The Bluetooth CYBORG: Analysis of the Full Human-Machine Passkey Entry AKE Protocol

Michael Troncoso*
Naval Postgraduate School
michael.troncoso@nps.edu

Britta Hale*
Naval Postgraduate School
britta.hale@nps.edu

Abstract

In this paper, we computationally analyze Passkey Entry in its entirety as a cryptographic authenticated key exchange (AKE) – including user-protocol interactions that are typically ignored as out-of-band. To achieve this, we model the user-to-device channels, as well as the typical device-to-device channel, and adversarial control scenarios in both cases. In particular, we separately capture adversarial control of device displays on the initiating and responding devices as well as adversarial control of user input mechanisms using what we call a *CYBORG* model. The CYBORG model enables realistic real-world security analysis in light of published attacks on user-mediated protocols such as Bluetooth that leverage malware and device displays. In light of this, we show that all versions of Passkey Entry fail to provide security in our model. Finally, we demonstrate how slight modifications to the protocol would allow it to achieve stronger security guarantees for all current variants of passkey generation, as well as a newly proposed twofold mode of generation we term *Dual Passkey Entry*. These proof-of-concept modifications point to improved design approaches for user-mediated protocols. Finally, this work points to categories of vulnerabilities, based on compromise type, that could be exploited in Bluetooth Passkey Entry.

Keywords: Bluetooth · Authenticated Key Exchange · CYBORG Protocols · Secure Connections · Secure Simple Pairing · Passkey Entry · Computational Analysis

1 Introduction

In traditional cryptographic protocols, user authentication is achieved via a trusted third party such as a certificate authority. However, in some settings, such as the Internet of Things (IoT), a reliable connection to such an authority cannot be guaranteed, and in some lightweight cases certificates cannot be used or reliably updated. Such protocols instead

*The views expressed in this document are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

rely on the human user to affirm the identities of communicating parties via an out-of-band (OOB) channel, usually via the user inputting a PIN code or password. Among these is the Bluetooth protocol [10]. Naturally, if we assume a perfectly secure OOB channel, analysis of the direct device-to-device communication is simplified, and consequently the approach was sensible in the earlier days of protocol design and analysis. Assuming that an adversary cannot effect such channels is no longer realistic, as demonstrated by recent attacks leveraging malware to target the user interface, as well as more established social engineering attacks. Notably the *Tap 'n Ghost* attack [33], which specifically targets Bluetooth devices, refutes the idea that user communications should be ignored in protocol analysis. As a result, it is critical to analyze the protocols in their entirety, accounting for both user interaction and device-to-device communication – namely, the *cyborg* protocol.

Leading authenticated key exchange (AKE) models do not capture user-to-device (UtD) attacks due to the sole focus on device-to-device (DtD) communications. However user-oriented attacks arise precisely from adversarial control of the UtD channel, or from a combination of control on UtD and DtD channels. While models capturing the UtD channel in the analysis of authentication protocols have emerged [18, 23], there has been a lack of such modeling for key exchange until now despite the standarized use of such protocols in practice. This is in stark contrast to the widespread implementation and standardization of AKE protocols employing the UtD channel [10, 25].

We model adversarial abilities against key exchange on what is normally the OOB channel. As noted before, Tap 'n Ghost is one attack which focuses heavily on vulnerabilities in the (previously assumed to be perfect) OOB channel. The attack is notable as it is a two-pronged assault on a user interface (without actively attacking device memory itself). The attacker first executes a Tag-based Adaptive Ploy, which forces a pop-up to display on a user's device, and then activates a Ghost Touch Generator, which spoofs touches on unwanted areas of the screen, to force pairing with a corrupted device. Thus, the attack requires adversarial ability to create messages to be sent to a user from a device, and adversarial ability to modify communications back from the user to the device (see Figure 1). We reference the Tap n' Ghost attack as an illustrative example throughout, but it is not the only attack leveraging the UtD communication channel. Touchloggers [12, 16], the StrandHogg vulnerability [24], social engineering, and shoulder-surfing attacks also fall into this category. All such attack vectors are systematically accounted for in our CYBORG model.

