# Fake It till You Make It: Enhancing Security of Bluetooth Secure Connections via Deferrable Authentication

Marc Fischlin Olga Sanina

Cryptoplexity, Technische Universität Darmstadt, Germany
`www.cryptoplexity.de`
`{marc.fischlin,olga.sanina}@tu-darmstadt.de`

**Abstract.** The Bluetooth protocol for wireless connection between devices comes with several security measures to protect confidentiality and integrity of data. At the heart of these security protocols lies the Secure Simple Pairing, wherewith the devices can negotiate a shared key before communicating sensitive data. Despite the good intentions, the Bluetooth security protocol has repeatedly been shown to be vulnerable, especially with regard to active attacks on the Secure Simple Pairing.

We propose here a mechanism to limit active attacks on the Secure Connections protocol (the more secure version of the Secure Simple Pairing protocol), without infringing on the current Bluetooth protocol stack specification. The idea is to run an authentication protocol, like a classical challenge-response step for certified keys, within the existing infrastructure, even at a later, more convenient point in time. We prove that not only does this authentication step ensure freshness of future encryption keys, but an interesting feature is that it—*a posteriori*—also guarantees security of previously derived encryption keys. We next argue that this approach indeed prevents a large set of known attacks on the Bluetooth protocol.

This is the full version of the paper appearing in the proceedings of the *31st ACM Conference on Computer and Communications Security (CCS 2024)*.

## Contents

# 1 Introduction

Bluetooth enables short-ranged wireless communication between and for users. Predicted by ABI Research to incline the shipment into the market in the next 5 years, Bluetooth technology is already used in billions of devices in smart homes, cars, monitoring and control systems, and wearables. With this spread, Bluetooth channels send more and more data, including sensitive medical information, hence data confidentiality is of concern. This includes authentication that ensures only intended devices can be connected.

Bluetooth was invented in the '90s and, naturally, has evolved over time. Each major change is depicted as a version in a Core Specification maintained by the Bluetooth Special Interest Group (SIG). The latest version 5.4 [Blu23] was introduced on the 31st of January 2023, and adds, to give an example, the use of electronic price tags. The modifications resulted in different versions of the Bluetooth protocol suits, which include (outdated) legacy protocols and are available as two major branches today: classical Bluetooth BR/EDR and low-energy Bluetooth BLE. All the so-called authentication methods suggested in the Bluetooth Core Specification are not secure and prone to numerous attacks, including very recent ones.[1]

## 1.1 Bluetooth's Etiology

To protect the communication, Bluetooth Core Specifications cover various cryptographic mechanisms. At foremost, starting with version 4.2, BR/EDR and BLE support Secure Connections (SC), which should enable parties to communicate securely, based on approved cryptographic primitives such as AES and HMAC.

The core of SC is the Secure Simple Pairing (SSP) protocol that allows establishment of a shared key between the devices and is based on the elliptic-curve Diffie–Hellman (ECDH) protocol. To obtain some form of authentication, devices rely on the input/output capabilities (*IOcap*) that they negotiate during the pairing. This includes the association models NC (numeric comparison, users comparing displayed digits), PKE (passkey entry, users entering digits), and OOB (out-of-band transfer of authentication data without the direct user involvement into the protocol). However, the protocol may also run in JW (just works) without authentication, user involvement, and dependency on *IOcap*. The concrete instantiation of SSP slightly differs between BR/EDR and BLE.

From the link key (BR/EDR) resp. long-term key (BLE), created in the SSP protocol, the parties can then derive encryption keys, which can be used to secure the actual communication. Reconnections also use this mechanism to create fresh encryption keys from the link key resp. long-term key, avoiding the need to pair again.

Although Bluetooth incorporates cryptographic mechanisms, many security design choices are not motivated well and appear to be ad hoc. Indeed, the history of the Bluetooth security protocol suite abounds with devastating attacks and thorny security analyses. Most of the attacks against Bluetooth focus on the SSP protocol and the authentication property. This includes attacks like the downgrading to JW attack [HH07], enforcing that intended authentication is omitted; the Method Confusion Attack [vTPFG21] and the Pairing Confusion Attack [CADE23], in which the adversary unnoticedly modifies the IO capabilities in the pairing step and thus changes the authentication method. The Ghost Keystroke Attacks [JZL23, ZWD$^+$20] exploit cross-connected executions, which leak passkeys used for authentication to MitM adversaries. Other attacks decrease the strength of the encryption scheme by downgrading the keysize, such as the KNOB Attack [ATR20b, ATR19] and the Keysize Confusion Attack [SCH$^+$23]. They can be used in combination with authentication attacks, like the BIAS [ATR20a] and BLUFFS [Ant23] Attacks, to cause even more dramatic consequences. For legacy versions of Bluetooth, further attacks aim at the possibility of using brute-forceable low-entropic secrets [JW01, Rya13]. Some attacks use the

[1]The reported attacks can be found at: `https://www.bluetooth.com/learn-about-bluetooth/key-attributes/bluetooth-security/reporting-security/`

protocol support to switch between the BR/EDR and BLE modes, including the BLURtooth attack [ATRP22] and the CSIA attack [WWX+22]. The BLESA attack [WNK+20] against reconnections in BLE exploits the lack of proper authentication. Table 1 gives an overview on the effectiveness of some suggested countermeasures [SCH+23, TH21] against various attacks. Note that our remedy works within the Secure Connections type on the application level and thus cannot thwart attacks on the Legacy protocols. To highlight the efficacy of our measure, we discuss the Ghost Keystroke attack and its mitigation more closely in Section 8; all other attacks appear in Section 4.1.

Table 1: Proposed patches and their resilience to the attacks. Symbols: ✗ means the attack is possible, ✓ means the attack is prevented, (✓) is not claimed but prevented. All attacks are possible for both BLE and BR/EDR transport, apart from [Rya13] resp. [Ant23, ATR20a, JW01, LÇA+04] that affect only the BLE resp. BR/EDR transport. Note that our approach cannot prevent attacks on the Legacy versions.

| Attack | Exploited Stage | Solution | | |
|---|---|---|---|---|
| | | [TH21] | [SCH+23] | Ours |
| Role Confusion [TH21] | PKE | ✓ | (✓) | ✓ |
| Method Confusion [vTPFG21] | NC, PKE | ✗ | ✓ | ✓ |
| Pairing Confusion [CADE23] | NC, PKE, Legacy | ✗ | ✗ | ✓ |
| Ghost Keystroke [JZL23, ZWD+20] | PKE, (NC) | ✗ | ✗ | ✓ |
| Downgrade to JW [HT10, HH07] | SC and SSP | ✗ | ✗ | ✓ |
| KNOB [ATR20b, ATR19, SCH+23] | Key Size Negotiation | ✗ | ✗ | ✗ |
| Key Cracking [Rya13, JW01] | Legacy | ✗ | ✗ | ✗ |
| BIAS [ATR20a] | Reconnection | ✗ | ✗ | ✗ |
| BLUFFS [Ant23] | Legacy Reconnection | ✗ | ✗ | ✗ |
| BLURtooth [ATRP22], CSIA [WWX+22] | Cross-transport | ✗ | ✗ | ✗ |

## 1.2 Contributions

Our contribution enhances the security of the existing Bluetooth Secure Connections protocol stack in a backward-compatible manner and consists of the three main parts.

First, unlike the usually suggested fixes for Bluetooth, we propose a (cryptographic) method to thwart many attacks (see the right column in Table 1). Based on the observation that most vulnerabilities root in the lack or misuse of message and identity authentication in association models, we propose an authentication protocol that builds on the established paradigm of challenge-response schemes. This protocol is executed on top of (i.e., after) the current steps, ensuring that the communication partner is authentic and uses the same link/long-term key. Such a solution requires some form of authentication means, like certified signing keys linked to the Bluetooth address of the device. We propose different variants of authentication protocols but our method only yields a provably secure solution for BR/EDR.[2]

Our protocols blend in well with the existing Bluetooth protocol stack: We rely on the defined cryptographic functions supported by the Core Specification; the protocols can run as an application over the Bluetooth channel and they do not require access to secrets or other security-relevant data beyond the link/long-term keys. The authentication step can be executed later, at any point in time, and still ensures the security of the keys "backwards". It means that, upon successful completion of the authentication part, one can deduce that previous encryption keys must have been secure. This feature differs from perfect forward secrecy (PFS), which guarantees the previous session keys to remain secure if the adversary compromises the long-term secret. For PFS, the keys are usually secure from the beginning; in contrast,

[2]The reason is that link keys in BR/EDR are derived via (truncated) HMAC and thus obey collision resistance. In contrast, BLE uses an AES-CMAC-based key derivation function, which is not known to provide the required level of robustness, as it is malleable [CE21]. We discuss this more in Section 1.3.

our security property says that authentication transfers the keys with the unknown status to secure keys. While in both cases something happens at a later point in time (compromise vs. authentication), the consequences are different: the keys remain secure after the compromise vs. they are made secure against active attacks.

Post-handshake protocols have been considered in the literature before, e.g., for the Internet Key Exchange protocol IKEv2 [KHN$^{+}$14] in [SSL20] or for TLS 1.3 in [JKSS10] and [Kra16]. The difference to our setting here is that these authentication steps are usually executed immediately after the key establishment, assuming that the key establishment data (e.g., communication transcript) is still available. For Bluetooth communication, the authentication process may run much later: either as a part of the application execution, or after a software update (e.g., installing the authentication data), or because the battery was low during the pairing procedure—but no fresh pairing is needed in these cases anymore. We discuss the relationship to existing approaches in more detail in Section 4.

While our solution complies with the protocol flow of Bluetooth, it requires some form of certification of identities and keys, e.g., in the form of a certificate issued by trustworthy vendors. The vendors themselves may be certified with a root certificate held by the Bluetooth SIG. Such certificates for vendors could, for example, be issued as part of the mandatory *Bluetooth Qualification Process* for new products. This requires additional administrative steps but is conceivably easier to integrate than making changes at the level of the specification and obeying legacy compatibility.

Since certification is already part of the Bluetooth ecosystem in Bluetooth Mesh (Mesh specification supports X.509 digital certificates since version 1.1 but requires OOB methods for incorporating such authentication methods), we find it plausible to assume that certification can be implemented in Bluetooth BR/EDR and LE, too, both administratively and also technically. In terms of the computational costs, Bluetooth devices already implement procedures like ECDH computations, so, e.g., EdDSA should not be more expensive than this. Furthermore, we alternatively suggest a signature-free authentication method, which also employs ECDH.

Second, we extend the trust-on-first-use (TOFU) security model for key establishment and reconnections in Bluetooth [FS21], which is a game-based model in the Bellare–Rogaway (BR) style [BR94]. While the TOFU model opts for passive adversaries during the initial connections to ensure the security of derived keys, we add authentication requirements to allow deferrable-outside-first-use (DOFU) authentication. That is, the authentication might be postponed to any later point in time and this way yields TOFU-OR-DOFU security.

Finally, we prove our proposed solution to provide an adequate level of cryptographic strength in the sense of authenticated key exchange protocols. That is, we show the new protocols enjoy the security in our extended TOFU-OR-DOFU model. Note that Fischlin and Sanina [FS21] showed the best result one can hope for, for the “vanilla” Bluetooth protocol stack: session keys (called encryption keys in Bluetooth) are secure as long as the adversary has been passive in the initial connection, wherein the link key (BR/EDR) resp. long-term key (BLE) has been distributed and whence all encryption keys are derived. We augment this result by stating that all encryption keys are secure if the initial pairing is trustworthy, *or if the authentication step has been carried out successfully.*

## 1.3 Applicability to BR/EDR and BLE

We emphasize here that our positive results in BR/EDR with HMAC as a key derivation function (KDF) for *LK* do not immediately transfer to BLE because BLE uses AES-CMAC as a KDF function for *LTK*. HMAC in BR/EDR gives collision resistance of the derived link key as an additional feature, which we require in the proof for deferrable authentication. The AES-CMAC function in BLE, however, is not collision-resistant and the attacks in [CE21] showed how such weaknesses can be exploited. We still managed to prove our protocols for BLE to have provided match security, authentication, and leakage resistance as long as AES-CMAC is used (but key secrecy only in the TOFU scenario). This is not a shortcoming of our proof, but rather the lack of the collision resistance of the KDF in BLE opens up the following attack strategy if only the long-term key can be used for authentication. The adversary can first make two unpartnered honest initial-connection sessions accept and hold the same long-term key *LTK* due to the lack of the collision resistance of the KDF in BLE. If the adversary connects these two sessions in subsequent authentication sessions, then authentication would succeed for the same *LTK*, although the initial-connection sessions were not partnered. This breaks the idea of the authentication intuitively and can also be exploited formally by the adversary. If the KDF in BLE was replaced by a collision-resilient variant, the proof of key secrecy would go through smoothly for BLE as well. Unfortunately, the BLE specification currently employs only AES and AES-CMAC as KDFs.

### 1.4 Paper Structure

The paper is organized as follows. The used notations, both general and Bluetooth-related, are introduced in Section 2 and acronyms in Appendix B. Section 3 shortly explains the Bluetooth technology and employed protocols, followed by Section 4 with an overview of the attacks and suggested countermeasures as well as analysis of Bluetooth and related cryptographic frameworks that could be used for analysis.

The paper continues with the introduction of the modified TOFU model in Section 5 and the security proof in a new TOFU-or-DOFU model in Section 7. Bluetooth-tailored notion of authentication is presented in Section 6. The example of how the suggested solution helps against some of the attacks is given in Section 8. The results and recommendations are discussed in Section 9.

## 2 Notation

We briefly recap the notation used throughout the paper and in Bluetooth. The model-specific definitions are given in Section 5. Acronyms used in Bluetooth and in the paper can be found in Appendix B.

### 2.1 General Notation

By $\mathbb{Z}_m$ we denote the set of integers $0, \ldots, m-1$, often also view it as the group or ring with common modular addition and multiplication as operations. With $x \leftarrow\!\$\, X$ we denote a uniformly sampled element from the set $X$. In particular, $x \leftarrow\!\$\, \mathbb{Z}_{(q+1)/2} \setminus \{0\}$ means to uniformly pick an integer $x$ from the range $1, \ldots, (q+1)/2$, where $q$ is odd. We denote by $g^x$ the $x$-fold multiplication of the generator $g$ of a group. The common Diffie–Hellman share of public data $g^a$ and $g^b$ is thus given by $g^{ab}$. When using elliptic curves, as in Bluetooth, it is understood that $g^a$ is the $a$-fold addition of the curve point $g$. In this case, we denote by $\langle g^a \rangle_x$ the $x$-coordinate of the curve point. For more background on elliptic curves see [Sil09].

The notation $V \leftarrow v$ means that value $v$ is assigned to variable $V$, $V := W$ means the variable $V$ is defined to be equal to variable $W$, and $\perp$ denotes an uninitialized value or an error. For example, for an array $A[]$ we let $A[i]$ be the $i$th element, where we assume $A[i] = \perp$ if the value has not been set previously. Given a vector $A$ of named attributes $a, b, c, \ldots$, we often denote the values of the attributes as $A.a, A.b, A.c, \ldots$ or simply as $a, b, c, \ldots$ if the context $A$ is clear. For strings $s$ and $r$ we denote by $s|r$ the concatenation of the two strings, and the bit-wise XOR of equal-length strings $s$ and $r$ by $s \oplus r$.

The deterministic resp. randomized output $y$ of an algorithm $\mathcal{A}$, run with input $x$, is denoted by $y \leftarrow \mathcal{A}(x)$ resp. $y \leftarrow\!\$\, \mathcal{A}(x)$. We use square brackets to denote an optional input to an algorithm, e.g., $y \leftarrow\!\$\, \mathcal{A}(x, [a])$ means that algorithm $\mathcal{A}$ may receive auxiliary input $a$ in addition to $x$. We write $\mathcal{A}^{\mathsf{O}_1, \ldots, \mathsf{O}_n}$ if the algorithm has access to oracles $\mathsf{O}_1, \ldots, \mathsf{O}_n$.

The security parameter is denoted by $\lambda$ or, written in unary, by $1^\lambda$. If adversary $\mathcal{A}$ interacts in some security experiment Game with some scheme $\Pi$ for security parameter $\lambda$, then we write $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{Game}}(\lambda) = 1$

to denote that the experiment outputs 1. This usually indicates an adversary's success, such that $\Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\text{Game}}(\lambda) = 1\right]$ represents the probability of this success. The advantage of the adversary is then denoted by $\mathbf{Adv}_{\mathcal{A},\Pi}^{\text{Game}}$ and measures the adversary's success probability over trivial strategies, e.g., over the guessing probability $\frac{1}{2}$ for decisional experiments.

### 2.2 Bluetooth-specific Notation

Bluetooth employs a number of variables, whose notation follows the standards closely and which we present here. Each party has a Bluetooth address, which we denote by `BD_ADDR` or sometimes as $A$. Furthermore, we use the common values appearing in the pairing step, including the nonce $N$, random passkey $r$, commitment value $C$, verification value $V$, and check value $E$. Devices usually provide or take input and output capabilities *IOcap*, authentication requirements *AuthReq*, and out-of-band authentication data *OOB*.

The established keys in Bluetooth are denoted as the link key *LK* (for BR/EDR), the long-term key *LTK* (for BLE), or either of them *L(T)K*. To derive further session-specific keys, devices use the device key *dk*, the authentication random number *AU_RAND*, the signed response *SRES*, the session key diversifier *SKD*, the initialization vector *IV*, the authentication ciphering offset *ACO*, and the encryption key $\mathsf{k}_{\text{AES}}$. The latter is used to protect the actual communication.

Bluetooth often uses truncated strings, i.e., we write $s/2^n$ if the $n$ leftmost bits of string $s$ are taken, and $s \bmod 2^{32}$ if least significant bits of $s$ are considered. If the string is given in hexadecimal representation, this is indicated by the prefix `0x`.

## 3 Bluetooth Background

Further we give a concise explanation of the main aspects of the Bluetooth technology. Today, Bluetooth exists in two main versions, low-energy (BLE) and classic (BR/EDR).[3] Both versions have similar discovery and connection mechanisms but they deviate in the technical aspects and applications, making them incompatible with each other. Devices might implement either of the versions (single-mode) or both (dual-mode). The latter may also switch the mode (transport) afterwards in reconnections.

Devices supporting BR/EDR are equipped with a unique and registered 48-bit MAC-address, called public (identity) address. BLE extends the address space by static random addresses of the same size, which are generated once upon booting up but otherwise stay fixed. For improved privacy, BLE also introduces resolvable and non-resolvable random private addresses. The former type can be used in reconnections, after the initial connection has been carried out with a static address. It can be resolved by the other party in a reconnection and mapped back internally to the static addresses via entries of already established connections.

The Bluetooth technology consists of the protocols on different layers, which are split into two components: the *host* for higher-layer aspects and the *controller* for lower-layer aspects—connected through a host-controller interface. The controller is usually harder to modify since it is closer tied to the hardware design. This is why we target the higher-layer protocols to integrate our authentication subprotocol. We emphasize that the cryptographic part is treated differently in the different versions: in BR/EDR, the Link Manager Protocol (LMP) is responsible for the cryptographic components, while in BLE, the cryptographic parts are moved to Security Manager Protocol (SMP).

To connect, devices make links on several layers: physically, logically, and cryptographically. The example protocol flow for the connection in BR/EDR on a cryptographic level is shown in Figure 1. During the link establishments, devices learn the Bluetooth Address `BD_ADDR` of each other, the devices'

[3]We neglect the Legacy protocols and the Mesh transport here.

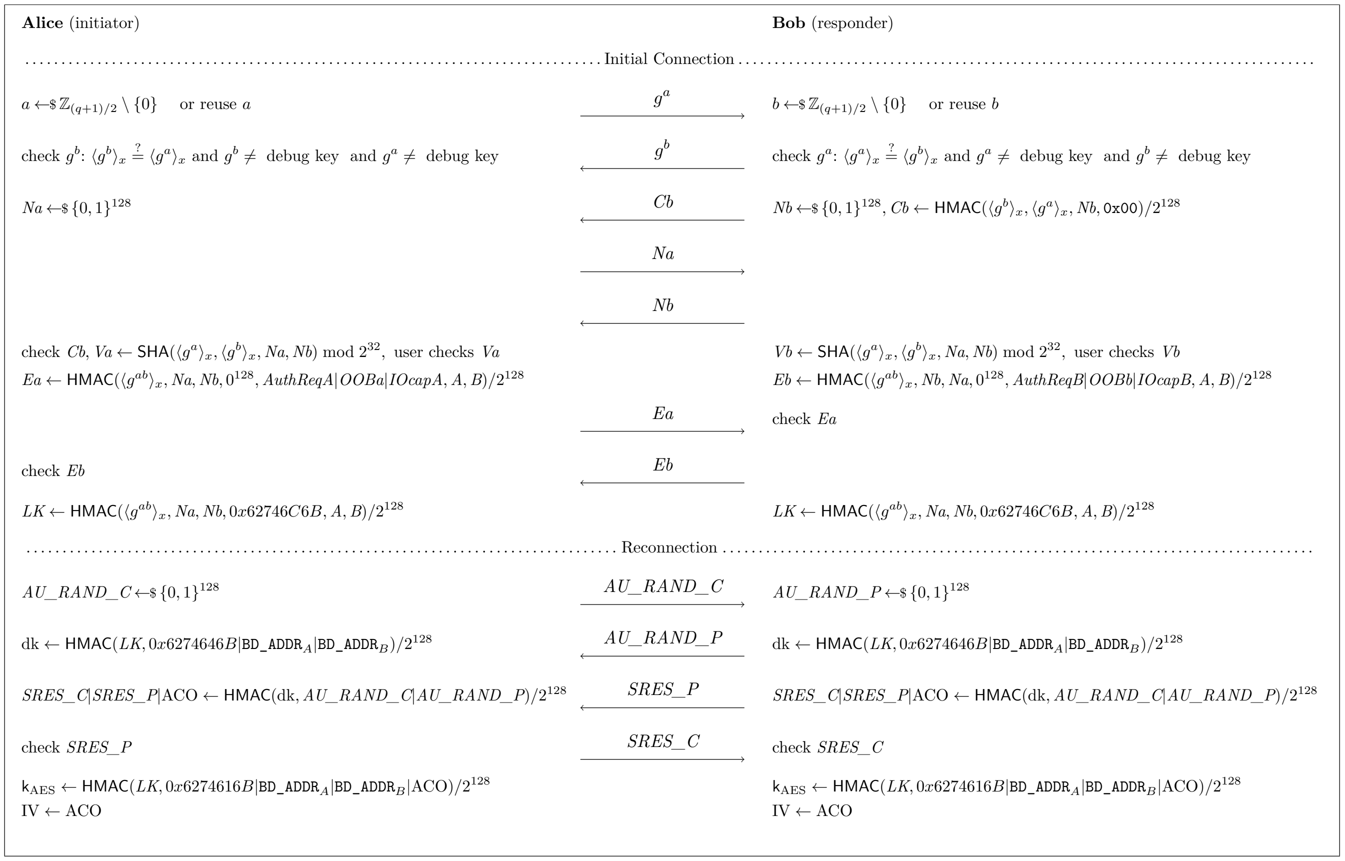
**Alice** (initiator)
**Bob** (responder)
Initial Connection
$a \leftarrow\!\$\, \mathbb{Z}_{(q+1)/2} \setminus \{0\}$ or reuse $a$
$g^a$
$b \leftarrow\!\$\, \mathbb{Z}_{(q+1)/2} \setminus \{0\}$ or reuse $b$
check $g^b$: $\langle g^b\rangle_x \overset{?}{=} \langle g^a\rangle_x$ and $g^b \neq$ debug key and $g^a \neq$ debug key
$g^b$
check $g^a$: $\langle g^a\rangle_x \overset{?}{=} \langle g^b\rangle_x$ and $g^a \neq$ debug key and $g^b \neq$ debug key
$Na \leftarrow\!\$\, \{0,1\}^{128}$
$Cb$
$Nb \leftarrow\!\$\, \{0,1\}^{128}$, $Cb \leftarrow \mathsf{HMAC}(\langle g^b\rangle_x, \langle g^a\rangle_x, Nb, \mathtt{0x00})/2^{128}$
$Na$
$Nb$
check $Cb$, $Va \leftarrow \mathsf{SHA}(\langle g^a\rangle_x, \langle g^b\rangle_x, Na, Nb) \bmod 2^{32}$, user checks $Va$
$Ea \leftarrow \mathsf{HMAC}(\langle g^{ab}\rangle_x, Na, Nb, 0^{128}, AuthReqA|OOBa|IOcapA, A, B)/2^{128}$
$Vb \leftarrow \mathsf{SHA}(\langle g^a\rangle_x, \langle g^b\rangle_x, Na, Nb) \bmod 2^{32}$, user checks $Vb$
$Eb \leftarrow \mathsf{HMAC}(\langle g^{ab}\rangle_x, Nb, Na, 0^{128}, AuthReqB|OOBb|IOcapB, A, B)/2^{128}$
$Ea$
check $Ea$
check $Eb$
$Eb$
$LK \leftarrow \mathsf{HMAC}(\langle g^{ab}\rangle_x, Na, Nb, 0x62746C6B, A, B)/2^{128}$
$LK \leftarrow \mathsf{HMAC}(\langle g^{ab}\rangle_x, Na, Nb, 0x62746C6B, A, B)/2^{128}$
Reconnection
$AU\_RAND\_C \leftarrow\!\$\, \{0,1\}^{128}$
$AU\_RAND\_C$
$AU\_RAND\_P \leftarrow\!\$\, \{0,1\}^{128}$
$\mathrm{dk} \leftarrow \mathsf{HMAC}(LK, 0x6274646B|\mathtt{BD\_ADDR}_A|\mathtt{BD\_ADDR}_B)/2^{128}$
$AU\_RAND\_P$
$\mathrm{dk} \leftarrow \mathsf{HMAC}(LK, 0x6274646B|\mathtt{BD\_ADDR}_A|\mathtt{BD\_ADDR}_B)/2^{128}$
$SRES\_C|SRES\_P|\mathrm{ACO} \leftarrow \mathsf{HMAC}(\mathrm{dk}, AU\_RAND\_C|AU\_RAND\_P)/2^{128}$
$SRES\_P$
$SRES\_C|SRES\_P|\mathrm{ACO} \leftarrow \mathsf{HMAC}(\mathrm{dk}, AU\_RAND\_C|AU\_RAND\_P)/2^{128}$
check $SRES\_P$
$SRES\_C$
check $SRES\_C$
$\mathsf{k}_{\mathrm{AES}} \leftarrow \mathsf{HMAC}(LK, 0x6274616B|\mathtt{BD\_ADDR}_A|\mathtt{BD\_ADDR}_B|\mathrm{ACO})/2^{128}$
$\mathrm{IV} \leftarrow \mathrm{ACO}$
$\mathsf{k}_{\mathrm{AES}} \leftarrow \mathsf{HMAC}(LK, 0x6274616B|\mathtt{BD\_ADDR}_A|\mathtt{BD\_ADDR}_B|\mathrm{ACO})/2^{128}$
$\mathrm{IV} \leftarrow \mathrm{ACO}$

Figure 1: Bluetooth initial pairing (SSP in mode NC) and reconnection (Secure Authentication and Encryption Key Derivation). The session identifier in the initial connection is $\mathsf{sid} = (\langle g^a\rangle_x, \langle g^b\rangle_x, A, B, Na, Nb)$ and in the reconnection $\mathsf{sid} = (AU\_RAND\_C, AU\_RAND\_P, A, B)$.

capabilities (i.e., what algorithms they support and security they request). To establish cryptographic keys and start secure data exchange, devices need to pair. At this step, they can decide if they want to create a bond during the pairing (i.e., store the derived keys and delete all the temporal pairing information) or opt for one-time connection only. Here the devices also decide if they want to use Secure Connections Only (SCO) or agree on weaker versions of the protocol.

