Formal Analysis of BLE Secure Connection Pairing and Revelation of the PE Confusion Attack

Min Shi*, Yongkang Xiao*, Jing Chen*✉, Kun He*, Ruiying Du*, Meng Jia†
*Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education,
School of Cyber Science and Engineering, Wuhan University.
✉Corresponding author: Jing Chen, email: {itachi, xiaoyongkang, chenjing, hekun, duraying}@whu.edu.cn
†Department of Computing, The Hong Kong Polytechnic University. email: jiameng@comp.polyu.edu.hk

***Abstract*—The Secure Connection (SC) pairing is the latest version of the security protocol designed to protect sensitive information transmitted over Bluetooth Low Energy (BLE) channels. A formal and rigorous analysis of this protocol is essential for improving security assurances and identifying potential vulnerabilities. However, the complexity of the protocol flow, difficulties in formalizing pairing method selection, and overly idealized user assumptions present significant obstacles to such analysis. In this paper, we address these challenges and present an accurate and comprehensive formal analysis of the BLE-SC pairing protocol using Tamarin. We extract state machines for each participant as the blueprint for modeling the protocol, and we use an equational theory to formalize the pairing method selection logic. Our model incorporates subtle user behaviors and considers stronger adversary capabilities, including the potential compromise of private channels such as the temporary out-of-band channel. We develop a verification strategy to automate protocol analysis and implement a script to parallelize verification tasks across multiple servers. We verify 84 pairing cases and identify the minimal security assumptions required for the protocol. Moreover, our results reveal a new Man-in-the-Middle (MitM) attack, which we call the PE confusion attack. We provide tools and Proof-of-Concept (PoC) exploits for simulating and understanding this attack within a controlled environment. Finally, we propose countermeasures to defend against this attack, improving the security of the BLE-SC pairing protocol.**

I. Introduction

Bluetooth Low Energy (BLE), introduced in 2010 [1], is a wireless standard designed to connect devices with minimal energy consumption. BLE-enabled devices are widely used in the world, with annual shipments expected to reach 7.37 billion by 2027 [2].

The popularity of BLE has brought attention to private data, such as keyboard strokes and health data in wearable devices, transmitted over BLE links. To establish a secure connection, two devices execute the BLE Secure Connection (BLE-SC) pairing protocol, possibly with the assistance of a user or a temporary Out-of-Band (OOB) channel such as Near-Field Communication, to authenticate each other and establish a shared Long-Term Key (LTK). The LTK is then used to derive a Session Key (SK) to encrypt the Link Layer (LL) connection.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240779
www.ndss-symposium.org

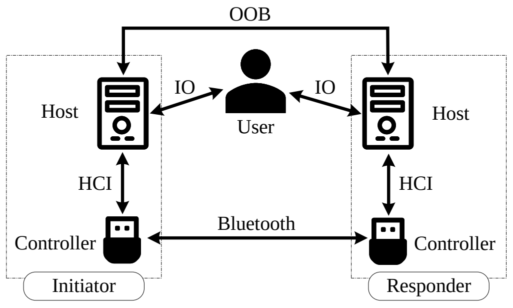


Fig. 1. Detailed BLE architecture with HCI channels.

A thorough formal analysis of the BLE-SC pairing protocol can help increase confidence in the security of BLE connections. However, there are three challenges when formally analyzing the BLE-SC pairing protocol.

The first challenge concerns modeling the complex interaction flow. To have better flexibility, interoperability, and efficiency when implementing Bluetooth technology across various devices, the Bluetooth specification [3] splits the BLE stack into the host part and the controller part, and defines an interactive interface, Host Controller Interface (HCI), to connect them. The pairing protocol involves five agents and six channels between them, as shown in Fig. 1. Existing works [4], [5], [6] implicitly assume that the HCI channel within the device is secure and model the device as a single entity to simplify the model. Unfortunately, these simplified models fail to faithfully capture the behaviors and interactions in pairing, since the HCI channel can be controlled through technologies such as Bumble [7].

The second challenge involves formalizing the selection logic of the association model, also known as the pairing method. The pairing protocol defines four association models for device authentication. The specific association model used in the pairing process depends on the capabilities and security requirements of both devices. Although the specification explicitly specifies the selection logic in tabular form, translating it into a formal representation remains challenging. Most existing models [4], [5] overlook this selection logic. Shi et al. [6] attempt to formalize it through individual rules for every possible combination of capabilities and security requirements. However, their approach contains 101 rules to model the state machine of the Secure Manager (SM), which executes on the host part. This results in a model that is too complex to be effectively integrated into the architecture depicted in Fig. 1.

The last challenge relates to assumptions about user actions. The pairing protocol may involve a user in the pairing process by asking the user to compare or enter numbers. The security in these cases relies on the assumptions that the user only confirms on BLE devices when both devices display the same numbers, and that the user faithfully enters the displayed number on one device to the other. Additionally, the specification implicitly assumes that users always select random numbers and never reuse them. However, Tschirschnitz et al. [8] point out that users may deviate from the idealized user assumption defined in the specification. Therefore, we need more appropriate user assumptions to capture real-world situations and accurately analyze the security of the protocol.

In this paper, we aim to overcome the aforementioned challenges and perform an accurate and comprehensive formal analysis of the BLE-SC pairing protocol using Tamarin [9]. First, we delve into the specification to extract the state machines for each participant shown in Fig. 1. Our model incorporates various channel types connecting these participants. Additionally, we consider a powerful adversary capable of compromising the confidentiality and integrity of private channels, such as HCI, Input/Output (IO), and OOB. Second, considering interactions between devices and users, we refine the four association models into six. We define two function symbols to formalize the selection logic and specify their properties using 23 equations. Third, we make more subtle user assumptions, including the possibility that users may select easily guessable numbers or reuse numbers they randomly chose in previous sessions, while Shi et al. [6], Wu et al. [4], and Jangid et al. [5] consider only the user assumptions outlined in the specification and the user study [8].

The novel aspects of our model are that it offers a highly fine-grained formalization capturing the detailed behavior of all participants and users, while also considering a stronger attacker model. By verifying this fine-grained model, we identify the minimal assumptions necessary to achieve the security goals for each pairing case. For instance, when the OOB channel between devices is unidirectional, security is ensured only if the channel provides both confidentiality and integrity. In contrast, with a bidirectional OOB channel, the protocol remains secure even if an adversary compromises the confidentiality of this channel. Furthermore, benefiting from our customized equation theory, we can model the complex state machine of the BLE-SC pairing protocol, rather than represent each device as implementing only a specific association model and combine different devices to analyze security properties when different association models are used in the same pairing session like Wu et al. [4] and Jangid et al. [5]. We would like to highlight that our customized equation theory can serve as an inspiration for other researchers. Specifically, this theory, which formalizes association model selection logic, can be extended to formalize additional protocol behaviors, such as protocol version selection and more complex branching logic.

We conduct an automated analysis of the BLE-SC pairing protocol using customized heuristics and an auxiliary formula. We design verification algorithms that leverage previously verified results to expedite the analysis process. In addition to identifying previously known attacks, we discovered a new Man-in-the-Middle (MitM) attack, called Passkey Entry (PE) confusion attack. By investigating the User Interface (UI) of Bluetooth devices, we found that the existing interface would prompt users to enter bad numbers, increasing security risks.

Contributions. Our main contributions are as follows:

- We present the most detailed symbolic model of the BLE-SC pairing protocol, modeling the host and controller components of BLE devices as separate entities, in alignment with the protocol specification. Furthermore, we formalize the association model selection logic within our model.
- We make more subtle user assumptions than prior works. We aim to clarify the relationship between user actions and the security properties of the protocol. Our model captures known attacks, such as the method confusion attack, as well as a novel MitM attack, which we term the PE confusion attack. To facilitate understanding and simulation of this attack, we provide tools and proof-of-concept (PoC) exploits within a controlled environment. We propose countermeasures designed to enhance the security of the BLE-SC pairing protocol against such attacks.
- We verify the pairing cases for devices with different capabilities and security requirements. We make 84 models for these pairing cases. For each pairing case, we find the minimal security assumptions to ensure security.

II. Background

A. Overview of BLE-SC Pairing Protocol

The BLE-SC pairing protocol involves two devices: the initiator and the responder, each comprising a host and a controller connected by the HCI channel, as illustrated in Fig. 1. The HCI channel transmits three types of messages: Command, Event, and ACL (Asynchronous Connection-oriented Logical transport). Command messages enable the host to send commands to the controller. The controller responds to the host with Event messages indicating the command's completion or requesting further messages. ACL messages carry pairing protocol packets between the host and the controller.

Controllers communicate mutually over the BLE channel, broadcasting messages or exchanging point-to-point packets. In this process, ACL messages from the HCI channel are transmitted to the peer controller via LL Data (LD) packets, which are then forwarded to the host through the HCI channel as ACL messages. Additionally, controllers can exchange LL Control (LC) packets to initiate an encrypted link connection or update connection parameters. Hosts use the IO channel to communicate with the user, either by displaying messages on the screen or accepting input from the keyboard. When both devices support OOB communication, the host can communicate directly through the OOB channel.

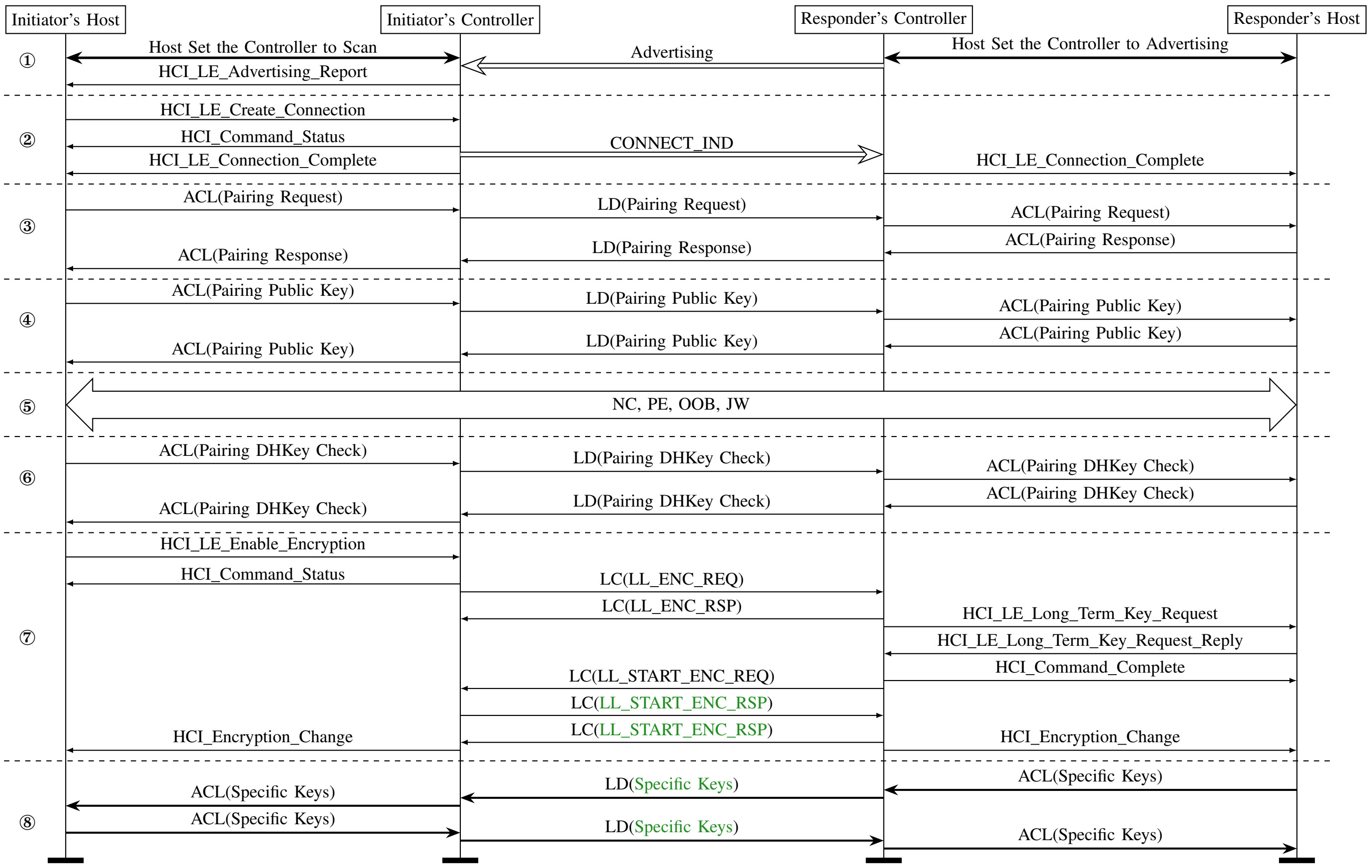


Fig. 2. Message sequence chart of the BLE-SC pairing protocol.

Advertising, Scanning, and Link Establishment. As shown in stages ①-② in Fig. 2, hosts send commands to their controllers to enter the scanning and advertising states, respectively. The initiator discovers the responder, which broadcasts its Bluetooth address. Then, the host of the initiator commands the controller to establish a BLE link with the peer device. Once this link is established, both controllers send a connection complete event to their respective hosts with a connection handle to identify the link.

Security Manager Protocol (SMP). The hosts of the two devices execute the SMP to authenticate each other and negotiate an LTK, which is then provided to the controller to enable encryption of the BLE link. Once encrypted, the BLE link allows the hosts to exchange keys for specific purposes, such as the Connection Signature Resolving Key (CSRK) used for authenticating unencrypted communication. We outline the three phases of SMP in the following.

