)			
		Display Only	**DisplayYesNo**	**KeyboardOnly**	**NoInputNoOutput**
Device B (Responder)	**DisplayOnly**	Numeric Comparison with automatic confirmation on both devices. Unauthenticated	Numeric Comparison with automatic confirmation on device B only. Unauthenticated	Passkey Entry: Responder Display, Initiator Input. Authenticated	Numeric Comparison with automatic confirmation on both devices. Unauthenticated
	DisplayYesNo	Numeric Comparison with automatic confirmation on device A only. Unauthenticated	Numeric Comparison: Both Display, Both Confirm. Authenticated	Passkey Entry: Responder Display, Initiator Input. Authenticated	Numeric Comparison with automatic confirmation on device A only and Yes/No confirmation whether to pair on device B. Device B does not show the confirmation value. Unauthenticated
	Keyboard Only	Passkey Entry: Initiator Display, Responder Input. Authenticated	Passkey Entry: Initiator Display, Responder Input. Authenticated	Passkey Entry: Initiator and Responder Input. Authenticated	Numeric Comparison with automatic confirmation on both devices. Unauthenticated
	NoInputNoOutput	Numeric Comparison with automatic confirmation on both devices. Unauthenticated	Numeric Comparison with automatic confirmation on device B only and Yes/No confirmation on whether to pair on device A. Device A does not show the confirmation value. Unauthenticated	Numeric Comparison with automatic confirmation on both devices. Unauthenticated	Numeric Comparison with automatic confirmation on both devices. Unauthenticated

Table 5.7: IO capability mapping to authentication stage 1

Figure 3.1: Mapping from IO Caps to pairing protocol on pp. 1280 of [1]

der/Peripheral (the target device of the pairing procedure) and the user (the operator(s) of the pairing devices).

Unlike the models in [9] and [10], where the pairing devices are directly initialized with a designated pairing protocol, the pairing devices in our model are initialized only with their respective IO Caps and device addresses. The devices then exchange their IO Caps via the attacker-controlled public channel, and work out the pairing protocol to use based on what they receive, which is generally prone to adversarial modification.

For example, the process for the Initiator goes like this:

```
process Initiator =
  // Start of IO cap exchange
  [ IO_Cap_A(addr, iocap) ]
    -->
    [ 'BTaddrA := addr, 'IOcapA := iocap, Out(<addr, iocap>) ];
  [ In(<addr, iocap>) ]
    -->
    [ 'BTaddrB := addr, 'IOcapB := iocap, 'Stage1_Protocol := tbi() ];
  determine_stage_1_protocol('IOcapA, 'IOcapB, 'Stage1_Protocol);
  // End of IO cap exchange
```

3.5 Different Kinds of Users

The Method Confusion Attack [14] demonstrates that the users themselves are also part of the attack surface of the pairing process of Bluetooth.

In light of this observation, we differentiated two kinds of users in the model. The first kind of user is presumed to have knowledge of the expected behavior of the pairing devices. for example, a user of this kind knows that NC should be invoked when 2 mobile phones are pairing with each other, and therefore will abort the pairing procedure when a pop-up shows up in one of the phones asking him/her to type in the 6-digit value displayed on the screen of the other device. Contrary to the first kind is the second kind of user, who is presumed not to be familiar with these technical details and is therefore prone to the confusion attack.

As for the representation in the formalization, since users are presumed to know the IO Caps of the pairing devices, they can also work out the specified pairing protocol to use by themselves. Hence, the first kind of user will only proceed if the states of the pairing devices match the expected states of the pairing protocol derived from the user's knowledge of IO Caps of the pairing devices. For example, the user's state transition for NC is as follows:

```
choice {
  {
    [ 'Stage1_Protocol cas Numeric_Comparison() ] --> [];

    [ Va(addr_a, confirm_val), Vb(addr_b, confirm_val) ]
```

```
      -->
      [ Auth_Stage_1_End(addr_a), Auth_Stage_1_End(addr_b) ]
  };
  // Cases for other protocols..
}
```

Here, **Auth_Stage_1_End()** can be thought of as the confirm button that the user would press if he/she wishes the pairing to proceed, and the processes of the pairing devices can only proceed after consuming this state registered with their own device addresses. In this case, even when both devices are displaying the same 6-digit value, the user will only press the button if he/she expects the devices to pair over NC.

In contrast, the second kind of user has no idea about what NC is, and all he/she knows is that when both devices are displaying the same 6-digit value, it seems OK to allow them to proceed.

```
choice {
  {
    [ Va(addr_a, confirm_val), Vb(addr_b, confirm_val) ]
      -->
      [ Auth_Stage_1_End(addr_a), Auth_Stage_1_End(addr_b) ]
  };
  // Cases for other protocols..
}
```

In our model, both kinds of users are analyzed. Since all the other components (as discussed in the previous sections) of the formalization are exactly the same, it would be better to model both implementations in a single file. In Tamgram, this is realized with the combination of a non-deterministic choice statement and labeled dummy transitions, shown as follows:

```
choice {
  {
    [] --[ User_Ideal() ]-> []; // A user familiar with the pairing protocol

    choice {
      // Branches for different pairing protocols
    }
  };
  {
    [] --[ User_Ignorant() ]-> []; // A user unfamiliar with the pairing
        protocol

    choice {
      // Branches for different pairing protocols
    }
  }
}
```

An example of how to use such construct in formal property specification in Tamgram

is given in a following section.