The basis of SCO is the Secure Simple Pairing (SSP) protocol, a Diffie–Hellman-based key exchange protocol. SSP comes in four possible association models: Out-of-Band (OOB) with the exchange of data over the secure channel outside of the Bluetooth transmission; PasskeyEntry (PKE) resp. NumericComparison (NC) with the user entering resp. comparing 6 displayed digits; and JustWorks (JW) with no attempt to protect against MitM attacks. As a result, SSP with the corresponding association model outputs the key, which is then used for the derivation of the encryption keys with the lifespan of a session. Usually the derived keys (link key $LK$ for BR/EDR and long-term key $LTK$ for BLE) are stored in the host and passed to the controller when the latter requests them.

The most vulnerable stage is the exchange of capabilities, as this stage influences the choice of the association model, encryption key size[4], and types of keys to derive. Most of the attacks happen here, e.g., modifying IO capabilities allows the adversary to launch the downgrade attack [HH07] to the JW association model, or the Method Confusion Attack [vTPFG21], or an impersonation attack [ZWD+20]; modifying OOB flag leads to the drop of the OOB association model [HT10]; avoiding SCO downgrades

[4]In BR/EDR, the negotiation of the encryption key size happens during the reconnection.

the protocol to the legacy version, which enables Pairing Confusion Attack [CADE23], BIAS [ATR20a], BLUFFS [Ant23], or attacks on the Legacy protocols [Ros13, Rya13]. Adversary can also downgrade the size of the keys [ATR20b, ATR19] or confuse the devices on what key size to use [SCH$^+$23].

Since the encryption keys are valid for one session, the devices can reconnect using the stored link/long-term key and authenticate each other with this key. In BR/EDR, reconnection is a challenge-response scheme. The parties each choose a fresh nonce (challenge), exchange it, and authenticate each other with the responses, which are based on these fresh values and the stored key. In BLE, the procedure is simpler and consists of the two exchanged fresh values to which the *LTK* is applied to generate the encryption key.

**Cryptographic Details.** When SSP starts, the devices possess the Bluetooth addresses `BD_ADDR` as identities and the *IOcap* of each other and know what cryptographic algorithms the partner is able to use. This includes the used elliptic curves, which are FIPS-approved and given in the Core Specification. For curve points, Bluetooth makes use of the x-coordinate only (although the y-coordinate is also shared to prevent the Invalid Curve Attack [BN19]), which we mark as $\langle g^a \rangle_x$. Both parties initially establish a Diffie–Hellman key, then run the corresponding association model, and finally confirm the derived keys via values *Ea* and *Eb*.

Both versions of Bluetooth employ different cryptographic algorithms: AES-CMAC in BLE and SHA with HMAC in BR/EDR. To compute the commit and confirmation values, the link key, the device key (a spin-off key for authentication in reconnections), the encryption key, and the response to a challenge, BR/EDR uses HMAC; the derived DHKey is straight used as MACKey; for the computation of the displayed digits in NC, SHA-256 is directly employed. BLE uses AES-CMAC for all these cases, apart from the encryption key derivation in which simply AES encryption is used. Bluetooth usually truncates the output and uses the leftmost $n$ bits, which we denote by $/2^n$.

## 4 Related Work

In the following, we detail related works about attacks on Bluetooth and potential countermeasures, existing analyses of the protocols and cryptographic techniques related to our enhancement of Bluetooth.

### 4.1 Attacks on Bluetooth

Cäsar et al. [CPST22] give an overview of the attacks on BLE in both Legacy and SC. They look into the versions of the Core Specification from 4.0 to 5.2 and cover attacks that violate both security and privacy, even if they have been mitigated. Wu et al. [WWX$^+$24] extend the overview to BR/EDR and Mesh and include the attacks on a soft- and hardware level. We refer to these papers for an overview of the vulnerabilities in Bluetooth.

Most of the attacks are mitigated by our solution (Table 1) since they result in an honest device and a compromised device deriving different *L(T)K*s in their corresponding sessions, and hence the adversary will not be able to go through the authentication process if the signature or MAC is not calculated over the same keys. In the following, we describe the attacks that are relevant for this paper and why other proposals patch (or not) these attacks.

**Legacy Attacks.** Legacy protocols are subject to attacks in both BLE [Rya13] and BR/EDR [JW01]. The problem with the protocols is that they do not even aim to protect against the eavesdropping attackers.

In BLE, the Legacy protocol is based on the commit-reveal scheme, i.e., both parties must first commit the temporal key *TK* using some random numbers and then reveal those numbers. It is possible to brute-force *TK* and find a collision, such that the revealed random number would (together with the brute-forced

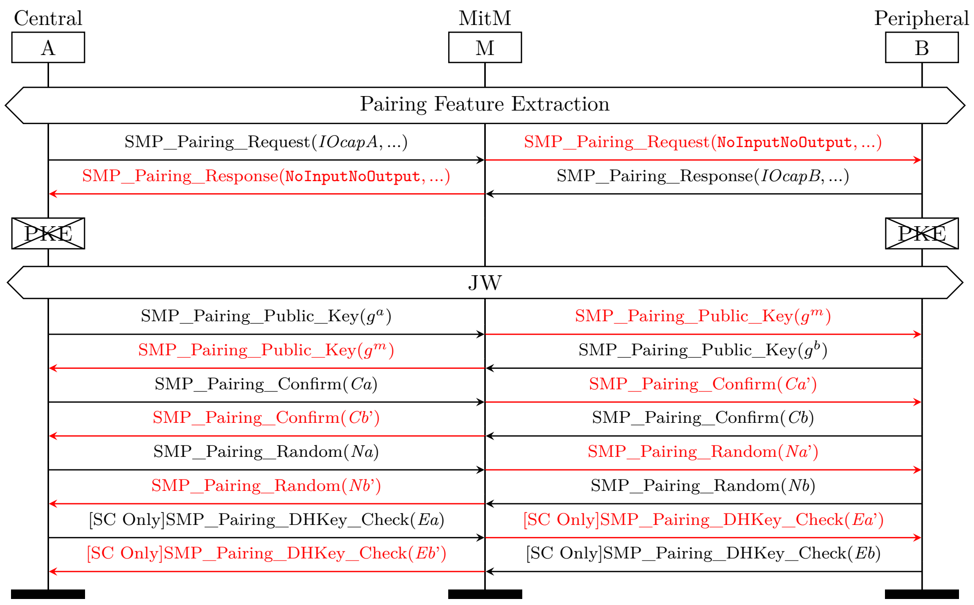
Central
A
MitM
M
Peripheral
B
Pairing Feature Extraction
SMP_Pairing_Request(IOcapA, ...)
SMP_Pairing_Request(NoInputNoOutput, ...)
SMP_Pairing_Response(NoInputNoOutput, ...)
SMP_Pairing_Response(IOcapB, ...)
PKE
PKE
JW
SMP_Pairing_Public_Key(g^a)
SMP_Pairing_Public_Key(g^m)
SMP_Pairing_Public_Key(g^m)
SMP_Pairing_Public_Key(g^b)
SMP_Pairing_Confirm(Ca)
SMP_Pairing_Confirm(Ca')
SMP_Pairing_Confirm(Cb')
SMP_Pairing_Confirm(Cb)
SMP_Pairing_Random(Na)
SMP_Pairing_Random(Na')
SMP_Pairing_Random(Nb')
SMP_Pairing_Random(Nb)
[SC Only]SMP_Pairing_DHKey_Check(Ea)
[SC Only]SMP_Pairing_DHKey_Check(Ea')
[SC Only]SMP_Pairing_DHKey_Check(Eb')
[SC Only]SMP_Pairing_DHKey_Check(Eb)

Figure 2: Message flow for the NINO-attack on [TH21] and [SCH$^+$23].

*TK*) end up to be equal to the value committed by the attacker. Such attack is found by Rosa [Ros13] and called Flawed Bit Commitment. (Note that the LE Legacy JW and PKE protocols with short passkeys are not secure even against passive adversaries, as shown by Ryan in [Rya13], since the temporal key *TK* can be recovered from the conversation and the short-term key *STK* is derived from the *TK* using two random values revealed in plaintext).

In the BR/EDR Legacy protocol, a PIN of size from 1 byte to 16 bytes is used. This PIN together with the two exchanged in plaintext random values is used to derive a $\mathsf{K}_{\mathsf{init}}$, which in turn is used for exchanging the combination keys, and the latter is used for the *LK* derivation. Devices that use low-entropy PINs are prone to PIN brute-forcing (e.g., when confused with a 6-digited passkey). The example of such an attack is presented by Jakobsson and Wetzel [JW01].

The Legacy protocols can be only secure, if no adversary was present during the pairing process and could not record the transcript for the key recovery. Our solution allows the device to be sure that the key was derived with the genuine device, but this does not, however, protect the key from being computed. Hence, the only way to secure protocols is to discard the Legacy protocols.

**NINO / Downgrade to JW.** The lack of IO and OOB capability authentication leads to the most dramatic attack: association model downgrade. In several works (e.g., [HH07, HT10]), researchers managed to show (also in practice) that the specification is prone to downgrading to the "weaker" association models. While the fixes suggested both in [TH21] and [SCH$^+$23] suggest modifications to the standalone protocols, they do not take into account the interplay of the different association models. That is, the adversary can still downgrade the communication to JW association model by simply modifying the *IOcap* of devices. Thus, the devices will not even enter the modified protocol and simply stay at the less secure association model. The flow of the attacks on fixes is similar to the standard downgrade attacks [HH07] and is shown in Figure 2. Our proposed fix mitigates the downgrade because the parties derive different *LK*s in such attacks, hence the adversary would need to forge the signature or MAC to pass the authentication step.

**Method Confusion.** Von Tschirschnitz et al. [vTPFG21] introduced the Method Confusion Attack that exploits the similarity of passkeys displayed or entered in the PKE and NC association models. That is, an attacker tampers devices' *IOcap*, so that each device chooses a different from another association model, i.e., one opts for PKE and another goes with NC instead of the agreement on the same association model. Since the user cannot make a distinction between the two association models, and devices do not authenticate the capabilities, the attack will remain unnoticed.

Neither of the suggestions in [TH21] is secure against the Method Confusion Attack, as the Dual Passkey Entry can be simply avoided by replacing the *IOcap*. The flow of the attack when one of the connections is replaced with another method is shown in Figure 3. There, the adversary still can manage to go through the Dual Passkey Entry because the other connection in NC can be used to make the user enter the passkey on Device $B$ and trigger a new session in PKE to make the user enter the passkey on Device $A$. In NC, the adversary learns the passkey as it will be displayed on its device, while in PKE, the adversary gradually reveals passkey as the protocol does not protect the passkey.

In [SCH$^+$23], displaying or entering the passkey happens twice, and the second passkey consists of the first passkey concatenated to the *IOcap*, which are different for the two association models, so the attack is mitigated. Our scheme does not allow the adversary to pass the authentication step since both devices compute different *L(T)K*s.

**Pairing Confusion.** The Pairing Confusion Attacks [CADE23] are close to the Method Confusion Attacks [vTPFG21] with the difference that the confusion between the SC and Legacy association models is added. While in the Method Confusion Attack, the adversary changes the *IOcap* flag, in Pairing Confusion, the attacker tampers with the *SC* flag in BLE or sends LMP_IN_RAND (instead of LMP_AU_RAND) PDU in BD/EDR.

The attack flow is similar to the Method Confusion Attack, so the solution in [TH21] is neither secure against the Pairing Confusion Attacks. The solution proposed in [SCH$^+$23] is not secure against pairing confusion as their fix does not aim to prevent attacks on the Legacy protocols. That is, the attacker can downgrade one of the connections to Legacy and use this connection to learn the passkey used in another connection. Analogously to the Method Confusion Attack, the user and the devices are not aware of the capabilities and do not notice the mismatch of the protocols' run on both sides due to their similar nature. Our suggestion prevents this attack as in both connections, the different keys will be derived, hence the adversary will not pass the authentication procedure without forging.

**KNOB and Keysize Confusion.** Antonioli, Tippenhauer, and Rasmussen [ATR19] showed an attack on the Key Negotiating mechanism Of Bluetooth (KNOB) in BR/EDR that happens during the Authentication and Encryption stage after the initial connection. The KNOB attack allows the adversary to downgrade the size of the encryption key in BR/EDR to 1 byte what makes the key easy to brute-force. In the subsequent work [ATR20b], the same authors found that KNOB mechanism in BLE (that is done during the Pairing Feature Extraction) is also subject to the same attack. The KNOB attacks are possible because keys' size lacks authentication.

Shi et al. [SCH$^+$23] discovered a new attack they call the Keysize Confusion Attack. The core idea behind the attack is similar to the KNOB attack with the adversary manipulating the key size field, but instead of downgrading the key size to the lowest, the adversary makes the parties accept with different key sizes. This attack is possible due to the same reason as the KNOB attack: there is no security mechanism to ensure and authenticate the partner's choice of the key sizes. It is worth to note that the adversary does not learn the resulting *L(T)K*s of the parties but only makes the parties accept with different key sizes (and hence also keys).

The same paper [SCH$^+$23] attempts to mitigate both KNOB and Keysize Confusion attacks by includ-

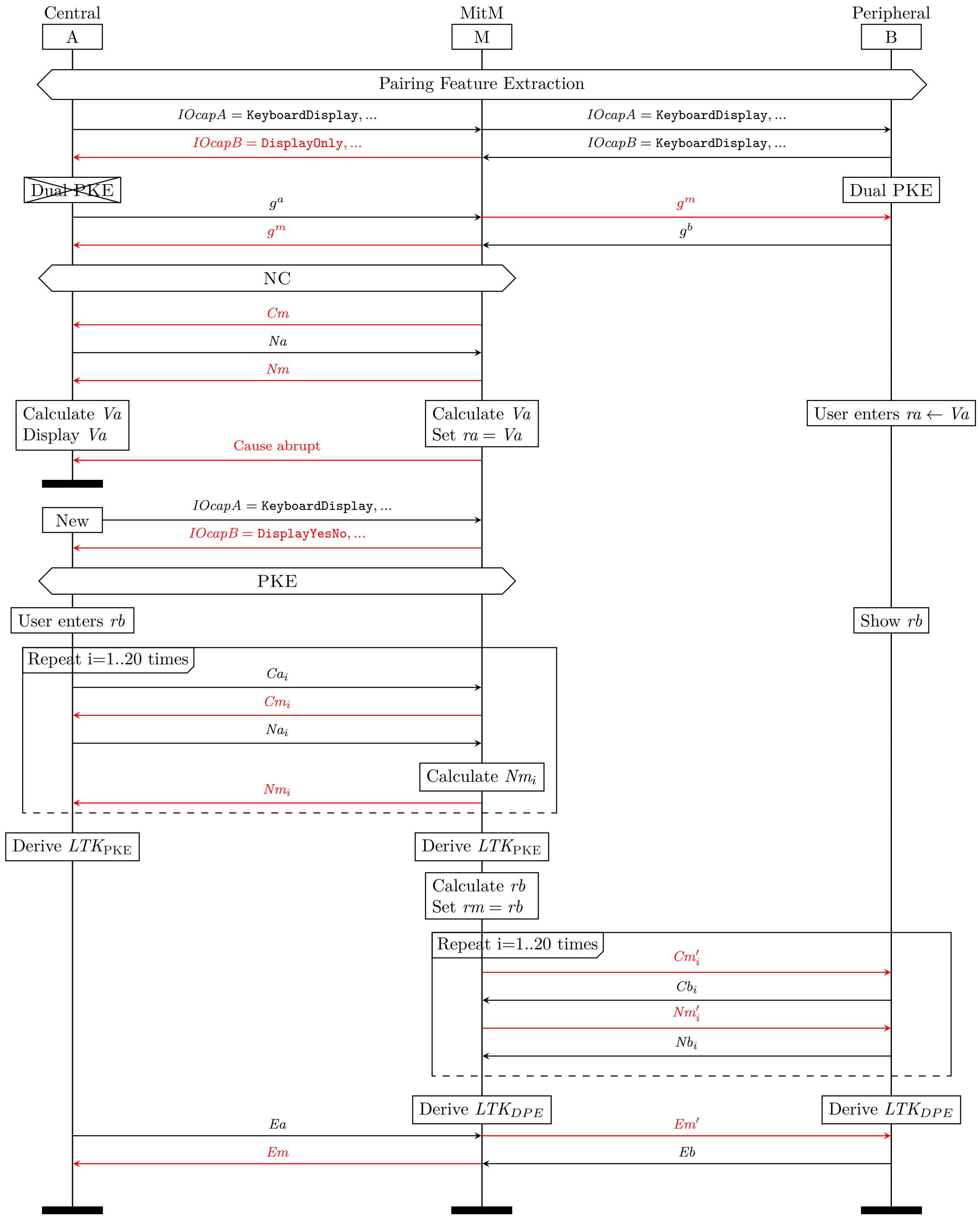
Central
A
MitM
M
Peripheral
B
Pairing Feature Extraction
$IOcapA$ = KeyboardDisplay, ...
$IOcapB$ = DisplayOnly, ...
$IOcapB$ = KeyboardDisplay, ...
~~Dual PKE~~
Dual PKE
$g^a$
$g^m$
$g^b$
NC
$Cm$
$Na$
$Nm$
Calculate $Va$ Display $Va$
Calculate $Va$ Set $ra = Va$
User enters $ra \leftarrow Va$
Cause abrupt
New
$IOcapB$ = DisplayYesNo, ...
PKE
User enters $rb$
Show $rb$
Repeat i=1..20 times
$Ca_i$
$Cm_i$
$Na_i$
Calculate $Nm_i$
$Nm_i$
Derive $LTK_{\text{PKE}}$
Calculate $rb$ Set $rm = rb$
$Cm'_i$
$Cb_i$
$Nm'_i$
$Nb_i$
Derive $LTK_{DPE}$
$Ea$
$Em'$
$Em$
$Eb$

Figure 3: Message flow for the Method Confusion Attack [vTPFG21] on fixes in [TH21].

ing the supported key size *MaxEnc* along with other values into the passkey computation. This, however, limits the power of their patch, as other than NC and PKE association models in BLE and the whole key negotiation mechanism in BR/EDR remain vulnerable to both attacks. Troncoso and Hale's modification [TH21] only includes *IOcap* of the devices and hence does not prevent the change of the key size. Our scheme could prevent the Keysize Confusion and KNOB attacks for BLE if a collision-resistant cryptographic function is used instead of AES-CMAC. Yet, the key negotiation mechanism in BR/EDR is not secured by our solution, as the negotiation happens during each reconnection step (and not during the pairing, as in BLE) and cannot be patched in a backward-compatible manner. Moreover, we have to assume that devices only use the maximum session key size—16 bytes—in order for the proof for BR/EDR to work.

**Initiator / Responder Role Confusion.** Since the parties in the conversation are not always equal, there is a space for role confusion or mismatching attacks. Troncoso and Hale [TH21] have found a confusion attack on the initiator/responder roles in the PKE association model, when a user generates the passkey and enters it on the both devices[5]. While good AKE protocols usually have a protection against such attacks (e.g., the role is included into the key computation), Bluetooth prevents this only by the fixed order of Central's and Peripheral's (i.e., initiator's and responder's) input values in the *L(T)K*s computation. That is, the devices might notice the attack via *L(T)K*s mismatch at the reconnection step, when computing a new session key (yet the both derived *L(T)K*s remain unknown to the adversary).

Troncoso and Hale [TH21] suggest a fix to prevent the role confusion attack they discovered by including the roles (initiator or responder) into the nonce computation. Shi et al. [SCH+23] implicitly fix this attack by adding pairing and response request messages into passkey computation in PKE and NC. As the role influences the input in *L(T)K*s computation, what makes parties derive different keys, in our solution, the adversary needs to commit forgery in order to pass the authentication stage.

Initiator resp. responder roles are also tied to the Central resp. Peripheral roles in Bluetooth. Antonioli, Tippenhauer, and Rasmussen [ATR20a] found several attacks they call Bluetooth Impersonation AttackS (BIAS). The first attack is based on the one-wayness of the challenge-response procedure in Legacy reconnections, where only the Central had to ask the Peripheral to respond to the challenge itself. If the Central device was the target, the attacker could ask for a role switch and exploit the first attack. If two devices had used Secure Connections for pairing, it was possible to downgrade to Legacy reconnections. The attacks themselves do not leak or help the adversary to learn neither the *LK*, nor the encryption keys. The current version of the Core Specification forbids the downgrade to Legacy reconnections for Secure Connections pairing, disallows role switching during the reconnection in progress, and enforces the mutual authentication in Legacy reconnections.

**Cross-platform Key Derivation.** Bluetooth's Core Specification allows the dual-mode devices to pair only once for either mode and derive the cross-platform key during this pairing. While facilitating the usability for users, the process of the cross-platform key derivation was shown to be prone to attacks: BLURtooth [ATRP22] and Cross Stack Illegal Access (CSIA) [WWX+22].

The BLURtooth attack [ATRP22] allows rewriting a "secure" key derived from genuine device interaction with a "less secure" cross-platform key from another transport, e.g., "unauthenticated" or with the lower entropy. The Cross Stack Illegal Access (CSIA) [WWX+22] attack considers a semi-compromised setting, when a malicious app, installed on a device with the access to the key of one transport, could use the cross-platform key on another transport to access the data it should have not in the first place. Since this setting requires compromised devices, the SIG treats this attack as out of scope.

Neither of the suggested solution prevents BLURtooth nor CSIA attacks, as they do not prevent the downgrade attack (what makes BLURtooth possible) and do not consider the compromised setting

[5]While [TH21] showed the attack only for the user-generated `passkey` entered on both `KeyboardOnly` devices, we do not see a reason why this attack does not apply to other cases in PKE as well as in the JW and OOB association models. NC prevents this attack by displaying the digits computed from the fixed-order input.

like in CSIA. If the app has the access to the *L(T)K*s only, but not the signing key, this attack could have been mitigated with our solution. However, our scheme also leaves this attack out of scope, since conversion is done with the help of a non-collision-resistant function AES-CMAC, i.e., even if devices derive different *L(T)K*s during the pairing, the converted keys might collide. The solution could be to replace the cryptographic function used for key derivation or to disable the possibility to convert the key.

**Other Attacks.** Claverie and Esteves [CE21] investigated reflection attacks on PKE they call BlueMirror that led to the successful impersonation of parties during the initial connection. The attack exploited the symmetric nature of PKE and the fact that the passkey is revealed after the full Authentication stage 1 execution. The reflection attack involved a MitM adversary, which used connection with one party to learn the passkey by simply reflecting the values sent by this party. The learned passkey was then used in the second connection to impersonate the genuine party from the first connection. This attack was mitigated in the Bluetooth Core Specification v.5.3, which mandates to check the equality of the received values to their own sent values.

A similar idea to KNOB and BIAS is used in BLUetooth Forward and Future Secrecy (BLUFFS) attacks [Ant23]. The attacks exploit the Legacy encryption key derivation, which bases the randomness of the key on only one nonce from a Central. Hence, the adversary can impersonate the Central and reuse the same nonce in multiple sessions.