1) *Pairing Feature Exchange (Stage ③).* The hosts exchange pairing request and response messages by transmitting ACL messages over the HCI channel. These messages contain fields to indicate the IO capabilities (IO_I and IO_R), whether OOB data has been received, whether MitM protection is required, and the maximum encryption key size of the two devices. The first three fields are crucial for selecting the association model in the next phase, while the final field is used to negotiate the encryption key size.
2) *LTK Generation (Stages ④-⑥).* Devices (BT_I and BT_R) negotiate an LTK during this phase, using three pseudorandom functions `f4`, `f5`, and `f6` (detailed in Appendix A). At the first stage (stage ④), two hosts exchange their Diffie-Hellman (DH) public keys, Pk_I and Pk_R, to share a DH key $DHKey$. At the second stage (stage ⑤), they execute one of the four association models and exchange two random values N_I, N_R as well as two numbers r_I, r_R. The third stage (stage ⑥) derives the $MacKey$ and an intermediate LTK' by $MacKey||LTK' = \mathtt{f5}(DHKey, N_I, N_R, BT_I, BT_R)$. The two hosts exchange and verify the confirmations E_I and E_R, which are calculated by
$$E_I = \mathtt{f6}(MacKey, N_I, N_R, r_R, IO_I, BT_I, BT_R),$$
$$E_R = \mathtt{f6}(MacKey, N_R, N_I, r_I, IO_R, BT_R, BT_I).$$
The LTK' is then resized to the negotiated encryption key size to get the final LTK.
3) *Transport Specific Key Distribution (Stage ⑧).* The two hosts exchange their specific keys over the encrypted BLE link after receiving an encryption change event.

Encrypted Link Establishment. The host of the initiator begins this phase (stage ⑦) by sending an enable-encryption command, along with the LTK, to its controller. The controller then generates a random Session Key Diversifier SKD_I for the initiator and transmits it to the responder's controller within an LL_ENC_REQ packet. In response, the responder's controller sends its session key diversifier, SKD_R, in the LL_ENC_RSP packet and requests the LTK from its host via an LTK request event. Both controllers derive the session key $SK = \texttt{KDF}(SKD_I || SKD_R, LTK)$. Using this session key, the controllers encrypt the LL_START_ENC_RSP packet and exchange it over the BLE link. Finally, each controller sends an encryption change event to its host, indicating that the BLE link is now encrypted.

B. Association Models

In the pairing protocol, association models play a crucial role in establishing a secure connection. At the stage ⑤, devices select one of four association models — Passkey Entry (PE), Numeric Comparison (NC), Out-of-Band (OOB), and Just Work (JW) — based on the first three fields in the pairing request and response messages. We provide a detailed description of the PE association model along with concise summaries of other association models. A full description of all the association models and an excerpt of the selection logic are available in Appendix B.

In the PE association model, illustrated in Fig. 3, two hosts share a 6-digit number $passkey$ with the assistance of the user, using one of two methods. In the first method, one host displays a randomly generated $passkey$, and the user is prompted to enter this number on the other host. In the second method, both hosts prompt the user to enter a self-selected number. Once the user enters the $passkey$, the hosts assign it to the numbers r_I and r_R. For each bit of the $passkey$, both hosts independently generate random values N_I^i and N_R^i, and calculate the pairing confirmations $C_I^i = \texttt{f4}(Pk_I, Pk_R, N_I^i, r_I^i)$ and $C_R^i = \texttt{f4}(Pk_R, Pk_I, N_R^i, r_R^i)$. After exchanging C_I^i and C_R^i, the initiator's host sends the N_I^i to the responder's host, enabling verification of C_I^i. If this verification succeeds, the responder's host then sends the N_R^i to the initiator's host, which subsequently verifies C_R^i. This process is repeated 20 times, covering each bit of the $passkey$. Upon completing 20 rounds, the hosts finish the PE association model by setting $N_I = N_I^{20}$ and $N_R = N_R^{20}$.

In the NC association model, each host calculates a 6-digit number and displays it, prompting the user to confirm if the two numbers match by pressing a button on each device. The OOB association model uses a temporary secure channel to transmit the authentication data. For the JW association model, the protocol flows are the same as the NC association model, but always assume that the generated numbers match. It is important to highlight that the JW association model does not provide any authentication.

C. Tamarin Prover

We analyze the BLE-SC pairing protocol using Tamarin Prover [10], [9], a powerful symbolic verification tool that is widely used for the analysis of real-world security protocols such as TLS 1.3 [11], [12], [13], 5G-AKA [14], [15], and

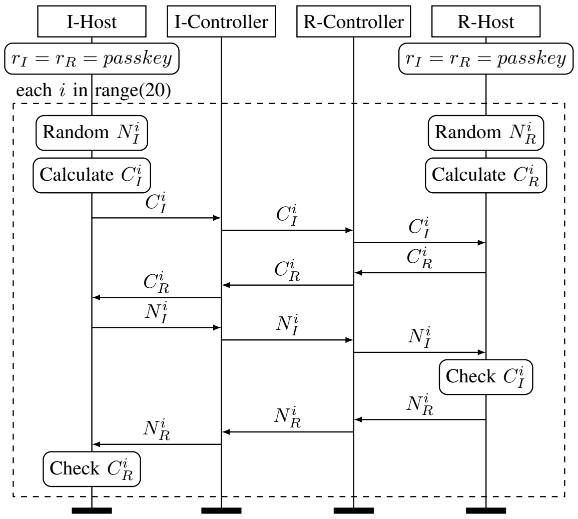


Fig. 3. Overview of the PE Association Model Flow.

EMV [16], [17]. Tamarin accepts as input a security protocol model, which specifies the adversary's behavior as well as the actions performed by agents in various roles (such as the initiator, the responder, and the user), along with the desired properties of the protocol. It verifies whether the provided model satisfies the desired properties by attempting to construct a counterexample that violates the property. Below, we provide a brief overview of Tamarin, and for further details, we refer the reader to the Tamarin manual [18].

Messages as Terms. In Tamarin, messages are represented as terms constructed from variables, constants, and function symbols. Cryptographic primitives are modeled using abstract function symbols, with their properties formalized by equations over terms. For example, the function symbols `senc` and `sdec` represent the symmetric encryption and decryption functions, respectively. The equation $\texttt{sdec(senc(}m\texttt{,}k\texttt{),}k\texttt{)=}m$ formalizes the property that symmetric decryption with the same key reverses symmetric encryption. Tamarin supports various built-in equational theories for modeling common cryptographic primitives, such as DH groups. Besides, Tamarin allows users to define functions in the form `funName`/n, where `funName` is the function name and n is the number of function parameters. Particularly, when $n = 0$, the `funName` represents a constant.

Protocols and Adversary as Rules. Tamarin uses Multiset Rewriting (MR) rules to model the protocol agents and the adversary as a Labeled Transition System (LTS), whose state is a multiset of facts. The initial state of this LTS is an empty multiset, and rules define how the LTS transitions to new states. Facts are atomic predicates applied to message terms, encoding the states of agents and the adversary. An MR rule is written as $[\ L\ ] \!-\![\ A\ ]\!\!\rightarrow [\ R\ ]$. When a rule is applied, it consumes the left-side facts L in the current state and produces right-side facts R. The application of a rule is recorded in the *trace* by appending the action facts to the trace. Persistent facts prefixed with a '!' can be consumed multiple times.

Properties as first-order formulas. Tamarin defines security properties as trace properties by using first-order logic formulas over action facts and timepoints (writing as $\tilde{i}$, $\tilde{j}\ldots$). For example, to specify that a fresh value n is unique across all rules labeled with the action fact `Dist(n)`, we can use a first-order formula that asserts the following: if an action with the same fresh value appears twice, both instances must occur at the same timepoint (i.e., $\tilde{i} = \tilde{j}$). This property is expressed as:

$$\forall\ n\ \tilde{i}\ \tilde{j}.\ \texttt{Dist}(n)\ @\tilde{i}\ \wedge\ \texttt{Dist}(n)\ @\tilde{j} \Rightarrow (\tilde{i} = \tilde{j}).$$

III. Protocol Modeling

We outline the assumptions underlying our model and present the core of our formal model, including the channels, the association model selections, and the participants.

A. Assumptions

Adversary and Cryptography Assumptions. We consider the Dolev-Yao adversary [19] who can read, intercept, reorder, replay, and send any message in public channels. The adversary's actions are constrained solely by the limitations of the cryptographic methods in use. We use standard symbolic models of the cryptographic primitives. The properties of the cryptographic primitives are explicitly specified by the equations over function symbols and terms.

Furthermore, to accurately capture the property of the function `f4`, defined by

$$\begin{aligned}&\texttt{f4}(x1, x2, x3, x4)\\ =&\texttt{AES}_{x3}(\texttt{AES}_{x3}(\texttt{AES}_{x3}(x1) \oplus x2) \oplus x4 \oplus \texttt{KDF}(x3)),\end{aligned}$$

we introduce a function `brkf4` to retrieve the fourth parameter when given the first three parameters and the output of `f4`. The equation to specify the above property is

$$\texttt{brkf4}(C = \texttt{f4}(x1, x2, x3, x4), x1, x2, x3) = x4.$$

This equation captures the core property of the function `brkf4` while abstracting from specific implementation details. It provides an abstraction for calculating $x4$ by

$$x4 = \texttt{AES}^{-1}_{x3}(C) \oplus \texttt{AES}_{x3}(\texttt{AES}_{x3}(x1) \oplus x2) \oplus \texttt{KDF}(x3).$$

User and Device Assumptions. We consider an honest user aiming to pair two trustworthy BLE devices. We assume that the user can compare numbers and input the displayed number. Besides, the user can input a self-selected number into both devices when prompted to do so. As for the $passkey$ used in the PE association model, the Bluetooth specification [3] states that it should be randomly selected and not reused. We assume that the devices comply with this specification. However, it is difficult to ensure that all users randomly choose their numbers and do not reuse them. In our model, we assume that the user faithfully inputs the displayed number but may reuse a randomly selected number or use a guessable number.

Channel Assumptions. The BLE channel between controllers is considered public and is assumed to be fully controlled by a Dolev-Yao adversary. In contrast, the OOB, IO, and HCI channels are assumed to be private, providing both confidentiality and integrity. To evaluate the necessity of these assumptions, we model scenarios in which the adversary is allowed to compromise confidentiality by reading information from these channels, as well as integrity by injecting or modifying messages over these channels.

B. Channels

To model private channels, we define a generalized channel model that can be instantiated for various channel types. Furthermore, we define two rules to enable the adversary to break the confidentiality and integrity of these channels.

Generalized Channel Model. We define the following two rules to model the generalized communication model.

$$\begin{aligned}&\texttt{S2DSrc2Chn}:\\ &\quad [\ \texttt{Out\_S2D}(chanType, Src, Dst, msg)\ ] \dashv[\ ]\!\!\rightarrowtail\\ &\quad [\ \texttt{!ChanState}(chanType, Src, Dst, msg)\ ]\\ &\texttt{S2DChn2Dst}:\\ &\quad [\ \texttt{!ChanState}(chanType, Src, Dst, msg)\ ] \dashv[\ ]\!\!\rightarrowtail\\ &\quad [\ \texttt{In\_S2D}(chanType, Src, Dst, msg)\ ]\end{aligned}$$

The rule `S2DSrc2Chn` models the process of sending messages to the channel, while the rule `S2DChn2Dst` represents the reception of messages from the channel. Messages are transmitted from message sources to channels through the first rule, and are delivered from the channel to the message destination using the second rule. The facts `Out_S2D` and `In_S2D` have four terms, representing the channel type, the message source, the message destination, and the message.

Channel Types. As shown in Fig. 2, there are four channel types in the BLE-SC pairing protocol: OOB, IO, HCI, and BLE. To model the BLE channel as a public channel, we use Tamarin's built-in facts, `Out` and `In`, to model sending and receiving of messages over the BLE channel. For the other channels, we use `Out_S2D` and `In_S2D`, as defined in our generalized channels model, to model sending and receiving of messages over these channels. By assigning the variable $channelType$ to 'OOB', the generalized channel is instantiated as an OOB channel. To achieve a more fine-grained channel model, we further divide the $channelType$ into $\langle mainType, subType\rangle$. For the IO channel, the $mainType$ is 'IO', and the $subType$ specifies the specific IO method, such as 'Display' and 'DisplayWithYN' (display a number with yes/no buttons). For the HCI channel, the $mainType$ is 'HCI', and the $subType$ further specifies the HCI types as 'Command', 'Event', and 'ACL'.

Break Channels. To enable the adversary to compromise private channels, we define the following two rules.

$$\begin{aligned}&\texttt{ChannelBrkC}:\\ &\quad [\ \texttt{!ChanState}(chanType, Src, Dst, msg)\ ]\\ &\ \dashv[\ \texttt{BrkC}(chanType, Src, Dst)\ ]\!\!\rightarrowtail\\ &\quad [\ \texttt{Out}(chanType, Src, Dst, msg)\ ]\end{aligned}$$

$$
\begin{aligned}
&\texttt{ChannelBrkI}:\\
&\quad [\ \texttt{In}(\langle \mathit{chanType}, \mathit{Src}, \mathit{Dst}, \mathit{msg}\rangle)\ ]\\
&\ -\![\ \texttt{BrkI}(\mathit{chanType}, \mathit{Src}, \mathit{Dst})\ ]\!\!\rightarrow\\
&\quad [\ \texttt{In\_S2D}(\mathit{chanType}, \mathit{Src}, \mathit{Dst}, \mathit{msg})\ ]
\end{aligned}
$$

The first rule `ChannelBrkC` sends messages from the private channel to the public channel, enabling the adversary to know the message and thereby compromising confidentiality. The second rule `ChannelBrkI` models that messages from the private channel can be relayed from the public channel, allowing the adversary to inject the message into private channels and compromise their integrity.

With the above two rules, the adversary can break the confidentiality and integrity of all private channels. To precisely extend the adversary's capabilities, we label these rules with action facts `BrkC` and `BrkI`, each including terms that specify the channel type, message source, and message destination. These action facts enable the definition of predicates used to define security properties under different channel assumptions. For example, the following two predicates represent that the adversary can break the confidentiality and integrity of the OOB channel, respectively.

$$
\begin{aligned}
\mathsf{BrkOOBC}() &\Leftrightarrow \exists\ \mathit{src}\ \mathit{dst}\ \tilde{i}.\ \texttt{BrkC}(\text{‘OOB’}, \mathit{src}, \mathit{dst})@\tilde{i}\\
\mathsf{BrkOOBI}() &\Leftrightarrow \exists\ \mathit{src}\ \mathit{dst}\ \tilde{i}.\ \texttt{BrkI}(\text{‘OOB’}, \mathit{src}, \mathit{dst})@\tilde{i}
\end{aligned}
$$

C. Association Model Selection

Devices select an association model according to the pairing request and response messages. The latest Bluetooth Core Specification [3] (Vol 3, Part H, 2.3.5.1 Selecting key generation method, p1640) defines two tables (Table 2.7 and Table 2.8, detailed in Appendix B) to describe the association model selection logic. To model this association model selection logic, we define two functions `selectAM` and `mapIOCaps2AM`. Parameters of the function `selectAM` are the OOB flags, the MitM flags, and the IO capabilities of the two devices. For the IO capabilities, the Bluetooth Core Specification [3] defines five types of IO capabilities:

- NoInputNoOutput: The device neither provides a means for the user to indicate ‘yes’ or ‘no’, nor is it capable of displaying a 6-digit decimal number.
- DisplayOnly: The device is capable of displaying a 6-digit decimal number, but does not provide a means for the user to indicate ‘yes’ or ‘no’.
- KeyboardOnly: The device allows the user to input a 6-digit decimal number, but lacks the capability to display such a number.
- DisplayYesNo: The device can display a 6-digit decimal number and enables the user to indicate ‘yes’ or ‘no’, but it does not allow input of numbers.
- KeyboardDisplay: The device can display a 6-digit decimal number and enables user input, including both numeric entries and confirmation via ‘yes’ or ‘no’.

