BlueBrothers: Three New Protocols to Secure Bluetooth

Tommaso Sacchetti
EURECOM
France
tommaso.sacchetti@gmail.com

Kasper Rasmussen
University of Oxford
United Kingdom
kasper.rasmussen@cs.ox.ac.uk

Daniele Antonioli
EURECOM
France
antonioli.daniele@gmail.com

Abstract

Bluetooth is a pervasive wireless standard that, despite numerous revisions, remains vulnerable to multiple design-level security flaws. Specifically, its pairing and session establishment security protocols do not provide integrity protection, forward secrecy, or strong authentication mechanisms, enabling critical impersonation and man-in-the-middle attacks. These risks are compounded by complex and fragmented specifications, which hinder secure implementation and formal analysis.

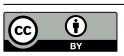
To address these issues, we present BlueBrothers, three new protocols to serve as a secure alternative to the current ones. BB-Pairing combines pairing and session establishment in a single protocol that provides integrity protection and robust user-assisted authentication. BB-Session establishes authenticated, secure sessions with forward secrecy guarantees. BB-Rekey provides forward and future secrecy within a session via a lightweight key-refresh mechanism.

We model BlueBrothers in ProVerif and verify confidentiality, integrity, and entity-authentication properties. We implement the protocols on constrained nRF52 devices and evaluate performance against the Bluetooth baseline. Our results show up to a 59% reduction in latency with comparable energy consumption.

ACM Reference Format:
Tommaso Sacchetti, Kasper Rasmussen, and Daniele Antonioli. 2026. BlueBrothers: Three New Protocols to Secure Bluetooth. In *Proceedings of the 19th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '26), June 30-July 03, 2026, Saarbrücken, Germany*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3765613.3811675

1 Introduction

Bluetooth is a pervasive wireless technology used by billions of devices and specified in the Bluetooth Core Specification v6.2 [8]. It offers two transport protocols: Bluetooth Classic (BC) for high-throughput, connection-oriented services such as audio streaming, and Bluetooth Low Energy (BLE) for ultra-low-power communication. Bluetooth has two security protocols: *pairing*, an (optionally) authenticated long-term key establishment protocol; and *session establishment*, a symmetric key derivation protocol that derives a fresh Session Key (SK) to encrypt each session. Although BC and BLE differ in various aspects, their pairing and session establishment protocols are structurally similar. We abstract their message flows and highlight the differences where necessary.



WiSec '26, Saarbrücken, Germany

ACM ISBN 979-8-4007-2201-1/2026/06
https://doi.org/10.1145/3765613.3811675

Despite significant updates and revisions to the Bluetooth standard, *17 design-level attacks* (Table 1) remain effective against pairing and session establishment [1, 5, 14]. Design-level attacks have severe and widespread consequences, enabling eavesdropping, impersonation, or Man-in-the-Middle (MitM) against any deployed Bluetooth device. We systematically review the protocol specifications and identify the *four root causes* of the 17 attacks: (C1), lack of message integrity; (C2), absence of replay and reflection protection; (C3), weak or absent forward and future secrecy; and (C4), insufficient entity authentication.

Additionally, the complexity and the informal, scattered nature of the Bluetooth specification hinder understanding, implementation, and formal verification of security protocols, contributing to additional vulnerabilities, including implementation-level ones [23, 37]. Based on these two reasons, we believe that Bluetooth security protocols need a clean-slate redesign.

To address these challenges, we present **BlueBrothers**, three new Bluetooth security protocols designed to replace existing Bluetooth pairing and session establishment. BlueBrothers are *secure-by-design* and *performant*. They feature a *streamlined specification* that facilitates integration into existing BC and BLE stacks while mitigating the C1–C4 vulnerability classes and the 17 attacks. The three protocols are: BB-Pairing, BB-Session, and BB-Rekey.

BB-Pairing bootstraps long-term (static) public keys and provides message integrity, replay and reflection resistance, and forward secrecy against the compromise of static keys. The protocol supports flexible authentication via pre-shared keys or user-assisted mechanisms, thus retaining the Bluetooth Trust On First Use (TOFU) model.

BB-Session authenticates an ephemeral Diffie-Hellman (DH) exchange using Elliptic Curve Digital Signature Algorithm (ECDSA) signatures under long-term keys established by BB-Pairing. It provides inter-session forward secrecy, ensuring that past communications remain secure even after a long-term key compromise. BB-Rekey introduces a novel intra-session rekey that provides both forward and future secrecy (post-compromise security), protecting past and future communication if a session key is compromised.

We formally verify BlueBrothers using ProVerif [7] under a Dolev-Yao attacker model, proving session key confidentiality and agreement, mutual authentication, and forward and future secrecy.

We implement BlueBrothers on two layers: a stack-layer integration for BLE by modifying the NimBLE stack, and an application-layer implementation via a standalone, portable C library that works with both BC and BLE. We evaluate their performance on constrained devices, specifically the nRF52840 (Cortex-M4). Compared to BLE Secure Connections (SC) pairing and standard session establishment, our protocols reduce latency by up to 59%. In terms of energy consumption, BB-Pairing reduces it, while BB-Session and

BB-Rekey can slightly increase it. Overall, our results indicate that BlueBrothers are suitable for practical, real-world deployment.

We summarize our contribution as follows:

- We systematically review Bluetooth pairing and session establishment, identifying 17 still-effective design-level attacks stemming from four vulnerability classes (C1–C4).
- We design BlueBrothers, three new protocols to replace the insecure ones from the Bluetooth standard. Our protocols address the 17 attacks and their root causes (C1–C4); they are more secure, performant, and easy to integrate.
- We formally model the protocols in ProVerif and verify their security properties, i.e., mutual authentication, confidentiality, forward and future secrecy, and KCI resistance.
- We implement BlueBrothers for BLE and evaluate them on constrained nRF52 devices. We demonstrate up to a 59% reduction in latency compared to the standard protocol and quantify the trade-offs in energy consumption.

Our implementation, evaluation data, and ProVerif models are available at anonymous.4open.science/r/bb-protocols-FBB1.

2 Bluetooth Preliminaries

Bluetooth is a widely used wireless technology specified by the Bluetooth Core Specification (v6.2) [8]. The standard defines BC for high-throughput services and BLE for low-power communication; the connection initiator is the Central and the responder is the Peripheral. The Bluetooth stack comprises two components: a Host and a Controller, connected via the Host Controller Interface (HCI).

Bluetooth defines two security protocols: *pairing*, which establishes a long-term Pairing Key (PK), and *session establishment*, which derives a SK to encrypt the connection. Their specifications are informal, differ between BC and BLE, and include (legacy) subprotocols, modes, features, and crypto primitives. For example, they support two security modes: Legacy Secure Connections (LSC), which uses legacy primitives (e.g., the E0 cipher [21]), and SC, which uses FIPS-compliant ones (e.g., Elliptic Curve Diffie-Hellman (ECDH)).

2.1 Bluetooth Pairing

Pairing is a key-establishment protocol that derives and optionally authenticates a PK. It is based on a Trust-On-First-Use (TOFU) authentication model and does not rely on certificates or a Public Key Infrastructure (PKI). The standard defines authentication as "association"; devices can enable it via user interaction or out-of-band channels (e.g., NFC). BC and BLE have distinct pairing protocols that share the same high-level messages and phases, i.e., Feature Negotiation, to exchange options; Association, for optional authentication; and Key Derivation to compute the PK.

Figure 1 provides an abstract overview of pairing between Central and Peripheral. The devices exchange identities (C, P) and pairing features (F_C, F_P). The features include supported security mode, I/O capabilities, and Cross-Transport Key Derivation (CTKD). CTKD allows deriving PKs for BC and BLE with a single pairing. For BLE, the features also include an integer between 7 and 16 to negotiate the PK entropy.

Then, the devices generate ECDH key pairs, exchange public keys (K_C^{Pub}, K_P^{Pub}) and compute a DH shared secret (SS). Based on I/O capability and MitM protection settings, they then run one of four association methods: JustWorks (JW), Numeric Comparison (NC), Passkey Entry (PE), or Out of Band (OOB). Successful association yields an authenticated binding, except for JW, which does not; after that, the protocol derives the PK from SS and the negotiated parameters.