Wu et al. [WNK$^+$20] considered reactive and proactive Spoofing Attacks on BLE (BLESA) in accessing the attributes: The proactive attack was an implementation drawback with the wrong handling of the errors, and the reactive attack allows "impersonation" of the server/Peripheral towards the client/Central with the spoofed responses. The latter attack may be fixed only by enforcing the encryption by default, as it does not happen during the key establishment or authentication stages but rather after the encryption should have been enabled.

Levi et al. [LÇA$^+$04] found two relay attacks (two-sided and one-sided) on the Legacy reconnection in BR/EDR. The two-sided simply relays the messages in the Legacy mutual reconnection step between two parties, and the one-sided attack allows relaying the challenge from one party to another by terminating one of the connections. These attacks do not result in the adversary learning the key, hence we leave the attacks out of scope.

## 4.2 Analyses of Bluetooth Security Protocols

Bluetooth was also a target of several security analyses. While some results purely focused on single protocols, others tried to consider the whole protocol stack. Both security proofs and automated tools were employed for the analysis.

**Complexity-based Analyses.** Lindell has analyzed the NC association model in [Lin09] and showed NC to be secure as comparison-based key exchange, notion of which is also introduced in that paper. The analysis of the protocol is done in isolation and assumes that the Bluetooth addresses (which are not constant identities) are included in the confirmation value computation. Sun and Sun [SS19] have also analyzed NC and OOB in isolation and showed their security. Their result is the same as [Lin09], but the model is more restricted, as they do not allowing the adversary to communicate with the parties after testing.

Troncoso and Hale [TH21] have developed a model for the analysis of protocols with user interaction. The model, called CYBORG, captures an adversarial influence on the User-to-Device (UtD) channel. They analyze different cases (which depend on the *IOcap* of the devices) in the PKE association model of Bluetooth SSP and show that PKE does not provide security in the CYBORG model: in the case of the user-generated passkey, the adversary can perform a role confusion attack and make the sessions accept with role disagreement; in case of the device-generated passkey, the adversary is able to perform a guessing attack and win with non-negligible probability. Troncoso and Hale do not provide a detailed analysis of NC subprotocol, but conclude it does not achieve the security with respect to the CYBORG model.

Yin [Yin23] has analyzed the OOB and NC association models in terms of the CYBORG security model. The analysis showed only bidirectional OOB to be CYBORG-secure: For NC, they found an attack where the adversary successfully substitutes the user's response to the device from negative to positive; in unidirectional OOB, they allowed the adversary to modify the data in the OOB channel and showed a successful attack in this setting. Yin has also extended the CYBORG model to address Tap'n'Ghost attacks, where the channel is authenticated but not confidential, and called it Authentication Protocols assisted by EXtra (APEX) model. They showed the APEX-security of NC and bidirectional OOB but presented an attack on the improved PKE from Troncoso and Hale [TH21] with the leaked passkey and unidirectional OOB (where the non-authenticated party is impersonated). Both scenarios are not intended to be secure according to the Bluetooth Core Specification. The analysis considered all the protocols in isolation, what does not represent the real-world scenario and ignores the known downgrade and confusion attacks on Bluetooth, such as [HH07, vTPFG21, CADE23].

Fischlin and Sanina [FS21] have analyzed all Secure Connections protocols together (i.e., excluding the Legacy protocols) and showed the security of Bluetooth in the trust-on-first-use (TOFU) model, where the adversary can eavesdrop on the initial connection and be active during the reconnections. While that analysis opted for the unmodified "vanilla" protocol and showed the best security the current version can achieve, this paper aims to improve the Bluetooth security guarantees by modifying the protocol in a backward-compatible manner.

**Formal Analyses.** Chang and Shmatikov [CS07] have conducted the first formal analysis on Bluetooth: namely, the BR/EDR Legacy protocol and the NC association model of the Bluetooth Simple Pairing protocol (the predecessor of SSP). They have used ProVerif for the analysis and modeled the user as a human oracle. The analysis confirmed the insecurity of BR/EDR Legacy and found a vulnerability that results in the difficulty for the user to distinguish for which concurrent sessions the numbers are being matched. As a solution, the authors suggest adding session identificators into the displayed digits' computation.

Wu et al. [WNK$^+$20] have used ProVerif to analyze the reconnection stage in BLE and found the Bluetooth Low Energy Spoofing Attacks (BLESA) that allows the adversary to spoof the attribute value from the Peripheral and use the value to disable encryption on the Central's side.

Wu et al. [WWX$^+$22] have analyzed all association models and their interplay in all stages in BR/EDR, BLE, and Bluetooth Mesh using ProVerif. They confirmed 5 known vulnerabilities and discovered two new attacks. The first attack is the Authentication Neutralization (BlueMAN) attack, in which the adversary forwards the provisioning information in Bluetooth Mesh and records the transcript to derive the session key and decrypt the data. The second attack is the Cross Stack Illegal Access (CSIA) attack, which, like the BLURtooth attack [ATRP22], exploits the cross-platform mechanism but requires one of the devices to be compromised.

Jangid, Zhang, and Lin [JZL23] have analyzed the PKE association model using Tamarin. They confirmed known attacks, such as the Method Confusion Attack [vTPFG21], the reflection attack [CE21], the static passkey reuse attack [SMS18], and uncovered two new attacks: Group Guessing and Ghost Keystroke. While the first one is a type of the static reuse attack, when several connections are in place and devices use the same passkey for concurrent sessions, the second is a type of semi-honest connection, similar to the attack on PKE shown by Zhang et al. in [ZWD$^+$20]. They also suggested countermeasures for the already known and newly discovered attacks: ban on passkey reuse, check on the equality to the own values, distinguished UI for different association models, and locking of the devices to avoid compromise.

Claverie et al. [CADE23] have analyzed using Tamarin all the association models and protocols available in BR/EDR. They repeated the analysis separately for BLE and Bluetooth Mesh. They found an attack they call pairing confusion, in which the adversary downgrades to a Legacy protocol in one of the connections, what results into confusion with the SSP association models.

Shi et al. [SCH$^+$23] have analyzed all association models and their interplay in all stages of the BLE SC protocol using Tamarin. They confirmed the Method Confusion Attack [vTPFG21] and found a new confusion attack for the key size. To overcome these two attacks, the authors proposed countermeasures for PKE and NC: run the protocol twice and include the previously generated passkey/digits together with the data from the pairing request/response as an input into the second stage.

## 4.3 Suggested Countermeasures

Researchers have suggested various fixes for Bluetooth protocols. Some fixes add modifications to existing protocols, some design completely new protocols. In the following, we describe the most relevant suggested fixes and discuss their advantages and drawbacks.

After the analysis of PKE, Troncoso and Hale [TH21] proposed two modifications to partly achieve CYBORG-security for this association model. The first modification (called Secure Hash) aims to authenticate various values: input and output capabilities should be included in the computation of the confirmation values; the nonces produced during the 20 iterations in PKE should be concatenated together with the initiator/responder role in the session, and the resulting concatenation is used in the computation of the confirmation values. The second modification (Dual Passkey Entry) suggests to use two independent `passkeys` that are generated by the both devices and transmitted to the other device via the user as a secure channel. This requires the devices to have keyboard-input capability (i.e., `KeyboardOnly` or `KeyboardDisplay` as *IOcap*), which limits the use of the subprotocol to the 4 cases only in total (out of 9 cases for PKE and 3 cases for NC in the current version of the protocol). It is noteworthy to mention that, according to the Bluetooth Core Specification, if both devices have `KeyboardDisplay` as *IOcap*, they agree on using the NC association model rather than PKE. These modifications still allow a number of the attacks and only mitigate the role confusion attack that the authors discovered.

To mitigate the Method Confusion, KNOB, and Keysize Confusion attacks, Shi et al. [SCH$^+$23] suggested two fixes they claim to be backward compatible. Their patch implies including the parameters, negotiated during the pairing feature extraction, into the computation of the displayed passkey both in NC and PKE during the additional run of the Authentication stage 1. Besides the additional computation, this fix would imply creation of a new cryptographic function: It should take the previously generated passkey as an input, concatenate it with the both initiator and responder devices' parameters and truncate the output to 6 digits. The proposed fix does not help to guarantee the resistance to the KNOB and Keysize Confusion attacks for JW, nor achieve authentication, since the downgrade to the JW protocol remains possible.

There are also possible non-cryptographic ways to address the attacks, e.g., safety number verification as in Signal or OOB communication. These are, however, of limited generality, as devices have different OOB capabilities, and require a technical add-on like an NFC chip, a camera, or a screen, which many Bluetooth devices do not have.

## 4.4 Related Cryptographic Frameworks

Jager et al. [JKSS10] suggested two generic compilers for BR-secure (security against passive adversaries is already sufficient) key exchange (KE) protocol, after which the derived key together with the transcript is handed over to the authentication protocol. The first compiler makes use of signatures and MACs, where the transcript from KE (and a random value) is signed using secret keys of parties and MAC is computed over the signatures using a MAC key derived from the resulting KE key. The second compiler reduces one additional round of exchanged messages by hashing the transcript, the MAC key, and a random value and signing the result with the secret key of the corresponding party. Their approach allows treating some protocols as pure KE protocols without authentication, which is exactly the case with Bluetooth.

Krawczyk [Kra16] has introduced the notion of post-handshake authentication for TLS 1.3. This notion helps to capture the feature of TLS 1.3 with an optional authentication of the client that happens after the client and a server performed a handshake (key exchange in TLS) and the client already authenticated the server. Krawczyk suggested a compiler based on the signature and MAC solution (which can be generalized to other authentication mechanisms). The compiler functions as follows: the parties first derive a session key and a MAC key out of the obtained shared secret key (the option when a session key and a MAC key are concatenated and outputted as a shared secret is also possible but is not considered in the paper). Then the client signs the transcript exchanged during the handshake and sends it to the server together with the MAC-ed identities of the server and the client. The reason to have identities included in a separate MAC is to have a deniability notion.

TLS 1.3 also has a mode for usage of a pre-shared key (PSK), i.e., when two parties have a mechanism to establish a shared secret key, which can be used later for authenticating a TLS connection. This mode either requires the PSK to be shared out-of-band before, or the PSK must be derived from a previous *authenticated* connection, based on certified signatures in TLS 1.3. Link/long-term keys in Bluetooth, however, are usually established during an unauthenticated key exchange protocol, which is vulnerable to machine-in-the-middle attacks. Thus, to use such PSK methods in Bluetooth, the adversary in the initial connection must be limited to being passive, which exactly corresponds to the TOFU-setting of [FS21]. Hence, deploying PSKs, which have been created in the pairing step, in subsequent Bluetooth connections does not help to protect against adversaries, which are active in the initial connection.

Schage et al. [SSL20] have analyzed RFC for IKEv2 [KHN$^+$14] as a privacy-preserving authenticated KE (PPAKE) protocol. That is, apart from achieving basic security guarantees, PPAKE ensures that the communication is deniable against the passive attacker, who can only eavesdrop the conversation. Their definition of the original key corresponds to our definition of TOFU connection. Although IKEv2 follows the similar 2-stage (KE-then-A) structure and allows two parties to derive somewhat long-term keys that might be used for deriving encryption keys later, it does not fully correspond to the way that Bluetooth devices are connecting: IKEv2 requires parties to store the transcript from the KE stage to sign it at the authentication stage. In addition, IKEv2 does not allow parties to create session keys before authentication, what should be allowed for Bluetooth solutions.

The solutions and compilers from [JKSS10] and [Kra16] are not applicable to the Bluetooth setting, since the transcript of the pairing (KE) is not stored on the devices: After the successful initial pairing, only identities, L(T)K, and own DH share are available for reconnections and authentication. In addition, Bluetooth does not follow the regular structure of KE with the authentication happening right after the KE, but rather consists of the KE with the following reconnections.

Pietrzak [Pie20] looked into the notion of the delayed authentication with the application to contact tracing. This notion implies committing a randomized identifier via MAC, which is revealed later on in order to authenticate the exposed party. The notion is somewhat similar to the delayed-key MAC proposed by Fischlin and Lehmann [FL10]. Therein, the key becomes known in the end of the stream, and the MAC is committed on the all the way through. Both solution do not meet our requirements, as the Bluetooth devices do not store the transcripts and in order to make the commit, we would need to modify the pairing protocol, what is not backward compatible.

Another approach is to rely on wireless communication to extract the key from probing and compose it with authentication mechanism as in WiKE-then-PAKE approach from Arriaga, Šala, and Škrobot [ASS23]. However, this approach relies on the adversary being passive during the probing period. Although some methods exist to enforce the security against active adversaries as well, this would make the Bluetooth protocol incompatible with the older devices.

Fischlin and Günther [FG14] proposed a model for multi-stage KE. In the model, participants can establishes new keys at different stages and use these keys across other stages for deriving new keys. While their model allows accounting for the keys' dependencies, the model is redundant for the Bluetooth case, as the only dependency we consider is the encryption keys in reconnections derived from the *L(T)K*s established in the first initial connections. While new reconnections in Bluetooth are consecutive, they are independent from each other and only originate from one "parental" initial session.

## 5 Enhancing the TOFU Security Model

In this section, we give the security model for TOFU-OR-DOFU-authenticated key exchange protocols. Besides the basic properties of key secrecy and match security in the TOFU-model [FS21], we also aim for authentication and weak forward secrecy. Since the authentication is optional and can be performed by any party at any point of time, we consider unilateral authentication, leaving out the execution of the authentication protocol once more with the reversed roles for mutual authentication. Mutual authentication follows from unilateral authentication in both directions, since signing or MAC-computation is done over fresh challenges from both sides, and the party applies the ConnectKey to at least own challenge (derived from the input that corresponds to the sid).

We also capture the presence of a passive adversary, who only eavesdropped on the conversation during the initial connection without intervening, by introducing TOFU-authentication. While it is not possible to achieve forward secrecy for Bluetooth with regard to the link/long-term key or reused DH shares, we still aim for the weaker version: that is, the session keys remain secure as long as the initial connection was not under active attacks, even if the authentication key of the party was corrupted. This corresponds to the cases, when, e.g., the creation process of the vendor was compromised.

For convenience, we follow the style of [FS21] to model different steps separately: we use separate sessions of the initial-connection step for link/long-term key agreement (with empty session keys); a reconnection step for encryption key derivation; and an authentication step. While the reconnection step normally follows the initial connection, the authentication step can be performed at any point of time, therefore, we let adversary decide on when to perform either of them.

### 5.1 TOFU-or-DOFU Security Model

Our TOFU-OR-DOFU model follows the TOFU-model from [FS21] which, in turn, is based on the common game-based security model of [BR94]. The gist of the model is to use the flag isTOFU to indicate if the connection key (i.e., the link key in BR/EDR or the long-term key in BLE) has been established in an execution with an honest partner or not. If the flag is not set, then the model does not consider this key to be a viable target, as it could be trivially known by the adversary anyway. We modify the TOFU-model to add authentication requirements, saying that even if the isTOFU flag is not set, but the authentication step has been carried out by the partner (flag isPartnerAuth is set), then the connection key is an admissible attack target. The modifications are highlighted with a different color.

Transmitted identities of the parties are defined as their Bluetooth addresses `BD_ADDR` and are usually denoted as $A$ and $B$ for the corresponding parties. The parties learn the identity of the intended partner before the KE execution through the corresponding discovery mechanisms. Note that the Bluetooth address does not correspond to the true identity (i.e., MAC-addresses) and can easily change in different sessions. To distinguish the transmitted address `BD_ADDR` from the actual device address, which we authenticate cryptographically, we denote the latter by `MAC_ADDR`. The true identity is globally unique and authenticated only later during the authentication step, and the Bluetooth address itself is not authenticated. We denote by $\mathcal{I}$ the universe of all possible identities `MAC_ADDR`.

**Authentication Keys Linked to Identities.** The Bluetooth protocols themselves are not equipped with public keys linked to the identities. Instead, some weak form of authentication should be provided through comparison or input of digits via PKEand NC, or out-of-band in OOB. Since such methods turn out to be highly vulnerable, and numerous attacks prove that they do not provide the common security guarantees, we instead revert to the classical authentication methods via certified long-term keys.

We assume each party, which intends to authenticate, possesses a long-term authentication key *authsk* and a matching verification key *authpk*. This can, for example, be a pair of signing and verification keys. The authentication keys are used for mutual or unilateral authentication. The certificate should be issued to the device's (static) identity $i = \texttt{MAC\_ADDR}$, either the registered public address in BR/EDR or BLE, or the static random address in BLE. The latter requires some form of bookkeeping of the certification authority to ensure that static random addresses do not repeat. Since we focus on BR/EDR with our solution, one may assume that the identity is the public address.

The verification key comes with a certificate *cert* issued by some trustworthy entity. We denote the key pair of the certification authority as $(\mathit{certsk}, \mathit{certpk}) \leftarrow_{\$} \mathsf{CertKGen}(1^{\lambda})$, generated by algorithm $\mathsf{CertKGen}$. We assume that all parties, including the adversary, have access to the certification authority's public key *certpk*. The certification of a public authentication key *authpk* is denoted by $\mathit{cert} \leftarrow_{\$} \mathsf{Cert}(\mathit{certsk}, \mathit{authpk})$.

While we allow parties to reauthenticate themselves (in case of credential change), we still consider this pair of keys as long-term secrets and allow the corruption of *authsk* via a corresponding $\mathsf{Corrupt}$-oracle. This corruption does not model only inadvertent leakage of the secret key but also successful attacks on the public key. If oracle $\mathsf{Corrupt}$ is called with the input $i \in \mathcal{I}$, then the identity $i$ is added to the (initially empty) set $\mathcal{C}$ of corrupt parties.

**Further Keys.** During the communication, parties derive a connection key $\mathsf{ConnectKey}$ for authentication during future reconnections. Although $\mathsf{ConnectKey}$ can be used for a long period of time, we treat it as an ephemeral key that can be derived anew via a corresponding $\mathsf{InitSession}$-oracle. Parties also possess DH key pairs that they might use only once or in several initial connections. We allow parties to reuse Diffie–Hellman key pairs in several executions (as defined in the Bluetooth standard [Blu23, Vol 2, Part H, Section 5.1]) by modeling initialization of keys[6] $(\mathit{dhsk}_i, \mathit{dhpk}_i) \leftarrow_{\$} \mathsf{DHKGen}(1^{\lambda})$ for each party $i$ at the beginning of the game. The option to change the key pair used by party $i$ is given to the adversary via the $\mathsf{NextDH}$-oracle. This oracle models the key pair update via counter $\mathsf{dhctr}_i$. The value of the counter is initially set to 0 and is incremented with each query to the oracle. Although these keys might be used in several connections, we do not allow the adversary to learn them (e.g., via a $\mathsf{Corrupt}$ query). This can reflect the behavior when a malicious app manages to get an access to the *L(T)K* but not to ephemeral keys like DH. We could capture the corruption of DH keys by tracking all the *L(T)K*s and sessions keys derived from the corrupted DH share and forbidding the adversary from testing these sessions keys, but this sophisticates the model and the proof and does not help to yield forward secrecy of sessions keys.

**Sessions.** We call the $k$-th session run by the party with identity $i \in \mathcal{I}$ as a protocol session and mark it via an administrative label $\mathsf{lbl} = (i, k), \mathsf{lbl} \in \mathcal{L}$, where $\mathcal{I}$ resp. $\mathcal{L}$ is the set of all identities resp. labels. Each session $\mathsf{lbl}$ consists of the following entries with boxed $\boxed{\text{initialized value}}$:

- $\mathsf{id} \in \mathcal{I}$ defines the party's identity.
- $\mathsf{pid} \in \mathcal{I}$ defines the identity of the intended partner. Note that this partner identity may only be authenticated later, when executing the authentication subprotocol.

[6]We renamed these keys *dhsk*, *dhpk* from the terminology $\mathsf{sk}, \mathsf{pk}$ used in [FS21] in order to distinguish them better from the other keys we have introduced here, like *authpk* and *certpk*.

- $\mathsf{mode} \in \{\mathsf{init}, \mathsf{reconnect}, \mathsf{auth}\}$ corresponds to the session in an initial-connection, reconnection, or authentication step.

- $\mathsf{parent} \in \mathcal{L}$ for a reconnection or authentication session points to the initial session from which this session $\mathsf{lbl}$ is spawned off, or to itself if this is a parental session.

- $\mathsf{state} \in \{\mathsf{running}, \mathsf{accepted}, \mathsf{rejected}\}$ describes the state of the session: $\mathsf{running}$ once the session is initialized, and $\mathsf{accepted}$ or $\mathsf{rejected}$ if the party has accepted.

- $\mathsf{aux}$ stands for auxiliary information such as the association model (JW, PKE, NC, or OOB) in use; additional data transmitted out of band, e.g., $\mathsf{passkey} \in \{0, 1, \ldots, 9\}^* \cup \{\perp\}$ in the PKE association model; or in authentication steps, the identity of the party aiming to authenticate (the "prover").

- $\mathsf{dhctr}$ stands for the counter of a Diffie–Hellman key pair that party $i$ uses in the session. While the party always uses the same key pair according to the counter during one protocol session, in different sessions, the counter can be incremented and lead to the different key pairs used by the party.

- $\mathsf{ConnectKey} \in \{0, 1\}^* \cup \{\boxed{\perp}\})$ is the connection key that corresponds to the link key in Bluetooth BR/EDR and long-term key in Bluetooth LE. The connection key can be set during the initial connection. It is used to derive session keys during the reconnection and to provide key authentication during authentication.

- $\mathsf{key} \in \{0, 1\}^* \cup \{\boxed{\perp}\}$ is the session key that is set in reconnections ($\mathsf{lbl.mode} = \mathsf{reconnect}$) after the session accepts ($\mathsf{lbl.state} = \mathsf{accepted}$). For authentication ($\mathsf{lbl.mode} = \mathsf{auth}$) or initial connection ($\mathsf{lbl.mode} = \mathsf{init}$), the key is set to $\perp$ even after the session accepts.

- $\mathsf{sid} \in \{0, 1\}^* \cup \{\boxed{\perp}\}$ is the session identifier. The session identifier can be set only once when executing the protocol.

- $\mathsf{isTested} \in \{\mathsf{true}, \boxed{\mathsf{false}}\}$ is a Boolean flag defining whether the session key has been tested before.

- $\mathsf{isRevealed} \in \{\mathsf{true}, \boxed{\mathsf{false}}\}$ is the Boolean flag defining whether the session key has been revealed.

- $\mathsf{isTOFU} \in \{\mathsf{true}, \boxed{\mathsf{false}}\}$ is a Boolean flag defining whether the session key has been derived in the honest execution during the initial connection.

- $\mathsf{isPartnerAuth} \in \{\mathsf{true}, \boxed{\mathsf{false}}\}$ is a Boolean flag defining if the partner in entry $\mathsf{pid}$ has been authenticated through the authentication subprotocol. Note that this can only be set to $\mathsf{true}$ if the partner is still honest ($\mathsf{pid} \notin \mathcal{C}$).

Note that although Bluetooth connections are asymmetric in nature (i.e., Bluetooth uses initiator resp. responder roles, which transform into the Central resp. Peripheral roles—Bluetooth's analogue for client resp. server roles), we do not include roles into our model. The reason is that the choice of the roles is not fixed (while there are devices that can be only in the Peripheral role, dual-role devices also exist) and is not authenticated throughout the protocol. Cryptographically, it is neither relevant for the security property of the established authenticated key.

We follow the approach to model partnered sessions via the same session identifier.

**Definition 5.1 (Partnered Sessions)** *We say two sessions* $\mathsf{lbl}$ *and* $\mathsf{lbl}'$ *are partnered if* $\mathsf{lbl} \neq \mathsf{lbl}'$ *and* $\mathsf{lbl.sid} = \mathsf{lbl}'\mathsf{.sid} \neq \perp$.

**Adversary's Capabilities.** We let an adversary $\mathcal{A}$ to be active and interact with the protocol through the access to the following oracles:

- $\mathsf{InitSession}(i, j, [\mathsf{aux}])$ creates a new session at party $i$ with intended partner $j$ and automatically incremented number $k$ and returns $\mathsf{lbl}$ to the adversary. The identity of the party is assigned to the corresponding entry $\mathsf{lbl.id} \leftarrow i$, the identity of the intended partner to $\mathsf{lbl.pid} \leftarrow j$, the initial connection is set as the mode $\mathsf{lbl.mode} \leftarrow \mathsf{init}$, and the state of the session changes to $\mathsf{lbl.state} \leftarrow \mathsf{running}$. If optional auxiliary information $[\mathsf{aux}]$ is present, then it is stored in parameter $\mathsf{lbl.aux}$, otherwise the parameter is set to $\perp$. The flags $\mathsf{lbl.isTested}, \mathsf{lbl.isRevealed}, \mathsf{lbl.isPartnerAuth}, \mathsf{lbl.isTOFU}$ are set to their initial values $\mathsf{false}$. The pointer to the parent $\mathsf{lbl.parent}$ is set to $\mathsf{lbl}$. The session counter of a key pair receives the value of the party's current counter $\mathsf{lbl.dhctr} \leftarrow \mathsf{dhctr}_i$.