The function `mapIOCaps2AM` takes the IO capabilities of the two devices and maps them to the corresponding association model. Properties of these two functions are formalized with 23 equations shown in Fig. 4. Note that for the PE association model, we subdivide it into three different cases:

```
functions:
  mapIOCaps2AM/2, selectAM/6,
// Constants: functions with 0 argument
  // IO capabilities
  DisplayOnly/0, DisplayYesNo/0, KeyboardOnly/0,
    NoInputNoOutput/0, KeyboardDisplay/0,
  // Association models
  JW/0, NC/0, OOB/0, PEII/0, PEID/0, PEDI/0,
  // Flag value
  True/0, False/0
// selectAM(OOBFlagI, OOBFlagR, MITMflagI, MITMflagR
    , IOCapI, IOCapR)
equations:
  selectAM(True,x2,x3,x4,x5,x6)              = OOB,
  selectAM(x1,True,x3,x4,x5,x6)              = OOB,
  selectAM(False,False,False,False,x5,x6)     = JW,
  selectAM(False,False,True,x4,x5,x6)
                          = mapIOCaps2AM(x5,x6),
  selectAM(False,False,x3,True,x5,x6)
                          = mapIOCaps2AM(x5,x6),
  mapIOCaps2AM(DisplayOnly,DisplayOnly)       = JW,
  mapIOCaps2AM(DisplayOnly,DisplayYesNo)      = JW,
  mapIOCaps2AM(DisplayOnly,KeyboardOnly)    = PEDI,
  mapIOCaps2AM(DisplayOnly,KeyboardDisplay)= PEDI,
  mapIOCaps2AM(DisplayYesNo,DisplayOnly)      = JW,
  mapIOCaps2AM(DisplayYesNo,DisplayYesNo)     = NC,
  mapIOCaps2AM(DisplayYesNo,KeyboardOnly)   = PEDI,
  mapIOCaps2AM(DisplayYesNo,KeyboardDisplay) = NC,
  mapIOCaps2AM(KeyboardOnly,DisplayOnly)    = PEID,
  mapIOCaps2AM(KeyboardOnly,DisplayYesNo)   = PEID,
  mapIOCaps2AM(KeyboardOnly,KeyboardOnly)   = PEII,
  mapIOCaps2AM(KeyboardOnly,KeyboardDisplay)=PEID,
  mapIOCaps2AM(KeyboardDisplay,DisplayOnly)= PEID,
  mapIOCaps2AM(KeyboardDisplay,DisplayYesNo) = NC,
  mapIOCaps2AM(KeyboardDisplay,KeyboardOnly)=PEDI,
  mapIOCaps2AM(KeyboardDisplay,KeyboardDisplay)
                                            =NC,
  mapIOCaps2AM(NoInputNoOutput,x)             = JW,
  mapIOCaps2AM(x,NoInputNoOutput)             = JW
```

Fig. 4. Model of association model selection.

- `PEII`: both devices prompt the user to input a number.
- `PEID`: the responder displays a number and the initiator prompts the user to input a number.
- `PEDI`: the initiator displays a number and the responder prompts the user to input a number.

This finer-grained division allows for a more precise modeling of host behavior, as the host can determine the appropriate action based on the selected association model.

The Tamarin prover allows users to define custom functions and equations to extend its built-in equational theory. However, user-defined equations must satisfy either the subterm convergence property or the finite variant property [20]. Equations that fall outside these theoretical classes may lead to non-termination or produce erroneous results during verification. Consequently, it is essential to ensure that all user-defined equations conform to either subterm convergence or the finite variant property. Although the equations we defined do not meet the subterm convergence property which equires that the right-hand side of an equation be either a strict subterm of

```
functions:
  sc/0, legacy/0, verSelect/2,
// verSelect(verFieldI, verFieldR)
equations:
  verSelect(sc,sc)      = sc,
  verSelect(legacy,x2) = legacy,
  verSelect(x1,legacy) = legacy
```

Fig. 5. Model of pairing version selection.

its left-hand side or a constant term., we verified that they possess the finite variant property using Cheval's automated tool FVPgen [21] and present the result in the supporting material [22]. This ensures that the defined equations are suitable for use in Tamarin, thereby enabling the formal verification of the BLE-SC pairing protocol.

To our knowledge, our work is the first to faithfully model the association model selection logic. Shi et al. [6] proposed the first model that covers all the possible execution paths based on the capabilities of the two devices. Unfortunately, they do not explicitly model the association model selection logic, but define a lot of rules and each rule is for a specific device pair.

Remark. Our modeling method for association model selection logic is extensible and can be applied to other aspects of protocol behavior, such as protocol version selection or other branching logic. For example, Fig. 5 illustrates how our method models the protocol version selected logic when considering both the legacy and SC pairing protocol. This approach can be extended to other protocols like EMV [17], where it can model kernel selection, and TLS 1.3 [12], where it can model handshake method selection. Furthermore, scenarios like key size negotiation [23], [24] can also benefit from this versatile modeling method.

D. Participants

Modeling the participants in the BLE-SC pairing protocol is non-trivial due to the inherent complexity of the protocol. Furthermore, the specification lacks explicit definitions for the participants' state machines. To address this, we extract the state machines of these participants from the specification and use them as the blueprint to help model the pairing protocol. To reduce the complexity of the model, we followed the approach of Jangid et al. [5] to abstract the PE association model into two rounds, rather than modeling all 20 rounds in full detail. This simplified approach captures the essential behavior of the hosts while maintaining the model's comprehensibility.

Hosts. Fig. 6 and Fig. 7 show the state machines of the initiator's and the responder's hosts, respectively. The initiator's host state machine comprises 21 states and 30 transitions, while the responder's host state machine contains 18 states and 27 transitions. Each transition corresponds to a specific rule used to model the hosts. The labels above the arrows in the state machine present the rule names. For example, in Fig. 6, to model the transition from state i_5, where the initiator's host

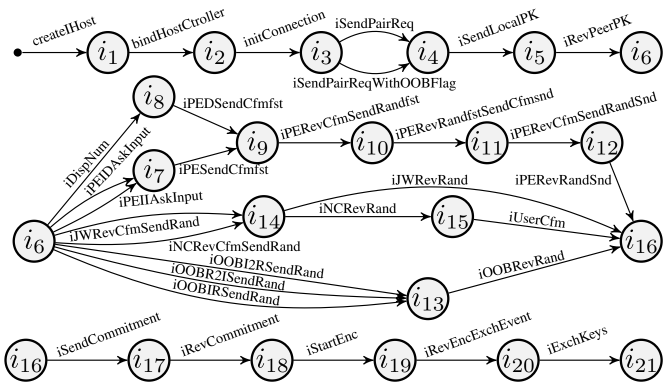


Fig. 6. State machine of initiator's host.

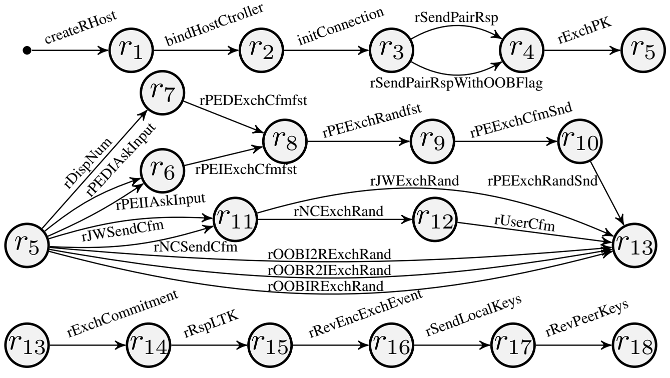


Fig. 7. State machine of responder's host.

has sent its public key, to state i_6, where the initiator's host has exchanged the public key, we define the following rule.

$$
\begin{aligned}
&\texttt{iRevPeerPK}:\\
&\quad [\ \texttt{HostWaitPeerPK}(i_5, \boldsymbol{L}, \boldsymbol{P}, ..., Lsk, ...),\\
&\quad\ \ \texttt{InS2D}(..., Ppk, ...)\ ] \mathrel{-}[\ \ ]\!\!\rightarrow\\
&\quad [\ \texttt{HostExchedPK}(i_6, ..., Lsk, Ppk, ..., AS)\ ]
\end{aligned}
$$

In this rule, Lsk and Ppk represent the local private key and the peer's public key from the initiator's view. The vector $\boldsymbol{L}$ contains the OOB flag, authentication requirement, and IO capabilities of the initiator's host. The vector $\boldsymbol{P}$ represents the corresponding information for the peer host. The AS is crucial for determining the branches that start from the state i_6. This value is computed by the function `selectAM()` using the parameters in $\boldsymbol{L}$ and $\boldsymbol{P}$. If $AS = \texttt{PEDI}$, the initiator's host executes the PE association model, where it displays a number, then the initiator's host transitions to the state i_8. This behavior is modeled by the rule `iDispNum` in Fig. 6.

Defining rules based on the state transitions illustrated in Figures 6 and 7 allows us to comprehensively model the behavior of the hosts. These state machines provide essential guidance in the modeling process by visually clarifying host behavior. Moreover, they enable a systematic approach to handling complex branches, helping to reduce errors and maintain accuracy throughout the model process.

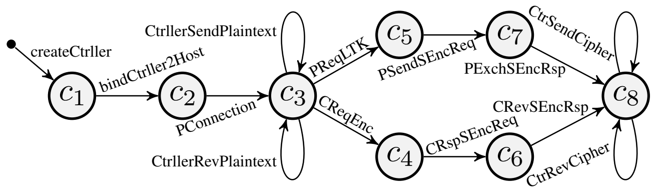


Fig. 8. State machine of controller.

Controllers The state machine of the controller is depicted in Fig. 8. In state c_3, the controller operates in a connection state, relaying plaintext data, such as pairing requests and public keys, between the LL and HCI. In state c_8, the controller is in the encryption state where it relays the ciphertext, such as the encrypted specific keys and application data, between the LL and HCI. The initiator's controller transitions from the connection state c_3 to the encryption state c_8 through states c_4 and c_6, while the responder's controller follows a path through states c_5 and c_7. After the link encryption begins, the controller uses AES-CCM to encrypt the LL packets. The AES-CCM key is derived by encrypting the SKD with the LTK, which is modeled as $SK = \texttt{KDF(}SKD\texttt{,}LTK\texttt{)}$, where the function `KDF` denotes a key derivation function. We use the built-in symmetric encryption theory to model the AES-CCM cipher. Moreover, we restrict the controller from accepting plaintext LL or LC packets after the link encryption has started.

IV. Formalizing Properties

In this section, we formalize the security properties of the pairing protocol under different assumptions. The pairing protocol is an authenticated key exchange protocol, and we principally consider its *authentication* properties and *secrecy* properties. Moreover, we also consider the *association model consistency* property, which is not verified by other formal works but is essential to the protocol's security.

A. Association Model Consistency

The association model consistency property guarantees that two devices execute the same association model during the identical pairing session. The importance of this property is inspired by the method confusion attack disclosed by Tschirsinger et al. [8]. In this attack, an adversary induces one device to execute the NC association model while the other executes the PE association model.

We formalize the association model consistency property by the following first-order logic formula.

$$\begin{aligned}
&\forall\ I\ R\ pidI\ pidR\ ASI\ ASR\ \tilde{i}\ \tilde{j}\ \tilde{k}.\\
&\ \texttt{DeviceConnect}(I, R, pidI, pidR)\ @\tilde{i}\\
&\quad \wedge \texttt{IFinishedAuth2}(I, pidI, ASI)\ @\tilde{j}\\
&\quad \wedge \texttt{RFinishedAuth2}(R, pidR, ASR)\ @\tilde{k}\\
&\quad \Rightarrow ASI = ASR\\
&\quad\ \vee \neg[\text{Assumptions}]
\end{aligned}$$

TABLE I
Meta assumptions

UNR	The user does not reuse the passkey.
UNG	The user does not use a guessable passkey.
UNC	The user does not confuse PE with NC.
IOS	The adversary does not break the IO channel.
HICS	The adversary does not break the HCI channel.
OOBS	The adversary does not break the OOB channel.

This formula expresses that if devices I and R finished the authentication phase 2 (state ⑥ in Fig. 2) in a pairing session labelled by $(pidI, pidR)$, then they execute the same association model ASI and ASR. The final line of this formula specifies its underlying assumption, which is constructed by the meta assumptions defined in the TABLE I. We consider the assumption IOS as the base assumption if devices have no OOB capabilities and build stronger assumptions from it by combining it with the UNR, UNG, and UNC assumptions. If the devices have OOB capabilities, we consider the assumption (IOS $\wedge$ OOBS) as the base assumption. It is worth mentioning that we allow the adversary to break the HCI channel when verifying the association model consistency property. Finally, we define 8 formulas to formalize the association model consistency property under different assumptions.

B. Authentication

We formalize the authentication properties which guarantee both hosts agree on the crucial messages, such as the IO capabilities and LTK, as well as both controllers agree on the same session key. We define the authentication properties based on the Non-Injective (NI) agreement introduced by Lowe [25]. The NI agreement property asserts that whenever an agent A completes a protocol run with agent B, then B must have previously been running the protocol with A, and both agents agree on the message t. Importantly, the Bluetooth specification [3] does not require authentication when the device executes the JW association model. Therefore, the pairing protocol should guarantee the agreement of the crucial messages unless the JW association model is executed.