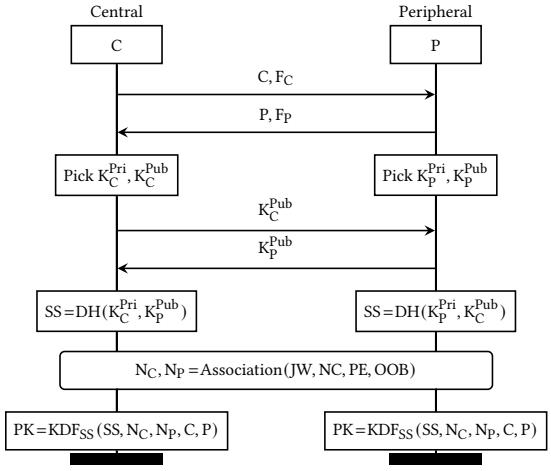


Figure 1: Bluetooth SC pairing. Central and Peripheral negotiate pairing features, derive a DH shared secret (SS), run one of four association protocols, and derive a long-term PK.

Figure 2 illustrates NC association. The devices select fresh random nonces (N_C, N_P). Peripheral computes a confirmation value $C_P = KDF(K_P^{Pub}, K_C^{Pub}, N_P, 0)$ that Central verifies. After verification, both parties compute a numeric value (V_C, V_P) from their public keys and the exchanged nonces, and display it on their screen. The user, whose interactions are shown as dashed lines, confirms the association only if the displayed values are identical; otherwise, he aborts the pairing. The JW method uses the same message sequence but skips user confirmation, providing no mutual authentication and leaving the connection vulnerable to MitM attacks. PE and OOB are described in Appendix 9 for space constraints.

After the association phase, the devices derive PK using a Key Derivation Function (KDF) with the SS, device identities, and negotiated features as input. In BLE, devices enforce the negotiated entropy by setting the most significant bytes of the PK to zero.

2.2 Bluetooth Session Establishment

Session establishment enables paired devices to derive a fresh SK from the long-term PK and session nonces, and to use SK to protect subsequent traffic. Typically, devices pair once and then establish a session on each reconnection.

Figure 3 shows a high-level representation of session establishment, abstracting the differences between BC and BLE. Central and Peripheral negotiate session features (F_C, F_P), like SC support or

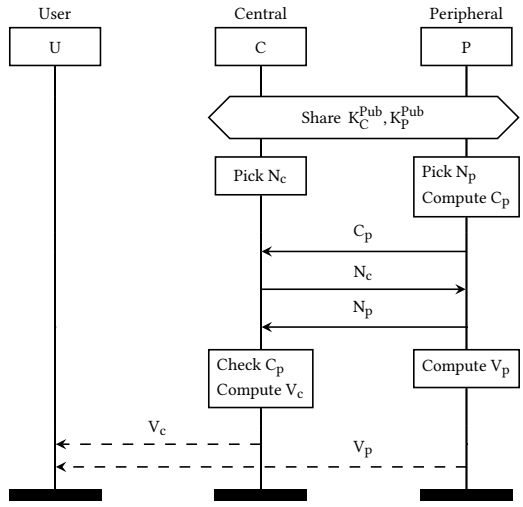


Figure 2: Bluetooth NC and JW association. In NC, the user confirms that V_C equals V_P. In JW, there is no user interaction. Dashed messages represent user-device interactions.

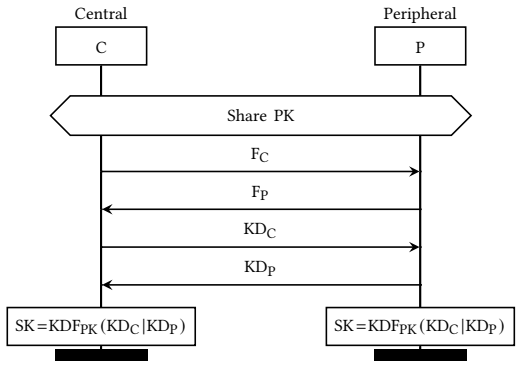


Figure 3: Bluetooth session establishment. Devices sharing a PK negotiate their session features and SK entropy (BC only), exchange key diversifiers, and derive SK to encrypt a session. BC has a PK authentication phase before the exchange of key diversifiers.

for BC the entropy of SK. Next, BC includes a PK- based authentication phase using a challenge-response mechanism (omitted from the MSC for readability). The parties then exchange session key diversifiers (KD_C, KD_P), which are input to a KDF keyed with PK to derive SK. In both BC and BLE, these messages are not encrypted nor integrity-protected. Upon deriving the SK, the devices protect the channel using authenticated encryption with AES-CCM. BC in LSC will use the E0 stream cipher, which does not provide integrity protection.

Forward secrecy and future secrecy (post-compromise security) capture how well a protocol limits the damage of key compromise. *Forward secrecy* means that compromising long-term keys (e.g, PK) or current session keys does not allow an adversary to decrypt *previous* traffic. *Future secrecy* means that, after a compromise, the parties can *recover* so that subsequent traffic becomes confidential again after the key update.

In the Bluetooth standard, an SK remains static throughout the connection lifetime. Consequently, a compromise of the SK during an active session enables an adversary to decrypt all previous and subsequent traffic within that session. Furthermore, the standard does not provide inter-session forward secrecy: since session keys are derived from the PK with only symmetric contributions, a compromise of the PK allows an adversary to decrypt past sessions and impersonate devices in future ones retroactively.

Table 1: BC and BLE security protocols are vulnerable to 17 design-level attacks stemming from four vulnerability categories (C1–C4) and affecting SC and LSC security modes.

Type	Attack	Year	Mode	Vulns.	Protocol
BLE	Pairing confusion [14]	2023	SC	C1	Pairing
BC	Pairing confusion [14]	2023	SC	C1	Pairing
BLE	Method confusion 2 [14]	2023	SC	C1	Pairing
BC	Method confusion 2 [14]	2023	SC	C1	Pairing
BC	BLUR [4]	2022	SC	C1, C2	Pairing
BLE	BLUR [4]	2022	SC	C1, C2	Pairing
BC	Method confusion [41]	2021	SC	C1	Pairing
BLE	Method confusion [41]	2021	SC	C1	Pairing
BC	BlueMirror A [15]	2021	LSC	C1, C2	Pairing
BLE	BlueMirror A [15]	2021	LSC	C1, C2	Pairing
BLE	BlueMirror PE-A1 [15]	2021	SC	C1, C2	Pairing
BLE	BlueMirror PE-A2 [15]	2021	SC	C1, C2	Pairing
BLE	KNOB [3]	2020	SC	C1	Pairing
BC	BLUFFS [1]	2023	SC	C3, C4	Session Est.
BC	BIAS [2]	2020	SC	C2, C4	Session Est.
BLE	BLESA [45]	2020	SC	C4	Session Est.
BC	KNOB [5]	2019	SC	C2, C4	Session Est.

3 Motivation

We systematically review the state of the art in Bluetooth security and identify *two issues* that motivate a clean-slate redesign of Bluetooth security protocols. First, despite years of revisions and patches, pairing and session establishment remain susceptible to multiple *design-level attacks* in multiple configurations. Second, the protocol specifications are complex, informal, and fragmented across many options, modes, and transports (BC and BLE), which hinders understanding, implementation, and formal analysis, and contributes to additional implementation-level vulnerabilities.

3.1 Bluetooth Protocols Design Issues

We conducted a systematic review of the Bluetooth Core Specification v6.2 and the academic literature to assess which known

protocol-level attacks still apply to the current version of the protocols. From this analysis, we identified 17 design-level attacks (Table 1) that have not been addressed and we additionally identify their root causes and categorize them: (C1) lack of message integrity, (C2) absence of replay and reflection protection, (C3) weak or missing forward and future secrecy, and (C4) weak or missing entity authentication. Each attack exploits at least one category; for example, pairing confusion [14] exploits C1, whereas BIAS [2] leverages C2 and C4.

The first part of Table 1 lists 13 attacks targeting pairing, which is the most affected protocol. Attacks such as KNOB [3] target feature negotiation and exploit the absence of message integrity to downgrade PK entropy, or BLUR [4] enables PK overwrite across transports. Attacks on the association phase, such as BlueMirror [15], exploit missing integrity and reflection protections to bypass user-assisted authentication. Method- and pairing-confusion attacks [14, 41] combine weaknesses across both these phases to bypass authentication.

Table 1's second block lists attacks on session establishment. The protocol is vulnerable because it lacks integrity protection, relies on weak or absent entity authentication, and provides no resistance to replay or reflection attacks. As a result, an attacker can achieve device impersonation (e.g., by combining KNOB [5] and BIAS [2]). Similarly, BLUFFS [1] exploits weak session key reuse to spoof BC devices, while BLESA [45] enables Peripheral spoofing during reconnections.