- $\mathsf{Reconnect}(\mathsf{lbl}, [\mathsf{aux}])$ establishes a new session $\mathsf{lbl}' = (i, k')$ from the parental session $\mathsf{lbl}$ and returns $\mathsf{lbl}'$ if there exists a session with $\mathsf{lbl.ConnectKey} \neq \perp$ and $\mathsf{lbl.parent} = \mathsf{lbl}$, otherwise returns $\perp$. A new session is established via calling $\mathsf{InitSession}(i, \mathsf{lbl.pid}, [\mathsf{aux}])$ and overwriting $\mathsf{lbl}'.\mathsf{mode} \leftarrow \mathsf{reconnect}$ as well as $\mathsf{lbl}'.\mathsf{parent} \leftarrow \mathsf{lbl}$. The new session $\mathsf{lbl}'$ inherits the TOFU parameter $\mathsf{lbl}'.\mathsf{isTOFU} \leftarrow \mathsf{lbl.isTOFU}$, the authentication parameter $\mathsf{lbl}'.\mathsf{isPartnerAuth} \leftarrow \mathsf{lbl.isPartnerAuth}$, and the connection key $\mathsf{lbl}'.\mathsf{ConnectKey} \leftarrow \mathsf{lbl.ConnectKey}$ from the parental session.

- $\mathsf{Authenticate}(\mathsf{lbl}, [\mathsf{aux}])$ establishes a new session $\mathsf{lbl}' = (i, k')$ from parental session $\mathsf{lbl}$ and returns $\mathsf{lbl}'$ if there exists the session with $\mathsf{lbl.ConnectKey} \neq \perp$ and $\mathsf{lbl.parent} = \mathsf{lbl.parent}$, otherwise returns $\perp$. This oracle is used to establish a session for unilateral authentication. The choice is determined by placing the identity ($\{i\}$ or $\{j\}$) of the authenticating party (the "prover") into $\mathsf{aux}$. Establishing a new session is done through calling $\mathsf{InitSession}(i, \mathsf{lbl.pid}, [\mathsf{aux}])$ and overwriting $\mathsf{lbl}'.\mathsf{mode} \leftarrow \mathsf{auth}$ as well as $\mathsf{lbl}'.\mathsf{parent} \leftarrow \mathsf{lbl}$. The new session $\mathsf{lbl}'$ inherits the TOFU parameter $\mathsf{lbl}'.\mathsf{isTOFU} \leftarrow \mathsf{lbl.isTOFU}$ and the connection key $\mathsf{lbl}'.\mathsf{ConnectKey} \leftarrow \mathsf{lbl.ConnectKey}$ that it is about to authenticate from the parental session. We allow parties to reauthenticate themselves, therefore this oracle can be queried multiple times, even for the same $\mathsf{lbl}$ or same identity.

- $\mathsf{Send}(\mathsf{lbl}, m)$ sends a protocol message $m$ to the session $\mathsf{lbl}$ and returns the answer. If the session does not exist or is not established, the oracle returns $\perp$. During the oracle execution, the session may change the state $\mathsf{lbl.state}$ to $\mathsf{accepted}$ or $\mathsf{rejected}$ and set session identifier $\mathsf{lbl.sid}$. If the session accepts ($\mathsf{lbl.state} = \mathsf{accepted}$), then the following happens:

  – If $\mathsf{lbl.mode} = \mathsf{init}$ and there exists a session $\mathsf{lbl}'$ partnered to $\mathsf{lbl}$, then set $\mathsf{lbl.isTOFU} \leftarrow \mathsf{true}$ and $\mathsf{lbl}'.\mathsf{isTOFU} \leftarrow \mathsf{true}$.

  – If $\mathsf{lbl.mode} = \mathsf{auth}$ and the partner is authenticated ($\mathsf{lbl.pid} \in \mathsf{lbl.aux}$) and is currently not corrupt ($\mathsf{lbl.pid} \notin \mathcal{C}$), then the partner authentication flag is set $\mathsf{lbl.isPartnerAuth} \leftarrow \mathsf{true}$. Only in this case, authenticate all other related sessions by setting $\mathsf{lbl}'.\mathsf{isPartnerAuth} \leftarrow \mathsf{true}$ for all sessions $\mathsf{lbl}'$ with $\mathsf{lbl}'.\mathsf{parent} = \mathsf{lbl.parent}$ as authentication spans over the connection key. Note that this also sets the authentication flag for the parental session to $\mathsf{true}$.

  – If there exists a partnered session $\mathsf{lbl}'$ with $\mathsf{lbl}'.\mathsf{isTested} = \mathsf{true}$, then the flag of the session here also receives $\mathsf{lbl.isTested} \leftarrow \mathsf{true}$.

  – If there exists a partnered session $\mathsf{lbl}'$ with $\mathsf{lbl}'.\mathsf{isRevealed} = \mathsf{true}$, then the flag of the session here also receives $\mathsf{lbl.isRevealed} \leftarrow \mathsf{true}$.

- $\mathsf{NextDH}(i)$ updates the Diffie–Hellman key pair used by party $i$ and returns the public key part to the adversary. To update, the oracle increments counter $\mathsf{dhctr}_i$ and generates a new key pair $(dhsk_i[\mathsf{dhctr}_i], dhpk_i[\mathsf{dhctr}_i]) \leftarrow\$ \mathsf{DHKGen}(1^\lambda)$. The new key pair will be used only in future sessions, not in currently running sessions.

- $\mathsf{Reveal}(\mathsf{lbl})$ returns the session key $\mathsf{key}$ of session $\mathsf{lbl}$. If the session does not exist or if $\mathsf{lbl.state} \neq \mathsf{accepted}$, the oracle returns $\perp$. Else sets $\mathsf{lbl.isRevealed} \leftarrow \mathsf{true}$ and for all partnered sessions $\mathsf{lbl}'$ with $\mathsf{lbl}'.\mathsf{sid} = \mathsf{lbl.sid}$ also sets $\mathsf{lbl}'.\mathsf{isRevealed} \leftarrow \mathsf{true}$. Note that if adversary queries this oracle about a session in mode $\mathsf{mode} = \mathsf{init}$ or $\mathsf{mode} = \mathsf{auth}$, the oracle returns $\mathsf{key} = \perp$ and not the connection key $\mathsf{ConnectKey}$.

- $\mathsf{Corrupt}(i)$ returns the secret authentication key *authsk* of party $i \in \mathcal{I}$. If the party does not have an authentication key, the oracle returns $\perp$, otherwise sets $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$. Note that although the connection keys might be used for several connections, we do not allow their corruption, but only of the secret authentication key.

- $\mathsf{Test}(\mathsf{lbl})$ is the oracle for testing the session key $\mathsf{key}$ of session $\mathsf{lbl}$. If the session does not exist, or has not accepted ($\mathsf{lbl.state} \neq \mathsf{accepted}$), or has been already revealed ($\mathsf{lbl.isRevealed} = \mathsf{true}$) or tested ($\mathsf{lbl.isTested} = \mathsf{true}$), or is neither the result of the genuine pairing ($\mathsf{lbl.isTOFU} = \mathsf{false}$) nor authenticated the partner yet ($\mathsf{lbl.isPartnerAuth} = \mathsf{false}$), or $\mathsf{key} = \perp$, then the query returns $\perp$. Otherwise, for the challenge bit $\mathsf{b}$, the query returns either the real key $\mathsf{key}$ if $\mathsf{b} = 1$, or a random string of length $|\mathsf{key}|$ if $\mathsf{b} = 0$. To prevent the adversary from an easy win by querying this oracle to the same or partnered session again, the session $\mathsf{lbl}$ sets the flag $\mathsf{lbl.isTested} \leftarrow \mathsf{true}$ and all partnered sessions $\mathsf{lbl}'$ with $\mathsf{lbl}'.\mathsf{sid} = \mathsf{lbl.sid}$ set the flag $\mathsf{lbl}'.\mathsf{isTested} \leftarrow \mathsf{true}$.

Note that according to our model, the authentication step is not executed over the secured communication channel, protected under an encryption key (as potentially Bluetooth would do), but run "in plain". We can model encrypted authentication steps by having the parties execute a reconnection step first (to establish an encryption key), immediately revealing this encryption key, followed by an authentication step in which the revealed encryption key can be used to incorporate encryption. Since our solutions would still be secure if the authentication steps were not encrypted, we stick to the easier model here.

## 5.2 Match Security and Key Secrecy

We define the two desired properties of authenticated key exchange protocols: match security and (session) key secrecy. The first property stands for the attacks where the adversary manages to confuse honest parties about their intended partners or with whom they share the session key. Key secrecy guarantees that the key is hidden from the adversary. For all security properties, including also the authentication property later, we assume the same initialization procedure, which generates the users' keys and certifies them. This is captured by the following description:

```
$\mathsf{Initialize}(\lambda)$
$b \leftarrow\$ \{0,1\}, \mathcal{C} \leftarrow \emptyset$
$(certsk, certpk) \leftarrow\$ \mathsf{CertKGen}(1^\lambda)$
forall $i \in \mathcal{I}$ do
    $\mathsf{dhctr}_i \leftarrow 0$
    $(dhsk_i[0], dhpk_i[0]) \leftarrow\$ \mathsf{DHKGen}(1^\lambda)$
    $(authsk_i, authpk_i) \leftarrow\$ \mathsf{AuthKGen}(1^\lambda)$
    $cert_i \leftarrow\$ \mathsf{Cert}(certsk, authpk_i)$
```

All values are available in the subsequent games.

**Match Security.** Match security ensures that two partnered sessions hold the same session key (1), and at most two sessions are partnered (2). Since the same ConnectKey can be used in several reconnections, we also need to require the ConnectKey array used for reconnection or authentication to match the one used in the initial connection. This is reflected in the split of the first condition into two cases: for initial connections, where we require matches on the session key and the connection keys, and for reconnections/authentication, where we require session key matching under the condition that the sessions are partnered and start with the same connection key array.

**Definition 5.2 (TOFU-or-DOFU Match Security)** *A key exchange protocol* $\Pi$ *provides TOFU-or-DOFU Match Security if for any PPT adversary* $\mathcal{A}$ *and identity set* $\mathcal{I}$

$$\boldsymbol{Adv}^{\textit{Match Security}}_{\mathcal{A},\Pi,\mathcal{I}}(\lambda) := \Pr\left[\boldsymbol{Exp}^{\textit{Match Security}}_{\mathcal{A},\Pi,\mathcal{I}}(\lambda) = 1\right]$$

*is negligible in the following experiment:*

```
$\boldsymbol{Exp}^{\textit{Match Security}}_{\mathcal{A},\Pi,\mathcal{I}}(\lambda)$
$\mathsf{Initialize}(\lambda)$
$\mathcal{A}^{\mathsf{InitSession},\mathsf{Reconnect},\mathsf{Authenticate},\mathsf{Send},\mathsf{NextDH},\mathsf{Reveal},\mathsf{Corrupt},\mathsf{Test}}(certpk, \{(i, dhpk_i[0], authpk_i, cert_i)\}_{i\in\mathcal{I}})$
$\boldsymbol{return}$ $1$ $\boldsymbol{if}$ $\exists$ $\textit{pairwise distinct}$ $\mathsf{lbl}, \mathsf{lbl}', \mathsf{lbl}'' \in \mathcal{L}$ :
$(1a)$ $\mathsf{lbl.sid} = \mathsf{lbl'.sid} \neq \bot$ $\boldsymbol{and}$ $\mathsf{lbl.mode} = \mathsf{init}$ $\boldsymbol{and}$ $(\mathsf{lbl.key} \neq \mathsf{lbl'.key}$ $\boldsymbol{or}$ $\mathsf{lbl.ConnectKey} \neq \mathsf{lbl'.ConnectKey})$,
$(1b)$ $\mathsf{lbl.sid} = \mathsf{lbl'.sid} \neq \bot$ $\boldsymbol{and}$ $(\mathsf{lbl.mode} = \mathsf{reconnect}$ $\boldsymbol{or}$ $\mathsf{lbl.mode} = \mathsf{auth})$ $\boldsymbol{and}$ $\mathsf{lbl.ConnectKey} = \mathsf{lbl'.ConnectKey}$
        $\boldsymbol{and}$ $\mathsf{lbl.key} \neq \mathsf{lbl'.key}$,
$(2)$ $\mathsf{lbl.sid} = \mathsf{lbl'.sid} = \mathsf{lbl''.sid} \neq \bot$
```

**Key Secrecy.** This property ensures that the session key remains secret if an adversary mounts an attack. Note that we capture the secrecy of the trustworthy or authenticated sessions only, what is ensured by the attack model (i.e., adversary is forbidden from querying Test-oracle to sessions with isTOFU = false and isPartnerAuth = false).

**Definition 5.3 (TOFU-or-DOFU Key Secrecy)** *A key exchange protocol* $\Pi$ *provides TOFU-or-DOFU Key Secrecy if for any PPT adversary* $\mathcal{A}$ *and identity set* $\mathcal{I}$

$$\boldsymbol{Adv}^{\textit{Key Secrecy}}_{\mathcal{A},\Pi,\mathcal{I}}(\lambda) := \Pr\left[\boldsymbol{Exp}^{\textit{Key Secrecy}}_{\mathcal{A},\Pi,\mathcal{I}}(\lambda) = 1\right] - \frac{1}{2}$$

*is negligible in the following experiment:*

```
$\boldsymbol{Exp}^{\textit{Key Secrecy}}_{\mathcal{A},\Pi,\mathcal{I}}(\lambda)$
$\mathsf{Initialize}(\lambda)$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{InitSession},\mathsf{Reconnect},\mathsf{Authenticate},\mathsf{Send},\mathsf{NextDH},\mathsf{Corrupt},\mathsf{Reveal},\mathsf{Test}}(certpk, \{(i, dhpk_i[0], authpk_i, cert_i)\}_{i\in\mathcal{I}})$
$\boldsymbol{return}$ $1$ $\boldsymbol{if}$
    $b' = b$ $\boldsymbol{and}$ $\textit{there are no}$ $\mathsf{lbl}, \mathsf{lbl}' \in \mathcal{L}$ $\textit{with}$ $\mathsf{lbl.sid} = \mathsf{lbl'.sid}$ $\textit{but}$ $\mathsf{lbl.isRevealed} = \mathsf{false}$ $\textit{and}$ $\mathsf{lbl'.isTested} = \mathsf{true}$
```

## 6 Authentication

We next define what we expect from the authentication subprotocol. Note that this part is merely a means to an end, and that a successful execution should provide key secrecy for (past and future) session keys. The goal is to make sure that a party can reliably authenticate the intended partner. However, we actually require more: We want to ensure that the intended partner also confirms to hold the same link key resp. long-term key. This, in turn, adds another requirement to the authentication procedure, namely, that this step needs to authenticate the identity and the connection key but, at the same time, should not infringe with the secrecy of the connection key.

## 6.1 Security Model

We denote by $\mathsf{P}$ and $\mathsf{V}$ the two parties executing the authentication protocol $\Psi$. The prover $\mathsf{P}$ holds an authentication key pair, $(\mathit{authsk}, \mathit{authpk}) \leftarrow\$\ \mathsf{AuthKGen}(1^\lambda)$, and both parties know a value $\mathsf{ConnectKey}$, corresponding to the connection key in our Bluetooth scenario. To link identities reliably to keys $\mathit{authpk}$, we also involve a certification scheme $\Gamma = (\mathsf{CertKGen}, \mathsf{Cert}, \mathsf{CertVf})$. When executing the protocol, we assume that the verifier $\mathsf{V}$ eventually outputs a decision bit, $\mathsf{accepted}$ or $\mathsf{rejected}$, to indicate if the verifier is convinced of the authenticity of the prover and the data $\mathsf{ConnectKey}$.

For the sake of compatibility, we will use the same session information as for the security model for key exchange protocols (albeit we do not need all entries here). We let the adversary interact with both the prover and verifier's instances as in the case of key exchange protocols. That is, we have sessions with labels of the form $\mathsf{lbl} = (i, k)$ or $\mathsf{lbl} = (j, k)$ and each session holds entries $\mathsf{state}$ for the status ($\mathsf{accepted}, \mathsf{rejected}$, or $\mathsf{running}$), $\mathsf{aux}$ for the data to be authenticated (i.e., the connection key), and $\mathsf{sid}$ for the session identifier (to be determined by the protocol). We assume that the party starting the execution always takes the role of the verifier $\mathsf{V}$. Only this party eventually sets the entry $\mathsf{lbl.isPartnerAuth} \leftarrow \mathsf{true}$ upon acceptance (from the initial value $\mathsf{false}$). For mutual authentication, the protocol needs to be run once more with the reverse roles of the prover and verifier.

We note that, when initializing a new session, the adversary has three options for the data to be authenticated (i.e., the connection key): either it sets the value to $v$ by providing auxiliary input $(\mathsf{set}, v)$, or it asks for a hidden random value $(\mathsf{secret}, \ell)$ of some length $\ell$, or it asks to inherit the data from some other session by setting the auxiliary information to $(\mathsf{inherit}, \mathsf{lbl}')$. The latter corresponds to reconnections.

We capture leakage-proofness of the connection key in authentication steps via the $\mathsf{AuthTest}$ oracle. This oracle will either initiate a session by inheriting the random key of another session, or by establishing an independent (but consistent) random connection key, the choice made in dependency of the secret challenge bit $b$. More precisely, the game will keep track of such independent random keys in authentication steps by maintaining a table $T[]$ indexed by connection keys $\mathsf{ConnectKey}$, where $T[\mathsf{ConnectKey}]$ contains the connection key $\mathsf{ConnectKey}$ to be used in the test session (either equal to $\mathsf{lbl.ConnectKey}$ if $b = 0$, or to the independent random value if $b = 1$). Later sessions with the same connection key will also use $T[\mathsf{ConnectKey}]$ to ensure consistency; this is why we index by key values. While this test oracle is irrelevant for the authentication property, in the secrecy game, the task of the adversary is to distinguish the two cases.

Since we aim to authenticate also the connection key in the authentication step, we have to use it in authentication protocol. To make sure that this usage of the connection key does not interfere with the usage to derive session keys in reconnection steps, we grant the adversary access to a "side-channel" oracle $\mathsf{AuthSide}$, which it can query about any pair $(\mathsf{lbl}, x)$ to receive the value for $f(\mathsf{lbl.ConnectKey}, x)$ under key $\mathsf{lbl.ConnectKey}$, if $\mathsf{lbl.ConnectKey} \neq \perp$ has been set already. Here $f$ is some protocol-specific function, e.g., $\mathsf{AES}$ for BLE and $\mathsf{HMAC}/128$ for BR/EDR. The only stipulation is that the adversary cannot see values for inputs $x$, which also appear in the authentication step. More precisely, we assume that the protocol defines a blacklist set $\mathcal{X}^{\text{auth}}_{\text{test}}$ of values $x$, which can only be used in authentication test sessions but never in $\mathsf{AuthSide}$. Once more, this oracle is only necessary for the secrecy game but we include it in all games for the sake of uniformity.

In summary, the adversary can interact with the instances as follows:

- $\mathsf{AuthInitSession}(i, j, \mathsf{aux})$ generates a new label $\mathsf{lbl} = (i, k)$ and returns $\mathsf{lbl}$ to the adversary. It sets $\mathsf{lbl.id} \leftarrow i$, $\mathsf{lbl.pid} \leftarrow j$, $\mathsf{lbl.state} \leftarrow \mathsf{running}$, $\mathsf{lbl.isPartnerAuth} \leftarrow \mathsf{false}$, and $\mathsf{lbl.sid} \leftarrow \perp$. The auxiliary information may be one of the following three types:
  - $\mathsf{aux} = (\mathsf{set}, v)$ sets $\mathsf{lbl.ConnectKey} \leftarrow v$ for value $v$.
  - $\mathsf{aux} = (\mathsf{inherit}, \mathsf{lbl}')$ checks if $\mathsf{lbl}'$ exists and, if not, executes the next item instead (i.e., for $\mathsf{secret}$ of some predetermined length parameter array $\ell$); else $\mathsf{lbl.ConnectKey} \leftarrow \mathsf{lbl}'.\mathsf{ConnectKey}$.
  - $\mathsf{aux} = (\mathsf{secret}, \ell)$ picks $\mathsf{lbl.ConnectKey} \leftarrow_{\$} \{0,1\}^{\ell}$ at random.

  In all cases, we store $\mathsf{aux}$ in $\mathsf{lbl.aux}$. We say that *the session* $\mathsf{lbl}$ *has a secret connection key of length* $\ell$ if $\mathsf{lbl.aux} = (\mathsf{secret}, \ell)$ or if it (recursively) leads to such a session with a secret connection key via a sequence of $\mathsf{aux}$ entries of the form $(\mathsf{inherit}, \mathsf{lbl}')$.

- $\mathsf{AuthSend}(\mathsf{lbl}, m)$ sends a protocol message to the session with label $\mathsf{lbl}$. If the session does not exist, the oracle immediately returns $\perp$. If the state $\mathsf{lbl.state}$ of the session changes to $\mathsf{accepted}$, then it also sets $\mathsf{lbl.sid} \neq \perp$. If, on top, the party is the verifier and sent the first protocol message, then it sets $\mathsf{lbl.isPartnerAuth} \leftarrow \mathsf{true}$ upon acceptance if $\mathsf{lbl.pid} \notin \mathcal{C}$.

- $\mathsf{AuthCorrupt}(i)$ returns the secret authentication key *authsk* of party $i \in \mathcal{I}$ to the adversary. It also adds $i$ to the initially empty set $\mathcal{C}$ of corrupt users, $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$.

- $\mathsf{AuthSide}(\mathsf{lbl}, x)$ checks that $\mathsf{lbl.ConnectKey} \neq \perp$ and $x \notin \mathcal{X}_{\text{test}}^{\text{auth}}$ and, if so, returns $f(\mathsf{lbl.ConnectKey}, x)$.

- $\mathsf{AuthTest}(\mathsf{lbl}')$ first checks if session $\mathsf{lbl}'$ exists and has a secret connection key of some length $\ell$; if not, it returns $\perp$. Else, it checks if the initially empty table $T[]$ has already been set for the connection key of session $\mathsf{lbl}'$, i.e., $T[\mathsf{lbl}'.\mathsf{ConnectKey}] \neq \perp$, and sets $\mathsf{aux} \leftarrow (\mathsf{set}, T[\mathsf{lbl}'.\mathsf{ConnectKey}])$ if so, independently of the game's challenge bit $b$. If not, it sets $\mathsf{aux} \leftarrow (\mathsf{set}, \mathsf{lbl}', \mathsf{ConnectKey})$ and $T[\mathsf{lbl}'.\mathsf{ConnectKey}] \leftarrow \mathsf{lbl}'.\mathsf{ConnectKey}$ for $b = 0$. For $b = 1$ it picks the entry $T[\mathsf{lbl}'.\mathsf{ConnectKey}]$ at random and sets $\mathsf{aux} \leftarrow (\mathsf{set}, T[\mathsf{lbl}'.\mathsf{ConnectKey}]$. It initiates a new session by calling $\mathsf{AuthInitSession}(\mathsf{lbl}'.\mathsf{id}, \mathsf{lbl}'.\mathsf{pid}, \mathsf{aux})$ and returns the response $\mathsf{lbl}$ to the adversary as a newly created session, which either inherits the random connection key or gets a freshly (but consistently) chosen random connection key.

We require *completeness* of the authentication protocol $\Psi$ in the sense that, if an honest verifier communicates with an honest prover, then the verifier session eventually sets $\mathsf{isPartnerAuth} = \mathsf{true}$. All our suggested protocols have this property. In addition, we next define the three security properties—match security, authentication, and leakage resistance:

**Definition 6.1 (Match Security)** *An authentication protocol $\Psi$ with a certification scheme $\Gamma$ provides Match Security, if for any PPT adversary $\mathcal{A}$ and identity set $\mathcal{I}$*

$$\boldsymbol{Adv}^{\textit{Match Security}}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda) := \Pr\left[\boldsymbol{Exp}^{\textit{Match Security}}_{\mathcal{A},\Psi,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda) = 1\right]$$

*is negligible in the following experiment:*

```
$\boldsymbol{Exp}^{\textit{Match Security}}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda)$
$\textsf{\textit{Initialize}}(\lambda)$
$\mathcal{A}^{\mathsf{AuthInitSession},\mathsf{AuthSend},\mathsf{AuthCorrupt},\mathsf{AuthSide},\mathsf{AuthTest}}(certpk, \{(i, authpk_i, cert_i)\}_{i\in\mathcal{I}})$
$\boldsymbol{return}$ 1 $\boldsymbol{if}$ $\exists$ pairwise distinct $\mathsf{lbl}, \mathsf{lbl}', \mathsf{lbl}''$ :
  $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid} = \mathsf{lbl}''.\mathsf{sid} \neq \perp$
```

Next we define authentication. This requires that any session $\mathsf{lbl}$, which claims to have reliably identified a (non-corrupt) partner ($\mathsf{lbl.isPartnerAuth} = \mathsf{true}$ and $\mathsf{lbl.pid} \notin \mathcal{C}$), there must be an honest session, which is actually partnered to $\mathsf{lbl}$ and which also holds the same connection key. The latter implies the authentication of the connection key. Note that we could actually demand there to be an honest partner session with the correct identity $\mathsf{lbl.pid}$, but since honest sessions do not adopt false identities, we can equally ask that there is some honest session. The adversary wins now if there is no such session, i.e., the session $\mathsf{lbl}$ has communicated with the adversary and the adversary managed to impersonate an honest user $\mathsf{lbl.pid}$, or the honest parties partnered but held different connection keys.