Authentication between Hosts. To formalize the authentication properties of the IO capabilities, the random values, the $DHKey$, the $MacKey$, and the LTK between hosts, we define the following first-order logic formula template and instantiate it with defined strings.

$$\begin{aligned}
&\forall\ A\ B\ t\ host\ pid\ AS\ \tilde{i}.\ \texttt{Commit}(A, B, \langle\text{`}\,Class\,\text{'}, t\rangle)\ @\tilde{i}\\
&\quad \wedge \texttt{Session}(host, pid, AS)\ @\tilde{i}\\
&\quad \Rightarrow (\exists\ \tilde{j}.\ \texttt{Running}(B, A, \langle\text{`}\,Class\,\text{'}, t\rangle)\ @\tilde{j}) \vee (AS = \texttt{JW})\\
&\quad\ \vee \neg[\text{Assumptions}]
\end{aligned}$$

For example, to formalize the authentication property of the IO capabilities between hosts, we replace '$Class$' with 'IOCaps'. The instantiated formula implies that, under the given assumption, if the host A believes that the host B's IO capability is t, then the host B is indeed running the protocol with A using the IO capability t, unless A executes the JW association model.

This formula defines mutual authentication between hosts, as Tamarin considers scenarios where A is the initiator and B is the responder, and vice versa. As for the given assumption, we use the same base assumption as the association model consistency property in the authentication between hosts, and we also allow the HCI channel to be broken. We consider different assumptions constructed from the base assumption and the UNR, UNG, and UNC assumptions. To formalize the authentication property under different assumptions, we define 8 formulas for each authentication property between hosts. Ultimately, we verify 40 authentication formulas between hosts for each pairing case.

The NI agreement property guarantees that the two hosts agree on the same value t, but multiple sessions may use the same value t. The IO capabilities of the devices are immobile, and the $DHKey$ can be identical between different sessions because the public key of the device can be reused as defined in the specification. The uniqueness of the $MacKey$ and the LTK is achieved by the fact that they are derived from the random values that are fresh on each session.

Authentication between Controllers. We formalize the mutual authentication property of the session key between controllers by the following formula.

$$\begin{aligned}
&\forall\ Ladd\ A\ B\ t\ host\ role\ chn\ AS\ \tilde{i}\ \tilde{j}.\\
&\quad \texttt{CCommit}(Ladd, A, B, \langle\text{`SK'}, t\rangle)\ @\tilde{i}\\
&\quad \wedge \texttt{HostInfo}(host, role, chn)\ @\tilde{i}\\
&\quad \wedge \texttt{SelectASHandle}(host, role, chn, AS)\ @\tilde{j} \wedge (\tilde{j} < \tilde{i})\\
&\Rightarrow (\exists\ \tilde{k}.\ \texttt{CRunning}(Ladd, B, A, \langle\text{`SK'}, t\rangle)\ @\tilde{k})\\
&\quad \vee (AS = \texttt{JW})\\
&\quad \vee \neg[\text{Assumptions}]
\end{aligned}$$

This formula indicates that, under the specified assumption, if controller A believes that controller B's session key for a session identified by $Ladd$ is t, then controller B is indeed engaging in the protocol with A using the session key t in the same session, unless the host of A has previously executed the JW association model. When devices lack OOB capabilities, we use the assumption (IOS $\wedge$ HICS) as the base assumption. For devices with OOB capabilities, the base assumption is (IOS $\wedge$ OOB $\wedge$ HICS). We define 8 formulas to capture the mutual authentication property of the session key between controllers under different assumptions. It is worth noting that we assume the adversary cannot compromise the HCI channel, as the LTK is transferred over this channel. If the HCI channel were compromised, it would trivially allow an adversary to violate the authentication property of the session key between controllers. This formula defines an authentication property where both controllers agree on the same session key, but do not guarantee uniqueness. The uniqueness is achieved by the fact that the session key is derived from the session key diversifier, which is fresh on each session.

C. Secrecy

The secrecy properties guarantee that, under the given assumption, an adversary cannot know the LTK, the session key, or the specific keys. In verifying these properties, we consider the same assumptions used for the authentication property of the session key between controllers.

Secrecy of LTK and Specific Keys. We formalize the secrecy of LTK as the following formula.

$$\begin{aligned}
&\forall\ host\ pid\ LTK\ AS\ \tilde{i}.\ \texttt{SecLTK}(host, pid, LTK)\ @\tilde{i}\\
&\quad \wedge \texttt{Session}(host, pid, AS)\ @\tilde{i}\\
&\Rightarrow \neg(\exists\ \tilde{j}.\ \texttt{K}(LTK)\ @\tilde{j}) \vee (AS = \texttt{JW})\\
&\quad \vee \neg[\text{Assumptions}]
\end{aligned}$$

Intuitively, this formula expresses that, under the specified assumption, the adversary cannot know (signifying by the action fact K) the LTK of the session identified by $(host, pid)$, unless the $host$ executes the JW association model.

The formula used to formalize the secrecy of specific keys is similar to the one described above. We define an action fact `SecSPKs()` within the rules that model the hosts exchange the specific keys, such as rule `iExchKeys` in Fig. 6 and rule `RevPeerKeys` in Fig. 7. In this context, we substitute the action fact `SecLTK()` with `SecSPKs()` in the above formula. Then we get the formula that states that, under the given assumption, the adversary cannot know the specific keys of the session identified by $(host, pid)$, unless the $host$ executes the JW association model in this session.

Secrecy of SK. We formalize the secrecy property of the session key using the following formula.

$$\begin{aligned}
&\forall\ host\ role\ chn\ SK\ \tilde{i}.\ \texttt{SecSK}(host, role, chn, SK)\ @\tilde{i}\\
&\Rightarrow \neg(\exists\ \tilde{j}.\ \texttt{K}(SK)\ @\tilde{j})\\
&\quad \vee (\exists\ \tilde{k}.\ \texttt{SelectASHandle}(host, role, chn, \texttt{JW})\ @\tilde{k}\\
&\qquad \wedge (\tilde{k} < \tilde{i}))\\
&\quad \vee \neg[\text{Assumptions}]
\end{aligned}$$

Intuitively, this formula indicates that, under the specified assumption, the adversary cannot know the session key SK of the controller unless the controller's $host$ executes the JW association model in the session identified by $(host, role, chn)$.

V. Analyzing and Results

In this section, we first present the details of our formal verification process, including the verified pairing cases and the efforts made to facilitate the verification process. Next, we summarize the verification results in Table II and provide an explanation of these results.

A. Formal Verification

The verification process is time-consuming due to the large number of pairing cases, numerous lemmas to verify, and the overall complexity of our model. To address this, we develop a script that distributes the verification task across multiple servers, allowing for parallel processing and significantly accelerating the verification timeline. We utilize 6 servers and deploy a total of 20 Docker containers to handle our verification tasks in parallel. For violated formulas, we implement a script to automatically extract the corresponding attack trace under Tamarin's interactive mode in a distributed manner. Furthermore, we apply filtering rules to reduce the number of pairing cases requiring analysis. Additionally, we develop heuristics, introduce a helping lemma, and propose a verification strategy to ensure the termination of the verification process and reduce the verification time.

1) Pairing Cases: The pairing cases are derived by taking the Cartesian product of the two devices' IO capabilities, OOB capabilities, authentication requirements, and maximum key size. To streamline the analysis, we focus only on reasonable cases. To reduce cases, we restrict consideration to devices with a high maximum key size. We do not consider devices that lack both OOB capability and IO capability but still require authentication. We assume that the adversary cannot establish an OOB channel with the devices, but the adversary may interact with OOB data in already-established channels. Under this assumption, certain cases become equivalent. For example, the case where a device possessing both OOB sending and receiving capabilities pairs with a device lacking OOB capabilities is equivalent to the case where both devices lack OOB capabilities. By applying these filters, we reduce the number of pairing cases from 6,400 to 84, while preserving comprehensive analytical coverage.

2) Heuristics and Helping Lemma: Tamarin provides two modes to analyze models: automated mode and interactive mode. Our goal is to automatically analyze the BLE-SC pairing protocol. However, when verifying formulas in our model, Tamarin fails to terminate under its built-in heuristics, which are used to guide the verification process. To address this problem, we utilize the interactive mode to gain insights and develop a set of customized heuristics to prevent the verification process from entering infinite loops, thereby enabling Tamarin to successfully analyze our model automatically. While investigating the interactive mode, we observe that Tamarin frequently attempts to construct a counterexample to disprove a lemma by searching for a trace in which the attacker can obtain a device's DH private key. However, in our model, the device's DH private key is never revealed to the attacker. To take advantage of this fact, we introduce a helping formula, which is annotated with 'reuse' in Tamarin and used as hypotheses in proof steps to facilitate the proofs of subsequent lemmas. This lemma helps avoid unnecessary proof attempts when proving other lemmas.

3) Verification Strategy: The verification algorithm for trace properties in Tamarin is sound [10]. This soundness ensures that if a formula A is implied by a formula B, then whenever B is satisfied in a model M, formula A must also be satisfied in M. By leveraging the algorithm's soundness and the fact that *a formula formalizing a security property under a certain assumption implies the same property under any stronger assumption*, we develop a verification strategy to optimize the verification process for a security property across multiple assumptions. In the verification process, we start by verifying the formula that formalizes a security property under the base assumption. If this formula is satisfied, then this security property is satisfied under all stronger assumptions. Otherwise, we verify the formula that formalizes this security property under the strongest assumption. If this formula is violated, then the security property is violated under all remaining unverified weaker assumptions. Otherwise, we continue by verifying this security property under the second weaker assumption, and so on. This approach improves efficiency by systematically reusing the previously verified results.

Despite our efforts, the verification process requires approximately 5 days to complete, while the extraction of attack traces takes about 6 hours. The verification results and the attack traces are available in the supporting material [22].

B. Results

Table II summarizes the verification results. We first discuss the verification results for pairing cases where both devices lack OOB capabilities (No. 1-13). In pairing cases No. 1-5, all security properties are satisfied under the base assumption. In these cases, the devices can only execute the JW association model. These verification results appear to contradict the fact that the JW association model is vulnerable to MitM attacks. However, they are consistent with the design expectations of the BLE-SC pairing protocol. This indicates that our formulas accurately reflect the security goals of this protocol.

In pairing cases No. 6-10, all security properties of these pairing cases are violated under the base assumption. For these cases, we identify the minimal security assumptions necessary to ensure the desired properties. For example, when one device is KeyboardOnly and the other is KeyboardDisplay, the minimal security assumption required to ensure a specific property is the conjunction of the base assumption and UNR $\wedge$ UNG $\wedge$ UNC. During analysis of pairing cases No. 6, 7, 9, and 10, we uncovered the *method confusion attack* [8], whose corresponding counterexample violates all properties under the assumption that the UNC assumption is not considered. Furthermore, when verifying pairing cases No. 6-8, we identify a new variant confusion attack, termed the *PE confusion attack*. The corresponding counterexamples violate all properties under the assumption that either the UNR or UNG assumptions are not considered. We discuss this attack in Section VI, and provide more details about the relationship between the attacks and the counterexamples in Appendix E.

For pairing cases No. 11-13, and for cases No. 14-16 where devices have OOB capabilities, the security properties, except the authentication of LTK between hosts, are satisfied under the base assumption. Notably, even under the strongest assumption, the LTK authentication property is violated in No. 6-16. The counter-example of this property describes the LTK *keysize confusion attack* disclosed by Shi et al. [6]. For properties that are satisfied in the base assumption, we relax the base assumption to find the necessity of this assumption and find subtle assumptions about the OOB channel. When a bidirectional OOB channel is used, an authenticated OOB channel is sufficient to ensure the security of this pairing case. However, when the OOB channel is unidirectional, the OOB channel provides both confidentiality and integrity to guarantee the security of this pairing case.

TABLE II
ANALYSIS RESULTS OF BLE-SC PAIRING

No.	Initiator	Responder	CAS	AIO-R-DHK-MACK	ASK	SLTK-SK-SP	ALTK
			$Base_1$	$Base_1$	$Base_2$	$Base_2$	$Base_2$
1	NN-NoMITM	**¬NN**	✓	✓	✓	✓	✓
2	**¬NN**	NN-NoMITM	✓	✓	✓	✓	✓
3	NN-NoMITM	NN-NoMITM	✓	✓	✓	✓	✓
4	**DO**	**DO/DYN**	✓	✓	✓	✓	✓
5	**DYN**	**DO**	✓	✓	✓	✓	✓
6	**KD**	**KD/KO**	UNR ∧ UNG ∧ UNC	UNR ∧ UNG ∧ UNC	UNR ∧ UNG ∧ UNC	UNR ∧ UNG ∧ UNC	✗
7	**KO**	**KD**	UNR ∧ UNG ∧ UNC	UNR ∧ UNG ∧ UNC	UNR ∧ UNG ∧ UNC	UNR ∧ UNG ∧ UNC	✗
8	**KO**	**KO**	UNR ∧ UNG	UNR ∧ UNG	UNR ∧ UNG	UNR ∧ UNG	✗
9	**DYN**	**KD/KO**	UNC	UNC	UNC	UNC	✗
10	**KD/KO**	**DYN**	UNC	UNC	UNC	UNC	✗
11	**DO**	**KD/KO**	✓	✓	✓	✓	✗
12	**DYN**	**DYN**	✓	✓	✓	✓	✗
13	**KD/KO**	**DO**	✓	✓	✓	✓	✗
14	OOBRev	OOBSend	✓(OOBS)	✓(OOBS)	✓(OOBS)	✓(OOBS)	✗
15	OOBSendRev	OOBSendRev	✓(OOBA)	✓(OOBA)	✓(OOBA)	✓(OOBA)	✗
16	OOBSend	OOBRev	✓(OOBS)	✓(OOBS)	✓(OOBS)	✓(OOBS)	✗

- CAS: Consistency of Association Models. AIO-R-DHK-MACK: Authentication of IO capabilities, Random Numbers, DH Key, and MAC Key. ALTK: Authentication of LTK. ASK: Authentication of Session Key. SLTK-SK-SP: Secrecy of LTK, Session Key, and Specific Keys.
- $Base_1$: Base assumption IOS $Base_2$: Base assumption IOS ∧ HICS
- DYN: DisplayYesNo   KD: KeyboardDisplay   DO: DisplayOnly   KO: KeyboardOnly   ¬NN: DYN/KD/DO/KO   NN-NoMitM: NoInputNoOutput-Not requre MitM protection   OOBSend/OOBRev/(OOBSendRev): Devices can send/receive/(send and receive) message through the OOB channel.
- Devices considered in No. 1-13 do not have OOB capabilities.
- **Bolded IO capabilities** indicate consideration of both with and without MitM protection requirements.
- OOBS: OOB channel provides both confidentiality and integrity.   OOBA: OOB channel provides only integrity.
- ✓: property verified in the base assumption   ✗: property falsified even in the strongest assumption

VI. PE CONFUSION ATTACK

This section introduces the PE confusion attack, including its two attack cases, and discusses its significance and impact. Subsequently, we present our implementation of this attack in a controlled environment. Finally, we propose countermeasures to mitigate the effects of this attack.