Bluetooth's Passkey Entry [10] generates and shares a random value (a *passkey*) via the user to effectively achieve entity authentication via the user mediation. Passkey Entry was primarily designed for pairing when at least one device has a keyboard but not a numerical display, such as a Bluetooth keyboard attempting to pair with a modern computer. In that case, a user could input a computer-displayed passkey into the keyboard for pairing. However, recently Passkey Entry has seen additional application when both devices have numerical keyboard and display capability, such as the manual pairing method between an Apple Watch and iPhone [3]. Bluetooth itself has enjoyed a long history of design advances [9, 36] and analysis [14, 15, 32, 41], as well as published attacks [1, 2, 8, 22, 31, 33]. The Secure Simple Pairing mechanism, first published in Bluetooth v2.1 and updated to Secure Connections in v4.2, allowed for flexibility by providing four methods of authenticated key exchange dependent on device input/output (IO) capabilities: Just Works, Numeric Comparison, Out-of-Band, and Passkey Entry. Of these, Numeric Comparison and Passkey Entry both rely

on the user playing an active role in an authenticated key exchange (AKE). While Numeric Comparison has been analyzed computationally [32, 41], Passkey Entry has until now not received a detailed analysis. This raises the question, *as a cyborg protocol, is Bluetooth Passkey Entry secure and, if so, under what conditions?*

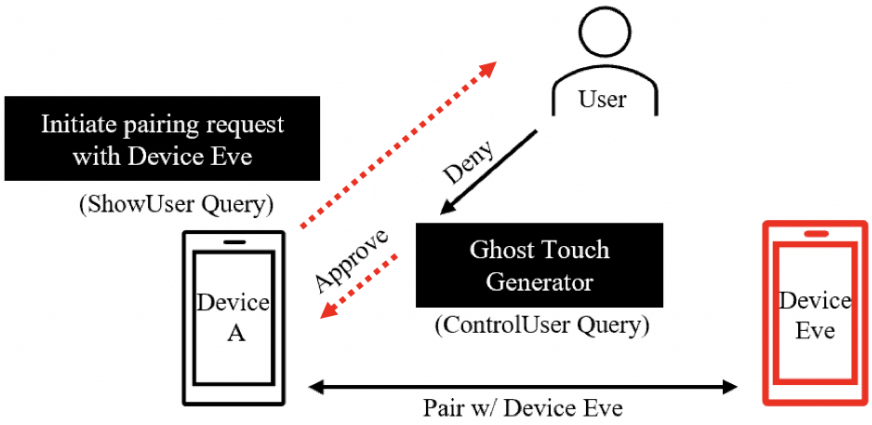

Figure 1: Tap 'n Ghost. Adversary uses a Tag-based Adaptive Ploy to display a message from device A to the user requesting to pair to a malicious device Eve. When the user denies this request, a Ghost Touch Generator corrupts the input and forces acceptance by device A. These actions map to a ShowUser query and ControlUser query, respectively, in the CYBORG model.

We address this question, presenting a computational analysis of Bluetooth Basic Rate / Extended Data Rate (BR/EDR) versions 2.1–5.2 Passkey Entry (henceforth abbeviated Passkey Entry). The analysis applies to Passkey Entry under either the Secure Simple Pairing or Secure Connections Bluetooth security framework, and is based on transcript matching as part of entity authentication. To achieve this, we first present a computational CYBORG AKE model, capturing adversaries capable of exploiting user-to-device transmissions. Our CYBORG model systematically covers all combinations of adversarial ability to create and modify messages from the device to the user and vice versa. As a result, our analysis not only demonstrates potential insecurities in Passkey Entry, but clarifies under which conditions it is secure.

1.1 Attacking User-to-Device Communications.

Social engineering and shoulder-surfing are well-accepted techniques for targeting user-device communication, but recent attacks on the UtD channel of cyborg-style protocols extend well beyond these and lead us to a baseline classification of attack vectors. For example, Touchloggers [12] attempt to mimic the role of keyloggers by logging a user's presses on a screen and then predicting the buttons that were pressed. The StrandHogg vulnerability [24] functions by disguising malware as legitimate apps for the user to interact with and unknowingly allow a hacker to compromise their device. Summarizing known attacks, we have the following four baseline attack vectors over the UtD channel: compromise of the communication channel from the initiating device to the user (type *iu*), the responding

device to the user (type ru), the user to the initiating device (type ui), and the user to the responding device (type ur). We illustrate these attacks in Fig. 2. We use these classifications to model adversarial capabilities by allowing for combinations of adversarial ShowUser and ControlUser queries in Section 3.