**Definition 6.2 (Authentication)** *An authentication protocol $\Psi$ with a certification scheme $\Gamma$ provides Authentication, if for any PPT adversary $\mathcal{A}$ and identity set $\mathcal{I}$*

$$\boldsymbol{Adv}^{Authentication}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda) := \Pr\Big[\boldsymbol{Exp}^{Authentication}_{\mathcal{A},\Pi,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda) = 1\Big]$$

*is negligible in the following experiment:*

```
$\boldsymbol{Exp}^{Authentication}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda)$
$\mathit{Initialize}(\lambda)$
$\mathcal{A}^{\mathsf{AuthInitSession},\mathsf{AuthSend},\mathsf{AuthCorrupt},\mathsf{AuthSide},\mathsf{AuthTest}}(certpk, \{(i, authpk_i, cert_i)\}_{i\in\mathcal{I}})$
$\boldsymbol{return}\ 1\ \boldsymbol{if}$
  $\exists \mathsf{lbl}$ with $\mathsf{lbl.isPartnerAuth}$ and $\mathsf{lbl.pid} \notin \mathcal{C}$, but
    $\forall \mathsf{lbl}' \neq \mathsf{lbl} : (\mathsf{lbl.sid} \neq \mathsf{lbl}'.\mathsf{sid}$ $\boldsymbol{or}$ $\mathsf{lbl.ConnectKey} \neq \mathsf{lbl}'.\mathsf{ConnectKey})$
```

**Definition 6.3 (Leakage Resistance)** *An authentication protocol $\Psi$ with a certification scheme $\Gamma$ is leakage-resistant, if for any PPT adversary $\mathcal{A}$ and identity set $\mathcal{I}$*

$$\boldsymbol{Adv}^{Leakage\ Resistance}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda) := \Pr\left[\boldsymbol{Exp}^{Leakage\ Resistance}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda) = 1\right] - \frac{1}{2}$$

*is negligible in the following experiment:*

```
$\boldsymbol{Exp}^{Leakage\ Resistance}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda)$
$\mathit{Initialize}(\lambda)$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{AuthInitSession},\mathsf{AuthSend},\mathsf{AuthCorrupt},\mathsf{AuthSide},\mathsf{AuthTest}}(certpk, \{(i, authpk_i, cert_i)\}_{i\in\mathcal{I}})$
$\boldsymbol{return}\ 1\ \boldsymbol{if}\ b = b'$
```

## 6.2 Leakage-resistant Authentication Protocols

We describe here four main variants of leakage-resistant authentication protocols, two more suited to BR/EDR and two more suited to BLE (albeit we cannot prove security of the overall key exchange protocol with deferred authentication for BLE). In all schemes we let the authenticating party include the *IO capabilities request* (BR/EDR) resp. *pairing request* data, describing the device's supported features. These data are transmitted before the secure simple pairing and include the device's IO capabilities, the OOB data flag, the authentication request (for BR/EDR), as well as the maximum encryption key size, and the initiator and responder key distribution entries (for BLE). For BR/EDR, the value *IOcap* can be described with 24 bits (as in key derivation) and the value *PairReq* in BLE with 48 bits. We let the authenticating device send this as part of the authentication protocol. Note that the KNOB attack on BLE [ATR19, ATR20b], for instance, uses the fact that the adversary is able to modify the maximum encryption key size when this value is transmitted in the initial pairing. Including this value here as part of *PairReq* in the authentication step implies that this would be detected later during authentication.

Our first protocol uses (certified) signatures to authenticate, where the signed data is computed in the same way as an encryption key from the long-term key, $\mathsf{AES}(\mathit{LTK}, \mathit{challB})$, where *challA* and *challB* are random 128-bit values, which are appended to the AES value to form *authdata*. The prover sends a signature $\sigma_B$ of *authdata* under its signature key *authsk* and appends the certificate for the verification key (which includes the verification key *authpk*). In this first variant, we rely on the key-collision resistance of AES in the sense that finding another key $\mathit{LTK}'$ mapping *challB* to the same value should be infeasible. The exact requirement appears below. The advantage is that the computation of the signed data complies with encryption key derivation in BLE with security function $e$ in [Blu23].

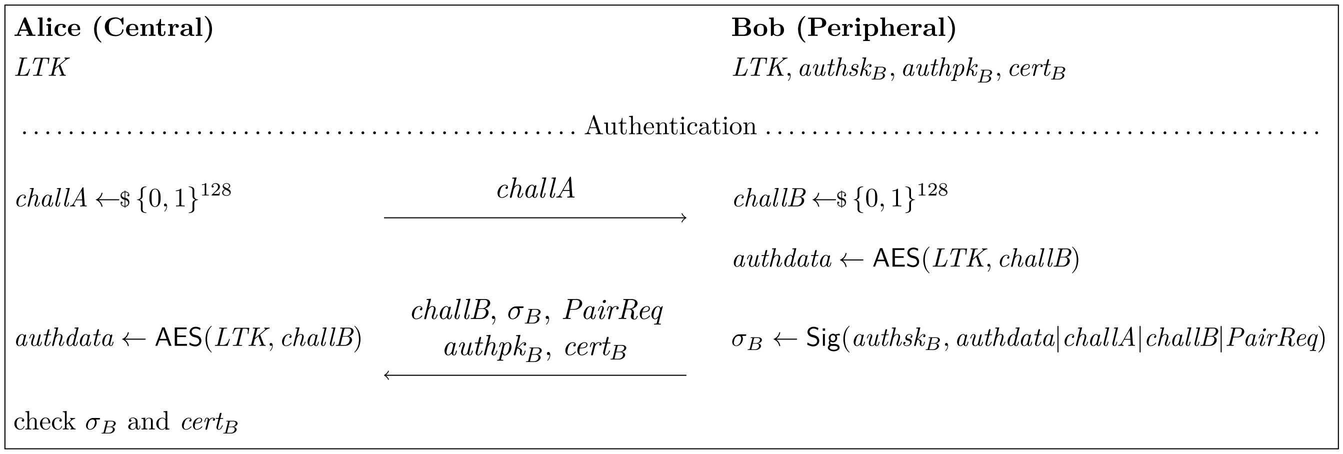
Alice (Central)
Bob (Peripheral)
$\mathit{LTK}$
$\mathit{LTK}, \mathit{authsk}_B, \mathit{authpk}_B, \mathit{cert}_B$
Authentication
$\mathit{challA} \leftarrow\!\$ \{0,1\}^{128}$
$\mathit{challA}$
$\mathit{challB} \leftarrow\!\$ \{0,1\}^{128}$
$\mathit{authdata} \leftarrow \mathsf{AES}(\mathit{LTK}, \mathit{challB})$
$\mathit{challB}, \sigma_B, \mathit{PairReq}$
$\mathit{authpk}_B, \mathit{cert}_B$
$\mathit{authdata} \leftarrow \mathsf{AES}(\mathit{LTK}, \mathit{challB})$
$\sigma_B \leftarrow \mathsf{Sig}(\mathit{authsk}_B, \mathit{authdata}|\mathit{challA}|\mathit{challB}|\mathit{PairReq})$
check $\sigma_B$ and $\mathit{cert}_B$

Figure 4: Signature-based Bluetooth Authentication Subprotocol for BLE. Recall that *PairReq* describes the device's preference transmitted also in the pairing request or response step. The session identifier here and for alternative authentication protocols is given as $\mathsf{sid} = (\mathit{challA}, \mathit{challB})$. The session key KDF in all cases is $\perp$.

We define here for simplicity the set of admissible test values $\mathcal{X}^{\mathrm{auth}}_{\mathrm{test}}$ to be $\{0,1\}^{128}$. Strictly speaking, since $\mathcal{X}^{\mathrm{auth}}_{\mathrm{test}}$ already covers all possible input values for $\mathsf{AES}(\mathit{LTK}, \cdot)$, the adversary could not call AuthSide about non-trivial values anymore. We could be more liberal and define $\mathcal{X}^{\mathrm{auth}}_{\mathrm{test}}$ to be the set of random values *challB* appearing in test queries only and allowing for other queries about $\mathsf{AES}(\mathit{LTK}, x)$, but then we would need to adapt the model slightly to set $\mathcal{X}^{\mathrm{auth}}_{\mathrm{test}}$ randomly. We omit this extension here.

Alternatively, and this constitutes our second variant, we may use a collision-resistant hash function Hash directly and, instead, compute $\mathit{authdata} = \mathsf{HMAC}(\mathit{LK}, \mathit{challA}\|\mathit{challB})$ and sign these data as before, this time with the *IOcap* features. The advantages here are that the security is based on the common collision-resistance of the underlying hash function Hash in HMAC and that this approach can easily support longer challenge values, like the concatenation of two 128-bit challenges. In fact, the HMAC computation resembles the device authentication confirmation function $h5$ in BR/EDR.[7] We set $\mathcal{X}^{\mathrm{auth}}_{\mathrm{test}}$ to be the set of all strings except for 128- and 192-bit values; note that BR/EDR calls $\mathsf{HMAC}(\mathit{LK}, \cdot)$ in reconnection steps only about these two input lengths.

The third variant for BLE avoids signatures and is thus more deniability-friendly. To this end, we assume that the prover holds a certified Diffie–Hellman share $g^y$ and the verifier provides a random Diffie–Hellman share $g^x$. Both parties also exchange 128-bit nonces *challA*, *challB* and compute a confirmation value as $\sigma_B \leftarrow \mathsf{PRF}(\langle g^{xy}\rangle_x, \mathit{challA}, \mathit{challB}, R, I_0, A_0, \mathit{PairReq})$ where $I_0 = 0^{|\mathit{IOcap}|}$ and $A_0 = 0^{|A|}$ are the placeholders for the prover's IO capabilities and the parties identities, *PairReq* are the 48-bit pairing request

[7]Strictly speaking, function $h5$ uses the device key as input, derived from the link key via function $h4$; this extra step may also be done here.

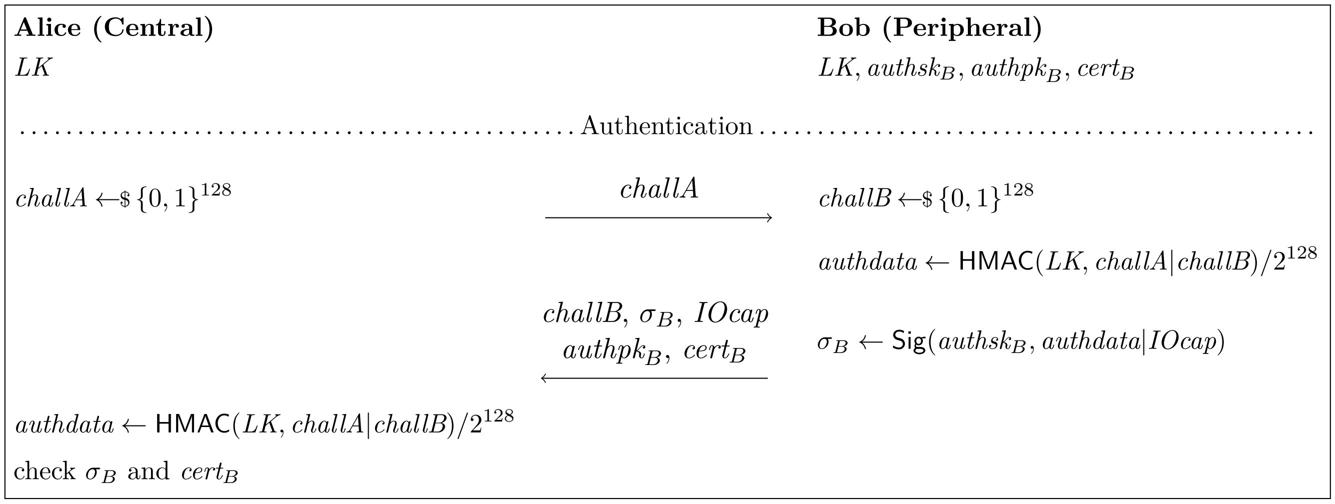
**Alice (Central)**
$LK$
**Bob (Peripheral)**
$LK, authsk_B, authpk_B, cert_B$
Authentication
$challA \leftarrow\$ \{0,1\}^{128}$
$challA$
$challB \leftarrow\$ \{0,1\}^{128}$
$authdata \leftarrow \mathsf{HMAC}(LK, challA|challB)/2^{128}$
$challB,\ \sigma_B,\ IOcap$
$authpk_B,\ cert_B$
$\sigma_B \leftarrow \mathsf{Sig}(authsk_B, authdata|IOcap)$
$authdata \leftarrow \mathsf{HMAC}(LK, challA|challB)/2^{128}$
check $\sigma_B$ and $cert_B$

Figure 5: Signature-based Bluetooth Authentication Subprotocol for BR/EDR. Recall that *IOcap* describes the device's IO capabilities (as used during secure simple pairing). Then notation $\mathsf{HMAC}(LK, challA|challB)/2^{128}$ means to take the leftmost 128 bits of the hash function's output.

data, and $R$ is given by $\mathsf{AES}(LTK, challB)$. The reason for computing $\sigma_B$ this way is that it corresponds to the confirmation value computation in the pairing step (LE Secure Connections check value generation function $f6$ for BLE in [Blu23]), and is based on AES-CMAC for BLE, with extra steps to map the Diffie–Hellman value to a 128-bit key (LE Secure Connections key generation function $f5$ in [Blu23]). Since the values $I_0, A_0$ do not serve any purpose here, beyond this compatibility issue, and may even not be available at this point, we set them to 0. The prover finally sends $\sigma_B$, the certificate, and the challenge value to the verifier. Once more let $\mathcal{X}^{\text{auth}}_{\text{test}} = \{0,1\}^{128}$.

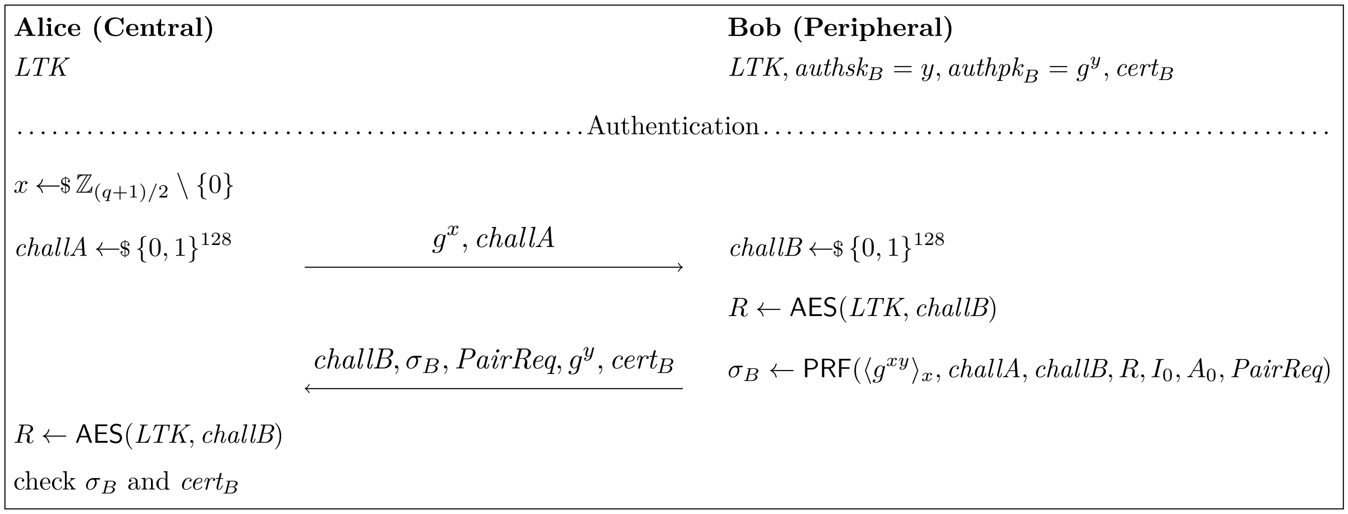
**Alice (Central)**
$LTK$
**Bob (Peripheral)**
$LTK, authsk_B = y, authpk_B = g^y, cert_B$
Authentication
$x \leftarrow\$ \mathbb{Z}_{(q+1)/2} \setminus \{0\}$
$challA \leftarrow\$ \{0,1\}^{128}$
$g^x, challA$
$challB \leftarrow\$ \{0,1\}^{128}$
$R \leftarrow \mathsf{AES}(LTK, challB)$
$challB, \sigma_B, PairReq, g^y, cert_B$
$\sigma_B \leftarrow \mathsf{PRF}(\langle g^{xy}\rangle_x, challA, challB, R, I_0, A_0, PairReq)$
$R \leftarrow \mathsf{AES}(LTK, challB)$
check $\sigma_B$ and $cert_B$

Figure 6: DH-based Bluetooth Authentication Subprotocol for BLE (with PRF being based on AES-CMAC), where $I_0 = 0^{|IOcap|}$ and $A_0 = 0^{|A|}$ are placeholders, and *PairReq* describes the 48 bits of the pairing request data.

The fourth variant maps the third scheme to BR/EDR, once more using Diffie–Hellman. The only difference to the previous version is that HMAC (truncated to 128 bits) is used to compute the value $R$ and that we place the 24-bit *IOcap* describing the device's capabilities into the corresponding entry instead of $I_0$ and now also set the second address entry $B_0 = 0^{|B|}$. Let $\mathcal{X}^{\text{auth}}_{\text{test}} = \{0,1\}^{256}$.

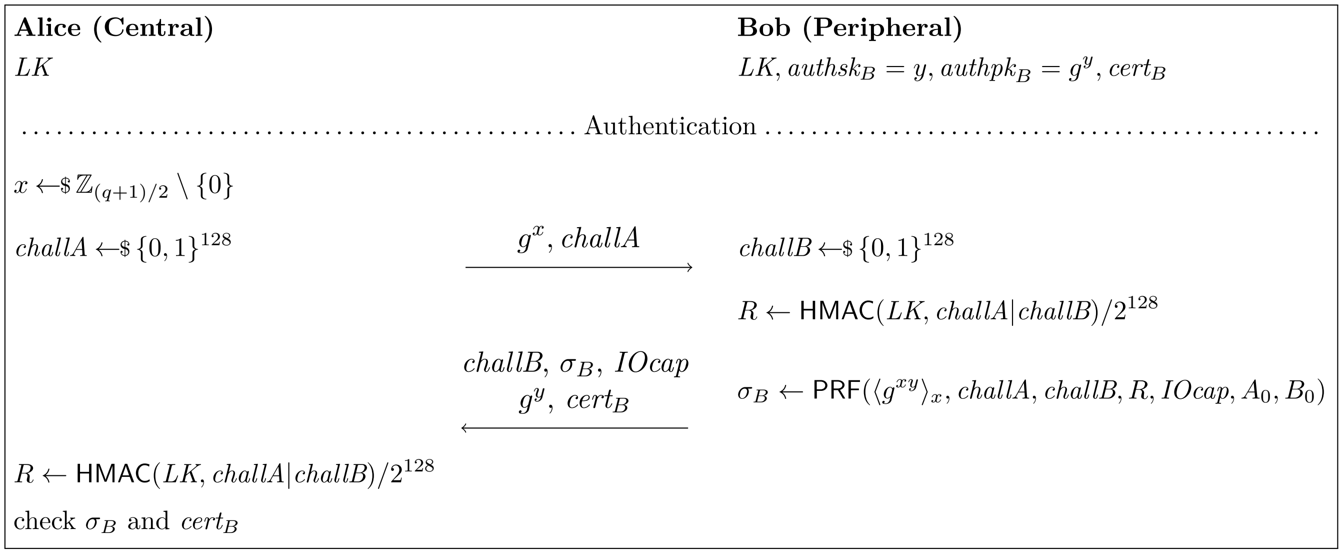
Alice (Central)
Bob (Peripheral)
$LK$
$LK, authsk_B = y, authpk_B = g^y, cert_B$
Authentication
$x \leftarrow\$ \mathbb{Z}_{(q+1)/2} \setminus \{0\}$
$challA \leftarrow\$ \{0,1\}^{128}$
$g^x, challA$
$challB \leftarrow\$ \{0,1\}^{128}$
$R \leftarrow \mathsf{HMAC}(LK, challA|challB)/2^{128}$
$challB, \sigma_B, IOcap$
$g^y, cert_B$
$\sigma_B \leftarrow \mathsf{PRF}(\langle g^{xy} \rangle_x, challA, challB, R, IOcap, A_0, B_0)$
$R \leftarrow \mathsf{HMAC}(LK, challA|challB)/2^{128}$
check $\sigma_B$ and $cert_B$

Figure 7: DH-based Bluetooth Authentication Subprotocol for BR/EDR (with $\mathsf{PRF}$ being being a truncated $\mathsf{HMAC}$), where *IOcap* are the device's IO capabilities, and $A_0 = 0^{|A|}$, $B_0 = 0^{|B|}$ are placeholders.

**Match Security of the Proposed Protocols.** For all four protocols, we let the session identifier $\mathsf{sid} = (challA, challB)$ to consist of the two 128-bit challenges. Match Security follows now easily via the birthday bound:

**Proposition 6.4 (DH and Sig BLE and BR/EDR Match Security)** *All protocols in Figures 4, 5, 6, and 7 provide Match Security. Specifically, for any adversary $\mathcal{A}$ initiating at most $q_s$ sessions, we have*

$$\boldsymbol{Adv}^{Match\ Security}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda) \leq q_s^2 \cdot 2^{-|chall|},$$

*where $|chall| = 128$ is the length of the challenge nonces.*

**Signature-based Bluetooth Authentication Protocol for BLE.** We next show security of the signature-based protocol for BLE. We define the session identifier $\mathsf{sid}$ to be $(challA, challB)$. To prove security, we rely on the unforgeability of the signature scheme $\Sigma$ and of the certification scheme $\Gamma$, as well as on the key-collision resistance of $\mathsf{AES}$ (see Appendix A for all definitions). The latter says that for algorithm $\mathcal{D}$, let $\mathbf{Adv}^{\text{key-coll}}_{\mathcal{D},\mathsf{AES}}(\lambda)$ be the probability that $\mathcal{D}$ outputs $(k, k', x)$, such that $k \neq k'$ but $\mathsf{AES}(k, x) = \mathsf{AES}(k', x)$.

**Proposition 6.5 (Sig BLE Authentication)** *The signature-based Bluetooth Authentication protocol for BLE in Figure 4 provides Authentication. That is, for any $\mathcal{A}$ initiating at most $q_s$ sessions with at most $q_p$ parties, there exist algorithms $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ (with approximately the same running time, and $\mathcal{B}$ making at most $q_s$ signature requests, and $\mathcal{C}$ making at most $q_p$ certification requests), such that*

$$\boldsymbol{Adv}^{Authentication}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},\mathsf{AES}}(\lambda) \leq 2q_s^2 \cdot 2^{-|chall|} + q_p \cdot \boldsymbol{Adv}^{EUF\text{-}CMA}_{\mathcal{B},\Sigma,\mathcal{I}}(\lambda) + \quad \boldsymbol{Adv}^{EUF\text{-}CMA}_{\mathcal{C},\Gamma,\mathcal{I}}(\lambda) + \boldsymbol{Adv}^{key\text{-}coll}_{\mathcal{D},\mathsf{AES}}(\lambda),$$

*where $|chall| = 128$ is the length of the challenges.*

*Proof.* We proceed by a sequence of game hops. The initial game $\mathsf{Game}_0$ is the original attack of $\mathcal{A}$ against authentication. For each game $\mathsf{Game}_i$, we denote by $\Pr[\mathsf{Game}_i]$ the probability that the corresponding game returns 1.

**Game 1.** We next declare $\mathcal{A}$ to lose if there are two honest sessions choosing the same *challA* resp. *challB* in $\mathsf{Game}_0$.

Note that the probability of this happening among the at most $q_s$ sessions is at most $q_s^2 \cdot 2^{-|\mathrm{chall}|}$ for each of the two values, and thus at most $2q_s^2 \cdot 2^{-|\mathrm{chall}|}$ for either part to collide. Hence,

$$\Pr[\mathsf{Game}_0] \le \Pr[\mathsf{Game}_1] + 2q_s^2 \cdot 2^{-|\mathrm{chall}|}.$$

**Game 2.** We next declare the adversary to lose if it manages, in some session, to send a public authentication key *authpk** under some identity $i$, different from the originally certified key of this party, together with a valid certificate.

We note that this immediately constitutes a breach of the unforgeability property of the certification scheme $\Gamma$. Namely, we can construct an algorithm $\mathcal{C}$, which receives the public key *certpk* of the certification authority and runs the setup of the authentication attack (according to $\mathsf{Game}_1$). For each required certificate, it calls its certification oracle. Then it waits for $\mathcal{A}$ to send, in some session, the forged certificate *cert** for *authpk** for identity $i$, and outputs $(\mathit{authpk}^*, i), \mathit{cert}^*$ as its forgery. Since each identity $i$ is issued only one certificate, and this has been for a different public key *authpk*, the output of $\mathcal{C}$ constitutes a valid forgery against $\Gamma$. The bound now follows:

$$\Pr[\mathsf{Game}_1] \le \Pr[\mathsf{Game}_2] + \mathbf{Adv}^{\text{EUF-CMA}}_{\mathcal{C},\Gamma,\mathcal{I}}(\lambda).$$

**Game 3.** We next declare the adversary to lose if, in some session, the honest verifier receives a valid signature $\sigma_B^*$ under the public key *authpk* of some honest party $i \notin \mathcal{C}$ for *authdata** (together with a valid certificate for $\mathit{authpk}, i$), such that this party has not signed *authdata** before.