A. Attack and Its Impact

In the PE confusion attack, as illustrated in Fig. 9, the adversary manipulates the IO capabilities fields within the pairing request and response messages to deceive the initiator, responder, and user into performing different types of PE association models. From the initiator's perspective, the PEID association model is being executed, while from the responder's perspective, the PEDI association model is in use. Meanwhile, the user believes that the PEII association model is being employed. In this attack, both devices await the user to enter a $passkey$. The pairing process can be completed once the user enters an identical $passkey$ on both devices. For the $passkey$, Bluetooth implementations that comply with the specification generate $passkey$ randomly and never reuse it. We assume the devices follow best practices as outlined in the Bluetooth specification. However, the user may unintentionally deviate from the specification, leading to two potential attack cases shown in Fig. 9.

In the first case, the user chooses a weak $passkey$, which is easily guessable by an adversary, such as "123456" for convenience. Even more concerning, we have found that the UIs of certain devices may inadvertently encourage the user to enter weak passkeys. We have tested smartphones equipped with Android versions 10 to 13. The UIs of these devices that execute the PE association model prompt the user to enter a $passkey$ are shown in Fig. 10. The misleading prompt "Usually 0000 or 1234" might lead the user to enter "0000" or "1234". According to the Bluetooth specification, the 4-digit value entered will be prefixed with zeros to form a 6-digit $passkey$. Although UIs of other devices (Appendix C) do not mislead users, they also fail to instruct users on how to ensure a secure pairing. If the user enters the weak passkey on both devices, an adversary who guesses the right passkey can perform a MitM attack, depicted in Fig. 9.

In the second case, the user reuses the passkey during two pairing processes that occur within a short time interval. When the user inputs a randomly selected passkey, the adversary observes both devices executing the PE association model and interrupts the pairing process once the PE association model is completed. With access to the transmitted packets, the adversary can recover the passkey by exploiting the property of the function `f4` defined in Section III-A. Afterwards, when the user restarts the pairing process, the adversary performs the PE confusion attack as mentioned before to let both devices wait for the user to input the passkey. Although specification-conformant devices can implement the randomly selected passkey and never reuse it from a previous pairing process, the specification cannot guarantee that the user will not reuse a previously selected passkey. Once the user inputs

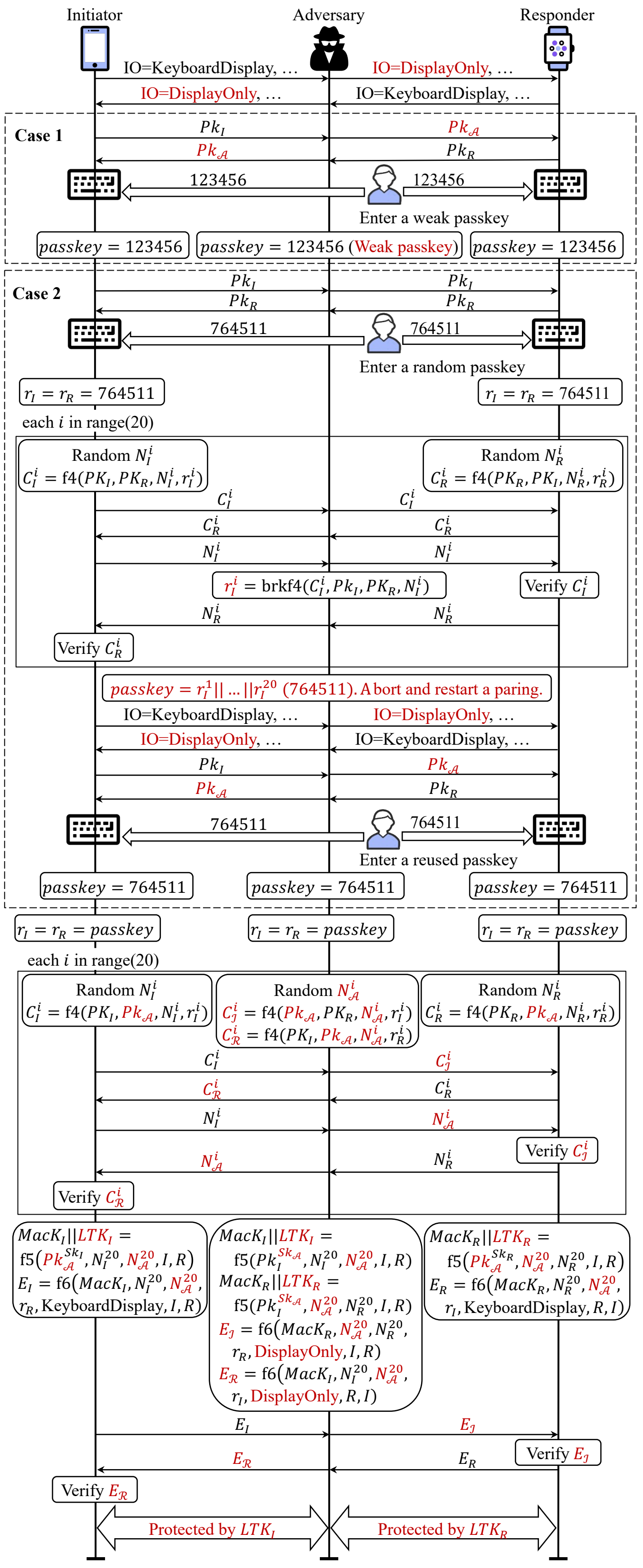
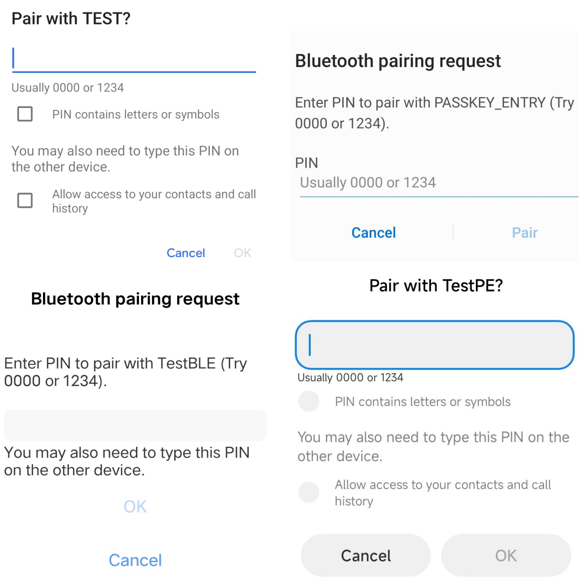

Initiator
Adversary
Responder
IO=KeyboardDisplay, …
IO=DisplayOnly, …
IO=DisplayOnly, …
IO=KeyboardDisplay, …
Case 1
Pk_I
$Pk_{\mathcal{A}}$
$Pk_{\mathcal{A}}$
Pk_R
123456
123456
Enter a weak passkey
$passkey = 123456$
$passkey = 123456$ (Weak passkey)
$passkey = 123456$
Case 2
Pk_I
Pk_I
Pk_R
Pk_R
764511
764511
Enter a random passkey
$r_I = r_R = 764511$
$r_I = r_R = 764511$
each i in range(20)
Random N_I^i
$C_I^i = \mathrm{f4}(PK_I, PK_R, N_I^i, r_I^i)$
Random N_R^i
$C_R^i = \mathrm{f4}(PK_R, PK_I, N_R^i, r_R^i)$
C_I^i
C_I^i
C_R^i
C_R^i
N_I^i
N_I^i
$r_I^i = \mathrm{brkf4}(C_I^i, Pk_I, PK_R, N_I^i)$
Verify C_I^i
N_R^i
N_R^i
Verify C_R^i
$passkey = r_I^1 || \ldots || r_I^{20}$ (764511). Abort and restart a paring.
IO=KeyboardDisplay, …
IO=DisplayOnly, …
IO=DisplayOnly, …
IO=KeyboardDisplay, …
Pk_I
$Pk_{\mathcal{A}}$
$Pk_{\mathcal{A}}$
Pk_R
764511
764511
Enter a reused passkey
$passkey = 764511$
$passkey = 764511$
$passkey = 764511$
$r_I = r_R = passkey$
$r_I = r_R = passkey$
$r_I = r_R = passkey$
each i in range(20)
Random N_I^i
$C_I^i = \mathrm{f4}(PK_I, Pk_{\mathcal{A}}, N_I^i, r_I^i)$
Random $N_{\mathcal{A}}^i$
$C_{\mathcal{J}}^i = \mathrm{f4}(Pk_{\mathcal{A}}, PK_R, N_{\mathcal{A}}^i, r_I^i)$
$C_{\mathcal{R}}^i = \mathrm{f4}(PK_I, Pk_{\mathcal{A}}, N_{\mathcal{A}}^i, r_R^i)$
Random N_R^i
$C_R^i = \mathrm{f4}(PK_R, Pk_{\mathcal{A}}, N_R^i, r_R^i)$
C_I^i
$C_{\mathcal{J}}^i$
$C_{\mathcal{R}}^i$
C_R^i
N_I^i
$N_{\mathcal{A}}^i$
Verify $C_{\mathcal{J}}^i$
$N_{\mathcal{A}}^i$
N_R^i
Verify $C_{\mathcal{R}}^i$
$MacK_I || LTK_I = \mathrm{f5}(Pk_{\mathcal{A}}^{Sk_I}, N_I^{20}, N_{\mathcal{A}}^{20}, I, R)$
$E_I = \mathrm{f6}(MacK_I, N_I^{20}, N_{\mathcal{A}}^{20}, r_R, \mathrm{KeyboardDisplay}, I, R)$
$MacK_I || LTK_I = \mathrm{f5}(Pk_I^{Sk_{\mathcal{A}}}, N_I^{20}, N_{\mathcal{A}}^{20}, I, R)$
$MacK_R || LTK_R = \mathrm{f5}(Pk_I^{Sk_{\mathcal{A}}}, N_{\mathcal{A}}^{20}, N_R^{20}, I, R)$
$E_{\mathcal{J}} = \mathrm{f6}(MacK_R, N_{\mathcal{A}}^{20}, N_R^{20}, r_R, \mathrm{DisplayOnly}, I, R)$
$E_{\mathcal{R}} = \mathrm{f6}(MacK_I, N_I^{20}, N_{\mathcal{A}}^{20}, r_I, \mathrm{DisplayOnly}, R, I)$
$MacK_R || LTK_R = \mathrm{f5}(Pk_{\mathcal{A}}^{Sk_R}, N_{\mathcal{A}}^{20}, N_R^{20}, I, R)$
$E_R = \mathrm{f6}(MacK_R, N_R^{20}, N_{\mathcal{A}}^{20}, r_I, \mathrm{KeyboardDisplay}, R, I)$
E_I
$E_{\mathcal{J}}$
Verify $E_{\mathcal{J}}$
$E_{\mathcal{R}}$
E_R
Verify $E_{\mathcal{R}}$
Protected by LTK_I
Protected by LTK_R

Fig. 9. Attack flow of PE confusion attack under two cases. Colored symbols indicate the value modified by the attacker.



Pair with TEST?

Usually 0000 or 1234

PIN contains letters or symbols

You may also need to type this PIN on the other device.

Allow access to your contacts and call history

Cancel OK

Bluetooth pairing request

Enter PIN to pair with PASSKEY_ENTRY (Try 0000 or 1234).

PIN

Usually 0000 or 1234

Cancel Pair

Bluetooth pairing request

Enter PIN to pair with TestBLE (Try 0000 or 1234).

You may also need to type this PIN on the other device.

OK

Cancel

Pair with TestPE?

Usually 0000 or 1234

PIN contains letters or symbols

You may also need to type this PIN on the other device.

Allow access to your contacts and call history

Cancel OK

Fig. 10. UIs of Android Phone when BLE-SC PE association model is used and the devices ask the user to enter a passkey. Devices and Versions (LTR-TTB): Pixel 4 (Android 10), Samsung Galaxy S10 (Android 11), IQOO Neo6 SE (Android 12), and Redmi K40 (Android 13).

the passkey from the previous pairing procedure, the adversary can perform a MitM attack, as shown in Fig. 9.

The PE confusion attack challenges the commonly accepted belief that weak or reused passkeys selected by users only affect the ‘KeyboardOnly-KeyboardOnly’ pairing cases that employ the PEII association model. Our analysis demonstrates that this vulnerability actually impacts 12 pairing cases—considering whether devices require MitM protection—as shown in Table III. Since the BLE-SC protocol has remained largely unchanged since its introduction in Bluetooth specification v4.2 [26], our findings are applicable to all versions from v4.2 through v6.0 [3]. While the attack's widespread impact is notable, successful exploitation of the PE confusion attack relies on the assumptions that users will enter weak or reused passkeys, potentially limiting its practicality in real-world scenarios.

B. Implementation

To implement the PE confusion attack, the attacker must be able to perform a MitM attack on BLE connections. Tschirschnitz et al. [8] employed the BtleJack [27] to perform the method confusion attack. While in this paper, we develop our own BLE tool suite to perform the PE confusion attack in a controlled environment. Our approach involves developing custom controller firmware based on the Zephyr OS [28] and a specialized MitM tool, both designed to manipulate the packet directly within the controller.

By leveraging our BLE tool suite, we implement the PE confusion attack between a Linux device running Ubuntu 24.04 LTS and a smartphone running Android 12. For the attack, the Linux device, which runs the BlueZ stack on version 5.72, is equipped with an nRF52840 dongle, a Bluetooth 5.4

TABLE III
AFFECTED CASES BY PE CONFUSION ATTACK

$\texttt{IOCap}_R$ \ $\texttt{IOCap}_I$	**KeyboardOnly**	**KeyboardDisplay**
KeyboardOnly	◑	●
KeyboardDisplay	●	●

●: Affected. ◑: PEII. **Bolded IO capabilities** indicate consideration of both with and without MitM protection requirements.