3.6 Cryptographic Functions

In SC, the pairing protocols rely on various cryptographic functions to implement the required functionalities, such as commitment checking and key generation etc. Under the hood, these cryptographic functions are constructed from *cryptographic primitives* at a lower level, such as hash functions and encryption functions, which are believed to be have certain properties that all the high-level security properties of the protocol can be based on. In reality, the details of the implementation of such cryptographic primitives can be arbitrarily complicated, in order to endow the primitives with desired properties.

On the other hand, Tamarin and Tamgram are based on the *Dolev-Yao Model*, in which the cryptographic primitives are modeled by abstract operators [15]. In other words, the model simply ignores the implementational details of the cryptographic primitives and directly works on the assumption that the desired properties hold.

For example, the function **f1**, as shown in Fig 3.2 , is a cryptographic function used to compute a commitment value for authentication purposes during SC pairing over BR/EDR. **f1** is defined in Vol. 2 Part H 7.7.1 of [1], as follows:

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

where both U and V are 256-bit values, X is a 128-bit value and Z is a 8-bit value.

Here, HMAC-SHA-256$_X$ is a hash-based message authentication code, which can be also thought of as a cryptographic function based on the hash function SHA-256 as the cryptographic primitive.

To define **f1** in Tamgram, we should start with the definition of SHA-256, which can be simply declared as an operator that takes in one argument:

```
fun SHA256/1
```

In Tamgram, functions defined like this are always one-way unless specified otherwise, which means they automatically qualify as hash functions. Now that SHA-256 is defined, HMAC-SHA-256$_X$ can be easily defined as follows:

```
fun HMAC_SHA256(key, m) = SHA256(<key,m>)
```

It is notable that such a definition slightly over-simplifies HMAC-SHA-256$_X$, but it does capture its properties in Dolev-Yao Model accurately.

The integer division and **f1** itself can also be defined in a similar fashion:

```
fun first_128bit/1
fun f1(U, V, X, Z) = first_128bit(HMAC_SHA256(X, <U,V,Z>))
```

All the cryptographic functions that occur in the following sections are defined similarly.

3.7 Pairing Protocols

Depending on the pairing protocol chosen and the role of a device, the actions to take during pairing are generally different. As a result, all of these different protocols are modeled in separate process macros, and are invoked in different branches of the main processes as appropriate.

```
choice {
  {
    [ 'Stage1_Protocol cas Just_Works() ] --> [];

    numeric_comparison_init('BTaddrA, 'PKa, 'PKb, 'ra, 'rb, 'Na, 'Nb, 'Cb,
        'Stage1_Protocol)
  };
  {
    [ 'Stage1_Protocol cas Numeric_Comparison() ] --> [];

    numeric_comparison_init('BTaddrA, 'PKa, 'PKb, 'ra, 'rb, 'Na, 'Nb, 'Cb,
        'Stage1_Protocol)
  };
  // Other pairing protocols..
};
```

3.7.1 Numeric Comparison / Just Works

Since JW can be deemed as NC without user confirmation, their pairing processes are structurally the same, and only differ in whether the user confirmation is asked for.

The detailed steps of the procedure are illustrated in Figure 3.2.

Since NC and JW are modeled together, both the Initiator and the Responder need to determine whether they should ask for user confirmation before proceeding to the next stage, depending on the pairing protocol used.

For example, the process macro for the Initiator using NC/JW is defined as follows:

```
process numeric_comparison_init (
   addr_a,
   'pka,
   'pkb,
   rw 'ra,
   rw 'rb,
   rw 'Na,
   rw 'Nb,
   rw 'Cb,
   'protocol
) =
   // 2a
   [ Fr(~na) ]
      -->
```

```
   // 3a
   [ 'Na := ~na, 'ra := "0", 'rb := "0"];
// 4
[ In(cb) ]
   -->
   // 5
   [ 'Cb := cb, Out('Na) ];
// 6
[ In(nb) ]
   -->
   [ 'Nb := nb ];
// 6a
if ('Cb cas f1('pkb, 'pka, 'Nb, "0")) then {
   [] --> [ Commitment_Checked('pka, 'pkb, 'Na, 'Nb) ]
};
// 7a
[ Commitment_Checked('pka, 'pkb, 'Na, 'Nb) ] --> [];
choice {
   {
      [ 'protocol cas Numeric_Comparison() ]
        -->
        [ Va(addr_a, g('pka, 'pkb, 'Na, 'Nb)) ]
   };
   {
      [ 'protocol cas Just_Works() ]
        -->
        [ Auth_Stage_1_End(addr_a) ]
   };
   {
      [ 'protocol cas Just_Works_AC() ]
        -->
        [ Va(addr_a, Pairing_Confirm()) ]
   };
   {
      [ 'protocol cas Just_Works_BC() ]
        -->
        [ Auth_Stage_1_End(addr_a) ]
   }
}
```