Analogously to the case of certificate forgeries, this event here would lead to a forgery against the signature scheme $\Sigma$. For this, first note that, according to the previous game hop, the public key *authpk* in question must be authentic. We give a reduction $\mathcal{B}$ against the signature scheme by running $\mathcal{A}$'s attack (in $\mathsf{Game}_2$) and using the externally provided key *authpk* as the key of party $i$. We do so by guessing the right party with probability $1/q_p$ and injecting the provided key there during the setup; all other keys are created by $\mathcal{B}$ itself. Subsequently, if the party $i$ is supposed to sign a value *authdata*, then we call the signing oracle to create the signature. If the adversary $\mathcal{A}$ succeeds in creating a forgery as defined by the game hop, then $\mathcal{B}$ outputs *authdata** together with $\sigma_B^*$ as its forgery. In conclusion,

$$\Pr[\mathsf{Game}_2] \le \Pr[\mathsf{Game}_3] + q_p \cdot \mathbf{Adv}^{\text{EUF-CMA}}_{\mathcal{B},\Sigma,\mathcal{I}}(\lambda).$$

**Game 4.** Finally, declare the adversary $\mathcal{A}$ to lose if, in some session, the honest verifier accepts a signature $\sigma_B$ for $\mathit{authdata}|\mathit{challA}|\mathit{challB}$ under certified key *authpk* of party $i \notin \mathcal{C}$, but such that the verifier holds a different connection key $\mathit{LTK}' \neq \mathit{LTK}$ than party $i$ did when creating the signature $\sigma_B$ in the unique matching session.

We note that this immediately gives a reduction $\mathcal{D}$ against key-collision resistance of $\mathsf{AES}$. If such event happens, then we can immediately output $k = \mathit{LTK}$ and $k' = \mathit{LTK}'$ together with $x = \mathit{challB}$ as the key collision for $\mathsf{AES}$. Thus,

$$\Pr[\mathsf{Game}_3] \le \Pr[\mathsf{Game}_4] + \mathbf{Adv}^{\text{key-coll}}_{\mathcal{D},\mathsf{AES}}(\lambda).$$

With the final game hop to $\mathsf{Game}_4$, we can now argue that $\mathcal{A}$ cannot break authentication in this final game. If this was the case—and we have already ruled out threefold collisions in session identifiers— then it must be that a verifier session $\mathsf{lbl}$ accepts an honest party $\mathsf{lbl.pid} \notin \mathcal{C}$ as legitimate ($\mathsf{lbl.isPartnerAuth} = \mathsf{true}$), but such that there is no matching prover session $\mathsf{lbl}'$ with the same session identifier $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid}$ and same connection key $\mathsf{lbl.ConnectKey} = \mathsf{lbl}'.\mathsf{ConnectKey}$. However, by the previous game hops, there must be a unique prover session, which created the valid signature for $\mathit{authdata}^*|\mathit{challA}|\mathit{challB}$, which the verifier session in question has accepted. It follows that this prover session has the same session identifier $\mathsf{sid} = (\mathit{challA}, \mathit{challB})$ as the verifier. According to the last game hop, it must be that this prover session also holds the same connection key as the verifier. This shows that the adversary $\mathcal{A}$ cannot violate the authentication property in $\mathsf{Game}_4$:

$$\Pr[\mathsf{Game}_4] = 0.$$

This yields the claimed bound.

**Proposition 6.6 (Sig BLE Leakage Resistance)** *The signature-based Bluetooth Authentication protocol for BLE in Figure 4 provides Leakage Resistance. That is, for any $\mathcal{A}$ initiating at most $q_s$ sessions, there exist algorithm $\mathcal{B}$ (with roughly the same running time as $\mathcal{A}$), such that*

$$\boldsymbol{Adv}^{\mathit{Leakage\ Resistance}}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda) \leq \frac{1}{2} + q_s^2 \cdot 2^{-|chall|} + q_s \cdot \boldsymbol{Adv}^{PRF}_{\mathcal{B},\mathsf{AES}}(\lambda),$$

*where $|chall| = 128$ is the length of the challenges.*

*Proof.* For the proof, we once more proceed via game hopping. Let $\mathsf{Game}_0$ be the original leakage-resistance experiment involving $\mathcal{A}$. Let again $\Pr[\mathsf{Game}_i]$ denote the probability that $\mathcal{A}$ correctly predicts the challenge bit $b$.

**Game 1.** In the first game hop, we (consistently) replace $\mathsf{AES}$ computations by a random function evaluation for those sessions, which have a secret connection key of length $\lambda = 128$. Recall that this means the connection key is chosen randomly or is inherited from another session but where this key, too, has been chosen randomly. In either case, the adversary $\mathcal{A}$ is oblivious about the key. By "consistently" we mean that we have a sequence of at most $q_s$ random function instances and, for each newly picked random connection key, assign the next available random function. Instead of computing $\mathsf{AES}(\mathit{LTK}, x)$ for this random key $\mathit{LTK}$, we instead call the assigned random function about $x$ and use the response. This holds for $\mathsf{AES}$ evaluations in protocol executions (via oracle $\mathsf{AuthSend}$) as well as queries to the side-channel oracle $\mathsf{AuthSide}$.

By a hybrid argument, we can reduce the advantage of replacing all $\mathsf{AES}$ instances to the case of a single $\mathsf{AES}$ instance, losing a factor $q_s$. Hence,

$$\Pr[\mathsf{Game}_0] \leq \Pr[\mathsf{Game}_1] + q_s \cdot \mathbf{Adv}^{\mathrm{PRF}}_{\mathcal{B},\mathsf{AES}}(\lambda).$$

**Game 2.** In the next game hop, we let the adversary lose if some sessions of honest provers choose the same value *challB*.

Since we have at most $q_s$ sessions, the birthday bound tells us that the probability of such a collision in the prover's challenge values is at most $q_s^2 \cdot 2^{-|\mathrm{chall}|}$. Therefore,

$$\Pr[\mathsf{Game}_1] \leq \Pr[\mathsf{Game}_2] + q_s^2 \cdot 2^{-|\mathrm{chall}|}.$$

In this final game, we apply the now random functions only to distinct inputs *challB*, independently of the potentially malicious challenge values *challA*. Furthermore, adversary $\mathcal{A}$ can only win if $\mathcal{S} \cap \mathcal{X}^{\mathrm{auth}}_{\mathrm{test}} = \emptyset$ for the set of queries $\mathcal{S}$ to the side-channel oracle, meaning that we can assume that all queries to secret connection keys via $\mathsf{AuthSide}$ have been for different inputs than the challenges *challB* used in test sessions. It follows that each evaluation of the random functions, even for the same connection key, yields independent responses. Put differently, all responses can be thought of as random and independent values. It follows that the two cases in the $\mathsf{AuthTest}$ oracle, inherited and dependent connection keys vs. random and independent connection keys, cannot be distinguished beyond the pure guessing probability of 1/2. In summary,

$$\Pr[\mathsf{Game}_2] = \frac{1}{2}.$$

Summing up the probabilities yields the claim.

**Signature-based Bluetooth Authentication Protocol for BR/EDR.** The proof for the signature-based authentication using $\mathsf{HMAC}$ is almost identical to the one for $\mathsf{AES}$. Only this time, we sign $\mathit{authdata} = \mathsf{HMAC}(LK, \mathit{challA}|\mathit{challB})/2^{128}$ directly, without repeating $\mathit{challA}, \mathit{challB}$ for signing. The reason is that $\mathsf{HMAC}$ is based on a collision-resistant hash function $\mathsf{Hash}$ (in case of Bluetooth 5.4 [Blu23], this is SHA-256) such that finding any colliding input triples $(LK, \mathit{challA}, \mathit{challB}) \neq (LK, \mathit{challA}', \mathit{challB}')$ should be infeasible, even if we truncate the output to the leftmost 128 bits. The session identifier once more consists of $\mathsf{sid} = (\mathit{challA}, \mathit{challB})$.

**Proposition 6.7 (Sig BR/EDR Authentication)** *The signature-based Bluetooth Authentication protocol for BR/EDR in Figure 5 provides Authentication. That is, for any $\mathcal{A}$ initiating at most $q_s$ sessions with at most $q_p$ parties, there exist algorithms $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ (with approximately the same running time, and $\mathcal{B}$ making at most $q_s$ signature requests, and $\mathcal{C}$ making at most $q_p$ certification requests), such that*

$$\boldsymbol{Adv}^{\mathit{Authentication}}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},\mathsf{HMAC}/128}(\lambda) \leq \; 2q_s^2 \cdot 2^{-|\mathit{chall}|} + q_p \cdot \boldsymbol{Adv}^{\mathit{EUF\text{-}CMA}}_{\mathcal{B},\Sigma,\mathcal{I}}(\lambda) + \boldsymbol{Adv}^{\mathit{EUF\text{-}CMA}}_{\mathcal{C},\Gamma,\mathcal{I}}(\lambda) + \boldsymbol{Adv}^{\mathit{coll\text{-}res}}_{\mathcal{D},\mathsf{HMAC}/128}(\lambda),$$

*where $|\mathit{chall}| = 128$ is the length of the challenges.*

Analogously, leakage resistance follows as $\mathsf{HMAC}$, even truncated to 128 bits, is considered to be a pseudorandom function. The proof is therefore easy to adapt from the BLE case with $\mathsf{AES}$.

**Proposition 6.8 (Sig BR/EDR Leakage Resistance)** *The signature-based Bluetooth Authentication protocol for* BR/EDR *in Figure 5 provides Leakage Resistance. That is, for any $\mathcal{A}$ initiating at most $q_s$ sessions there exist algorithm $\mathcal{B}$ (with roughly the same running time as $\mathcal{A}$), such that*

$$\boldsymbol{Adv}^{\mathit{Leakage\ Resistance}}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},\mathsf{HMAC}/128}(\lambda) \leq \frac{1}{2} + q_s^2 \cdot 2^{-|\mathit{chall}|} + q_s \cdot \boldsymbol{Adv}^{\mathit{PRF}}_{\mathcal{B},\mathsf{HMAC}/128}(\lambda),$$

*where $|\mathit{chall}| = 128$ is the length of the challenges.*

**DH-based Bluetooth Authentication Subprotocol.** We treat both Diffie–Hellman cases, for BLE and for BR/EDR, in one go. Authentication follows from the adapted PRF-ODH assumption, which has also been used to prove the security of the Bluetooth protocol in the TOFU setting [FS21] and which will be used in our proof as well. The assumption says that given $g^u, g^v$, it is infeasible to distinguish $\mathsf{PRF}(\langle g^{uv}\rangle_x, x^*)$ for some label $x^*$ from random. This should even hold if one is able to learn related values $\mathsf{PRF}(\langle g^{uw}\rangle_x, x)$ for inputs $(g^w, x) \neq (g^{\pm v}, x^*)$ and values $\mathsf{PRF}(\langle g^{vw}\rangle_x, x)$ for inputs $(g^w, x) \neq (g^{\pm u}, x^*)$. Some care must be taken in the Bluetooth setting since only the $x$-coordinate enters the computations and the secret exponents $u, v$ are chosen from $\mathbb{Z}_{(q+1)/2} \setminus \{0\}$ instead of $\mathbb{Z}_q$ or $\mathbb{Z}_q^*$.

The PRF-ODH assumption is described formally in Appendix A. As discussed in [FS21], it is plausible that this assumption holds for Bluetooth BLE, where $\mathsf{PRF}$ is an $\mathsf{AES\text{-}CMAC}$-based function, and for BR/EDR, where the function is defined by $\mathsf{HMAC}$ for SHA-256. Hence, we assume from now on that $\mathbf{Adv}^{\mathrm{PRF\text{-}ODH}}_{\mathcal{D},\mathsf{PRF},\mathbb{G}}(\lambda)$ is indeed small in both cases. With this, we can show the authentication security:

**Proposition 6.9 (DH BR/EDR and BLE Authentication)** *The Diffie–Hellman-based Bluetooth Authentication protocol in Figure 6 resp. in Figure 7 provides Authentication. That is, for any $\mathcal{A}$ initiating at most $q_s$ sessions with at most $q_p$ parties, there exist algorithms $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ (with approximately the same running time, and $\mathcal{B}$ making at most $q_s$ evaluation requests in the PRF-ODH assumption, and $\mathcal{C}$ making at most $q_p$ certification requests), such that*

$$\boldsymbol{Adv}^{\mathit{Authentication}}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},f}(\lambda) \leq 2q_s^2 \cdot 2^{-|\mathit{chall}|} + 2q_s q_p \cdot \boldsymbol{Adv}^{\mathit{PRF\text{-}ODH}}_{\mathcal{B},\mathsf{PRF},\mathbb{G}}(\lambda) \quad + 2q_s q_p \cdot 2^{-|\mathit{prf}|} + \boldsymbol{Adv}^{\mathit{EUF\text{-}CMA}}_{\mathcal{C},\Gamma,\mathcal{I}}(\lambda) + \delta,$$

*where $|\mathit{chall}| = |\mathit{prf}| = 128$ is the length of the challenges and* $\mathsf{PRF}$ *output. The value $\delta$ equals $\boldsymbol{Adv}^{\mathit{key\text{-}coll}}_{\mathcal{D},\mathsf{AES}}(\lambda)$ for the protocol in Figure 6, and $\boldsymbol{Adv}^{\mathit{CR}}_{\mathcal{D},\mathsf{HMAC}/128}(\lambda)$ for the protocol in Figure 7.*

*Proof.* In the first two game hops, the proof(s) are identical to the previous ones. Only in the hop to $\mathsf{Game}_3$, where we gave a reduction to the unforgeability of the signature scheme, we now give a reduction to the unpredictability of the $\mathsf{PRF}$ value. Unpredictability here says that no adversary can predict the value $\mathsf{PRF}(\langle g^{uv}\rangle_x, x^*)$, instead of distinguishing it from random. We remark that for any predicting adversary $\mathcal{A}$, we have a distinguishing adversary $\mathcal{B}$, such that $\mathcal{A}$'s advantage is bounded by $2 \cdot \mathbf{Adv}^{\text{PRF-ODH}}_{\mathcal{B},\mathsf{PRF},\mathbb{G}}(\lambda) + 2^{-|\text{prf}|}$.
**Game 3.** We next declare the adversary to lose if, in some session, the honest verifier receives a valid value $\sigma^*_B$ under the public key *authpk* of some honest party $i \notin \mathcal{C}$ for ($R^*$, *challA*$^*$, *challB*$^*$) (together with a valid certificate for *authpk*, $i$), such that this party has not computed a $\mathsf{PRF}$ value for ($R^*$, *challA*$^*$, *challB*$^*$) before.

The reduction now causes a contradiction to the unpredictability of the PRF-ODH assumption. That is, we can guess the verifier's session with probability $1/q_s$, and the certified public key of the prover with probability $1/q_p$. Then we insert the PRF-ODH challenge values $g^u$ (for the verifier) and $g^v$ (for the prover) and let the label $x^*$ be the value $(\mathit{challA}^*, \mathit{challB}^*, R^*, \mathit{IOcap}, A, B)$, where $\mathit{IOcap}, A, B$ are the values used in the corresponding session. Note that since the challenge values chosen by honest parties are unique according to $\mathsf{Game}_1$, we can use the $\mathsf{ODH}_u$ and $\mathsf{ODH}_v$ to compute any other value for different labels. Hence, if $\mathcal{A}$ manages to send the correct value $\sigma^*_B$, then this immediately gives a predictor against the PRF-ODH assumption (if the initial guesses have been correct). Therefore,

$$\Pr[\mathsf{Game}_2] \le \Pr[\mathsf{Game}_3] + 2q_sq_p \cdot \mathbf{Adv}^{\text{PRF-ODH}}_{\mathcal{C},\mathsf{PRF},\mathbb{G}}(\lambda) + 2q_sq_p \cdot 2^{-|\text{prf}|}.$$

The final step in the proof is as before, using the (key-)collision resistance of $\mathsf{AES}$ resp. of $\mathsf{HMAC}$, to conclude that there cannot be another session with the same session identifier but a different connection key. This, again, yields the claim.

We next note that leakage resistance follows as before in the signature cases. The reason is that the pseudorandomness of $\mathsf{AES}$ resp. $\mathsf{HMAC}$ ensures this property, independently of the concrete post-processing via signatures or Diffie–Hellman computations. We thus get:

**Proposition 6.10 (DH BR/EDR and BLE Leakage Resistance)** *The Diffie–Hellman-based Bluetooth Authentication protocol in Figure 6 resp. in Figure 7 provides leakage-resistant Authentication. That is, for any $\mathcal{A}$ initiating at most $q_s$ sessions, there exist algorithm $\mathcal{B}$ (with roughly the same running time as $\mathcal{A}$), such that*

$$\boldsymbol{Adv}^{\mathit{Leakage\ Resistance}}_{\mathcal{A},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{auth}_{test},\mathsf{PRF}}(\lambda) \le \frac{1}{2} + q_s^2 \cdot 2^{-|\mathit{chall}|} + q_s \cdot \boldsymbol{Adv}^{PRF}_{\mathcal{B},}(\lambda),$$

*where $|\mathit{chall}| = 128$ is the length of the challenges. Here,* $\mathsf{PRF} = \mathsf{AES}$ *for the protocol in Figure 6, and* $\mathsf{PRF} = \mathsf{HMAC}/128$ *for the protocol in Figure 7.*

## 7 Security of the BR/EDR Protocol with Deferrable Authentication

We next argue that the security of the BR/EDR Bluetooth protocol can be enhanced via deferrable authentication, lifting the guarantees from trust-on-first-use to trust-on-first-use *or* authenticated. This means that any session key, future or past, of an authenticated connection is also secret, even if the adversary has previously been active during the initial pairing.

The proof strategy is intuitive: The difference from the TOFU model is that the adversary may also test sessions where it was active during the initial interaction, as long as there has been a subsequent authentication step. Yet, the security of the authentication protocol ensures that only the intended honest partner could have executed this part and also holds the correct connection key. The latter, however, implies that this party must have indeed been the one that executed the initial connection and computed the connection key. Hence, the authentication step basically confines the adversary once more to a

TOFU attack. Leakage resistance of the authentication protocol ensures that the extra deployment of the connection key in the authentication step does not facilitate the adversary's TOFU-only attack.

## 7.1 Match Security

Before discussing the key secrecy property, we first look at match security. We define the session identifier $\mathsf{sid}$ of our protocol as follows. For the initial connections, $\mathsf{sid}$ consists of the $x$-coordinates of the DH-shares, Bluetooth addresses, and randomly generated nonces: $\mathsf{sid} = (\mathsf{init}, \langle g^a \rangle_x, \langle g^b \rangle_x, A, B, Na, Nb)$. For reconnections in BR/EDR, $\mathsf{sid}$ should contain the randomly generated nonces as well as the Bluetooth addresses of the devices: $\mathsf{sid} = (\mathsf{reconnect}, AU\_RAND\_C, AU\_RAND\_P, A, B)$. Finally, for authentication, we inherit the session identifier $\mathsf{sid}'$ of the authentication protocol but prepend the key word $\mathsf{auth}$, $\mathsf{sid} = (\mathsf{auth}, \mathsf{sid}')$.

**Proposition 7.1 (BR/EDR Match Security)** *The BR/EDR Bluetooth protocol* $\Pi$ *with an authentication protocol* $\Psi$ *and a certification scheme* $\Gamma$ *provides TOFU-or-DOFU-Match Security. That is, for any adversary* $\mathcal{A}$ *initiating at most* $q_s$ *sessions, there exists an adversary* $\mathcal{B}$ *(with approximately the same running time as* $\mathcal{A}$ *and initiating also at most* $q_s$ *sessions), such that*

$$\boldsymbol{Adv}^{\text{Match Security}}_{\mathcal{A},\Pi,\mathcal{I}}(\lambda) \leq q_s^2 \cdot 2^{-|nonce|} + \boldsymbol{Adv}^{\text{Match Security}}_{\mathcal{A},\Psi,\Gamma,\mathcal{I}}(\lambda),$$

*where* $|nonce| = 128$.

*Proof.* We first show that the partnered sessions derive the same session key. For initial connections, all the data in the session identifier $\mathsf{sid} = (\mathsf{init}, \langle g^a \rangle_x, \langle g^b \rangle_x, A, B, Na, Nb)$ completely determine the input to the key derivation function for the connection key, such that identical session identifiers imply identical connection keys. Furthermore, we set the session key of initial connections to be empty, such that parties trivially agree on the same session key.

For reconnection sessions with identical connection keys, the values in the session identifiers in BR/EDR together with the connection key deterministically define the session key. Hence, equality on session identifiers and connection keys also implies identical session keys for reconnections. Finally, authentication sessions, starting and keeping the same connection key, derive an empty session key such that the claim trivially holds for such sessions.

It remains to show that at most two sessions have the same identifier $\mathsf{sid}$. Since all identifiers for types $\mathsf{init}$ and $\mathsf{reconnect}$ contain random nonces (of which at least one is chosen by an honest party), the birthday bound tells us that colliding nonces can happen with probability at most $q_s^2 \cdot 2^{-|\text{nonce}|}$ in the total number $q_s$ of all sessions. For all sessions, the nonce length is 128.

For authentication steps, this follows from the corresponding property of the authentication protocol. It is straightforward to give a reduction $\mathcal{B}$, which perfectly simulates all other steps of the key exchange protocol in $\mathcal{A}$'s attack but instead interacts in the Match Security game of the authentication scheme. Since the key exchange protocol uses the same session identifiers in $\mathsf{auth}$ sessions as the authentication protocol (plus the prepended constant $\mathsf{auth}$), any threefold session-identifier collision on the key exchange protocol yields a corresponding collision for the authentication protocol. Hence, we can bound such cases by the Match Security of the authentication protocol.

## 7.2 Key Secrecy

As explained earlier in this section, the proof strategy is to reduce the adversary to a TOFU-only adversary. For technical reasons, yet, we still need to dive into the original TOFU proof in [FS21]. The underlying cryptographic assumptions (like the PRF-ODH assumption or collision resistance of truncated $\mathsf{HMAC}$) are stated formally in Appendix A.

**Proposition 7.2 (BR/EDR Key Secrecy)** *The BR/EDR Bluetooth protocol* $\Pi$ *with an authentication protocol* $\Psi$ *(with set* $\mathcal{X}_{test}^{auth} = \{0,1\}^* \setminus (\{0,1\}^{128} \cup \{0,1\}^{192})$ *and* $f = \mathsf{HMAC}/128$*) and a certification scheme* $\Gamma$ *provides TOFU-or-DOFU-Key Secrecy: for any adversary* $\mathcal{A}$ *with at most* $q_s$ *sessions, initiating only initialization and authentication sessions, there exist adversaries* $\mathcal{B}$*,* $\mathcal{C}$*,* $\mathcal{D}$*,* $\mathcal{E}$*, and* $\mathcal{F}$ *(with roughly the same running time as* $\mathcal{A}$*, and* $\mathcal{C}$ *making at most* $q_s$ *oracle queries), such that*

$$\boldsymbol{Adv}_{\mathcal{A},\Pi,\mathcal{I}}^{\textit{Key Secrecy}}(\lambda) \leq q_s^2 \cdot 2^{-|nonce|} + \boldsymbol{Adv}_{\mathcal{B},\Psi,\Gamma,\mathcal{I},\mathcal{X}_{test}^{auth},f}^{\textit{Authentication}}(\lambda) + \boldsymbol{Adv}_{\mathcal{C},\mathsf{HMAC}/128}^{\textit{CR}}(\lambda) \quad + q_s^3 \cdot \boldsymbol{Adv}_{\mathcal{D},\mathsf{PRF},\mathbb{G}}^{\textit{PRF-ODH}}(\lambda) +$$
$$2 \cdot \boldsymbol{Adv}_{\mathcal{E},\Psi,\Gamma,\mathcal{I}}^{\textit{Leakage Resistance}}(\lambda) \quad + 2q_s \cdot \boldsymbol{Adv}_{\mathcal{F},\mathsf{HMAC}/128}^{\textit{PRF}}(\lambda) + q_s^2 \cdot 2^{-|ACO|},$$

*where* $|nonce| = 128$ *and* $|ACO| = 64$.

Note that the bound roughly matches the one in [FS21] but adds the (un)forgeability bound for the authentication step, as well as the leakage bound for the link key in the authentication step. Indeed, the factor $q_s^3$ in the PRF-ODH advantage and the factor $q_s^2 \cdot 2^{-|\mathrm{ACO}|}$ mean that the bounds are only reasonable if one considers corresponding bounds on the number of sessions, which can occur in a setting.

Let us highlight the term $\mathbf{Adv}_{\mathcal{C},\mathsf{HMAC}/128}^{\mathrm{CR}}(\lambda)$ in the above bound. This term captures the likelihood of collisions when deriving the link key via truncated HMAC in BR/EDR and is used in the proof to argue that the authentication step (which ranges also over the link key) must be carried out by the same party as in the initial connection establishment (in which exactly this link key has been created). Our results would transfer to BLE almost immediately, if we could argue a similar term for the collision resistance of the long-term key derivation function AES-CMAC in BLE. Unfortunately, this would be hard to show, especially in light of known malleability attacks [CE21].

*Proof.* We prove the claim by performing the game hops, until we finally reach a game where the adversary always receives independent random keys from the Test oracle, no matter what value the challenge bit $b$ has. We let $\mathsf{Game}_0$ be the original attack of $\mathcal{A}$ on key secrecy. To simplify the presentation, we assume the adversary never reveals or tests an empty session key of a session in mode $\mathsf{mode} = \mathsf{init}$ or $\mathsf{mode} = \mathsf{auth}$. We also write $\Pr[\mathsf{Game}_i]$ to denote the probability that the adversary predicts the challenge bit in $\mathsf{Game}_i$ correctly, minus the guessing probability of 1/2.