The most recent Bluetooth security survey [47], published in 2024, provides a comprehensive historical taxonomy of Bluetooth vulnerabilities up to version 5.4. While that work catalogs a wide range of flaws regardless of their patch status, our research isolates explicitly design-level vulnerabilities that remain exploitable in the current specification up to version 6.2 and covers recent developments, such as the BLUFFS attacks [1]. We further extend the systematization by performing a root-cause analysis that identifies four distinct vulnerability categories (C1–C4) that are persistent in the current specification of pairing and session establishment.

3.2 Bluetooth Protocols Complex Specification

Another significant challenge for Bluetooth security is the complexity of the pairing and session establishment specifications. These protocols include numerous sub-protocols, optional features, and cryptographic primitives that vary depending on the transport (BC, BLE), the association method, the security mode, and the device capabilities. Moreover, the specification is informal and scattered across a large, multi-thousand-page document, errata, and paywalled supplementary material, making it difficult to understand, implement, maintain, and formally analyze it.

Prior work [28, 39, 45, 48] highlighted this complexity by producing formal models that cover subsets of pairing and session establishment. To date, no formal analysis has modeled all combinations of BC and BLE pairing and session establishment. Moreover, the standard's complexity further compounded the problem and contributed to serious implementation flaws. For example, researchers reported pairing and session establishment vulnerabilities leading to Denial of Service (DoS) and Remote Code Execution (RCE) [22, 31, 37], skipped DH confirmation, installation of zeroed PKs [23], failure to enforce SC-only mode [50], and missing association dialogs [51].

4 BlueBrothers Protocols

Motivated by the issues discussed in Section 3, we create **BlueBrothers**, three new Bluetooth security protocols (*BB-Pairing*, *BB-Session*, and *BB-Rekey*) designed from the ground up. The protocols are secure-by-design, address C1–C4, and provide *integrity protection* and *forward and future secrecy* across and within sessions.

BlueBrothers feature a streamlined specification which makes them easy to *formally verify* and integrate with the Bluetooth standard for both BC and BLE. The protocols support the Bluetooth TOFU model while remaining adaptable to vendor-specific scenarios that use PSKs or certificates. This flexibility allows for two integration paths: a stack-level adoption, in which the Bluetooth SIG integrates the protocols directly into the standard, or a vendor-driven approach, in which vendors implement them as a security overlay at the application layer.

4.1 Threat Model

System Model. We consider Central and Peripheral devices using BlueBrothers to pair and establish secure sessions over BC or BLE. These devices represent a broad spectrum of Bluetooth hardware, ranging from resource-constrained IoT sensors to smartphones and laptops, and have any I/O capabilities or none. Devices support the standard Bluetooth TOFU model via user-assisted authentication, or rely on a vendor-specific setup with PSKs or certificates. We assume that random number generators (RNGs) provide high-entropy outputs and that all cryptographic primitives are computationally secure and correctly implemented according to their specifications. We consider implementation-specific vulnerabilities, such as side-channel attacks, physical tampering, or platform-level RNG failures, to be out of scope.

For user-assisted association, we distinguish an *attentive* user who follows the device instructions (e.g., correctly compares/enters the displayed code) but may be confused, from a *inattentive* user who approves without verification, as the latter effectively provides no authentication guarantees.

Attacker model. We assume a wireless adversary within Bluetooth range of the victims, interested in exploiting protocol-level issues on BlueBrothers. Unless stated otherwise, it has Dolev-Yao capabilities [17], i.e., it can intercept, inject, modify, drop, replay, reflect, or redirect messages across concurrent protocol runs and has access to public device information (e.g., addresses, features). It can attempt known protocol-level attacks on pairing and session establishment, including those in Table 1.

To assess forward and future secrecy, we assume that the adversary compromises static private keys or session keys. For *inter-session forward secrecy*, we allow compromise of long-term static keys after a session ends. For *intra-session future secrecy*, we assume the attacker is passive during the rekey that follows a key leak. This assumption aligns with post-compromise security threat models, in which recovery requires at least one honest session without active interference [16].

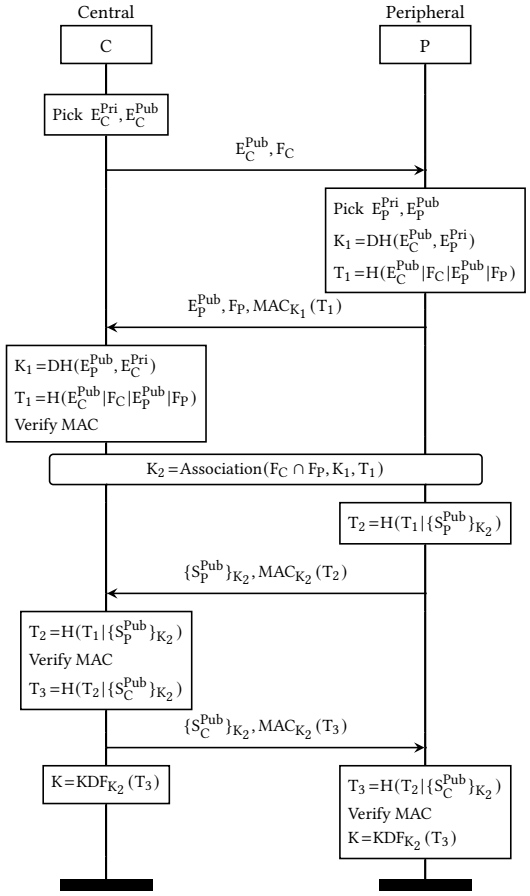


Figure 4: BB-Pairing. The devices securely negotiate their feature, authenticate using association, securely exchange their static keys, and derive a session key (K).

4.2 BB-Pairing Design

BB-Pairing (Figure 4) integrates pairing and session establishment into a single secure protocol that distributes per-device *asymmetric static keys* for subsequent authentication and relies on *asymmetric ephemeral keys* to derive a *session key* to encrypt the connection. BB-Pairing provides integrity protection via a hash transcript and Message Authentication Code (MAC), resistance to replay and reflection attacks, and inter-session forward secrecy. Moreover, it includes redesigned association methods based on Short Authentication Strings (SAS) derived from the hash transcript and uses explicit method identifiers to mitigate method confusion attacks.

The protocol has *four phases*: feature negotiation, association, static key exchange, and session key derivation.

Feature Negotiation. In the feature negotiation phase, the devices exchange their ephemeral public keys (E_C^{Pub}, E_P^{Pub}) and their feature sets (F_C, F_P), which include the supported association methods. The devices compute a hash transcript (T_1) over the exchanged

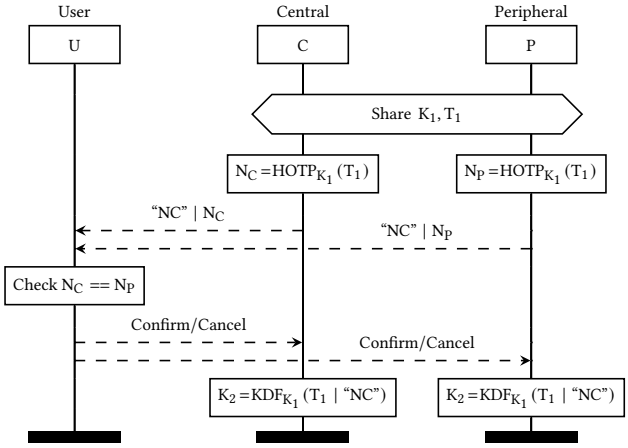


Figure 5: BB-NC. The devices compute and display N_C and N_P. If the user confirms $N_C = N_P$ the devices derive K_2.

messages, which is then used to integrity-protect the subsequent messages using a MAC.

Association. BB-Pairing introduces four novel authentication methods called BB-NC, BB-PE, BB-OOB, and BB-JW. While negotiation happens similarly to the standard methods using I/O capabilities, they differ by cryptographically binding the association phase to the handshake transcript.

Specifically, rather than relying on random codes, BB-NC implements user-assisted authentication via SAS derived from the cryptographic state. The devices generate a short code using HMAC-based One-time Password (HOTP) [30] over K_1 and the hash transcript T_1, and include an explicit association-method identifier. The user compares the code on both devices to prevent method confusion. On success, both parties derive K_2 using a KDF keyed with K_1 over the transcript and the association method identifier, thereby binding the human decision to the session. On failure (e.g., if an incorrect prefix is used), the protocol aborts.