System-on-Chip. This Bluetooth dongle has been flashed with our customized firmware. This setup enables the dongle to interact seamlessly with our attack tool, effectively performing the necessary BLE message interception and injection in the dongle. We provide the firmware, attack tool, and attack scripts implementing the attack described in Fig. 9 in the supporting material [22]. For demonstration purposes, our attack scripts complete only the pairing process and do not relay any encrypted data packets after pairing.

C. Countermeasures

We provide the following advice to alleviate the effects of the PE confusion attack.

1) The confirmation calculation should contain the IO capabilities of both devices. We recommend replacing the nonce in the `f4` function with the hash of the concatenation of the nonce, the pairing request, and response messages.
2) The Bluetooth specification should include a dedicated section on the UI design of the PE and NC association models, as this is a critical security matter. We recommend that the prompt message in the UI when the device is asking the user to enter a passkey should be: *"Enter the number displayed on the peer device and DO NOTHING on that device. If no number is displayed on the peer, enter a RANDOM number on both devices, and NEVER REUSE that number."* The first part of this message is intended to mitigate the method confusion attack, while the second part aims to prevent the PE confusion attack.
3) The pairing protocol should not support the PE association model in which both devices ask the user to enter a human-choice number, as asking users to randomly select and not reuse passkeys is unreliable. In situations where the IO capabilities of both devices are limited to 'KeyboardOnly', the JW association model should be used. This approach allows the community to guide the user to either compare numbers or input the numbers displayed on the peer device.

Disclosure. We notified the Bluetooth Special Interest Group (SIG) about the attack we had identified in the Bluetooth specification on July 15, 2024. Initially, the Bluetooth SIG believed that our attack was similar to the method confusion attack disclosed by Tschirschnitz et al. [8]. In our follow-up report on October 13, 2024, we clarified the distinctions between these two attacks. Specifically, we pointed out that they target different users who are confused between different association models. The method confusion attack [8], or more precisely, the NC-PE confusion attack, occurs when the user confuses the NC association model with the PE association model. *In contrast, the PE confusion attack involves confusion among three types of PE association models: the initiator executes the PEID association model, the responder executes the PEDI association model, while the user believes they are engaging with the PEII association model.* On February 7, 2025, the Bluetooth SIG responded to our disclosure and confirmed that our findings do affect BLE pairing.

VII. RELATED WORK

Initial symbolic work [29], [30], [31] on the pairing protocol focused on its security under a single association model. Recently, Wu et al. [4] and Jangid et al. [5] provided formal models considering multiple association models using ProVerif [32] and Tamarin, respectively. Their models abstract the devices' capabilities but just combine different association model flows. Therefore, their model cannot guide which pairing cases are vulnerable. In contrast, Shi et al. [6] provided a more comprehensive formal model of the BLE-SC pairing protocol using Tamarin. However, they modeled the association model selection logic by defining lots of rules, and each rule is for a specific device pair. This makes their model too complex to find the minimal security assumptions to ensure the security of the pairing protocol.

In the model of [5], they assumed that the device may reuse $passkey$ and found the group guessing attack. We assume that the devices use $passkey$ as expected by the Bluetooth specification, but the user may deviate from the specification. They also discussed the leaking of the $passkey$ and found the ghost attack. This assumption is covered by our model, as our model allows the adversary to break the confidentiality of the IO channel between the user and the device.

Claverie et al. [33] utilized Tamarin to verify the key agreement protocols of Bluetooth Classic (BC), BLE, and Bluetooth Mesh (BM). They considered both the legacy pairing mode and the secure connection pairing mode, revealing pairing mode confusion attacks in both BC and BLE. In these attacks, one device pairs under the secure connection mode while the other pairs under the legacy mode. The pairing mode confusion attack is outside the scope of our model since we only consider the secure connection mode. In our PE confusion attack, both devices are in the secure connection mode.

Cremers et al. [34] proposed methods for symbolically modeling non-prime order groups. To achieve this, they introduced a special group element, DH_neutral, into Tamarin's built-in DH theory. In our model, devices verify the peer device's DH public key and reject any instance of the DH_neutral element. This approach ensures that the devices accept only valid DH public keys. Basin et al. [35] formalized human errors in security protocols using Tamarin. They presented two approaches, skilled and rule-based, but only provided a human model of the latter. In our work, we model the human user based on our specific user assumptions.

In a different direction, Lindell [36], Troncoso et al. [37], and Fischlin et al. [38] analyze the security of Bluetooth within the computational model. These analyses employ manual provable security techniques. In contrast, our analysis is conducted fully automatically by the computer.

Numerous studies have investigated attacks targeting various aspects of Bluetooth security, including the pairing protocol [39], [40], [41], the session key negotiation protocol in BC [23], [42], and cross-transport key derivation [43]. For a comprehensive overview of research in Bluetooth security, we refer the reader to the survey by Wu et al. [44].

VIII. Conclusion

We perform an accurate and comprehensive formal analysis of the BLE-SC pairing protocol using Tamarin. By automatically verifying 84 pairing cases across distributed servers, we identify the minimal security assumptions necessary to secure each pairing case. Additionally, our analysis reveals a MitM attack, referred to as the PE confusion attack, which we implement in a controlled environment. Finally, we propose specific countermeasures to mitigate the PE confusion attack. For transparency and reproducibility, we provide the formal model, verification script, controller firmware, MitM tool, and attack code in the supporting materials, fostering collaboration and advancement within the research community.

Acknowledgments

We thank the anonymous reviewers and our shepherd for their constructive comments. This work was partially supported by the National Cryptologic Science Fund of China under Grant No. 2025NCSF02027, Wuhan Scientific and Technical Achievements Project under Grant No. 2024030803010172.

Ethics considerations

Upon discovering the attack in the Bluetooth specification, we promptly notified the Bluetooth SIG on July 15, 2024. This early notification reflects our duty to responsible disclosure, ensuring that the relevant stakeholders are aware of the issue and can take appropriate action to mitigate potential risks. By engaging with the SIG and providing detailed explanations of our attack and countermeasures, we aimed to contribute positively to the security community. We believe that sharing our findings responsibly not only helps in addressing the vulnerability but also advances the overall understanding and development of more secure systems. Through these actions, we have upheld the ethical standards expected in research, prioritizing both the integrity of our work and the safety of users potentially affected by the vulnerability.

References

[1] Bluetooth Specification Contributors, "Bluetooth Core Specification 4.0," Jan. 2010.

[2] M. Powell, "2023 bluetooth market update," [Online]. Available: https://www.bluetooth.com/2023-market-update/.

[3] Bluetooth Specification Contributors, "Bluetooth Core Specification 6.0," Aug. 2024.

[4] J. Wu, R. Wu, D. Xu, D. J. Tian, and A. Bianchi, "Formal model-driven discovery of bluetooth protocol design vulnerabilities," in *Proc. of S&P*, 2022.

[5] M. K. Jangid, Y. Zhang, and Z. Lin, "Extrapolating formal analysis to uncover attacks in bluetooth passkey entry pairing," in *Proc. of NDSS*, 2023.

[6] M. Shi, J. Chen, K. He, H. Zhao, M. Jia, and R. Du, "Formal analysis and patching of BLE-SC pairing," in *Proc. of USENIX Security Symposium*, 2023.

[7] Google, "Bumble: Bluetooth stack for apps, emulation, test and experimentation," [Online]. Available: https://github.com/google/bumble.

[8] M. von Tschirschnitz, L. Peuckert, F. Franzen, and J. Grossklags, "Method confusion attack on bluetooth pairing," in *Proc. of S&P*, 2021.

[9] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *Proc. of Computer Aided Verification*, 2013.

[10] B. Schmidt, S. Meier, C. Cremers, and D. A. Basin, "Automated analysis of diffie-hellman protocols and advanced security properties," in *Proc. of CSF*, 2012.

[11] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe, "Automated analysis and verification of TLS 1.3: 0-rtt, resumption and delayed authentication," in *Proc. of S&P*, 2016.

[12] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A comprehensive symbolic analysis of TLS 1.3," in *Proc. of CCS*, 2017.

[13] H. Lee, Z. Smith, J. Lim, G. Choi, S. Chun, T. Chung, and T. T. Kwon, "matls: How to make TLS middlebox-aware?" in *Proc. of NDSS*, 2019.

[14] D. A. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, "A formal analysis of 5g authentication," in *Proc. of CCS*, 2018.

[15] C. Cremers and M. Dehnel-Wild, "Component-based formal analysis of 5g-aka: Channel assumptions and session confusion," in *Proc. of NDSS*, 2019.

[16] D. A. Basin, R. Sasse, and J. Toro-Pozo, "The EMV standard: Break, fix, verify," in *Proc. of S&P*, 2021.

[17] ——, "Card brand mixup attack: Bypassing the PIN in non-visa cards by using them for visa transactions," in *Proc. of USENIX Security Symposium*, 2021.

[18] Tamarin Team, "Tamarin prover manual," [Online]. Available: https://tamarin-prover.com/manual/index.html.

[19] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[20] H. Comon-Lundh and S. Delaune, "The finite variant property: How to get rid of some algebraic properties," in *Proc. of Term Rewriting and Applications*, 2005.

[21] V. Cheval and C. Fontaine, "Automatic verification of Finite Variant Property beyond convergent equational theories," in *Proc. of CSF*, 2025.

[22] M. Shi, Y. Xiao, J. Chen, K. He, R. Du, and M. Jia, "Formal analysis of ble secure connection pairing and revelation of the pe confusion attack," Nov. 2025. [Online]. Available: https://doi.org/10.5281/zenodo.17677477

[23] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "The KNOB is broken: Exploiting low entropy in the encryption key negotiation of bluetooth BR/EDR," in *Proc. of USENIX Security Symposium*, 2019.

[24] ——, "Key negotiation downgrade attacks on bluetooth and bluetooth low energy," *ACM Trans. Priv. Secur.*, vol. 23, no. 3, pp. 14:1–14:28, 2020.

[25] G. Lowe, "A hierarchy of authentication specifications," in *Proceedings 10th computer security foundations workshop*, 1997.

[26] Bluetooth Specification Contributors, "Bluetooth Core Specification 4.2," Dec. 2014.

[27] D. Cauquil, "You'd better secure your ble devices or we'll kick your butts," *DEF CON*, vol. 26, 2018.

[28] Zephyr Community, "Zephyr project," [Online]. Available: https://zephyrproject.org/.

[29] R. Chang and V. Shmatikov, "Formal analysis of authentication in bluetooth device pairing," *FCS-ARSPA'07*, vol. 45, 2007.

[30] D. Jia and R. Hsu, "Formal modeling and analysis of bluetooth 4.0 pairing protocol," 2013.

[31] M. Sethi, A. Peltonen, and T. Aura, "Misbinding attacks on secure device pairing and bootstrapping," in *Proc. of AsiaCCS*, 2019.

[32] B. Blanchet, "Automatic proof of strong secrecy for security protocols," in *Proc. of S&P*, 2004.

[33] T. Claverie, G. Avoine, S. Delaune, and J. Lopes-Esteves, "Tamarin-based analysis of bluetooth uncovers two practical pairing confusion attacks," in *Proc. of ESORICS*, 2023.

[34] C. Cremers and D. Jackson, “Prime, order please! revisiting small subgroup and invalid curve attacks on protocols using diffie-hellman,” in *Proc. of CSF*, 2019.
[35] D. Basin, S. Radomirovic, and L. Schmid, “Modeling human errors in security protocols,” in *Proc. of CSF*, 2016.
[36] A. Y. Lindell, “Comparison-based key exchange and the security of the numeric comparison mode in bluetooth v2.1,” in *CT-RSA*, vol. 5473, 2009, pp. 66–83.
[37] M. Troncoso and B. Hale, “The Bluetooth CYBORG: Analysis of the Full Human-Machine Passkey Entry AKE Protocol,” in *Proc. of NDSS*, 2021.
[38] M. Fischlin and O. Sanina, “Cryptographic analysis of the bluetooth secure connection protocol suite,” in *Proc. of ASIACRYPT*, 2021.
[39] A. Y. Lindell, “Attacks on the pairing protocol of bluetooth v2.1,” in *Black Hat USA*, 2008.
[40] J. Barnickel, J. Wang, and U. Meyer, “Implementing an attack on bluetooth 2.1+ secure simple pairing in passkey entry mode,” in *TrustCom*, 2012.
[41] Y. Zhang, J. Weng, R. Dey, Y. Jin, Z. Lin, and X. Fu, “Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks.” in *Proc. of USENIX Security Symposium*, 2020.
[42] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, “BIAS: bluetooth impersonation attacks,” in *Proc. of S&P*, 2020.
[43] D. Antonioli, N. O. Tippenhauer, K. Rasmussen, and M. Payer, “Blurtooth: Exploiting cross-transport key derivation in bluetooth classic and bluetooth low energy,” in *Proc. of AsiaCCS*, 2022.
[44] J. Wu, R. Wu, D. Xu, D. J. Tian, and A. Bianchi, “Sok: The long journey of exploiting and defending the legacy of king harald bluetooth,” in *Proc. of S&P*, 2024.

Appendix A
Cryptographic Tools

The cryptographic functions `f4`, `f5`, and `f6` utilize `AES-CMAC` as their foundational building block. The `AES-CMAC` function is defined in RFC-4493 as a Cipher-based Message Authentication Code (CMAC) that uses `AES-128` as the block cipher. The confirmation calculation function `f4` used in the association models is defined as

$$\texttt{f4}(Pk_A, Pk_B, N_A, r_A) = \texttt{AES-CMAC}_{N_A}(Pk_A||Pk_B||r_A).$$

The LTK and MacKey generation function `f5` is defined as

$$\begin{aligned}&\texttt{f5}(DHKey, N_I, N_R, BT_I, BT_R)\\ =&\texttt{AES-CMAC}_T(0||\text{0x62746C65}||N_I||N_R||BT_I||BT_R||256)\\ &||\texttt{AES-CMAC}_T(1||\text{0x62746C65}||N_I||N_R||BT_I||BT_R||256),\end{aligned}$$

where $T = \texttt{AES-CMAC}_{SALT}(DHKey)$ and $SALT$ is a 128-bit constant value. The check value generation function `f6` is defined as

$$\begin{aligned}&\texttt{f6}(MacKey, N_I, N_R, r_X, IOCap_X, BT_I, BT_R)\\ =&\texttt{AES-CMAC}_{MacKey}(N_I||N_R||r_X||IOCap_X||BT_I||BT_R).\end{aligned}$$

The function `g2` used to generate the displayed number in the NC association model is defined as

$$\texttt{g2}(Pk_I, Pk_R, N_I, N_R) = \texttt{AES-CMAC}_{N_I}(Pk_I||Pk_R||N_R).$$

Appendix B
Association Models

We first present the selection logic of the association models and then present the complete description of the association models that are briefly introduced in Section II-B.