**Game 1.** In $\mathsf{Game}_1$, we assume that there are no three sessions in mode $\mathsf{mode} = \mathsf{init}$ with the same nonce entries $N_A, N_B$ in the session identifier, and thus in particular also with the same session identifier. This happens with probability at most $q_s^2 \cdot 2^{-|\mathrm{nonce}|}$ for the 128-bit nonces exchanged in the pairing step.

**Game 2.** Declare the adversary to lose if, in a session $\mathsf{lbl}$ of type $\mathsf{auth}$, the verifier accepts the partner $\mathsf{lbl.pid}$ as authentic ($\mathsf{lbl.isPartnerAuth} = \mathsf{true}$), implying that the partner was honest upon completion of the session ($\mathsf{lbl.pid} \notin \mathcal{C}$), but there is no honest session $\mathsf{lbl}'$ with the same session identifier ($\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid}$) and the same connection key ($\mathsf{lbl.ConnectKey} = \mathsf{lbl}'.\mathsf{ConnectKey}$).

This immediately gives a contradiction to the authentication property of protocol $\Psi$. Namely, we can simulate $\mathcal{A}$'s attack by choosing all other data, except for the authentication part, locally. We construct an algorithm $\mathcal{B}$ relaying all authentication steps to the corresponding oracles in the authentication game, setting the connection key in all initializations via $\mathsf{aux} = (\mathsf{set}, v)$ to the known value $v$ from the simulation. Note that key exchange sessions can only run in mode $\mathsf{auth}$ if they have a valid connection key in the first place. Also note that $\mathcal{B}$ does not need the AuthTest oracle when playing against the authentication property.

Since there is a bijection from session identifiers for the authentication step in the key exchange protocol to the ones of the plain authentication protocol (prepending resp. pruning the prefix $\mathsf{auth}$) and $\mathcal{B}$'s simulation is perfect, it follows that any violation according to $\mathsf{Game}_2$ immediately breaks authentication of protocol $\Psi$. Hence,

$$\Pr[\mathsf{Game}_1] \leq \Pr[\mathsf{Game}_2] + \mathbf{Adv}_{\mathcal{B},\Psi,\Gamma,\mathcal{I},\mathcal{X}_{\mathrm{test}}^{\mathrm{auth}},f}^{\mathrm{Authentication}}(\lambda).$$

*Remark:* *We would now like to conclude that, after a successful run of the authentication step, the connection keys have been generated genuinely between two honest parties, akin to the TOFU setting. But the previous game hop only means that there must be an honest party, which has been partnered in the authentication step and also holds the same connection key. We next conclude that these two parties must have also been partnered in the initialization phase. This follows from the fact that, in* BR/EDR*, the connection key is computed via a truncated* HMAC *and thus provides a form of collision resistance. We are not aware of one can draw the same conclusion for* BLE*, where the connection key is computed via the malleable* AES-CMAC *function.*

**Game 3.** Declare the adversary to lose if there exists two distinct accepting sessions ($\mathsf{lbl} \neq \mathsf{lbl}'$), both in the $\mathsf{mode} = \mathsf{init}$, which are not partnered ($\mathsf{lbl.sid} \neq \mathsf{lbl}'.\mathsf{sid}$) but hold the same connection keys, $\mathsf{lbl.ConnectKey} = \mathsf{lbl}'.\mathsf{ConnectKey}$.

Note that the distinct session identifiers $\mathsf{lbl.sid}, \mathsf{lbl}'.\mathsf{sid}$ are given by the $x$-coordinates of the DH values $\langle g^a \rangle_x$, $\langle g^b \rangle_x$, the nonces $N1$, $N2$, and the Bluetooth addresses $A1$, $A2$ from which the BR/EDR connection key/link key is derived as

$$\mathsf{HMAC}(W, N1|N2|\mathrm{kID}_{\mathrm{BR/EDR}}|A1|A2)/2^{128},$$

where $\mathrm{kID}_{\mathrm{BR/EDR}}$ is a constant and $W = \langle g^{ab} \rangle_x$ is the $x$-coordinate of the shared DH key. It follows that if there are two sessions $\mathsf{lbl}, \mathsf{lbl}'$ as above, then we get a collision in the truncated HMAC output and thus a collision in (truncated) SHA-256. Therefore,

$$\Pr[\mathsf{Game}_2] \leq \Pr[\mathsf{Game}_3] + \mathbf{Adv}^{\mathrm{CR}}_{\mathcal{C},\mathsf{HMAC}/128}(\lambda).$$

**Game 4.** In all sessions $\mathsf{lbl}$ of mode $\mathsf{init}$, upon acceptance, replace the connection key $\mathsf{ConnectKey}$ as follows:

1. If there is a partnered session $\mathsf{lbl}'$, which has accepted before—there can be at most one by the previous game hops—set $\mathsf{lbl.ConnectKey} \leftarrow \mathsf{lbl}'.\mathsf{ConnectKey}$.

2. Else, if there exists an (incomplete) session $\mathsf{lbl}'$, which upon acceptance would set the same session identifier $\mathsf{lbl.sid}$, i.e., which has sent and received the same data $g^x, g^y, N_A, N_B, A, B$ from which the connection key is derived, then pick the connection key $\mathsf{lbl.ConnectKey}$ randomly.

3. In any other case, compute the connection key according to the protocol.

Note regarding case 2 that we cannot have a pairing step in which session $\mathsf{lbl}$ accepts and another (incomplete) session $\mathsf{lbl}'$ of an honest party has not yet received the entire data $g^x, g^y, N_A, N_B, A, B$ but becomes partnered later with $\mathsf{lbl}$. The reason is that both parties eventually exchange the confirmation values $E_A$ and $E_B$ such that, upon acceptance of $\mathsf{lbl}$, the session $\mathsf{lbl}'$ must have sent its confirmation value already. The key for computing the confirmation value is based upon the data $g^x, g^y, N_A, N_B, A, B$.

We argue that these replacements are valid according to the PRF-ODH assumption. Recall that the PRF-ODH assumption states that an adversary receives challenge values $U = g^u$ and $V = g^v$, picks a challenge label $x^*$, and cannot distinguish the value $\mathsf{PRF}(\langle g^{uv} \rangle_x, x)$ from random, even when learning related PRF values (for $U$ and for $V$) for other labels via $\mathsf{ODH}$ oracles. See Appendix A for a formal definition.

As in the proof for the TOFU property [FS21], we now do a hybrid argument, gradually replacing all the (at most $q_s$) actual connection keys in case 2 by random values. Formally, for $i$ picked randomly between 1 and $q_s$, we replace the first $i-1$ of such keys by random values, and the $i$-th value by the challenge in the PRF-ODH game. To this end, we first guess the right DH-key shares among the at most $q_s^2$ many possibilities, and inject the given challenge values $g^u, g^v$ when the next DH key is triggered and reaches our guesses. If our guess has been wrong, then we output a random prediction for the PRF-ODH challenge bit. In any other case, we use the prediction of the key exchange adversary. All other connection keys, including the ones computed according to case 3, are derived by calling the $\mathsf{ODH}$ oracle. Note that, since nonces are unique per session according to $\mathsf{Game}_1$, any query to the $\mathsf{ODH}$ oracles are for a different label than the one in the challenge value.

The analysis in [FS21] reveals that we lose a factor $q_s^3$ in the PRF-ODH advantage when progressing from $\mathsf{Game}_3$ to $\mathsf{Game}_4$, due to the at most $q_s$ hybrids and the at most $q_s^2$ possibilities to inject the challenge values $g^u, g^v$. It follows that

$$\Pr[\mathsf{Game}_3] \leq \Pr[\mathsf{Game}_4] + q_s^3 \cdot \mathbf{Adv}^{\text{PRF-ODH}}_{\mathcal{D},\mathsf{PRF},\mathbb{G}}(\lambda).$$

*Remark: Note that this means that we have substituted all connection keys, which are created between two honest parties in an undisturbed pairing step by random values, e.g., if* $\mathsf{isTOFU} = \mathsf{true}$. *The previous games,* $\mathsf{Game}_2$ *and* $\mathsf{Game}_3$*, show that any other "testable" session (in which* $\mathsf{isPartnerAuth} = \mathsf{true}$*) must have such an undisturbed pairing step, too. It follows that we are already using random keys instead in all sessions, which could potentially be tested.*

Mark the sessions in which we replace $\mathsf{ConnectKey}$ by a random value according to case 2, or in which we set $\mathsf{ConnectKey}$ according to case 1 for an already substituted key $\mathsf{ConnectKey}$, as having *secret* connection keys. Observe that these are sessions, which have an initial (and unique) honest partner session by construction. In particular, we can identify sessions, which have the same (secret) connection key via such initial pairings.

The next step is to argue that the authentication step cannot facilitate the adversary's task in computing a session key (in a reconnection step), due to the leakage resistance of the authentication step. To this end, the game also keeps an initially empty table $T[]$ to store values, which we use in the authentication step.

**Game 5.** Next, in each initialized session $\mathsf{lbl}$ of mode $\mathsf{auth}$ with a secret connection key, use (yet another) random value $T[\mathsf{ConnectKey}] \leftarrow\$ \{0,1\}^{|\mathsf{ConnectKey}|}$ instead of $\mathsf{ConnectKey}$ (unless a value $T[\mathsf{ConnectKey}]$ has already been chosen for this connection key, in which case we reuse the value $T[\mathsf{ConnectKey}]$).

Note that the adaption corresponds to the behavior of the $\mathsf{AuthTest}$ oracle in the leakage resistance game. It is therefore straightforward to give a reduction $\mathcal{E}$ to leakage resistance. To this end, $\mathcal{E}$ upon asked by the key exchange adversary to create a new authentication session for some session $\mathsf{lbl}$ with a secret connection key, calls the $\mathsf{AuthTest}$ oracle to create a new authentication session, either with the actual connection key or with a fresh (but consistent) one, and then relays the communication with the adversary and this session. For genuine connection keys picked by $\mathsf{AuthTest}$, this corresponds to $\mathsf{Game}_4$, and for fresh keys this corresponds to $\mathsf{Game}_5$. Note that $\mathcal{E}$ can simulate additional reconnection steps towards $\mathcal{A}$ for such unknown connection keys, and potentially subsequent $\mathsf{Reveal}$ queries about such session keys, with the help of its side-channel oracle $\mathsf{AuthSide}$: Simulate the reconnection protocol according to the game and query $\mathsf{AuthSide}$ about the right input $(\mathsf{lbl}, x)$ to get the device key and the actual session key. By construction, we then query $\mathsf{HMAC}$ only about inpout values $x$ of 128 bits (for the device key) and 192 bits (for the session key). Since we excluded such inputs from the set $\mathcal{X}^{\text{auth}}_{\text{test}}$ of admissible test queries, reduction $\mathcal{E}$ can safely use $\mathsf{AuthSide}$ to get these values.

Taking into account the factor 2 for moving from prediction to indistinguishability in the authentication game, we get

$$\Pr[\mathsf{Game}_4] \leq \Pr[\mathsf{Game}_5] + 2 \cdot \mathbf{Adv}^{\text{Leakage Resistance}}_{\mathcal{E},\Psi,\Gamma,\mathcal{I},\mathcal{X}^{\text{auth}}_{\text{test}},f}(\lambda).$$

*Remark: Note that this means that authentication steps in sessions with secret connection keys actually do not use the connection key anymore. In particular, the only time connection keys are used to derive values is in reconnection steps.*

**Game 6.** In any session $\mathsf{lbl}$ of mode $\mathsf{reconnect}$ with secret connection key, (consistently) replace the computation $\mathsf{HMAC}(LK, \dots)/2^{128}$ by random values. Here, consistently means that fresh inputs yield fresh random outputs, but repeating inputs yield the previous answers again.

This easily follows from the pseudorandomness of $\mathsf{HMAC}$, i.e., we can straightforwardly simulate the game for the adversary and give a reduction to $q_s$ instances of the pseudorandom function $\mathsf{HMAC}$. In the simulation, we identify the right instance via the session identifiers in the pairing step. Thus,

$$\Pr[\mathsf{Game}_5] \leq \Pr[\mathsf{Game}_6] + q_s \cdot \mathbf{Adv}^{\mathrm{PRF}}_{\mathcal{F},\mathsf{HMAC}}(\lambda).$$

In particular, we now have replaced the device keys $\mathrm{dk} \leftarrow \mathsf{HMAC}(LK, \mathrm{kID}_{\mathrm{Dev}}|\mathtt{BD\_ADDR}_A|\mathtt{BD\_ADDR}_B)/2^{128}$ in such sessions by random values.

**Game 7.** In any session $\mathsf{lbl}$ of mode $\mathsf{reconnect}$ with secret connection key, (consistently) replace the values $SRES\_C|SRES\_P|\mathrm{ACO} \leftarrow \mathsf{HMAC}(\mathrm{dk}, AU\_RAND\_C|AU\_RAND\_P)/2^{128}$ by random values. Once more, since dk is already random, we can conclude from the pseudorandomness of $\mathsf{HMAC}$ that

$$\Pr[\mathsf{Game}_6] \leq \Pr[\mathsf{Game}_7] + q_s \cdot \mathbf{Adv}^{\mathrm{PRF}}_{\mathcal{F},\mathsf{HMAC}}(\lambda).$$

Here we use again the fact that we can ensure consistency via the session identifiers in the pairing step. (For sake of simplicity, we use the same adversary $\mathcal{F}$ for the bound here as in the previous game hop and account for this in the theorem's bound via a factor 2.)

**Game 8.** Declare the adversary to lose if there are two accepting sessions $\mathsf{lbl}$ and $\mathsf{lbl}'$, both in mode $\mathsf{reconnect}$, for the same secret connection key, which are not partnered but such that ACO and $\mathtt{BD\_ADDR}_A$ $\mathtt{BD\_ADDR}_B$ are identical in both sessions.

Fix two sessions $\mathsf{lbl}$ and $\mathsf{lbl}'$ as above. Since the sessions are not partnered but have the same Bluetooth addresses, and the session identifier is given by $\mathsf{sid} = (\mathsf{reconnect}, AU\_RAND\_C, AU\_RAND\_P, \mathtt{BD\_ADDR}_A, \mathtt{BD\_ADDR}_B)$, we must have that the parties' nonces $(AU\_RAND\_C, AU\_RAND\_P)$ in session $\mathsf{lbl}$ and $(AU\_RAND\_C', AU\_RAND\_P')$ in session $\mathsf{lbl}'$ must be distinct. It follows that the values $SRES\_C|SRES\_P|\mathrm{ACO}$ and $SRES\_C'|SRES\_P'|\mathrm{ACO}'$ in the two sessions are picked randomly and independently according to $\mathsf{Game}_7$. Hence, we have a collision on the ACO-part with probability at most $2^{-64}$. Since we have at most $q_s^2$ of such session pairs $\mathsf{lbl}, \mathsf{lbl}'$, we get

$$\Pr[\mathsf{Game}_7] \leq \Pr[\mathsf{Game}_8] + q_s^2 \cdot 2^{-64}.$$

We can now conclude that all session keys of $\mathsf{reconnect}$ sessions with secret connection keys are generated independently and randomly if they are not partnered. This holds as they either have identical connection keys, since they are partnered in the corresponding $\mathsf{init}$ session, but unless they are partnered in the reconnection sessions and use different input values to compute the session key $\mathsf{k}_{\mathrm{AES}} \leftarrow \mathsf{HMAC}(LK, \mathrm{kID}_{\mathrm{AES}}|\mathtt{BD\_ADDR}_A|\mathtt{BD\_ADDR}_B|\mathrm{ACO})/2^{128}$ according to $\mathsf{Game}_8$. Or, they are not partnered via an $\mathsf{init}$ session and thus have independent connection keys and all subsequent values are chosen independently. Note that the $\mathsf{Test}$ oracle forbids to test a $\mathsf{reconnect}$ session and reveal its partner, such that any admissible $\mathsf{Test}$ for a $\mathsf{reconnect}$ session yields an independent and random answer in both cases $b = 0$ and $b = 1$. Now the adversary has a view on two completely random session keys and can guess only with the probability of 1/2.

## 8 Known Attacks on Bluetooth and our Mitigation

In this section, we briefly describe in how far our solution prevents known attacks. We start with an example of the Ghost Keystroke attack on Bluetooth from [ZWD+20] and show how it is mitigated by our solution. We continue with a short summary of other known attacks on Bluetooth and explain the efficacy of our method against these attacks. More detailed explanation of the attacks and their fixes can be found in Section 4.1.

## 8.1 Example of the Ghost Keystroke Attack

The source of the Ghost Keystroke Attack is the impersonation attack on PKE discovered by Zhang et al. [ZWD+20] (see Figure 8 for a pictorial description). In this attack, a user tries to connect two devices, one of which (Central $A$) has a screen to display the digital passkey, and the second one (Peripheral $B$) has an input capability for the user to enter the digits. If an MitM-attacker $M$ had prior access to the input device $B$ and established a connection (by sending its own DH public share $g^m$ and computing the compromised $LK_{comp}$), it can receive the keystroke entered at this compromised device. When the user initiates a connection between devices $A$ and $B$, the attacker connects to the output device $A$, which displays the digits. The user enters those digits on the compromised input device $B$, which sends the keystroke with the passkey to the attacker. Eventually, the attacker successfully impersonates the input device $B$ and connects its fake input device to the user's honest output device $A$ without the user noticing.

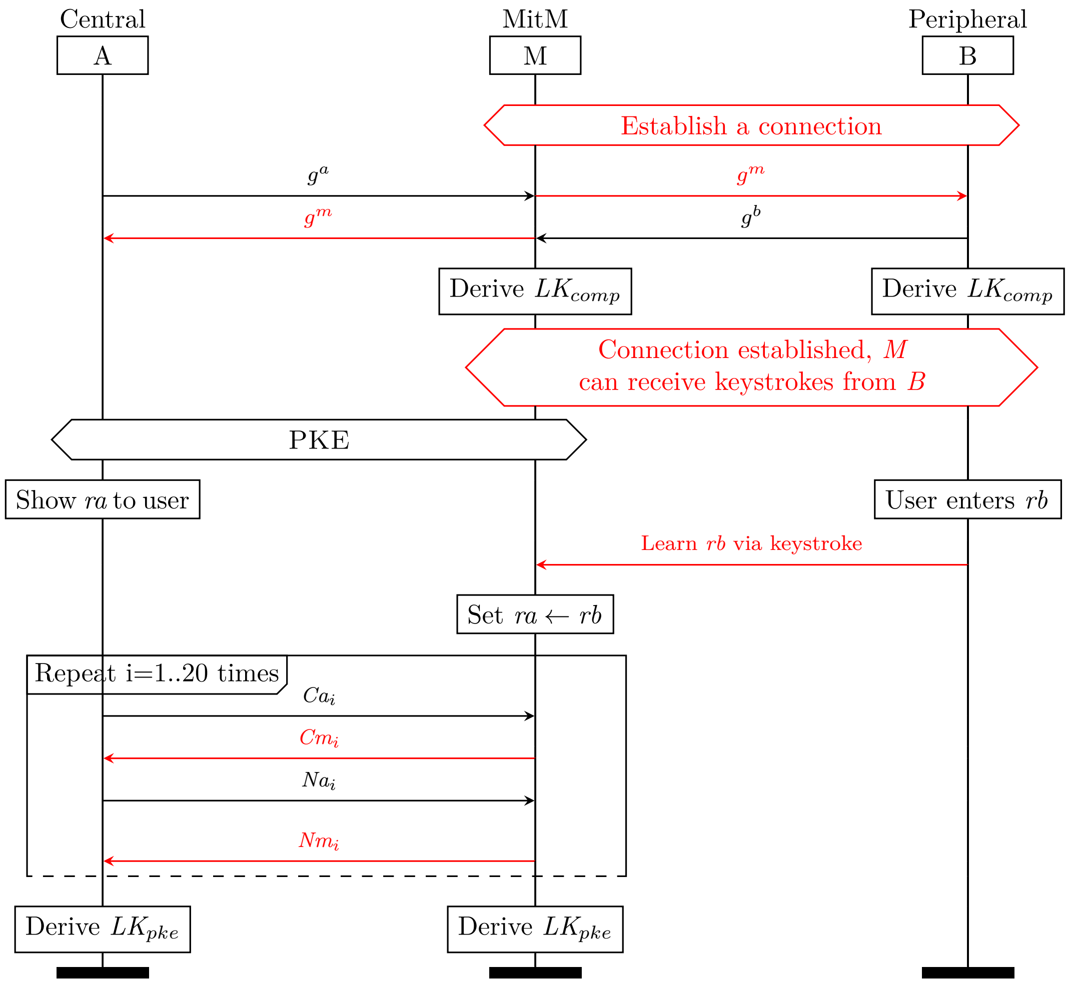

Figure 8: Message Flow for Ghost Keystroke Attack [ZWD+20, JZL23] on Secure Hash Modification [TH21]. The MitM adversary $M$ manages to impersonate $B$ by establishing a connection with it to receive the leakage of the passkey. Each connection is independent, and the honest devices derive different long-term keys in each: $LK_{pke}$ by Central $A$ resp. initially compromised $LK_{comp}$ by Peripheral $B$.

The impersonation attack on PKE from [ZWD$^+$20] was generalized by Jangid et al. [JZL23] as the Ghost Keystroke Attack. The attack assumes that one device is leaking the passkey: e.g., if the adversary had a physical access to the device and established the connection during the user's absence, then the attacker can receive the keystrokes via this connection whenever the user enters anything on this device. The Ghost Keystroke Attack is valid for PKE, and even the fix for PKE suggested in [SCH$^+$23] and the Secure Hash Modification fix in [TH21] do not protect against it (the principle of the attack is the same as shown in Figure 8). When the adversary not only can receive the keystrokes but also inject malicious strings on the compromised device's screen, the attack is even extendable to Dual Passkey Entry in [TH21], unmodified NC, and the fix for NC in [SCH$^+$23].

The adversary can launch the Ghost Attack on our scheme only if it has the power to forge in the authentication process, assuming it has been carried out at some point. Since this step also authenticates the link key $LK$, and the honest device derives different from the compromised device's $LK$ in the corresponding sessions, the adversary will not be able to successfully authenticate, unless it breaks unforgeability of the authentication procedure.

## 8.2 Efficacy of Authentication Against Attacks

**Prevented Attacks.** The attacks we prevent are commonly addressing the capabilities exchange stage, where the adversary can get in between the devices and replace the genuine capabilities with the needed for itself without any further authentication. For example, the adversary can replace the IO capabilities in the exchanged messages and trick devices into the DH-like JW association model, as shown in several works, e.g., [HH07, HT10]. If devices have screens, the adversary can launch even more sophisticated attacks, such as a method confusion attack [vTPFG21] and its extension, pairing confusion attack [CADE23]. With these attacks, the devices are tricked into using different association models, which look the same from the user perspective, and in case of the pairing confusion—even into different protocols that look alike. Changing initiator/responder packets [TH21] allows the adversary to trick the parties into the believe that each is the initiator/responder. While the attack itself does not lead to the key leakage, cryptographically such attacks should not be allowed in any good protocol.

In each of the above attacks, the devices will derive pairwise-different connection keys. Hence, we can expect the authentication step to bring forward this discrepancy, as the keys $LK$s and hence the genuine signatures will not match—unless the adversary breaks authenticity, e.g., forges a signature.

We note that to prevent capability mismatches, both fixes in [TH21] and [SCH$^+$23] suggest modifications to the concrete standalone protocols. They do not take into account the interplay of all the different association models. That is, the adversary can still downgrade the communication to the JW association model by simply modifying the *IOcap* of devices, which, in turn, will not even enter the modified protocol and simply stay at the less secure association model.

**Potentially Prevented Attacks.** The prevention of some attacks is not possible because of some vulnerable parts of the Bluetooth protocol. For instance, the protocols in BLE employ the weak function $\mathsf{AES\text{-}CMAC}$ that is not collision resistant. Bluetooth has a cross-platform mechanism to derive a key on one transport channel using the key derived from another transport. The cross-key derivation mechanism, however, uses AES-CMAC for these procedures: it would not be possible to carry out our proof for this function, due to similar weaknesses as in the standalone BLE setting. The KNOB attack [ATR20b] and the Keysize Confusion attack [SCH$^+$23] in BLE could have been mitigated in principle, as the negotiation is done during the capability exchange stage. Yet, due to the usage of the non-collision-resistant function $\mathsf{AES\text{-}CMAC}$, we cannot show security against these attacks in BLE.

**Attacks Escaping our Solution.** In BR/EDR, attacks on the negotiation mechanism [ATR19, ATR20b] cannot be prevented by our method, as the negotiation mechanism has its own protocol, independent from the Bluetooth pairing step. In addition, all Legacy protocols do not withstand against the passive adversary (as shown, e.g., in [Rya13, JW01]) and hence are not possible to be secured. Initiator resp. responder roles are tied to the Central resp. Peripheral roles in Bluetooth; by exploiting the challenge-response procedure in Legacy reconnections in BR/EDR [ATR20a], the adversary can make only the Central to have asked the Peripheral to respond to the challenge itself. If the Central device is a target, the attacker asks for a role switch and exploits the one-wayness again. For Secure Connections, the attacker can downgrade to Legacy reconnections. The attacks themselves do not leak or help the adversary to learn the link key $LK$, nor the encryption keys. The current version of the Core Specification forbids the downgrade to Legacy reconnections for Secure Connections pairing, disallows role switching during the reconnection in progress, and enforces the mutual authentication in Legacy reconnections.

**Other attacks.** The reflection attack [CE21] allowed the adversary to use a connection with one party to learn the secret passkey by simply reflecting the values sent by this party. This passkey was then used in a connection with another party to impersonate the first. This attack was mitigated in the Bluetooth Core Specification v.5.3, which mandates to check the equality of the received values to their own sent values.