BB-PE similarly binds the association phase to the handshake transcript, but instead of using SAS, it generates a random numeric code. It supports two modes: one in which a device displays the random code prefixed with a method identifier and the user enters it into the peer (Figure 9); and one in which the user generates the code and enters it on both devices (10).

BB-OOB derives K_2 by mixing a pre-shared key (PSK) obtained via an external channel (e.g., NFC), providing strong authentication without user interaction.

BB-JW, like the standard JW, is designed for devices with no I/O capabilities and provides no authentication. It should be restricted to scenarios in which authenticated methods are technically infeasible.

Static Key Exchange. The parties exchange their static public keys (S_C^{Pub}, S_P^{Pub}) encrypted under K_2 and integrity-protected using a MAC over the hash transcript. Successful decryption and MAC

verification binds both identities to the negotiated features and association outcome, yielding mutual authentication (except for BB-JW) and enabling subsequent session key derivation.

Session Key Derivation. After successful static key exchange and mutual authentication, both parties derive the session key K as: $K = KDF_{K_2}(T_3)$, i.e., a KDF keyed with K_2 over the final transcript. The resulting K is then used to protect session traffic. Because K is derived from K_2 and the full transcript (ephemeral keys, negotiated features, and static identifiers), agreement on K implies the transcript was authentic and untampered.

4.3 BB-Session Design

BB-Session (Figure 6) is a secure alternative to the standard session establishment protocol for devices that already possess each other's static public keys (e.g., provisioned via BB-Pairing). The key design goal is to establish a fresh session key while (1) mutually authenticating the ephemeral key exchange using the pre-provisioned static keys, (2) cryptographically binding feature negotiation to the execution to prevent downgrade attacks, and (3) providing explicit key confirmation so both parties detect tampering or desynchronization before sending protected application data. As in BB-Pairing, devices negotiate the rekey type during setup to enable later BB-Rekey. The protocol has *three phases*: feature negotiation, authenticated ephemeral exchange, and session key derivation and confirmation.

Feature Negotiation. The devices exchange their supported feature sets together with fresh ephemeral public keys (E_C^{Pub}, E_P^{Pub}). Each party computes a hash transcript T over all messages exchanged so far, including the negotiated features, role identifiers (Central vs. Peripheral), and both ephemeral public keys.

Authenticated Ephemeral Exchange. To authenticate the ephemeral DH exchange, each device signs the transcript hash T using its static private key corresponding to the peer-known static public key (S_C^{Pub}, S_P^{Pub}). Because the signature covers the full transcript (including both ephemeral keys and negotiated features), an active adversary cannot modify, replay, or reflect messages across roles without causing the signature verification to fail. This step provides mutual authentication (under the assumption that the stored static public keys are not compromised) while preserving the freshness and forward secrecy properties of ephemeral DH.

Session Key Derivation and Confirmation. After both signatures verify, both parties compute the ephemeral DH shared secret SS and derive the session key as a KDF output over SS and the transcript, i.e., K = KDF(SS | T). Including T in the key schedule ensures that agreement on K is cryptographically tied to the same negotiated features, roles, and ephemeral values that were authenticated by the signatures

Then, the Peripheral sends a fresh nonce encrypted (and integrity-protected) under K. Successful decryption and verification provide explicit key confirmation: the Central learns that the Peripheral derived the same K from the same transcript and shared secret. This prevents failures in which authentication succeeds, but the parties derive different keys (e.g., due to message reordering, state confusion, or partial transcript mismatch).

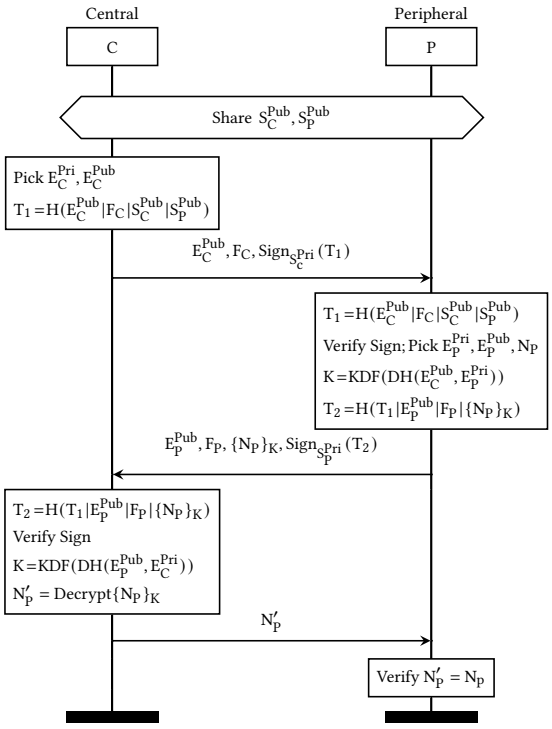


Figure 6: BB-Session. The devices use the shared static keys to authenticate the ephemeral key exchange and to negotiate features. Then they derive a session key (K).

4.4 BB-Rekey Design

BB-Rekey operates within an existing session established by BB-Pairing or BB-Session and updates the session key *within* a session. The design goal is to reduce the security impact of key compromise for long-lived sessions by periodically refreshing the encryption key. BB-Rekey introduces two new security guarantees not provided by the standard: *intra-session forward secrecy* (past keys remain safe if the current key is compromised) and *intra-session future secrecy* (a compromise of the current key does not permanently compromise all future keys once a rekey completes).

Rekeys can be initiated by either Central or Peripheral, and can be *symmetric* and/or *asymmetric*, depending on which devices negotiated during BB-Pairing or BB-Session.

Asymmetric Rekey. In the asymmetric mode (Figure 7), the parties exchange fresh ephemeral DH public keys (E_C^{Pub}, E_P^{Pub}) and nonces (N_C, N_P). Both compute the DH shared secret SS and derive the next session key as $K_N = KDF_K(SS \mid N_C \mid N_P)$. This mode provides both intra-session forward secrecy and intra-session future secrecy: even if an adversary learns K at some time, once a subsequent asymmetric rekey completes using fresh ephemeral secrets, the adversary cannot compute K_N without the DH private keys.

Symmetric Rekey. In the symmetric mode (Figure 11), the parties exchange fresh nonces N_C, N_P, deriving $K_N = KDF_K(N_C \mid N_P)$.

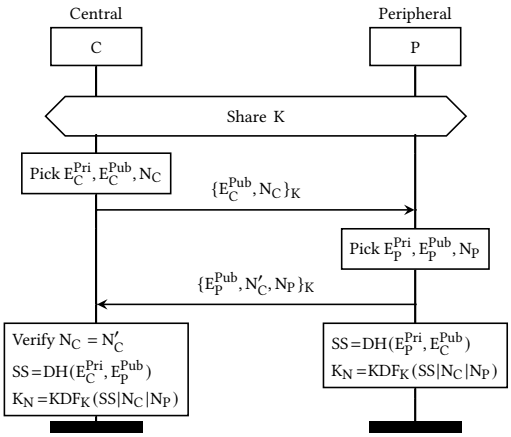


Figure 7: BB-Rekey asymmetric. The devices perform an ephemeral DH key exchange and use the shared secret to derive a new session key (K_N) within a session.

This mode provides intra-session forward secrecy in the sense that prior keys are unlinkable from K_N given only K_N; however, it does *not* provide future secrecy against an adversary that has compromised the current key K, as knowledge of K is sufficient to compute subsequent symmetric rekeys. Symmetric rekey is thus appropriate for routine key rotation and reduce key reuse, but not as a recovery mechanism after key compromise.

4.5 Security Properties

BlueBrothers fixes the 17 attacks identified in Section 3 by addressing their root causes (C1–C4).

Message Integrity (C1). All protocols maintain a handshake hash transcript (T) and compute MACs or digital signatures over it for each message, so any modification causes verification failure. This mechanism ensures the integrity of the entire protocol handshake.

Replay and Reflection Resistance (C2). All protocols incorporate fresh, execution-specific entropy into the hash transcript: from ephemeral DH keys in BB-Pairing, BB-Session, and asymmetric BB-Rekey and from random nonces in symmetric BB-Rekey. This mechanism prevents an adversary from reusing messages from previous sessions (replay) or reflecting a device's own message to itself (reflection) to complete a handshake.