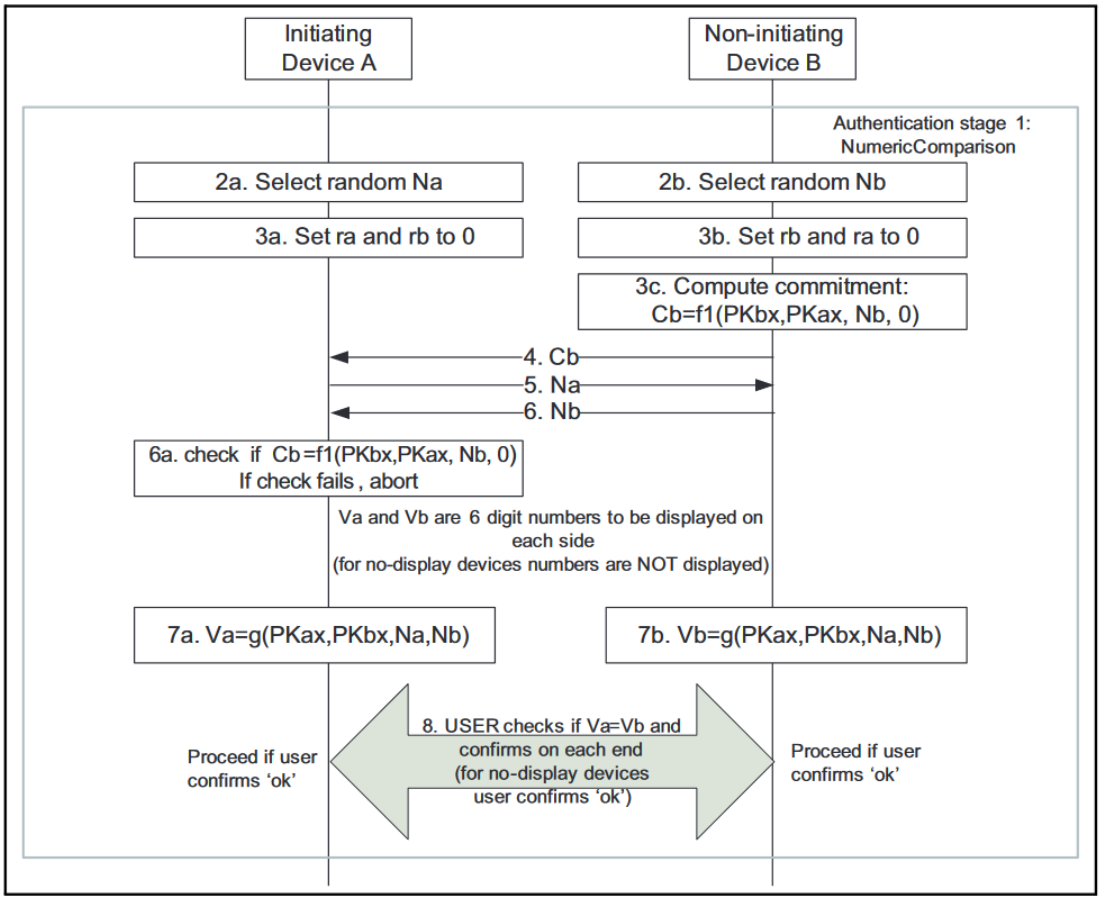

Figure 7.4: Authentication stage 1: Numeric Comparison protocol details

Figure 3.2: Procedure of NC & JW over BR/EDR (pp. 985 of [1])

3.7.2 Passkey Entry

The detailed steps of the procedure are illustrated in Figure 3.3.

Depending on the IO Caps of the pairing devices, both the Initiator and Responder may need the user to type the 6-digit passkey, which is either generated and displayed by the other pairing device or generated by the user, into their terminals. As a result, the corresponding process macros also need to determine whether it should generate and display a value or to wait for the user input.

For example, the process macro for the Initiator using PE is defined as follows:

```
process passkey_entry_init (
   pka,
   pkb,
   addr_a,
   rw 'ra,
   rw 'rb,
   rw 'Na,
   rw 'Nb,
   rw 'Cb,
   'protocol
) =
   // 2a
   if ('protocol cas Passkey_Entry_ADBI()) then {
      [ Fr(~passkey) ]
        -->
        [ Va(addr_a, ~passkey), 'ra := ~passkey, 'rb := ~passkey ]
   } else {
      [] --> [ Passkey_Prompt_Init(addr_a) ];
      [ Enter_Passkey_Init(addr_a, passkey) ]
        -->
        [ 'ra := passkey, 'rb := passkey ]
   };
   // 3a
   [ Fr(~na) ]
     -->
     [ 'Na := ~na ];
   // 4a, 5
   [] --> [ Out(f1(pka, pkb, 'Na, 'ra)) ];
   // 6
   [ In(cb) ]
     -->
     // 7
     [ 'Cb := cb, Out('Na) ];
   // 8
   [ In(nb) ]
     -->
     [ 'Nb := nb ];
   // 8a
   [ 'Cb cas f1(pkb, pka, 'Nb, 'rb) ]
     -->
     [ Auth_Stage_1_End(addr_a) ]
```