A similar idea to KNOB and BIAS is used in BLUFFS attacks [Ant23]. The attack exploits the Legacy encryption key derivation, which bases the randomness of the key on only one nonce from a Central. Hence, the adversary can force the Peripheral to reuse the same nonce in multiple sessions. This is only fixable by forbidding the usage of the same nonce.

An implementation drawback with the wrong handling of the errors [WNK$^+$20] allowed accessing the attributes, and spoofing the responses [WNK$^+$20] could lead to the "impersonation" of one device towards another. The latter attack may only be fixed by enforcing the encryption by default, as the attack does not happen during the key establishment or authentication stages but rather after the encryption should have been enabled.

Two relay attacks (two-sided and one-sided) [LÇA$^+$04] targeted the Legacy reconnection in BR/EDR. The first attack simply relays the messages in the Legacy mutual reconnection step between two parties, the second allows relaying the challenge from one party to another by terminating one of the connections. These attacks do not lead to the adversary learning the key, hence we leave the attacks out of our scope.

## 9 Conclusion

In this work, we proposed a scheme based on certificates to add authentication to the Bluetooth Secure Connections protocol and to prevent many known attacks on the protocol. We showed the cryptographic strength of the solutions in an extension of common game-based security models. It is noteworthy that our solution only yields a provably secure version for BR/EDR, wherein HMAC is used to derive link keys. In contrast, our proof technique falls short of working in the BLE setting, where the long-term key is computed via AES-CMAC. The reason lies in the lack of collision resistance of the "hashing function AES-CMAC" [Blu23]. It would be easy to lift our proof if, say, also HMAC would be used in BLE. Indeed, we are positive that one could then also extend our result to the cross-transport key derivation (CTKD) mechanism for dual-mode devices, wherewith a BR/EDR link key can be converted to a BLE long-term key and vice versa. This would allow us to also rule out corresponding CTKD attacks. Given the security issues with AES-CMAC, also the malleability problems in Bluetooth Mesh with this function [CE21], it may be worth to consider switching.

## Acknowledgments

We thank the anonymous reviewers and the shepherd for the valuable feedback that helped to improve the paper. This research work has been funded by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE. It has also been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) — 251805230/GRK 2050 and 23661529/SFB 1119.

## References

[Ant23] Daniele Antonioli. BLUFFS: Bluetooth forward and future secrecy attacks and defenses. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023: 30th Conference on Computer and Communications Security*, pages 636–650, Copenhagen, Denmark, November 26–30, 2023. ACM Press. (Cited on pages 3, 4, 9, 14, and 41.)

[ASS23] Afonso Arriaga, Petra Sala, and Marjan Skrobot. Wireless-channel key exchange. In Mike Rosulek, editor, *Topics in Cryptology – CT-RSA 2023*, volume 13871 of *Lecture Notes in Computer Science*, pages 672–699, San Francisco, CA, USA, April 24–27, 2023. Springer, Heidelberg, Germany. (Cited on page 17.)

[ATR19] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Bonne Rasmussen. The KNOB is broken: Exploiting low entropy in the encryption key negotiation of bluetooth BR/EDR. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019: 28th USENIX Security Symposium*, pages 1047–1061, Santa Clara, CA, USA, August 14–16, 2019. USENIX Association. (Cited on pages 3, 4, 9, 11, 26, and 41.)

[ATR20a] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. BIAS: Bluetooth impersonation AttackS. In *2020 IEEE Symposium on Security and Privacy*, pages 549–562, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press. (Cited on pages 3, 4, 9, 13, and 41.)

[ATR20b] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Key negotiation downgrade attacks on bluetooth and bluetooth low energy. *ACM Trans. Priv. Secur.*, 23(3), jul 2020. (Cited on pages 3, 4, 9, 11, 26, 40, and 41.)

[ATRP22] Daniele Antonioli, Nils Ole Tippenhauer, Kasper Rasmussen, and Mathias Payer. BLURtooth: Exploiting cross-transport key derivation in bluetooth classic and bluetooth low energy. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22: 17th ACM Symposium on Information, Computer and Communications Security*, pages 196–207, Nagasaki, Japan, May 30 – June 3, 2022. ACM Press. (Cited on pages 4, 13, and 15.)

[BFGJ17] Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. PRF-ODH: Relations, instantiations, and impossibility results. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 651–681, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. (Cited on page 47.)

[Blu23] Bluetooth Special Interest Group (SIG). Bluetooth core specification, January 2023. Ver. 5.4. (Cited on pages 3, 19, 27, 28, 32, and 41.)

[BN19] Eli Biham and Lior Neumann. Breaking the bluetooth pairing - the fixed coordinate invalid curve attack. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography*, volume 11959 of *Lecture Notes in Computer Science*, pages 250–273, Waterloo, ON, Canada, August 12–16, 2019. Springer, Heidelberg, Germany. (Cited on page 9.)

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany. (Cited on pages 5 and 18.)

[CADE23] Tristan Claverie, Gildas Avoine, Stéphanie Delaune, and José Lopes Esteves. Tamarin-based Analysis of Bluetooth Uncovers Two Practical Pairing Confusion Attacks. working paper or preprint, April 2023. (Cited on pages 3, 4, 9, 11, 15, 16, and 40.)

[CE21] Tristan Claverie and José Lopes Esteves. Bluemirror: Reflections on bluetooth pairing and provisioning protocols. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 339–351, 2021. (Cited on pages 4, 5, 14, 15, 35, and 41.)

[CPST22] Matthias Cäsar, Tobias Pawelke, Jan Steffan, and Gabriel Terhorst. A survey on bluetooth low energy security and privacy. *Comput. Netw.*, 205(C), mar 2022. (Cited on page 9.)

[CS07] Richard Chang and Vitaly Shmatikov. Formal analysis of authentication in bluetooth device pairing. In *Proceedings of LICS/ICALP Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA)*, volume 45, Wrocław, Poland, jul 2007. (Cited on page 15.)

[FG14] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 1193–1204, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. (Cited on page 18.)

[FL10] Marc Fischlin and Anja Lehmann. Delayed-key message authentication for streams. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 290–307, Zurich, Switzerland, February 9–11, 2010. Springer, Heidelberg, Germany. (Cited on page 17.)

[FOR17] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. Security of symmetric primitives under incorrect usage of keys. *IACR Transactions on Symmetric Cryptology*, 2017(1):449–473, 2017. (Cited on page 48.)

[FS21] Marc Fischlin and Olga Sanina. Cryptographic analysis of the bluetooth secure connection protocol suite. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 696–725, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany. (Cited on pages 5, 15, 17, 18, 19, 32, 34, 35, 36, 37, and 47.)

[HH07] Konstantin Hypponen and Keijo M.J. Haataja. "nino" man-in-the-middle attack on bluetooth secure simple pairing. In *2007 3rd IEEE/IFIP International Conference in Central Asia on Internet*, pages 1–5, 2007. (Cited on pages 3, 4, 8, 10, 15, and 40.)

[HT10] Keijo Haataja and Pekka Toivanen. Two practical man-in-the-middle attacks on bluetooth secure simple pairing and countermeasures. *IEEE Transactions on Wireless Communications*, 9(1):384–392, 2010. (Cited on pages 4, 8, 10, and 40.)

[JKSS10] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Generic compilers for authenticated key exchange. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 232–249, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany. (Cited on pages 5, 16, and 17.)

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. (Cited on page 47.)

[JW01] Markus Jakobsson and Susanne Wetzel. Security weaknesses in Bluetooth. In David Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 176–191, San Francisco, CA, USA, April 8–12, 2001. Springer, Heidelberg, Germany. (Cited on pages 3, 4, 9, 10, and 41.)

[JZL23] Mohit Kumar Jangid, Yue Zhang, and Zhiqiang Lin. Extrapolating formal analysis to uncover attacks in bluetooth passkey entry pairing. *NDSS 2023*, 2023. (Cited on pages 3, 4, 15, 39, and 40.)

[KHN$^+$14] Charlie Kaufman, Paul E. Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296, October 2014. (Cited on pages 5 and 17.)

[Kra16] Hugo Krawczyk. A unilateral-to-mutual authentication compiler for key exchange (with applications to client authentication in TLS 1.3). In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1438–1450, Vienna, Austria, October 24–28, 2016. ACM Press. (Cited on pages 5 and 17.)

[LÇA$^+$04] Albert Levi, Erhan Çetintaş, Murat Aydos, Çetin Kaya Koç, and M. Ufuk Çağlayan. Relay attacks on bluetooth authentication and solutions. In Cevdet Aykanat, Tuğrul Dayar, and İbrahim Körpeoğlu, editors, *Computer and Information Sciences - ISCIS 2004*, pages 278–288, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. (Cited on pages 4, 14, and 41.)

[Lin09] Andrew Y. Lindell. Comparison-based key exchange and the security of the numeric comparison mode in Bluetooth v2.1. In Marc Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 66–83, San Francisco, CA, USA, April 20–24, 2009. Springer, Heidelberg, Germany. (Cited on page 14.)

[Pie20] Krzysztof Pietrzak. Delayed authentication: Preventing replay and relay attacks in private contact tracing. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *Progress in Cryptology - INDOCRYPT 2020: 21st International Conference in Cryptology in India*, volume 12578 of *Lecture Notes in Computer Science*, pages 3–15, Bangalore, India, December 13–16, 2020. Springer, Heidelberg, Germany. (Cited on page 17.)

[Ros13] Tomas Rosa. Bypassing passkey authentication in Bluetooth low energy. Cryptology ePrint Archive, Report 2013/309, 2013. `https://eprint.iacr.org/2013/309`. (Cited on pages 9 and 10.)

[Rya13] Mike Ryan. Bluetooth: With low energy comes low security. In *7th USENIX Workshop on Offensive Technologies (WOOT 13)*, Washington, D.C., August 2013. USENIX Association. (Cited on pages 3, 4, 9, 10, and 41.)

[SCH$^+$23] Min Shi, Jing Chen, Kun He, Haoran Zhao, Meng Jia, and Ruiying Du. Formal analysis and patching of BLE-SC pairing. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 37–52, Anaheim, CA, August 2023. USENIX Association. (Cited on pages 3, 4, 9, 10, 11, 13, 16, and 40.)

[Sil09] Joseph H. Silverman. *Algorithmic Aspects of Elliptic Curves*, pages 363–408. Springer New York, New York, NY, 2009. (Cited on page 6.)

[SMS18] Da-Zhi Sun, Yi Mu, and Willy Susilo. Man-in-the-middle attacks on secure simple pairing in bluetooth standard v5.0 and its countermeasure. *Personal Ubiquitous Comput.*, 22(1):55–67, feb 2018. (Cited on page 15.)

[SS19] Da-Zhi Sun and Li Sun. On secure simple pairing in bluetooth standard v5.0-part i: Authenticated link key security and its home automation and entertainment applications. *Sensors*, 19(5), 2019. (Cited on page 14.)

[SSL20] Sven Schäge, Jörg Schwenk, and Sebastian Lauer. Privacy-preserving authenticated key exchange and the case of IKEv2. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 567–596, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany. (Cited on pages 5 and 17.)

[TH21] Michael Troncoso and Britta Hale. The bluetooth CYBORG: Analysis of the full human-machine passkey entry AKE protocol. In *ISOC Network and Distributed System Security Symposium – NDSS 2021*, Virtual, February 21–25, 2021. The Internet Society. (Cited on pages 4, 10, 11, 12, 13, 14, 15, 16, 39, and 40.)

[vTPFG21] Maximilian von Tschirschnitz, Ludwig Peuckert, Fabian Franzen, and Jens Grossklags. Method confusion attack on bluetooth pairing. In *2021 IEEE Symposium on Security and Privacy*, pages 1332–1347, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press. (Cited on pages 3, 4, 8, 11, 12, 15, 16, and 40.)

[WNK$^+$20] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave (Jing) Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. BLESA: Spoofing attacks against reconnections in bluetooth low energy. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, August 2020. (Cited on pages 4, 14, 15, and 41.)

[WWX$^+$22] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave Jing Tian, and Antonio Bianchi. Formal model-driven discovery of bluetooth protocol design vulnerabilities. In *2022 IEEE Symposium on Security and Privacy*, pages 2285–2303, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press. (Cited on pages 4, 13, and 15.)

[WWX$^+$24] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave (Jing) Tian, and Antonio Bianchi. Sok: The long journey of exploiting and defending the legacy of king harald bluetooth. 2024. (Cited on page 9.)

[Yin23] Haotian Yin. Security analysis of bluetooth secure simple pairing protocols with extended threat model. *Journal of Information Security and Applications*, 72:103385, 2023. (Cited on page 15.)

[ZWD$^+$20] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. Breaking secure pairing of bluetooth low energy using downgrade attacks. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020: 29th USENIX Security Symposium*, pages 37–54. USENIX Association, August 12–14, 2020. (Cited on pages 3, 4, 8, 15, 38, 39, and 40.)

## A Security Assumptions

In this section we define the necessary assumptions formally. We usually give them in the common asymptotic setting, but the definitions of the advantages can be readily applied to our concrete security statements as well.

**EUF-CMA of Signature and Certification Schemes.** A signature scheme $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ consists of the key generation algorithm outputting a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow\$ \mathsf{KGen}(1^\lambda)$, the signing algorithm $\sigma \leftarrow\$ \mathsf{Sign}(\mathsf{sk}, m)$ outputting a signature $\sigma$ for message $m$, and the verification algorithm $d \leftarrow \mathsf{Vf}(\mathsf{pk}, m, \sigma)$ returning a decision bit $d$. Completeness demands that for any security parameter $\lambda$, any valid key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow\$ \mathsf{KGen}(1^\lambda)$, any message $m$ and any signature $\sigma \leftarrow\$ \mathsf{Sign}(\mathsf{sk}, m)$, we always have $\mathsf{Vf}(\mathsf{pk}, m, \sigma) = 1$.

Existential unforgeability under chosen message attacks demands that no adversary can forge signatures, even after having seen signatures for other messages:

**Definition A.1 (EUF-CMA)** *A signature scheme $\Sigma$ is EUF-CMA if for any PPT adversary $\mathcal{A}$, we have*

$$\boldsymbol{Adv}^{EUF\text{-}CMA}_{\mathcal{A},\Sigma}(\lambda) := \Pr\Big[\boldsymbol{Exp}^{EUF\text{-}CMA}_{\mathcal{A},\Sigma}\Big]$$

*is negligible, where*

```
$\boldsymbol{Exp}^{EUF\text{-}CMA}_{\mathcal{A},\Sigma}$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow\$ \mathsf{KGen}(1^\lambda)$
$(m^*, \sigma^*) \leftarrow\$ \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})$
$\boldsymbol{return}$ 1 $\boldsymbol{if}$
  $\mathsf{Vf}(\mathsf{pk}, m^*, \sigma^*) = 1$ $\boldsymbol{and}$ $m^*$ has not been queried to $\mathsf{Sign}(\mathsf{sk}, \cdot)$
```

The definition of unforgeability is analogously for certification schemes $\Gamma = (\mathsf{CertKGen}, \mathsf{Cert}, \mathsf{CertVf})$, when viewing the issued certificate as the signature and the certified data as the message.

**Pseudorandom Function.** A pseudorandom function $\mathsf{PRF}$ for security parameter $\lambda$ takes as input a key $k \in \{0,1\}^\lambda$ and some input $x$ and returns a value $y \in \{0,1\}^\lambda$. The outputs should look random, i.e., they should be indistinguishable from random (but consistent) answers. In the definition below, we capture consistency of the random answers by maintaining an initially empty list $\mathcal{F}$ of already evaluated inputs.

**Definition A.2 (PRF Assumption)** *A function* $\mathsf{PRF}$ *is pseudorandom if for any PPT adversary $\mathcal{A}$, we have*

$$\boldsymbol{Adv}^{PRF}_{\mathcal{A},\mathsf{PRF}}(\lambda) := \Pr\Big[\boldsymbol{Exp}^{PRF}_{\mathcal{A},\mathsf{PRF}}\Big] - 1/2$$

*is negligible, where*

```
$\underline{\textbf{\textit{Exp}}^{PRF}_{\mathcal{A},\mathsf{PRF}}}$
$b \leftarrow\$ \{0,1\}$
$\mathcal{F} \leftarrow \emptyset$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{OPRF}}(1^\lambda)$
$\textbf{return } b = b'$
```

*where oracle* $\mathsf{OPRF}$ *on input* $x$ *checks if* $x$ *has the correct length, and returns* $\perp$ *if not. Else it checks if* $(x, y) \in \mathcal{F}$ *and returns* $y$ *if so. Otherwise it computes* $y \leftarrow \mathsf{PRF}(k, x)$ *if* $b = 0$ *resp. samples* $y \leftarrow\$ \{0,1\}^\lambda$ *if* $b = 1$*, stores* $(x, y)$ *in* $\mathcal{F}$*, and returns* $y$ *to the adversary.*

**PRF-ODH Assumption.** We employ the version of $\mathsf{PRF\text{-}ODH}$ assumption from [FS21] which has been modified to match the Bluetooth setting. The assumption intertwines a PRF-like assumption with a Diffie–Hellman-based key generation process. It has been originally proposed in [JKSS12] and analyzed further in [BFGJ17], and is handy when analyzing protocols like Bluetooth where Diffie–Hellman values can be reused throughout multiple sessions.

Let $\mathbb{G}$ be a cyclic (multiplicatively written) group of prime order $q = q(\lambda)$ with generator $g$, $\mathsf{PRF} : \mathbb{G} \times \{0,1\}^* \to \{0,1\}^*$ be a pseudorandom function, with a key $k \in \mathbb{G}$ and a string $s$ as input, and a string $\mathsf{PRF}(k, s)$ as output. For a given $w \in \mathbb{Z}_q$, let $\mathsf{ODH}_w : \mathbb{G} \times \{0,1\}^* \to \{0,1\}^*$ be the function with $X \in \mathbb{G}$ and string $s$ as input, and $\mathsf{PRF}(\langle X^w \rangle_x, s)$ as output. Let $g^u$ resp. $g^v$ with $u$ resp. $v$ be in the range 1 and $q/2$, or in the range $q/2$ and $q$. We let PRF-ODH to use only the $x$-coordinate and forbid the adversary from querying $\mathsf{ODH}_u$ and $\mathsf{ODH}_v$ about the challenge value $g^{uv}$ with label $x^*$ and about $g^{-uv}$ with $x^*$.

**Definition A.3 (PRF-ODH Assumption)** *The PRF-ODH assumption holds relative to* $\mathbb{G}$ *if for any PPT adversary* $\mathcal{A}$*, we have*

$$\textbf{\textit{Adv}}^{PRF\text{-}ODH}_{\mathcal{A},\mathsf{PRF},\mathbb{G}}(\lambda) := \Pr\Big[\textbf{\textit{Exp}}^{PRF\text{-}ODH}_{\mathcal{A},\mathsf{PRF},\mathbb{G}}\Big] - 1/2$$

*is negligible, where*

```
$\underline{\textbf{\textit{Exp}}^{PRF\text{-}ODH}_{\mathcal{A},\mathsf{PRF},\mathbb{G}}}$
$u, v \leftarrow\$ \mathbb{Z}_{(q+1)/2} \setminus \{0\}, b \leftarrow\$ \{0,1\}$
$U \leftarrow g^u, V \leftarrow g^v$
$(x^*, \mathit{st}) \leftarrow\$ \mathcal{A}^{\mathsf{ODH}_u(\cdot,\cdot),\mathsf{ODH}_v(\cdot,\cdot)}(U, V)$
$y_0 \leftarrow \mathsf{PRF}(\langle g^{uv} \rangle_x, x^*), y_1 \leftarrow \{0,1\}^{|y_0|}$
$b' \leftarrow\$ \mathcal{A}^{\mathsf{ODH}_u(\cdot,\cdot),\mathsf{ODH}_v(\cdot,\cdot)}(\mathit{st}, V, y_b)$
$\textbf{return } b = b'$
```

*where* $\mathcal{A}$ *never makes a query* $(A, x) = (V^{\pm 1}, x^*)$ *to oracle* $\mathsf{ODH}_u$ *resp.* $(B, x) = (U^{\pm 1}, x^*)$ *to* $\mathsf{ODH}_v$*.*

**Collision Resistance.** Collision resistance is more tricky to define since we cannot use the usual asymptotic behavior, unless we deal with keyed hash functions. Hence we only define the advantage here.

**Definition A.4 (Collision Resistance)** *For a deterministic function* $\mathsf{Hash}$ *and adversary* $\mathcal{A}$*, define the collision resistance advantage as*

$$\textbf{\textit{Adv}}^{CR}_{\mathcal{A},\mathsf{Hash}}(\lambda) := \Pr\Big[\textbf{\textit{Exp}}^{CR}_{\mathcal{A},\mathsf{Hash}}\Big],$$

*where*

$$\begin{array}{l}\boldsymbol{Exp}^{CR}_{\mathcal{A},\mathsf{Hash}} \\ \hline (x, x') \leftarrow\$\, \mathcal{A}(1^\lambda) \\ \boldsymbol{return}\ 1\ \boldsymbol{if}\ x \neq x'\ \boldsymbol{and}\ \mathsf{Hash}(x) = \mathsf{Hash}(x')\end{array}$$

**Key-collision Resistance of Block Ciphers.** We give the definition of key collision resistance abstractly for block ciphers, where a block cipher is a deterministic algorithm $\mathsf{BC}$, which takes as input a key $k$ and value $x$ of equal length, as well as a sign $\pm$ (to indicate if one should compute the answer in forward or backward direction), and returns a value $y$ of equal length as $x$. For each key $k \in \{0,1\}^\lambda$, the functions $\mathsf{BC}(k, \cdot, +)$ and $\mathsf{BC}(k, \cdot, -)$ are permutations over $\{0,1\}^\lambda$ such that $\mathsf{BC}(k, \mathsf{BC}(k, x, +), -) = x$ and $\mathsf{BC}(k, \mathsf{BC}(k, y, -), +) = y$.

Key-collision resistance of a block cipher now demands that it is infeasible to find $k \neq k'$ and $x$ with $\mathsf{BC}(k, x, +) = \mathsf{BC}(k', x, +)$. Once more, we cannot rely on the asymptotic behavior unless we introduce keyed versions. The notion here is related to the definition of full robustness of pseudorandom functions [FOR17] (given a $\mathsf{PRF}$, find $k \neq k'$ and $x, x'$ such that $\mathsf{PRF}(k, x) = \mathsf{PRF}(k', x')$ collide), albeit we work with block ciphers and explicitly ask the adversary to find a collision for the same input $x = x'$. The latter is indeed necessary for block ciphers since otherwise any values $k \neq k', x$ and $x' = \mathsf{BC}(k', \mathsf{BC}(k, x, +), -)$ would form a valid solution.

**Definition A.5 (Key-Collision Resistance of Block Ciphers)** *For block cipher $\mathsf{BC}$ and adversary $\mathcal{A}$, define the key-collision resistance advantage as*

$$\boldsymbol{Adv}^{\textit{key-coll}}_{\mathcal{A},\mathsf{BC}}(\lambda) := \Pr\left[\boldsymbol{Exp}^{\textit{key-coll}}_{\mathcal{A},\mathsf{BC}}\right],$$

*where*

$$\begin{array}{l}\boldsymbol{Exp}^{\textit{key-coll}}_{\mathcal{A},\mathsf{BC}} \\ \hline (k, k', x) \leftarrow\$\, \mathcal{A}(1^\lambda) \\ \boldsymbol{return}\ 1\ \boldsymbol{if}\ k \neq k'\ \boldsymbol{and}\ \mathsf{BC}(k, x, +) = \mathsf{BC}(k', x, +)\end{array}$$

## B Acronyms

| | |
|---|---|
| AKE | Authenticated Key Exchange |
| BIAS | Bluetooth Impersonation AttackS |
| BC | Block Cipher |
| BLE | Bluetooth Low Energy |
| BLESA | Bluetooth Low Energy Spoofing Attack |
| BLUFFS | BLUetooth Forward and Future Secrecy |
| BR/EDR | Basic Rate/Enhanced Data Rate |
| CR | Collision Resistance |
| CSIA | Cross Stack Illegal Access |
| DDH | Decisional Diffie–Hellman |
| DOFU | Deferrable outside first use |
| ECDH | Elliptic Curve Diffie–Hellman |
| EUF-CMA | Existential Unforgeability under Chosen Message Attack |
| IO | Input/Output |
| JW | Just Works |
| KE | Key Exchange |

| | |
|---|---|
| KNOB | Key Negotiation of Bluetooth |
| LE | Low Energy |
| LK | Link Key |
| LTK | Long-term key |
| L(T)K | Long-term and Link key |
| MAC | Media Access Control *or* Message Authentication Code |
| MitM | Machine in the Middle |
| ODH | Oracle Diffie–Hellman |
| NINO | No Input No Output |
| NumCom | Numeric Comparison |
| OOB | Out of Band |
| PDU | Protocol Data Unit |
| PKE | Passkey Entry |
| PKI | Public-key infrastructure |
| PPT | Probabilistic Polynomial-time |
| PRF | Pseudorandom Function |
| PRF-ODH | Pseudorandom-function oracle-Diffie–Hellman |
| SC | Secure Connections |
| SCO | Secure Connections Only |
| SIG | Bluetooth Special Interest Group |
| SMP | Security Manager Protocol |
| SSP | Secure Simple Pairing |
| STK | Short-term key |
| TK | Temporal key |
| TOFU | Trust on the first use |