Forward and Future Secrecy (C3). BB-Pairing and BB-Session provide inter-session forward secrecy via ephemeral DH. BB-Rekey updates encryption keys within a session. Its *asymmetric* mode provides intra-session forward and future secrecy under our threat model. Its *symmetric* mode provides intra-session forward secrecy but does not provide post-compromise recovery once the current key is compromised.

Entity Authentication and KCI Resistance (C4). BB-Pairing strengthen the association phase by using SAS with prefixed method identifiers, ensuring that an attentive user cannot confirm a pairing under an unintended association mechanism. BB-Session provides mutual entity authentication by having both parties sign the hash transcript using their static private keys via ECDSA. Unlike the current Bluetooth symmetric-key model, using signatures provides resistance to key compromise impersonation (KCI): compromising a device's static private key does not enable impersonation of another peer to that device. Furthermore, each execution of BB-Pairing uses a fresh static key pair, so compromising one pairing does not endanger others.

4.6 Comparison with State-of-the-Art Protocols

A Bluetooth-oriented Authenticated Key Exchange (AKE) must satisfy requirements that are atypical in other deployments such as Interned-based ones: (1) *first-contact bootstrapping* without pre-provisioned credentials (TOFU); (2) an explicit *user-assisted association* step that binds a human decision to the cryptographic transcript; (3) *negotiation safety* under heterogeneous device capabilities (preventing downgrade/method confusion); and (4) *tight message sizing and framing* under small MTUs and memory footprints. Existing protocols provide strong building blocks but do not address these Bluetooth-specific constraints.

Ephemeral Diffie-Hellman Over COSE (EDHOC). EDHOC is optimized for constrained IoT deployments with pre-shared credentials or keys. This fits scenarios where identities are configured out of band, but it does not address Bluetooth's TOFU pairing with user-assisted authentication. Using EDHOC would therefore require a separate, non-standard association and capability-negotiation mechanism, whereas BlueBrothers specifies these phases directly.

Noise Protocol Framework. Noise [34] provides a framework for DH-based protocols, but it intentionally leaves application-specific components unspecified. In Bluetooth, those components would include feature negotiation and user interaction. BlueBrothers can be viewed as a Bluetooth-specific instantiation: BB-Pairing specifies association and negotiation binding, BB-Session specifies mutually authenticated session establishment using stored static keys, and BB-Rekey specifies explicit rekeying operations.

TLS 1.3 and Double Ratchet. TLS 1.3 [27] targets IP-style transport and certificate-centric authentication, and its handshake size and framing are at least four times the typical BLE MTUs, increasing fragmentation and state complexity. The Double Ratchet [35] targets asynchronous messaging with per-message key refresh rather than explicit rekey operations within a connection. BB-Rekey instead provides explicit in-session rekeys that support long-lived Bluetooth links, with a lightweight nonce-based mode and an DH-based mode for post-compromise recovery.

4.7 Integration with Bluetooth

The BlueBrothers protocols are designed for practical deployment in the Bluetooth ecosystem and support two integration paths. The protocol can be deployed at the *stack layer* by updating the standard or at the *application layer* without modifying the standard.

Stack-layer Integration. The ideal deployment path is to incorporate BlueBrothers into the Bluetooth Core Specification. To achieve this, we propose adding a `BlueBrothers_Supported` bit to the Link-Layer (LL) Feature Set in BLE or the Extended Feature Page in BC. Devices would manage backward compatibility and security

enforcement through a local policy, analogous to the existing SC-Only mode. Specifically, a device configured with a BlueBrothers-Only policy would verify the peer's `BlueBrothers_Supported` bit during the initial connection phase. To ensure downgrade protection, devices must immediately terminate the connection if the remote peer does not indicate support for the new protocols. However, if devices are configured with a mixed-mode policy to maintain backward compatibility with legacy Bluetooth devices, they remain susceptible to downgrade attacks. This is an unavoidable trade-off as long as legacy devices are in use.While standardization poses a significant challenge, our stack-layer implementation on the nRF52840 (Section 6) demonstrates that replacing existing procedures with BlueBrothers is technically feasible even on highly constrained hardware. Moreover, because BlueBrothers retains the established TOFU model, it aligns with the current requirements of the Bluetooth ecosystem.

Application-layer integration. BlueBrothers can be deployed as application-layer protocols (e.g., over a dedicated data channel) without modifying the Bluetooth standard. This approach treats Bluetooth as an insecure transport, allowing vendors to provide seamless pairing experiences (e.g., Apple [25], Xiaomi [10, 11]) while avoiding the pitfalls of proprietary, "security-through-obscurity" designs.

The primary drawback of this integration path is ecosystem fragmentation. Without a centralized standard, vendors might implement BlueBrothers using divergent cryptographic primitives or handshake variations, complicating cross-vendor interoperability. To mitigate this risk, we could define a minimal mandatory-to-implement cipher suite to provide a common baseline for interoperable vendor implementations.

Despite fragmentation risks, application-layer integration remains attractive for two main reasons: first, it enables immediate adoption of advanced security features without relying on the SIG's update cycles. Second, vendors can pre-provision static keys or PSKs to enable authentication from the very first connection, bypassing the unauthenticated BB-JW association method.

5 BlueBrothers Formal Analysis

We formally verified BlueBrothers using ProVerif [7]. We provide *three separate* symbolic models, one per protocol (BB-Pairing, BB-Session, BB-Rekey), because each protocol has a distinct state machine, security goals, and compromise assumptions. BB-Pairing bootstraps trust and establishes long-term key material, BB-Session uses pre-established trust to run an authenticated key exchange, and BB-Rekey refreshes traffic keys within an existing session. We verify each protocol under explicit assumptions about what they receive from the others: authenticity of the static public keys for BB-Session, and a currently secure channel under K for BB-Rekey. We model all protocols for unbounded concurrent sessions using process replication (denoted by the ! operator in ProVerif syntax) and use the same notation as the MSCs in Section 4.

5.1 BB-Pairing Model

We define the BB-Pairing model with three processes: *Central*, *Peripheral*, and *User*. Central and Peripheral communicate with each other over a public channel and with the User process via private channels (*uvisc*, *uvisp*, *uinp*) that represent the physical I/O interface. We allow the attacker to control the negotiation messages on the public channel.

To formally evaluate resilience against method confusion attacks the *User* process includes an inattentive user ("confused behavior" branch). In this branch, the user blindly accepts mismatched association prompts, e.g., by reading a NC code displayed on the Peripheral and inputting it into a Central that is expecting a PE input.

Although practical user-assisted association is physically rate-limited by the human channel, we model BB-Pairing for unbounded concurrent sessions using process replication. This approach formally proves that the protocol is resilient to cross-session replay and confusion attacks, even during multiple simultaneous pairing attempts.

```
① not attacker_p1(SKtest[]) is true.
② event(PeripheralTerm(sk1,k,n1,n2)) && event(CentralTerm(sk2,k,
    n1,n2)) ==> sk1 = sk2 is true.
③ event(CentralTerm(sk,k,n1,n2)) && event(PeripheralTerm(sk,k,n1,
    n2)) ==> event(UserSaysOK(k)) is true.
④ inj-event(CentralTerm(sk,k,n1,n2,am1)) ==> inj-event(
    PeripheralAccepts(k,n1,n2,am2)) && am1 = am2 is true.
⑤ inj-event(PeripheralTerm(sk,k,n1,n2,am2)) ==> inj-event(
    CentralAccepts(k,n1,n2,am1)) && am1 = am2 is true.
```

Listing 1: BB-Pairing ProVerif results. All queries hold true under our attacker model (Section 4.1).

Listing 1 summarizes the verification results for BB-Pairing, confirming that the protocol satisfies all targeted security properties. Query ① proves confidentiality and forward secrecy by verifying that a secret test value (*SKtest*) encrypted with the session key remains inaccessible to the attacker even after a post-protocol static key compromise, which we model using phase transitions. Query ② confirms the integrity of the established channel by verifying the session key agreement using termination events. Queries ④ and ⑤ prove mutual entity authentication via injective correspondence. These queries ensure that both devices agree on the pairing parameters and the association method (am1 = am2), thereby confirming the protocol's resilience against method confusion attacks. Query ③ validates the user interaction by verifying that a *UserSaysOK* event is a strict prerequisite for any completed session, thereby enforcing explicit user consent.

5.2 BB-Session Model

The BB-Session model uses the same notation and cryptographic primitives of BB-Pairing and assumes peers possess trusted static public keys. Using analogous queries, ProVerif confirms that BB-Session guarantees *Confidentiality*, *inter-session forward secrecy*, *Session Key Agreement*, and *Mutual Entity Authentication*. Furthermore, by modeling dynamic key-leakage events, we explicitly verify the protocol's resistance to KCI.