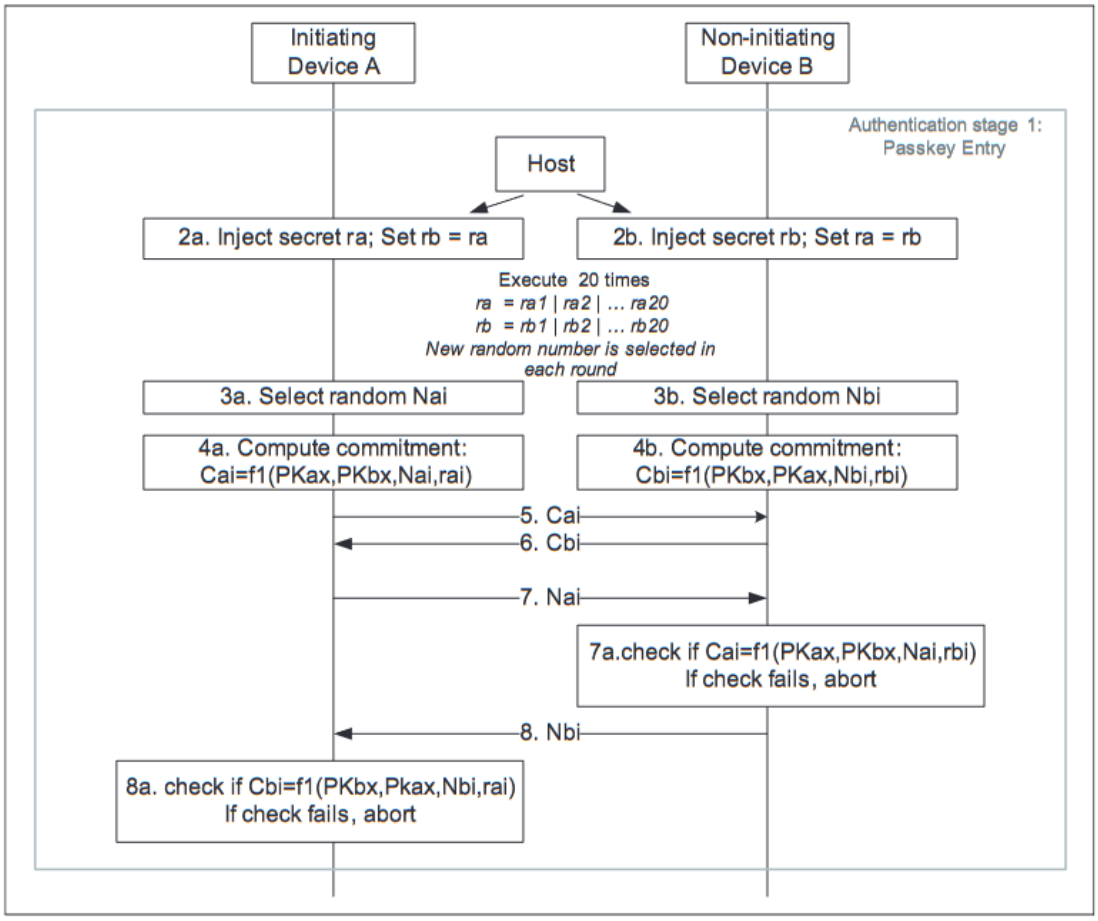

Figure 7.6: Authentication stage 1: Passkey Entry details

Figure 3.3: Procedure of PE over BR/EDR (pp. 989 of [1])

3.8 IRK Theft

The previous sections document the key components of the model that we originally planned to build, which hasn't involved anything related to the IRK distribution. As mentioned in section 3.2, this is because we found that model was already too computationally complicated for Tamarin to analyze, and therefore we decided not to complicate it further by integrating the features related to IRK distribution. Instead, we switched the gear to build a simplified model, which can only be used to analyze IRK distribution. In the remaining part of this section, we will introduce the properties of IRK distribution that we are interested in, as well as the observation which enables the simplification of our model.

The IRK Theft essentially refers to the event that a device distributes (leaks) its IRK to an adversary without asking for user confirmation or notifying the user of the distri-

bution. Hence, the property of IRK distribution that we want to analyze can be stated as:

For all event of IRK distribution, there exists an event of asking for user confirmation.

Or in the form of a *lemma* in Tamgram:

```
lemma IRKTheftProof =
   All #i. (Distribute_IdKey() @ #i)
     ==> (Ex #j. User_Confirmation() @ #j& j < i)
```

A device is deemed **IRK-theft-proof** if this property holds. Since IRK distribution happens after the key generation phase, the premise would be false whenever the key generation fails, and the whole argument would consequently be true. But these cases are not of interest as the focus is on the existence of user confirmation when IRK does get distributed. Thus, it is fine to assume the key generation always succeeds, in the context of analyzing IRK leakage. This observation leads to the aforementioned simplification of model for the analysis of IRK leakage, in which all the hairy details of the key generation procedure (including the generated key itself) are omitted from the formalization, and nothing but the pairing protocol used remains, as it relates to the presence of user confirmation.

3.9 User Confirmation

In both BR/EDR and BLE, NC and PE involve user interactions, and the event **User_Confirmation()** will be triggered whenever these 2 association models are used in our model. The problem gets slightly more complicated in the case of JW. In Figure 3.1, the mapping table from IO Caps to pairing protocols explicitly specifies that a device with *DisplayYesNo* IO Caps should ask for user confirmation while pairing over BR/EDR **even if the protocol is JW**, whereas such requirements are missing from the counterpart table for BLE on pp. 1573 of [1].