Selection Logic. The association model selection logic is defined in the Bluetooth specification [3] (Vol 3, Part H, 2.3.5.1 Selecting key generation method, p1640). We extract the selection logic from the specification and present it in Table IV and Table V. The function symbols `selectAM` and `mapIOCaps2AM`, along with the 23 equations in Fig. 4, faithfully formalize this selection logic.

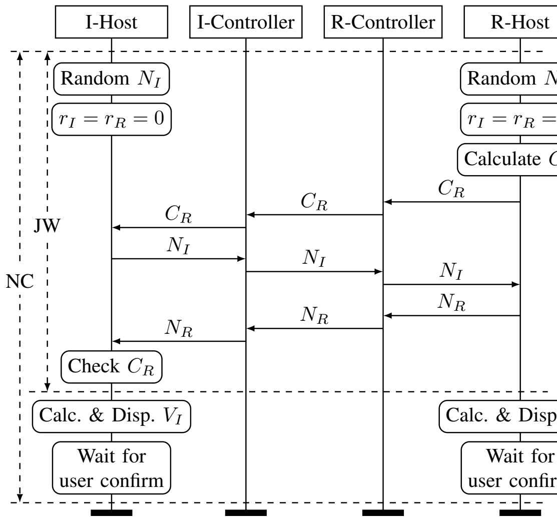


Fig. 11. Overview of JW and NC association models flows.

Just Work (JW). In the JW association model, as shown in Fig. 11, the two hosts generate random values N_I and N_R respectively and set the two numbers r_I and r_R to zero. The responder’s host calculates the pairing confirm $C_R = \texttt{f4}(Pk_R, Pk_I, N_R, r_R)$, and sends it to the peer. After receiving C_R, the initiator’s host sends the random number N_I to the responder’s host. The responder’s host then responds with the random value N_R. Then, the initiator’s host checks C_R after receiving N_R from the responder’s host. The JW association model is finished if the verification is successful.

Numeric Comparison (NC). In the NC association model, as shown in Fig. 11, the two hosts execute the flows like the JW association model. After this, both devices display numbers V_I and V_R, calculated by $V_I = V_R = \texttt{g2}(Pk_I, Pk_R, N_I, N_R)$, with buttons to indicate whether the two numbers are identical. The NC association model is finished if the user pushes ‘Yes’ on both devices.

Out-of-Band (OOB). In the OOB association model, as shown in Fig. 12, the two hosts transfer OOB data through an OOB channel before the pairing process. The OOB data includes Bluetooth address I or R of the device, a random number r_I or r_R, and a confirmation C_I or C_R calculated by $C_I = \texttt{f4}(Pk_I, Pk_I, r_I, 0)$ or $C_R = \texttt{f4}(Pk_R, Pk_R, r_R, 0)$. The OOB data can be unidirectional or bidirectional. If a host has received the peer’s OOB data, it will set the OOB flag in the pairing request or pairing response message to true. In the stage ⑤, the two hosts verify C_I or/and C_R. After this, they exchange random values N_I and N_R.

TABLE IV
RULES FOR USING OOB AND AUTHREQ FLAGS FOR BLE-SC PAIRING

Responder \ Initiator		OOB Set		OOB Not Set	
		MITM Set	MITM Not Set	MITM Set	MITM Not Set
OOB Set	MITM Set	OOB	OOB	OOB	OOB
	MITM Not Set	OOB	OOB	OOB	OOB
OOB Not Set	MITM Set	OOB	OOB	Use IOCaps	Use IOCaps
	MITM Not Set	OOB	OOB	Use IOCaps	JW

TABLE V
MAPPING OF IO CAPABILITIES TO KEY GENERATION METHOD

Responder \ Initiator	DisplayOnly	DisplayYesNo	KeyboardOnly	NoInputNoOutput	KeyboardDisplay
DisplayOnly	JW	JW	PEID	JW	PEID
DisplayYesNo	JW	NC	PEID	JW	NC
KeyboardOnly	PEDI	PEDI	PEII	JW	PEDI
NoInputNoOutput	JW	JW	JW	JW	JW
KeyboardDisplay	PEDI	NC	PEID	JW	NC

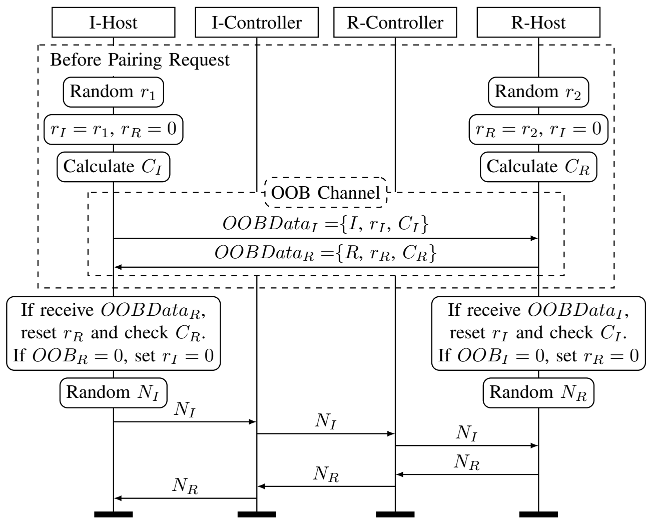
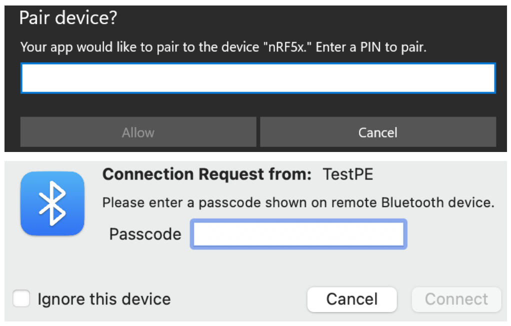

I-Host
I-Controller
R-Controller
R-Host
Before Pairing Request
Random r_1
Random r_2
$r_I = r_1, r_R = 0$
$r_R = r_2, r_I = 0$
Calculate C_I
Calculate C_R
OOB Channel
$OOBData_I = \{I, r_I, C_I\}$
$OOBData_R = \{R, r_R, C_R\}$
If receive $OOBData_R$, reset r_R and check C_R. If $OOB_R = 0$, set $r_I = 0$
If receive $OOBData_I$, reset r_I and check C_I. If $OOB_I = 0$, set $r_R = 0$
Random N_I
Random N_R
N_I
N_I
N_I
N_R
N_R
N_R

Fig. 12. Overview of the OOB association model flow.


Pair device?

Your app would like to pair to the device "nRF5x." Enter a PIN to pair.

Allow | Cancel

Connection Request from: TestPE

Please enter a passcode shown on remote Bluetooth device.

Passcode

Ignore this device | Cancel | Connect

Fig. 13. UIs of Windows and macOS.

APPENDIX C
UIS OF VARIOUS OPERATING SYSTEMS

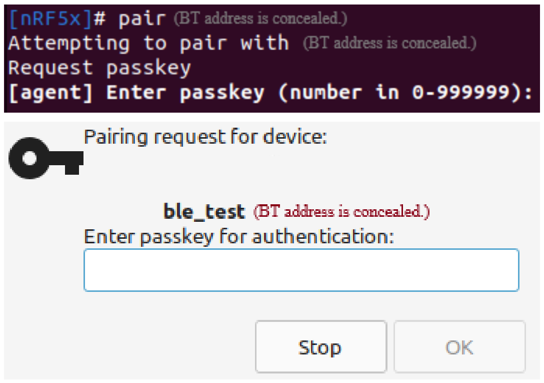
The UIs of Android have been discussed in Section VI. In this section, we show our exploration results of the UIs of widely used Operating Systems (OS) when executing the BLE-SC pairing protocol.

Windows and macOS. Fig. 13 presents the UIs of Windows 11 (above) and macOS 14.5 (below) when devices running the PE association model prompt the user to enter a passkey. While Windows does not mislead users into entering a guessable passkey, it also lacks guidance on where to find the passkey. In contrast, macOS explicitly instructs the user to enter the passkey displayed on the remote device.

Linux Distributions. We investigate two Bluetooth clients in the Linux distributions: BlueZ and Blueman. BlueZ is a command-line client, and Blueman is a graphical user interface client. The UIs of these clients when running the PE association model to prompt the user to enter a passkey are shown in Fig. 14, where the above is the UI of BlueZ 5.72, and the below is the UI of Blueman 2.4.

Although the UIs of Windows, macOS, and Linux do not mislead users into entering easily guessable passkeys, they also fail to instruct users on what to do to ensure a secure pairing. If a user enters the number displayed on the remote device operating under the NC association model, and subsequently pushes the 'Yes' button on that device, an adversary can perform the method confusion attack disclosed by Tschirschnitz et al. [8].

IOS. The UI of IOS 14.8 (iPhone XR) is shown in Fig. 15. If the user follows the prompt message on the UI of IOS, the method confusion attack can be protected.



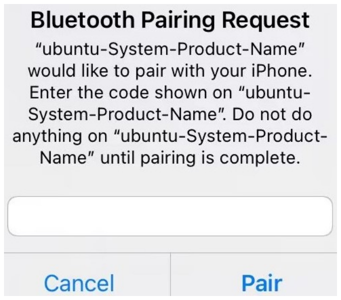
```
[nRF5x]# pair (BT address is concealed.)
Attempting to pair with (BT address is concealed.)
Request passkey
[agent] Enter passkey (number in 0-999999):
```

Pairing request for device:

ble_test (BT address is concealed.)

Enter passkey for authentication:

Fig. 14. UIs of Linux



Bluetooth Pairing Request

"ubuntu-System-Product-Name" would like to pair with your iPhone. Enter the code shown on "ubuntu-System-Product-Name". Do not do anything on "ubuntu-System-Product-Name" until pairing is complete.

Fig. 15. UIs of IOS

Appendix D
NC-PE Confusion Attack vs. PE Confusion Attack.

The main similarity between NC-PE confusion attacks [8] and PE confusion attacks is that, in both cases, the adversary manipulates the IO capabilities fields in the pairing request and response messages to cause the devices to execute different association models. The differences are outlined as follows:

- In the NC-PE confusion attack, the adversary tricks one device into executing the NC association model, which displays a 6-digit number with a confirmation button, while the other device executes the PE association model, which prompts the user to enter the number displayed on the peer device. The success of this attack relies on the user mistakenly believing that both devices are executing the PE association model: entering the displayed number into the device that prompts for it and confirming the action on the device that displays the number.
- In the PE confusion attack, the adversary tricks the initiator into executing the PEID association model, which prompts the user to enter a passkey, while the responder executes the PEDI association model, which also prompts the user to enter a passkey. From the user's perspective, both devices appear to be executing the PEII association model, prompting the user to enter a passkey. The success of this attack relies on the user entering a weak passkey or reusing a passkey from a previous pairing session.

Appendix E
Attacks and Counterexamples

In this section, we provide details on the relationship between PE confusion attacks and the corresponding counterexample traces generated by Tamarin. Due to the complexity of our model, the complete traces are too large to be presented in this paper. Therefore, we have cropped each trace to highlight the core steps that capture the key aspects of the relationship between the attacks and their corresponding counterexamples. The cropped traces are shown in Fig. 16 and Fig. 17, with the locations of certain entities adjusted for better readability. We use red boxes to highlight the critical terms to facilitate the understanding. Furthermore, we provide the full traces in our supplementary material [22].

The counterexample trace for the secrecy of the LTK in the 'KeyboardDisplay-KeyboardDisplay' cases, under the assumption that *users may enter weak passkeys*, reflects the steps of the first case of the PE confusion attack described in Section VI. The cropped trace shown in Fig. 16 illustrates the following key steps: ① The attacker has tricked the initiator into executing the PEID association model by modifying the peer's IO capability to 'DisplayOnly' in the pairing response message, while ② the attacker has tricked the responder into executing the PEDI association model by modifying the peer's IO capability to 'DisplayOnly' in the pairing request message. ③ Both devices prompt the user to enter a passkey and ④ transition to the 'WaitInput' state. ⑤ The user who is pairing these two devices encounters the situation where both devices prompt for a passkey, and ⑥ enters a publicly known passkey on both devices.

Accordingly, the counterexample trace for secrecy of the LTK in the 'KeyboardDisplay-KeyboardDisplay' cases, under the assumption that *users may reuse passkeys*, demonstrates the steps of the second case of the PE confusion attack described in Section VI. The cropped trace shown in Fig. 17 illustrates how the attacker successfully obtains the inputted passkey: In this figure, ① the attacker obtains the commitment to the left part of the passkey and the random value from the public channel, then derives the left part of the passkey using the function `brkf4` explained in Section III-A. Similarly, ② the attacker derives the right part of the passkey. ③ The attacker merges the left and the right parts to obtain the passkey. Notably, in actions ① and ②, the input of both devices' public key, which are obtained during the public key exchange stage, is omitted.