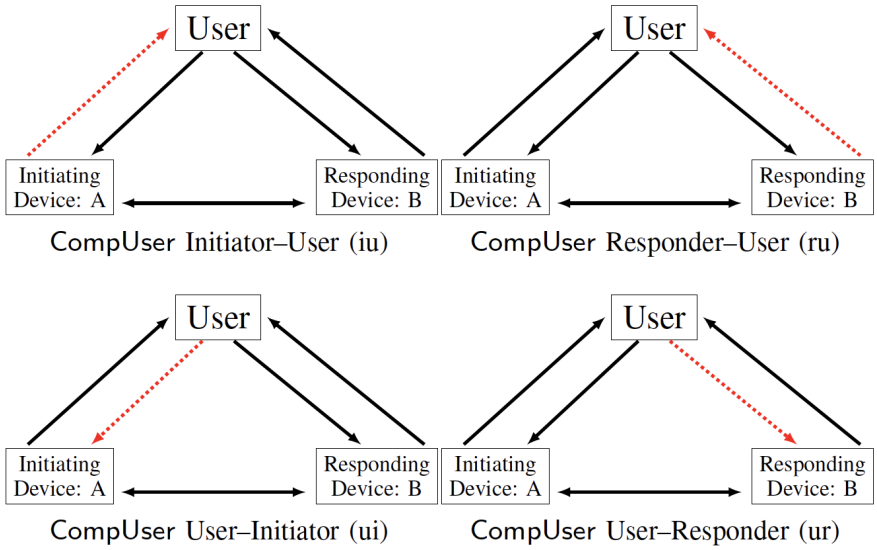

Figure 2: Visual depiction of the baseline four possibilities of user compromise (CompUser-freshness) where the adversary is allowed to corrupt only one direction of the user-to-device channel at a time, denoted by a red dashed line. We use the abbreviations $\mathsf{CompUser}_{[iu]}$, $\mathsf{CompUser}_{[ru]}$, $\mathsf{CompUser}_{[ui]}$, and $\mathsf{CompUser}_{[ur]}$ to describe the various attack scenarios.

For an example, consider how these combinations capture the Tap n' Ghost attack: This attack involves compromising communications between a single device and the user; both from the device to the user and from the user to the same device. Thus, we can model this attack as a simultaneous combination of either both a type iu and type ui compromise ($\mathsf{CompUser}_{[iu,ui]}$), if the initiating device is being targeted, or a type ru and type ur compromise ($\mathsf{CompUser}_{[ru,ur]}$), if the responding device is being targeted. Initiator and responder roles become important depending on the protocol and attack, such as in Initiator-Generated Passkey Entry where only the initiator displays a string to the user. In our model, this correlates to both a ShowUser query and a ControlUser query, as shown in Figure 1.

We allow for 16 different security frameworks (plus one to model the null case where the adversary gains no additional capabilities on the UtD channel) based on the possible combinations of the four baseline types $\{iu, ru, ui, ur\}$ in Fig. 2. All of these combinations are distinct due to the various forms that cyborg protocols may take. For example, Passkey Entry has three modes based on how the passkey is generated: by the initiating device, responding device, or the user. The effects of adversary action against a device display depends on the initiator/responder role of the device in question. Allowing for all possible combinations therefore enables analysis to not only adapt itself based on the protocol being investigated, but also pinpoint the exact circumstances under which a protocol fails to maintain security. This is especially useful information for those looking to assess where to devote security resources. If a protocol provides weak security against attacks afflicting the initiator device,

for example, but is resistant to attacks against a responder device, then developers and system administrators can plan and make design choices accordingly to limit vulnerability. Thus, analysis in our model can assist in limiting zero-day attacks against devices employing cyborg protocols.