In contrast, it is mentioned on pp. 1581 that "**When Just Works is used (over BLE), the commitment checks are not performed and the user is not shown the 6-digit values**", which leaves readers the impression that user confirmation is not needed whenever JW is used, and we couldn't find any description contradicting with such impression in the specification. As a result, in our model, the event **User_Confirmation()** will never be triggered when JW is invoked over BLE pairing, but it will be triggered over BR/EDR pairing if the victim device has *DisplayYesNo* IO Caps.

For Pairing over BLE, the condition for triggering **User_Confirmation()** can be captured by a dummy transition inside an if statement as follows:

```
if not ( 'protocol cas Just_Works() ) then {
  [] --[ User_Confirmation() ]-> []
};
```

As for BR/EDR, the condition is slightly more complicated, as we need to differentiate 3 sub-cases of JW, which are respectively the case where neither device has sufficient IO Cap to ask for user confirmation, and the cases where the Initiator or Responder can ask for user confirmation. It is impossible that the 2 devices ask for user confirmation simultaneously, as in that case NC would have been invoked. In our model, the convention is that **A** always refers to the Initiator while **B** always refers to the Responder, and the condition for triggering **User_Confirmation()** on the Initiator is as follows:

```
if not ( 'protocol cas Just_Works() ) then {
  if not ( 'protocol cas Just_Works_BC() ) then {
    [] --[ User_Confirmation() ]-> []
  }
};
```

To make things more complicated, it was specified on pp. 1279 that **if neither of the devices indicates the requirement for MITM protection** during pairing (over BR/EDR), then the pairing function should be selected "**as if both devices set their IO Caps to DisplayOnly**", which is essentially JW without user confirmation on both devices. As for BLE, the table on pp. 1573 also specifies that if neither device sets the MITM flag in the SMP Pairing Request/Response command, the pairing method should be JW:

		Initiator			
		OOB Set	**OOB Not Set**	**MITM Set**	**MITM Not Set**
Responder	**OOB Set**	Use OOB	Use OOB		
	OOB Not Set	Use OOB	Check MITM		
	MITM Set			Use IO Capabilities	Use IO Capabilities
	MITM Not Set			Use IO Capabilities	Use Just Works

Table 2.7: Rules for using Out-of-Band and MITM flags for LE Secure Connections pairing

Figure 3.4: pp. 1573 of [1]

This is another instance where the specification seems to imply that no user interaction should be involved anyway when JW over BLE is used.

In our model, this is captured by a non-deterministic choice statement, which directly sets the pairing protocol to JW when both devices unset their MITM flag:

```
choice {
  {
    [ 'mitm_a cas False(), 'mitm_b cas False() ]
      -->
```

```
        [ 'protocol := Just_Works() ]
    };
    {
      [ 'mitm_a cas True() ] --> [];

      determine_stage_1_protocol('iocap_a, 'iocap_b, 'protocol)
    };
    {
      [ 'mitm_b cas True() ] --> [];

      determine_stage_1_protocol('iocap_a, 'iocap_b, 'protocol)
    }
  };
```

It is also notable that [1] doesn't specify any user confirmation process specific to the keys distributed during TSKD, and the only mandatory user interactions throughout the entire pairing process are those involved during the key generation phase. As we will discuss in the following sections, the implication of such design choice may be worth more thorough consideration than one may imagine.

3.10 IRK Distribution

Both pairing devices can indicate their willingness to share their respective IRKs to the other party. During Feature Exchange, the initiator of pairing process indicates whether it wants to share its IRK and whether it requests that the other party share its IRK, by setting corresponding flags in the SMP Pairing Request. The Responder can deny any requested distribution by unsetting the corresponding flag in SMP Pairing Response.

On pp. 1611 of [1], it is specified that

"**The Peripheral (Responder) shall not set to one any flag ... that the Central (Initiator) has set to zero**"

which forbids the Responder from proposing distribution of flags not requested by the Initiator. However, the specification doesn't explicitly mention that the Initiator shall make sure that the Responder actually obeys the specification while setting the flags in the Pairing Response. Consequently, if the implementation of the Initiator simply presumes the Responder's obedience to the specification, an adversary may be able to trick it into distributing its IRK by maliciously setting the corresponding flag in Pairing Response.

In our model, both the implementations that do the appropriate check and those who don't are analyzed. Since all the other components of the two implementations are exactly the same, we can model both implementations in the same file. Similar to how we modeled the different kinds of users, this is also realized with the combination of a non-deterministic choice statement and labeled dummy transitions, shown as follows:

```
choice {
```

```
    {
      [] --[ Careful_Implementation() ]-> [];

      [ 'irk_a_req cas True(), 'irk_a_rsp cas True() ]
        --[ Distribute_IdKey() ]->
        []
    };
    {
      [] --[ Careless_Implementation() ]-> [];

      [ 'irk_a_rsp cas True() ]
        --[ Distribute_IdKey() ]->
        []
    }
  }
```