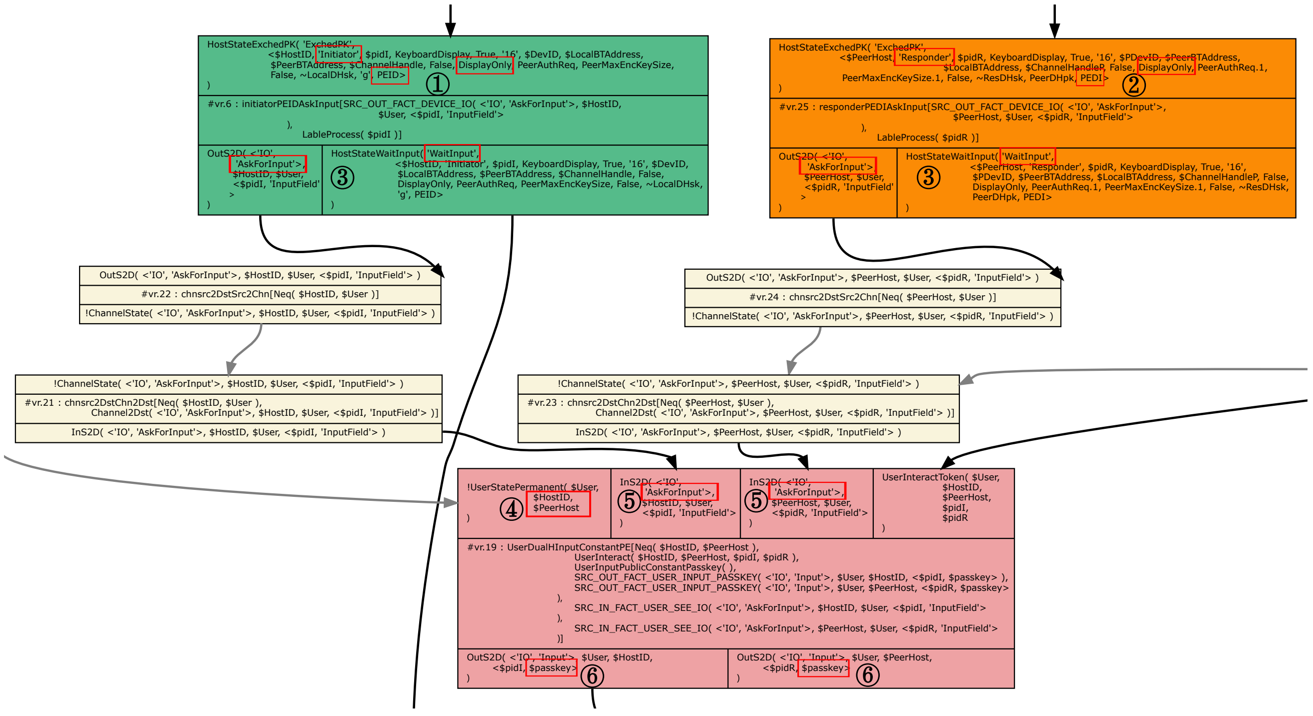


Fig. 16. Cropped and adjusted trace of the counterexample for secrecy of the LTK for ‘KeyboardDisplay-KeyboardDisplay’ cases under the assumption that users may enter weak passkeys. The symbol ‘$’ in Tamarin models publicly known terms.

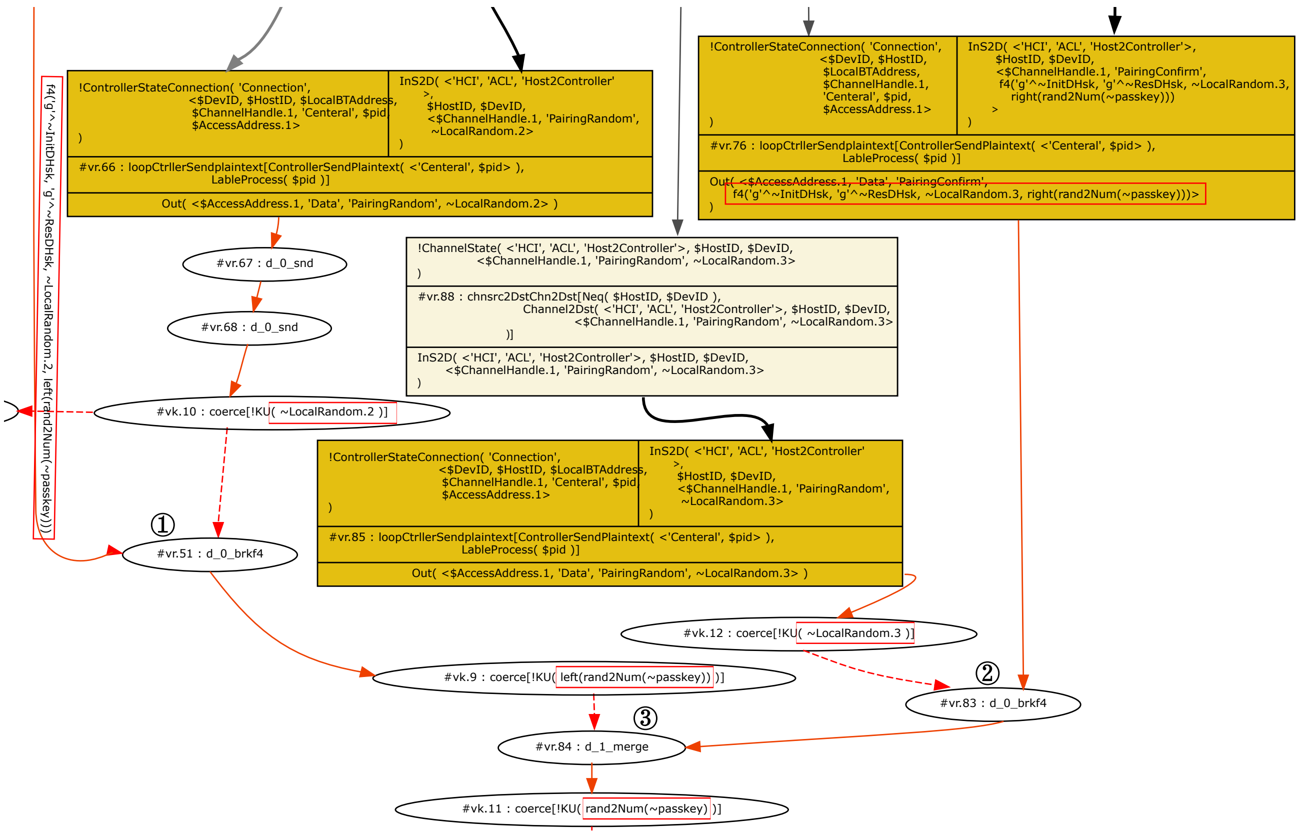


Fig. 17. Cropped and adjusted trace of the counterexample for secrecy of the LTK for ‘KeyboardDisplay-KeyboardDisplay’ cases under the assumption that users may reuse passkeys. Ellipse boxes represent the actions taken by the attacker or the public channel.

Appendix F
Artifact Appendix

A. Description & Requirements

This artifact provides the formal models and verification framework used to analyze the security of BLE Secure Connections pairing protocols. It includes the **Tamarin Prover** models, verification scripts, and corresponding verification results necessary to reproduce the security analysis presented in the paper.

1) How to Access: The artifact is publicly available at: https://github.com/itachiMin/BLE-Model-Results-Poc. For long-term archival access, the artifact is also published on Zenodo and can be obtained via DOI: https://doi.org/10.5281/zenodo.17677477.

2) Hardware Dependencies:

- Architecture: AMD64
- CPU: 8 cores
- Memory: 16 GB
- Storage: 128 GB of available disk space

3) Software Dependencies:

- Operating System: Ubuntu 24.04 LTS
- Required Packages and Tools:
  - `docker.io`
  - `openssh-server`
  - `make`
  - `m4`
  - `python3` (with virtual environment support)
  - `curl`
  - `git`

4) Benchmarks: None.

B. Artifact Installation & Configuration

This section describes the installation and configuration steps required to prepare the environment for evaluating the artifact.

1) Step 1: Install Dependencies.: Update the system and install all required packages:

```
$ sudo apt update
$ sudo apt install -y curl git m4 make \
    openssh-server docker.io python3-venv
```

Add the current user to `Docker` group (re-login required):

```
$ sudo usermod -aG docker $USER
```

Note: Ensure that both Docker and SSH services are running properly. The user must have permission to execute Docker commands without using `sudo`.

2) Step 2: Clone the Repository.: Clone the repository and switch to the `artifact-eval` branch:

```
$ git clone --branch artifact-eval \
    --single-branch --depth 1      \
    https://github.com/itachiMin/BLE-Model
-Results-PoC.git
$ cd BLE-Model-Results-PoC/
```

Note: All subsequent commands are executed from the `BLE-Model-Results-PoC/` directory.

3) Step 3: Set Up Python Environment.: Create and activate a Python virtual environment:

```
$ python3 -m venv .myvenv
$ source .myvenv/bin/activate
$ pip3 install -r requirements.txt
```

4) Step 4: Download Docker Image and Required Files.: Download the pre-built `Tamarin Prover` Docker image:

```
$ curl -L -o ./ExpRun/files/tamarin-conta
iner_1.8.0.tar "https://github.com/itachi
Min/BLE-Model-Results-PoC/releases/downlo
ad/v1.0.0/tamarin-container_1.8.0.tar"
```

After downloading, the `ExpRun/files` directory should contain the following files:

```
./ExpRun/files/
|-- hardware.py
|-- run_tamarin.sh
|-- tamarin-container_1.8.0.tar
`-- verify.py
```

a) Step 5: Configure Server Settings.: Edit `ExpRun/servers.json` to configure the artifact for local execution only:

```
[{
    "host": "127.0.0.1",
    "port": 22,
    "username": "your_username",
    "password": "your_password",
    "workdir": "/tmp/ble_exp_ae",
    "workers": 1,
    "weight": 1
}]
```

Note: Replace `your_username` and `your_password` with your actual system credentials.

C. Experiment Workflow

The experimental workflow involves formal model verification using the Tamarin Prover. The artifact provides:

- Automatic model generation scripts for BLE Secure Connections pairing protocols
- Distributed verification framework for parallel model checking
- Results collection and analysis tools for violated security properties

The AE version uses a reduced subset of pairing scenarios to demonstrate the key formal model generation and verification experimental process, while maintaining computational feasibility.

D. Major Claims

- (C1): The formal models correctly capture the security properties of BLE Secure Connections pairing protocols.

This is proven by experiment (E1) whose results demonstrate the verification of security lemmas and identification of violations.

- (C2): The artifact identifies critical security vulnerabilities in specific BLE pairing configurations. This is proven by experiment (E1) through the generation of attack graphs for violated lemmas.

E. Evaluation

To ensure that all experiments could be completed within the available time and computational resources, we scaled down the verification workload by selecting 16 representative cases with shorter verification times. Since the generation and verification processes of the formal models are identical for all cases, we believe this reduced experiment set is sufficient to demonstrate the correctness and effectiveness of our proposed formal model generation and verification framework.

1) Experiment (E1): **Model Verification** [30 human-minutes + 10 compute-hours]: This experiment generates formal models according to different capabilities of the devices and verifies them using the *Tamarin Prover* to check whether the specified security properties hold.

[Preparation]

1) Ensure that all installation steps described in Section F-B have been completed.
2) Verify that the Docker and SSH services are running and that the user has sufficient permissions.
3) Activate the Python virtual environment using the command: `source .myvenv/bin/activate`

[Execution] Run the reduced verification subset with the following command: `make subset` This command automatically launches the verification of 16 representative models from the full model set. The process performs distributed verification based on the current local machine configuration.

[Results] After approximately 10 hours of computation:

1) The verification results are stored in the `./ExpRun/results/` directory.
2) Each verified model has its own subdirectory containing the verification outcomes for all corresponding lemmas.
3) To check specific results, navigate to the relevant directory and utilize Tamarin Prover's interactive interface for detailed proof analysis results. For instance, for the case `I[NoInNoOut_NoOOB_NoAuthReq_KeyHigh]_R[NoInNoOut_NoOOB_NoAuthReq_KeyHigh]`, execute:
```
$ cd ./ExpRun/results/BLE-SC_I[NoInput
NoOutput_NoOOB_NoAuthReq_KeyHigh]_R[No
InputNoOutput_NoOOB_NoAuthReq_KeyHigh]
$ docker run --rm -it \
    -p 3001:3001 -v $(pwd):/root \
    ghcr.io/luojiazhishu/tamarin-docke
r/cli:latest \
    tamarin-prover interactive --inter
face=0.0.0.0 --derivcheck-timeout=0 .
```
4) Access http://localhost:3001 via a web browser to review the verification details.
5) Note that the Tamarin Prover web interface displays a maximum of 5 files. For comprehensive lemma analysis, either isolate each lemma in individual directories and inspect them sequentially, or employ the `crawler.py` script to capture images of all violated lemmas (automated in the full verification):
```
cd ./ExpRun
python3 crawler.py
```
6) Complete verification results and lemma violated graphs are accessible through the project repository at https://github.com/itachiMin/BLE-Model-Results-PoC/tree/artifact-eval?tab=readme-ov-file#verification-results.

The expected outcomes are successful proof derivations for verified properties and automatic graphs for violated ones, reproducing our paper's key findings and confirming the framework's validity and its ability to expose security vulnerabilities in BLE-SC pairing protocols.

F. Customization

To customize the verification process, modify the `ExpSubset/subset_cases.json` file to define different combinations of cases for verification. Each case comprises two devices: an initiator and a responder. Each device is characterized by four capabilities:

$$d = \big(\mathrm{IOcap}(d),\ \mathrm{OOB}(d),\ \mathrm{AuthReq}(d),\ \mathrm{KeySize}(d)\big),$$

where

$$\begin{array}{ll}
\mathrm{IOcap} \in \mathcal{I}, & \text{with } \mathcal{I} = \{\texttt{NoInputNoOutput}, \\
& \quad \texttt{DisplayOnly},\ \texttt{KeyboardOnly}, \\
& \quad \texttt{DisplayYesNo},\ \texttt{KeyboardDisplay}\} \\
\mathrm{OOB} \in \mathcal{O}, & \text{with } \mathcal{O} = \{\texttt{OOBSendRev},\ \texttt{OOBSend}, \\
& \quad \texttt{OOBRev},\ \texttt{NoOOB}\}, \\
\mathrm{AuthReq} \in \mathcal{A}, & \text{with } \mathcal{A} = \{\texttt{AuthReq},\ \texttt{NoAuthReq}\}, \\
\mathrm{KeySize} \in \mathcal{K}, & \text{with } \mathcal{K} = \{\texttt{KeyHigh},\ \texttt{KeyLow}\}.
\end{array}$$

Thus, each device capability tuple satisfies:

$$d \in \mathcal{I} \times \mathcal{O} \times \mathcal{A} \times \mathcal{K}.$$

Below is an example configuration that verifies only the case where both the initiator and responder have the capabilities "NoInputNoOutput", "NoOOB", "NoAuthReq", and "KeyHigh":

```
[{
    "init": [
        "NoInputNoOutput", "NoOOB",
        "NoAuthReq", "KeyHigh"
    ],
    "resp": [
        "NoInputNoOutput", "NoOOB",
        "NoAuthReq", "KeyHigh"
    ]
}]
```