5.3 BB-Rekey Model

We model BB-Rekey using separate models for forward secrecy and future secrecy. The rekey messages are protected using the current session key K. To model our *future secrecy* claim, we restrict

the adversary to be *passive during the rekey exchange* when K is compromised, consistently with our threat model in Section 4.1.

Forward and future secrecy. We model key compromise using ProVerif's *phase* construct. For forward secrecy, we show that a test value encrypted under a prior key (*attacker(K_old_test)*) remains secret after compromise of long-term keys or subsequent traffic keys. For future secrecy (PCS), we consider compromise of the current traffic key K and verify secrecy of a test value protected under the *post-rekey* key (*attacker(KNtest)*) under the assumption that the adversary is *passive during the rekey exchange*. ProVerif confirms future secrecy for the asymmetric rekey mode, whereas symmetric rekeying cannot recover once K is leaked.

Session key agreement. Analogous to query ②, this property is proven to hold for both modes, ensuring that honest parties derive identical session keys (K_N).

6 BlueBrothers Implementation

We implement BlueBrothers at the stack layer for BLE by modifying NimBLE [6], an open-source stack compliant with Bluetooth v5.4 and supporting embedded devices such as nRF51, nRF52, nRF53, and DA1469x. We also implement BlueBrothers at the application layer as a portable C library. This library uses WolfSSL [42] for its cryptographic primitives and, on Linux systems, interfaces with BlueZ [9] APIs. The application-layer implementation is transport-agnostic, i.e., works with BC and BLE.

6.1 Stack Layer Implementation

We integrate BlueBrothers into NimBLE by modifying its Security Manager (SM) in the Host and LL state machine in the Controller. Additionally, we introduced a new Host component, the Rekey Manager (Rekey Manager (RM)), to handle BB-Rekey.

Host Modifications. To support BB-Pairing and BB-Session, we repurpose the SMP Pairing Request/Response packets, expanding the payload to carry ECDH public keys and authentication values. Upon successful derivation, the Host pushes the session key K to the Controller via a custom HCI command. We implement the RM for BB-Rekey in a dedicated module (`ble_rm.c`). The RM communicates over a specific data channel with Channel ID 7 to exchange public keys (asymmetric) or nonces (symmetric). It exposes a Generic Access Profile (GAP) API to trigger key refreshes. Similar to the pairing phase, the new key K_N is pushed to the Controller using a dedicated HCI command.

Controller Modifications. In the Controller, we define four new LL control PDUs to synchronize key updates. Two empty unencrypted PDUs handle the initial session key agreement for BB-Pairing and BB-Session, while two authenticated PDUs manage the switch to new keys during BB-Rekey. These messages are not shown in Section 4, as they are specific to the BLE stack implementation.

Crypto Primitives. The implementation supports compile-time selection of X25519/Ed25519 and secp256r1 via build flags. To address performance bottlenecks on Cortex-M4 targets, we replace the default NimBLE secp256r1 backend with an architecture-optimized implementation. All cryptographic APIs, including transcript hashing, MACs and signatures, encryption, HOTP, and HKDF, are in `ble_sm_alg.c`.

6.2 Application Layer Implementation

We implement BlueBrothers as a standalone C library (bb-lib) to maximize portability. The library maintains protocol state within a dedicated `bbstate` structure. It exposes a platform-agnostic API (e.g., `bb_session_make_pkt`) that operates on raw byte buffers, ensuring consistent logic across different platforms.

Cryptographic primitives are managed through a modular abstraction layer (`crypto.c`), allowing a choice between a custom backend and WolfSSL. The current implementation uses X25519 for ECDH, ED25519 for signatures, ChaCha20-Poly1305 for authenticated encryption, and BLAKE2b for hashing and key derivation.

We developed a demonstrator for Linux systems that uses HCI User Channels. This approach bypasses the standard BlueZ settings and L2CAP restrictions, granting the application exclusive access to the Bluetooth controller. We implement a minimal user-space driver to inject raw data packets. L2CAP (Logical Link Control and Adaptation Protocol) is Bluetooth's data multiplexing and framing layer above the link layer, which carries multiple higher-level protocols over a single connection via per-channel identifiers (CIDs), enabling communication over a dedicated L2CAP channel (CID 81).

7 BlueBrothers Performance Evaluation

We evaluate the latency and energy consumption of our NimBLE implementation of BlueBrothers focusing on a worst-case scenario involving constrained BLE devices (nRF52).

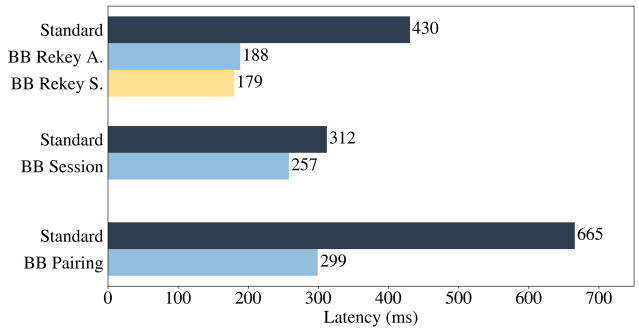
7.1 Experimental Setup

Our setup consists of two Nordic nRF52840 development boards running Mynewt v1.13.0 and a patched version of NimBLE v1.8.0 (as described in Section 6). We conducted the tests in a real-world environment characterized by active interference from other devices operating in the 2.4 GHz ISM band. The boards run two specialized NimBLE applications configured as Central and Peripheral. We control the Central via a Python script that interfaces with the board over UART, enabling automated execution of the protocols in loops. For the measurements, we use the BB-JW association method to eliminate the human-in-the-loop variable. This approach allows us to isolate and quantify the latency overhead inherent to the protocol logic and cryptographic primitives. To ensure a fair performance comparison, both the baseline Bluetooth implementations and BlueBrothers use the same elliptic curve (secp256r1) for ECDH and ECDSA operations where applicable.

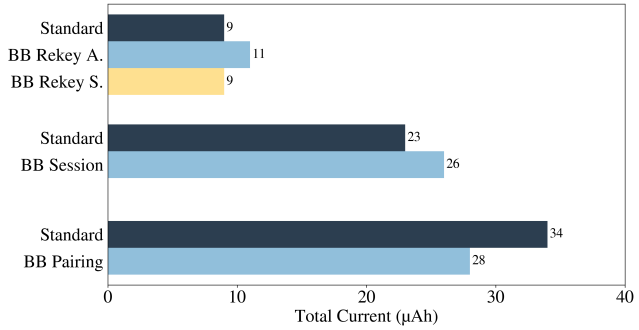
We measure protocol latency on the Central in microseconds (µs) using the Mynewt native time APIs. Simultaneously, we monitor the Peripheral's power draw using a Nordic Power Profiler Kit II. The profiler records instantaneous current samples in microampere (µA) at a sampling rate of 100 kHz. By aggregating these measurements over 100 sequential iterations for each protocol, we compute the resulting total energy consumption in milliampere-hours (mAh), providing a metric to assess the protocol's impact on battery life.

7.2 Latency and Energy Results

We evaluate the performance of BlueBrothers in terms of latency and energy consumption, comparing it against a baseline of the standard BLE SC protocols, specifically pairing, session establishment, and encryption refresh. To ensure a fair and consistent comparison,



(a) Latency (lower is better)



(b) Energy consumption (lower is better)

Figure 8: BlueBrothers Latency and Energy Consumption. The protocols are faster than the standard ones while having similar energy consumption.

we instantiate the protocols using secp256r1 for ECDH and ECDSA, as BLE SC uses this curve.

Latency. Latency results are shown in Figure 8a, representing the mean across 100 executions. Compared to the baseline, BB-Pairing reduces latency by 55.0% while BB-Session achieves a 17.6% reduction. Notably, BB-Rekey outperforms the standard encryption restart procedure in all configurations: latency is reduced by 58.4% in symmetric mode and 56.3% in asymmetric mode.

These results demonstrate that our protocols are more efficient than the standard ones, outperforming the baseline while providing superior security guarantees. We achieved this performance improvement through a streamlined design that minimizes the number of protocol round-trips and reduces state-machine complexity.