The event labels **Careful_Implementation()** and **Careless_Implementation()** marks the path taken, which can be used to specify the property we want to analyze and restrict the scope of exploration whilst conducting verification. For example, if we want to analyze whether an IRK theft could happen under the implementation that does the validity check, the lemma can be specified as follows:

```
  lemma IRKTheftProofCareful =
    All #i #k.
      (Distribute_IdKey() @ #i & Careful_Implementation() @ #k)
        ==>
        (Ex #j. User_Confirmation() @ #j & j < i)
```

3.11 Search Restriction

With the components discussed in previous sections, the model seems basically complete. Now, we can instantiate a process with the feature configuration of interest and let Tamarin analyze it. To instantiate a process from a process macro, we need to initialize all the cells before passing them to the process macro, including those that haven't been assigned a valid value. For example, we can instantiate a process which has IO Caps *DisplayYesNo*, requires MITM protection during pairing and does not want to distribute its IRK during TSKD, as follows:

```
process DisplayYesNo_Mitm_NoIrk =
  [ Fr(~id) ]
    -->
    [ 'idkey := False(), 'mitm := True(), 'iocap := DisplayYesNo(),
        'process_id := ~id,
      'irk_a_rsp := tbi(), 'mitm_b := tbi(), 'iocap_b := tbi(), 'protocol :=
          tbi() ];
  Initiator('process_id, 'idkey, 'mitm, 'iocap, 'irk_a_rsp, 'mitm_b,
      'iocap_b, 'protocol)
```

With at least one instance of process, we can ask Tamarin to analyze the properties specified in the lemmas. As for the feature configuration in the example above, we would expect the property of IRK-theft-proof to hold at least when the careful implementation is used, because it has set the IRK flag to false. However, it turns out that Tamarin will falsify both properties checked.

If we use the interactive mode of Tamarin to examine the attack trace found, we will find that the attack trace found by Tamarin actually combines the result from both the branch of the careful implementation and that of the careless implementation:

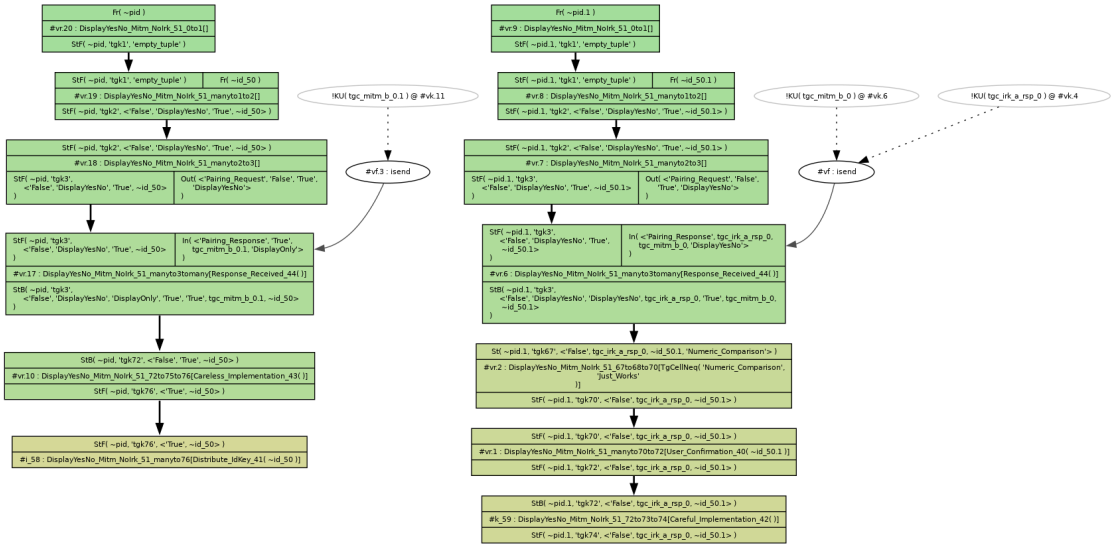

Figure 3.5: Trace of False Positive Attack Found by Tamarin

Obviously, this is a false positive, as in reality the two implementations will not co-exist and run in parallel like this on the same device.

In order to get rid of these false positive attack traces, we will need to restrict the cases Tamarin should consider, which can be specified using the built-in *restriction* clauses in Tamgram and Tamarin:

```
restriction either_careless_or_careful =
   (All pid #i.
      (Careful_Implementation(pid) @ #i) ==>
      (not (Ex #j. Careless_Implementation(pid) @ j)))
   &
   (All pid #i.
      (Careless_Implementation(pid) @ #i) ==>
      (not (Ex #j. Careful_Implementation(pid) @ j)))
```

More specifically, this restriction specifies that Tamarin should not consider traces where both implementations are running in parallel. After the addition of this this restriction, the model is finally completed, and the results of Tamarin's analysis are shown and

discussed in Chapter 4.

Chapter 4

Results & Analysis

As discussed above, we only managed to finish the verification of the model for IRK leakage, and the results for the other model is missing due to the high computational cost.

The 2 properties analyzed were the IRK-theft-proofness of the Initiator/Center respectively under careful and careless implementation. Due to the nature that the Responder/Peripheral only responds to the incoming pairing requests, it is not affected by such potential implementation issues. However, as we will discuss later, this does not mean the Responder will never leak its IRK without user's awareness.