1.2 Previous Work on Passkey Entry.

CYBORG Security Framework		*Bluetooth BR/EDR Secure Simple Pairing/Secure Connections Passkey Entry*						
		Original Protocol			**with Secure Hash Modification**			
		Init-Gen'd	*Resp-Gen'd*	*User-Gen'd*	*Init-Gen'd*	*Resp-Gen'd*	*User-Gen'd*	*Dual*
	UncUser	X	X	X	✓	✓	✓	✓*
	$\mathsf{CompUser}_{[iu]}$	X	X	X	X	✓*	✓*	✓*
	$\mathsf{CompUser}_{[ru]}$	X	X	X	✓*	X	✓*	✓*
	$\mathsf{CompUser}_{[ui]}$	X	X	X	✓*	X	X	✓*
	$\mathsf{CompUser}_{[ur]}$	X	X	X	X	✓*	X	✓*
	$\mathsf{CompUser}_{[iu,ru]}$	X	X	X	X	X	✓*	✓*
	$\mathsf{CompUser}_{[iu,ur]}$	X	X	X	X	✓*	X	✓*
	$\mathsf{CompUser}_{[ru,ui]}$	X	X	X	✓*	X	X	✓*
	$\mathsf{CompUser}_{[iu,ui]}$	X	X	X	X	X	X	✓*
	$\mathsf{CompUser}_{[ru,ur]}$	X	X	X	X	X	X	✓*
	$\mathsf{CompUser}_{[iu,ru,ui,ur]}$	X	X	X	X	X	X	✓

Figure 3: Table depicting levels of CYBORG security in the uncompromised (UncUser) and compromised user ($\mathsf{CompUser}_{[x]}$) settings achieved by the Passkey Entry protocol (Section 4), as well as Secure Hash Modification (SHM) Passkey Entry and Dual Passkey Entry (Section 6). Dual Passkey Entry is provably secure under all variants of the CYBORG security model (Section 3). All other versions of Passkey Entry were found to be insecure under the definitions not depicted. ✓ depicts proven secure in this work, ✓* depicts provably secure by implication, and X depicts insecure.

Passkey Entry is the primary version of Bluetooth authenticated key exchange employed when two devices want to pair and at least one does not have display capability [10]. Informally, it functions by having two devices exchange/receive a passkey for a user-to-device channel and proceed to commit to each bit of said passkey sequentially. This sequential construction, first proposed in [27], was devised to prevent offline dictionary attacks [39], such as those that plague legacy versions of Bluetooth authenticated key exchange.

Although Passkey Entry complicates an offline dictionary attack by committing to each bit of the passkey sequentially, it is more susceptible to other types of eavesdropping attacks [31]. Other investigations into the security of Passkey Entry have been largely ad-hoc [4, 31, 40], involving exploits in the re-use of random values across multiple executions of the protocol. A notable attack applicable to Passkey Entry outside of this construction is the *Fixed Coordinate Invalid Curve Attack* [8], which exploits devices not verifying the y-coordinate of a received Elliptic Curve Diffie-Hellman (ECDH) public key to insert an erroneous value and trivially compute the agreed upon DH key. Research has been conducted on the use of short, user-authenticated strings (i.e. a passkey no more than 20 bits long) for the authentication of a

Diffie-Hellman key [21,30,42], which demonstrated that security of such schemes is achievable in certain models and mainly dependent on the length of the passkey.

Other published attacks against Bluetooth pairing that use auxiliary mechanisms to compromise device communications include the *BT–Niño–MITM* attack [22], Bluetooth Impersonation AttackS (BIAS) [1], and Key Negotiation of Bluetooth (KNOB) [2]. *BT–Niño–MITM* exploits the un-authenticated exchange of IO capabilities to force a downgrade to Just Works for pairing, a protocol which Bluetooth acknowledges provides no protection against active adversaries [10]. Although *BT–Niño–MITM* was originally published in regards to Bluetooth BR/EDR v2.1-3, the attack remains current and recommendations to alleviate this vulnerability [22] include setting mandatory pairing methods for devices and displaying messages to the user confirming the pairing method desired. BIAS and KNOB both affect the Bluetooth BR/EDR v2.1-5.0 standardization and apply strictly to resumption after successful pairing.