Energy. Energy consumption results are shown in Figure 8b. The plot shows the total energy consumption (in mAh) of 100 sequential executions for each protocol. BB-Pairing reduces energy consumption by 17.65% while BB-Session shows an increase of 13.04%. BB-Rekey symmetric rekeying energy consumption is similar to the baseline, while the asymmetric one is 22.22% higher. We argue that the BlueBrothers energy consumption overhead is acceptable even for real-world constrained devices, since these protocols are intended to run infrequently.

8 Bluetooth Related Work

Security extensions. Prior work proposes Bluetooth security extensions to harden both protocols. [24] presents a lightweight certificate-based alternative to BLE pairing with JW. While [20] mitigates attacks against SC pairing in a backward-compliant way using deferred challenge-response authentication. Other work extended legacy pairing with asymmetric cryptography [43]. In contrast, BlueBrothers redesigns pairing, session establishment, and rekeying without introducing unrealistic assumptions (e.g., a PKI).

Surveys and Privacy. Recent surveys [18, 33, 47] provide a comprehensive taxonomy of Bluetooth vulnerabilities, covering physical-layer, implementation-specific, and already-patched issues. Other works focus on privacy and device tracking risks [26, 46, 49]. We focus on the security of the connection (confidentiality, integrity, and authentication) and treat these as orthogonal.

Intrusion Detection System. Bluetooth Intrusion Detection System (IDS) systems detect reconnaissance, DoS, spoofing, and session attacks using approaches such as fingerprinting, controller-based monitoring, or state-machine analysis [12, 13, 32, 44]. These defenses monitor exploitation but do not remove the underlying protocol weaknesses; BlueBrothers instead replaces vulnerable protocol logic.

Testing and fuzzing. A large body of work targets implementation security via differential testing [29] and fuzzing [22, 23, 36] and has revealed numerous stack-specific issues. More recent toolkits also broaden coverage to protocol-level checks [51]. Our contribution is complementary: we propose replacement protocols that eliminate many design-level issues, irrespective of stack quality.

Formal verification. Multiple phases of Bluetooth security protocols have been analyzed using symbolic verifiers. Tamarin models uncovered vulnerabilities in BLE SC association and key agreement, including pairing confusion [14, 28, 39], while using ProVerif, researchers identified misbinding and other attacks across BC, BLE, and Mesh [38, 48]. Complementary computational work studied human-assisted association and TOFU-style security [19, 40]. We build on these analyses and use ProVerif to verify our replacement protocols.

9 Conclusion

This work presents BlueBrothers, three new Bluetooth security protocols designed to replace pairing and session establishment in both BC and BLE. Our design is motivated by persistent protocol-level vulnerabilities and the excessive complexity of the current Bluetooth security specification, both of which facilitate implementation flaws and hinder formal verification. BB-Pairing establishes a root of trust, distributing long-term public keys and providing an encrypted session. BB-Session uses those keys to perform authenticated key agreement with inter-session forward secrecy. BB-Rekey introduces a novel intra-session key refresh mechanism that enables forward and future secrecy.

The BlueBrothers protocols advance the state-of-the-art of Bluetooth security by addressing four critical design flaws (C1–C4) and attacks stemming from them (Table 1). Moreover, they have a simple, open design, and can be integrated at the stack or application layer, providing a practical path for real-world deployment.

We formally verified BlueBrothers using ProVerif [7], confirming session-key confidentiality, mutual authentication, and all claimed security properties. We integrated the protocols into NimBLE for stack-layer deployment and as a C library for application-layer use. Our evaluation on constrained nRF52 hardware shows that BlueBrothers consistently outperforms the standard protocols, with BB-Pairing reducing latency by 55% and energy consumption by 17.7%. BB-Session and BB-Rekey follow this trend, achieving latency reductions of 17.6% and 58.4%, respectively.

These results confirm that BlueBrothers provides a more secure alternative to the standard Bluetooth protocols, while being more performant and preserving similar battery life.

Acknowledgments

This work was funded by the European Union under grant agreement no. 101070008 (ORSHIN) and partially supported by the French National Research Agency (ANR) under the France 2030 label, with the REV (ANR-22-PECY-0009) and NF-HiSec (ANR-22-PEFT-0009) research grants. Views and opinions are, however, those of the authors only and do not necessarily reflect those of the European Union or ANR. We would also like to thank Slasti Mormanti for his consistent support and valuable input.

References

[1] Daniele Antonioli. 2023. BLUFFS: Bluetooth Forward and Future Secrecy Attacks and Defenses. In *ACM conference on Computer and Communications Security (CCS)*.

[2] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2020. BIAS: Bluetooth impersonation attacks. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 549–562.

[3] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2020. Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy. In *ACM Transactions on Privacy and Security (TOPS)*, Vol. 23. Association for Computing Machinery (ACM), New York, NY, USA, 1–28. doi:10.1145/3394497

[4] Daniele Antonioli, Nils Ole Tippenhauer, Kasper Rasmussen, and Mathias Payer. 2022. BLURtooth: Exploiting Cross-Transport Key Derivation in Bluetooth Classic and Bluetooth Low Energy. In *Proceedings of the Asia conference on computer and communications security (AsiaCCS)*.

[5] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B Rasmussen. 2019. The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR. In *28th USENIX Security Symposium (USENIX Security 19)*. 1047–1061.

[6] Apache. 2024. NimBLE is an open-source BLE 5.4 stack. https://github.com/apache/mynewt-nimble.

[7] Bruno Blanchet et al. 2016. Modeling and verifying security protocols with the applied pi calculus and ProVerif. *Foundations and Trends® in Privacy and Security* 1, 1-2 (2016), 1–135.

[8] Bluetooth SIG. 2025. Bluetooth Core Specification v6.2. https://www.bluetooth.com/specifications/specs/core-specification-6-2/.

[9] BlueZ developers. 2024. BlueZ: Official Linux Bluetooth protocol stack. https://www.bluez.org/.

[10] Marco Casagrande, Riccardo Cestaro, Eleonora Losiouk, Mauro Conti, and Daniele Antonioli. 2023. E-Spoofer: Attacking and Defending Xiaomi Electric Scooter Ecosystem. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*.

[11] Marco Casagrande, Eleonora Losiouk, Mauro Conti, Mathias Payer, and Daniele Antonioli. 2022. BreakMi: Reversing, Exploiting and Fixing Xiaomi Fitness Tracking Ecosystem. *IACR Transactions on Cryptographic Hardware and Embedded Systems (CHES)* (2022), 330–366.

[12] Romain Cayre, Vincent Nicomette, Guillaume Auriol, Mohamed Kaâniche, and Aurélien Francillon. 2024. OASIS: An Intrusion Detection System Embedded in Bluetooth Low Energy Controllers. In *Asia Conference on Computer and Communications Security (AsiaCCS)*.

[13] Xijia Che, Yi He, Xuewei Feng, Kun Sun, Ke Xu, and Qi Li. 2024. BlueSWAT: A Lightweight State-Aware Security Framework for Bluetooth Low Energy. In *Conference on Computer and Communications Security (CCS)*.

[14] Tristan Claverie, Gildas Avoine, Stéphanie Delaune, and José Lopes Esteves. 2023. Tamarin-based Analysis of Bluetooth Uncovers Two Practical Pairing Confusion Attacks. In *European Symposium on Research in Computer Security*. Springer, 100–119.

[15] Tristan Claverie and José Lopes Esteves. 2021. Bluemirror: reflections on Bluetooth pairing and provisioning protocols. In *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 339–351.

[16] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. 2016. On Post-compromise Security. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. 164–178. doi:10.1109/CSF.2016.19

[17] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.

[18] John Dunning. 2010. Taming the blue beast: A survey of Bluetooth based threats. *IEEE Security & Privacy* 8, 2 (2010), 20–27.

[19] Marc Fischlin and Olga Sanina. 2021. Cryptographic analysis of the Bluetooth secure connection protocol suite. In *Advances in Cryptology–ASIACRYPT 2021*. Springer, 696–725.

[20] Marc Fischlin and Olga Sanina. 2024. Fake It till You Make It: Enhancing Security of Bluetooth Secure Connections via Deferrable Authentication. In *ACM conference on Computer and Communications Security (CCS)*.

[21] Scott Fluhrer and Stefan Lucks. 2001. Analysis of the E0 encryption system. In *Selected Areas in Cryptography: 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16–17, 2001 Revised Papers 8*. Springer, 38–48.

[22] Matheus E Garbelini, Vaibhav Bedi, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. 2022. BrakTooth: Causing Havoc on Bluetooth Link Manager via Directed Fuzzing. In *31st USENIX Security Symposium (USENIX Security 22)*. 1025–1042.