Under the assumption that phase 2 of pairing always succeeds, the IRK flag is the only factor determining whether IRK is distributed, whereas the MITM flag and the IO Cap of Initiator relates to the pairing protocol used, which in turn determines the presence of user confirmation. All the combinations of these 3 factors were exhaustively analyzed.

4.1 Pairing over BLE

See Table 4.1[1].

4.2 Pairing over BR/EDR

See Table 4.2.

[1] The first 3 columns show the configuration used and the last 2 columns shows whether the analyzed property holds.

Table 4.1: IRK-theft-proofness Over BLE

IRK	MITM	IO Caps	Careless Implementation	Careful Implementation
True	True	DisplayOnly	False	False
True	True	DisplayYesNo	False	False
True	True	KeyboardOnly	False	False
True	True	NoInputNoOutput	False	False
True	True	KeyboardDisplay	False	False
True	False	DisplayOnly	False	False
True	False	DisplayYesNo	False	False
True	False	KeyboardOnly	False	False
True	False	NoInputNoOutput	False	False
True	False	KeyboardDisplay	False	False
False	True	DisplayOnly	False	True
False	True	DisplayYesNo	False	True
False	True	KeyboardOnly	False	True
False	True	NoInputNoOutput	False	True
False	True	KeyboardDisplay	False	True
False	False	DisplayOnly	False	True
False	False	DisplayYesNo	False	True
False	False	KeyboardOnly	False	True
False	False	NoInputNoOutput	False	True
False	False	KeyboardDisplay	False	True

4.3 Interpretation

It is noticeable that, with the careless implementation, an adversary is almost always able to steal the IRK, only except when pairing happens over BR/EDR and the Initiator, with IO Caps *DisplayYesNo*, sets the MITM flag. The strategy of the adversary is actually pretty simple, which is to always claim that it has IO Cap of *NoInputNoOutput*, and always set the IdKey flag in the *Initiator Key Distribution* field of *SMP Pairing Response*. With this strategy, the adversary can always force the Initiator to choose JW as the pairing protocol, which circumvents the need for user confirmation under most circumstances, with the exception of the case where pairing takes place over BR/EDR and the Initiator has the IO Cap of *DisplayYesNo*. The exception exists because the existence of user confirmation is guaranteed under that circumstance, regardless of whether IRK is distributed.

With the careful implementation, the Initiator will not leak its IRK if it properly sets its IdKey flag. On the other hand, it will still send out its IRK, willingly and silently, if the IdKey flag is not set properly, which, as we will discuss below, may be more common than one may expect.

Table 4.2: IRK-theft-proofness Over BR/EDR

IRK	MITM	IO Caps	Careless Implementation	Careful Implementation
True	True	DisplayOnly	False	False
True	True	DisplayYesNo	True	True
True	True	KeyboardOnly	False	False
True	True	NoInputNoOutput	False	False
True	False	DisplayOnly	False	False
True	False	DisplayYesNo	False	False
True	False	KeyboardOnly	False	False
True	False	NoInputNoOutput	False	False
False	True	DisplayOnly	False	True
False	True	DisplayYesNo	True	True
False	True	KeyboardOnly	False	True
False	True	NoInputNoOutput	False	True
False	False	DisplayOnly	False	True
False	False	DisplayYesNo	False	True
False	False	KeyboardOnly	False	True
False	False	NoInputNoOutput	False	True

4.4 Incorrect Flag Configuration

With confidentiality at the first place, it is quite unlikely that people pay sufficient attention to the privacy features of Bluetooth, nor to their corresponding issues, until a pandemic sweeps the world and governments decided to develop apps to track where people have been to and the people who they have been in close contact with, before they test positive.

Even the specification itself does not seem to have paid too much attention to this privacy feature, as otherwise it should have highlighted what the consequences could be if one sets/unsets the IdKey flag. In [1], the IdKey flag that controls the distribution of IRK is introduced in Vol. 3 Part H section 3.5-3.6, but one needs to go more than 1000 pages back to Vol. 1 Part A section 5.4.5 to find a high level description of the purpose of IRK.

What's more, it is worth re-emphasizing that the user interactions during phase 2 of pairing is not meant for the keys distributed during TSKD, but for the LK/LTK. In other words, when the user presses the confirm button to allow pairing to proceed, it might be the case that all he/she wants is a one-time connection to transmit something to the other device, and he/she has no idea that such a confirmation may enable the other device to detect his/her presence in proximity until the IRK of his/her device is changed (which never happens in certain implementations of Bluetooth, such as the one on Android phones until factory reset [3]). Hence, even in the cases where the

property IRK-theft-proof is verified, the user may still end up surrendering its IRK without awareness.

At the end of the day, it is usually up to the implementation to set up the flags properly so that users wouldn not have to make a compromise between privacy and usability. However, without clear documentation, the developers are unlikely to bother thinking too much about the consequences of sharing one more non-critical key with other devices.

4.5 Inconsistencies between BR/EDR & BLE

As we already mentioned, JW over BR/EDR requires that the device with sufficient IO Cap ask for user confirmation during phase 2 of pairing. It was also because of this requirement, that BR/EDR did slightly better in terms of the IRK-theft-proofness.

On the other hand, if one looks closely at the Tables 4.1 and 4.2, he/she will find that the number of rows in Table 4.2 is actually less than that in Table 4.2. And this is because, apart from the 4 IO Cap values available in BR/EDR, BLE also supports the IO Cap of *DisplayKeyboard*. In BLE, a device sets its IO Cap to *DisplayKeyboard* if it has access to both a display and a keyboard (Vol. 3 Part H section 2.3.2 of [1]), whereas the same device will set its IO Cap to *DisplayYesNo* in BR/EDR (Vol. 3 Part C section 5.2.2.5 of [1]).

If we consider the case where the device A sets its MITM flag and wants to pair with a device B of IO Cap DesplayOnly, we will find that the device A can only trigger an unauthenticated JW when pairing over BR/EDR, in contrast to an authenticated PE while pairing over BLE.

In consideration of the ability to derive LK and LTK from each other via CTKD, these asymmetries seem pointless.

Chapter 5

Concluding Remarks

In this project, we used a Tamgram-based model to show that there are many scenarios where an adversary can acquire the IRK of a device without its user's awareness. Furthermore, we also argued that the design of Bluetooth does not seem as serious about its privacy feature as it is about other security properties, and there is a lack of emphasis on the implications of the distribution of IRK. As a result, we think Bluetooth Core Specification should document more explicitly about the expected use cases of its privacy features, as well as the privacy implications of different protocol configurations, to reduce the likelihood of inappropriate implementations.

More generally, in consideration of the current trend of adoption of Bluetooth-based offline tracking devices, we think SIG should revise the security model of its privacy features to adapt to the new use cases.

In addition, we identified several inconsistencies/asymmetries between the design of pairing over BR/EDR and pairing over BLE. In light of the unification of pairing process over the 2 transports introduced by SC and CTKD, we think these inconsistencies/asymmetries are pointless and therefore should be resolved. In particular, we think that DisplayKeyboard should also be added as a valid IO Cap value for pairing over BR/EDR, and JW over BLE should also require that the device with sufficient IO Cap ask for user confirmation during phase 2 of pairing.

Bibliography

[1] B. SIG, "Bluetooth core specification," 2021. [Online]. Available: https://www.bluetooth.com/specifications/specs/core-specification-5-3/ [Cited on pages 1, 3, 5, 7, 8, 9, 11, 14, 17, 19, 20, 21, 22, 29, and 30.]

[2] D. Antonioli, N. O. Tippenhauer, K. Rasmussen, and M. Payer, "Blurtooth: Exploiting cross-transport key derivation inbluetooth classic and bluetooth low energy," 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3488932.3523258 [Cited on pages 1 and 5.]

[3] Z. Brighton-Knight, J. Mussared, and A. Tiu, "Linkability of rolling proximity identifiersin google's implementation of theexposure notification system," 2021. [Online]. Available: https://github.com/alwentiu/contact-tracing-research/blob/main/GAEN.pdf [Cited on pages 1 and 29.]

[4] "Airtag." [Online]. Available: https://en.wikipedia.org/wiki/AirTag [Cited on page 2.]

[5] "Smarttag." [Online]. Available: https://en.wikipedia.org/wiki/AirTag [Cited on page 2.]

[6] T. Yu, J. Henderson, A. Tiu, and T. Haines, "Privacy analysis of samsung's crowd-sourced bluetoothlocation tracking system," 2022. [Online]. Available: https://arxiv.org/pdf/2210.14702.pdf [Cited on page 2.]

[7] J. Mussared and A. Tiu, "Covidsafe-cve-2020-12856: A silent pairing issue in bluetooth-based contact tracing apps," 2020. [Online]. Available: https://github.com/alwentiu/COVIDSafe-CVE-2020-12856 [Cited on page 2.]

[8] "Tamgram." [Online]. Available: https://github.com/Tamgram/tamgram#tamgram [Cited on pages 2 and 4.]

[9] J. Wu, R. Wu, D. Xu, D. Tian, and A. Bianchi, "Formal model-driven discoveryof bluetooth protocol design vulnerabilities," 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9833777 [Cited on pages 2, 4, 5, 7, and 12.]

[10] T. Claverie, G. Avoine, S. Delaune, and J. L. Esteves, "Tamarin-based analysis of bluetooth uncovers twopractical pairing confusion attacks," 2023. [Online]. Available: https://hal.science/hal-04079883 [Cited on pages 2, 4, 5, 7, and 12.]

[11] T. T. Team, "Tamarin-prover manual," 2023. [Online]. Available: https://tamarin-prover.github.io/manual/master/tex/tamarin-manual.pdf [Cited on page 4.]

[12] D. Li, "Tamgram: A frontend for large-scale protocol modeling in tamarin," unpublished. [Cited on page 5.]

[13] J. Wu, Y. Nan, V. Kumar, D. J. Tian, A. Bianchi, M. Payer, and D. Xu, "Tamarin-prover manual," 2020. [Online]. Available: https://www.usenix.org/conference/woot20/presentation/wu [Cited on page 5.]

[14] M. von Tschirschnitz, L. Peuckert, F. Franzen, and J. Grossklags, "Method confusion attack on bluetooth pairing," 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9519477 [Cited on page 12.]

[15] "Dolev-yao model." [Online]. Available: https://w.wiki/7ucz [Cited on page 14.]