[23] Matheus E. Garbelini, Chundong Wang, Sudipta Chattopadhyay, Sun Sumei, and Ernest Kurniawan. 2020. SweynTooth: Unleashing Mayhem over Bluetooth Low Energy. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 911–925.

[24] Chandranshu Gupta and Gaurav Varshney. 2023. An improved authentication scheme for BLE devices with no I/O capabilities. *Computer Communications* 200 (2023), 42–53.

[25] Dennis Heinze, Jiska Classen, and Felix Rohrbach. 2020. MagicPairing: Apple's take on securing Bluetooth peripherals. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 111–121.

[26] Jun Huang, Wahhab Albazrqaoe, and Guoliang Xing. 2014. BlueID: A practical system for Bluetooth device identification. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2849–2857.

[27] Internet Engineering Task Force (IETF). 2018. The Transport Layer Security (TLS) Protocol Version 1.3. https://www.rfc-editor.org/rfc/rfc8446.

[28] Mohit Kumar Jangid, Yue Zhang, and Zhiqiang Lin. 2023. Extrapolating Formal Analysis to Uncover Attacks in Bluetooth Passkey Entry Pairing. In *NDSS*.

[29] Imtiaz Karim, Abdullah Al Ishtiaq, Syed Rafiul Hussain, and Elisa Bertino. 2023. BLEDiff: Scalable and Property-Agnostic Noncompliance Checking for BLE Implementations. In *2023 IEEE Symposium on Security and Privacy (SP)*.

[30] David M'Raihi, Frank Hoornaert, David Naccache, Mihir Bellare, and Ohad Ranen. 2005. HOTP: An HMAC-Based One-Time Password Algorithm. RFC 4226. doi:10.17487/RFC4226

[31] Andy Nguyen. 2020. BleedingTooth: Linux Bluetooth Zero-Click Remote Code Execution. https://google.github.io/security-research/pocs/linux/bleedingtooth/writeup.

[32] Terrence OConnor and Douglas Reeves. 2008. Bluetooth network-based misuse detection. In *2008 Annual Computer Security Applications Conference (ACSAC)*. IEEE, 377–391.

[33] John Padgette, Karen Scarfone, and Lily Chen. 2022. Guide to Bluetooth Security NSP 800-121 Revision 2. *NIST special publication* 800, 121 (2022), 657–696.

[34] Trevor Perrin. 2018. The Noise Protocol Framework specification (Revision 34). https://noiseprotocol.org/noise.html.

[35] Trevor Perrin and Moxie Marlinspike. 2016. The double ratchet algorithm. *GitHub wiki* (2016).

[36] Jan Ruge, Jiska Classen, Francesco Gringoli, and Matthias Hollick. 2020. Frankenstein: Advanced wireless fuzzing to exploit new Bluetooth escalation targets. In *29th USENIX Security Symposium (USENIX Security 20)*. 19–36.

[37] Ben Seri and Gregory Vishnepolsky. 2017. The Attack Vector BlueBorne Exposes Almost Every Connected Device. https://armis.com/blueborne/.

[38] Mohit Sethi, Aleksi Peltonen, and Tuomas Aura. 2019. Misbinding attacks on secure device pairing and bootstrapping. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 453–464.

[39] Min Shi, Jing Chen, Kun He, Haoran Zhao, Meng Jia, and Ruiying Du. 2023. Formal analysis and Patching of BLE-SC pairing. In *32nd USENIX Security Symposium (USENIX Security 23)*. 37–52.

[40] Michael Troncoso and Britta Hale. 2021. The Bluetooth CYBORG: Analysis of the full human-machine passkey entry AKE protocol. In *Network and Distributed System Security Symposium (NDSS)*.

[41] Maximilian von Tschirschnitz, Ludwig Peuckert, Fabian Franzen, and Jens Grossklags. 2021. Method confusion attack on Bluetooth pairing. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1332–1347.

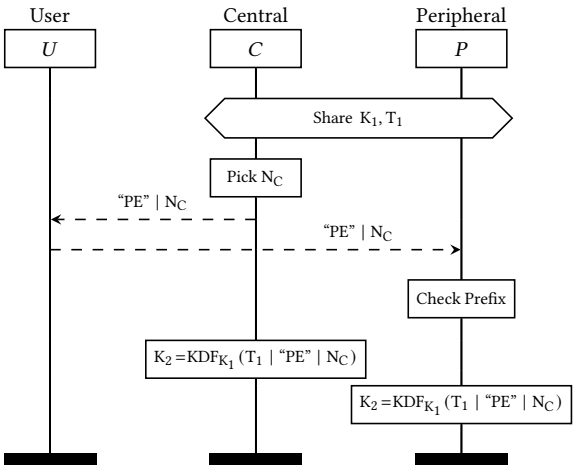


Figure 10: BB-PE input-output. One device generates a Passkey and displays it to the User, who inputs it on the other device.

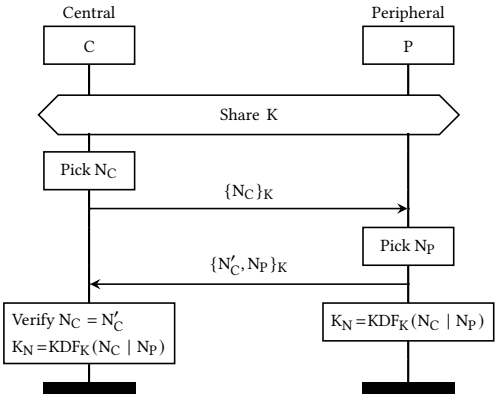


Figure 11: Symmetric BB-Rekey. The devices exchange nonces (N_C, N_P), and derive a new key (K_N) without aborting the session.

[42] wolfSSL Inc. 2025. wolfSSL Embedded SSL/TLS Library. https://www.wolfssl.com.

[43] Ford-Long Wong, Frank Stajano, and Jolyon Clulow. 2005. Repairing the Bluetooth pairing protocol. In *Proceedings of International Workshop on Security Protocols*. Springer, 31–45.

[44] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Mathias Payer, and Dongyan Xu. 2020. BlueShield: Detecting spoofing attacks in Bluetooth Low Energy networks. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. 397–411.

[45] Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave Jing Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. 2020. BLESA: Spoofing attacks against reconnections in bluetooth low energy. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*.

[46] Jianliang Wu, Patrick Traynor, Dongyan Xu, Dave (Jing) Tian, and Antonio Bianchi. 2024. Finding Traceability Attacks in the Bluetooth Low Energy Specification and Its Implementations. In *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA, 4499–4516.

[47] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave Tian, and Antonio Bianchi. 2024. SoK: The Long Journey of Exploiting and Defending the Legacy of King Harald Bluetooth. In *IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 23–23.

[48] Jianliang Wu, Ruoyu Wu, Dongyan Xu, Dave Jing Tian, and Antonio Bianchi. 2022. Formal model-driven discovery of Bluetooth protocol design vulnerabilities. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2285–2303.

[49] Yue Zhang and Zhiqiang Lin. 2022. When Good Becomes Evil: Tracking Bluetooth Low Energy Devices via Allowlist-based Side Channel and Its Countermeasure. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 3181–3194.

[50] Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. 2020. Breaking secure pairing of Bluetooth Low Energy using downgrade attacks. In *29th USENIX Security Symposium (USENIX Security 20)*. 37–54.

[51] Vladyslav Zubkov, Tommaso Sacchetti, Daniele Antonioli, and Martin Strohmeier. 2025. Bluetooth security testing with BlueToolkit: a large-scale automotive case study. In *WOOT 2025, 19th USENIX Conference on Offensive Technologies, co-located with the USENIX Security 2025, 11-12 August 2025, Seattle, WA, USA*, Usenix (Ed.). Seattle.

Appendix

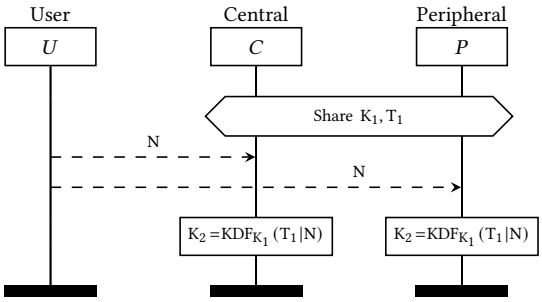


Figure 9: BB-PE input-only. The user inputs a Passkey on Central and Peripheral to authenticate the pairing.