The above attacks are relevant when discussing the holistic security of Bluetooth, yet these attacks do not address the underlying security achieved by Passkey Entry. While it is possible to require ephemeral passkeys in keeping with Bluetooth's recommendation, and verification of the y-coordinate of the ECDH public key, these changes alone do not provide guarantees of security, as a systematic analysis of Passkey Entry is lacking. Furthermore, the above mentioned analyses all treat user-to-device communications as secure OOB channels and disallow adversary actions on such channels – an assumption falsified by attacks like Tap 'n Ghost, Touchloggers, etc.

1.3 Modeling User-mediated Protocols.

Including user interaction in analysis saw early work in the symbolic setting, including an analysis of Bluetooth Numeric Comparison [15]. The user as a central component of the model was first introduced in the symbolic setting under the concept of *ceremonies* [20]. Ceremonies capture the intuition that there is no out-of-band communication, whether through the user, network, or other devices; all possible network communications, user to device interaction, displays, etc., can affect security. These ideas were later used for a second analysis of Numeric Comparison, which expanded adversarial capabilities over the user-to-device channel to include *Eavesdrop* and *Spoof* [14].

Using the user-to-device channel to exchange short, authenticated passkeys between devices saw security investigation under a computational setting by Peyrin and Vaudenay [37]. Although the user-to-device channel was considered as a secure OOB channel outside of adversarial attack abilities in that research, the user was included as an active participant with the ability to pick random numbers or compare values. Vaudenay [42] extended these modeling choices in a subsequent investigation of user-mediated protocols using passkeys where the adversary was allowed to delete, replay, or delay messages on the OOB channel, but not modify or create them. Thus, such modeling would not capture attacks such as Tap n' Ghost, as message modification is out-of-scope and channels. Notably, the user-device channels were also treated in a combined way, vs. allowing separate action on each device. Vaudenay's adversarial construction was also mirrored in Laur and Nyberg's security analysis of the MANA IV protocol, a precursor to Numeric Comparison [30]. We expand on these initial analyses by also giving the adversary the capability to modify and create user-to-device messages

dependent on the CYBORG security framework in use, as well as systematically considering separate adversarial action for the variants of the user-to-device channel.

Recent research in the computational setting [18, 23] incorporates user–device interaction in the analysis of authentication protocols. The 3-Party Possession User Mediated Authentication (3-PUMA) model, presented in [23], provided a computational model to capture communications sent over both the device-to-device channel and the user-to-device channels. That model was later adapted to allow for the adversarial queries ShowUser and ControlUser [18] and used to analyze entity authentication in Signal under the Mediated Epoch Three-Party Authentication (META) computational model. These queries capture adversarial ability to show erroneous information to the user (e.g. via malware on the device) as well as input erroneous information from the user (e.g. via modeling social engineering). These models form the foundation for our development of the CYBORG model.

While previous work focused on authentication protocols, we introduce a framework for the analysis of cyborg key exchange. Our key exchange framework is in keeping with other prominent standard AKE models [7, 13, 26, 29] where matching participant transcripts are required. We further add user modeling components from [18, 23]. Additionally, we extend earlier models for finer-grained analysis of the user-to/from-device channel security for individual devices and sessions, by separating out specific adversarial categories as shown in Figure 2.

Figure 3 summarizes our results, showing the security of the original Passkey Entry protocol in our model under all protocol variants (Initiator-Generated Passkey, Responder-Generated Passkey, and User-Generated Passkey) and all adversarial channel control variants (rows). Furthermore, the security of Bluetooth under our two proposed protocol modification variants is also shown (columns 4-7). While Dual Passkey Entry with Secure Hash Modification is the strongest protocol in our model, it also requires the most changes to the underlying protocol, and may not be feasible if both devices do not possess both input and output capabilities. Thus we demonstrate how even a minor change to the protocol can enable Passkey Entry to provide some security under a CYBORG adversary (columns 4-6).

1.4 Contributions

We build on research of user-mediated protocols with an analysis of Passkey Entry, and summarize our results in Fig. 3. Specifically,

- we provide the first security framework in computational analysis for cyborg AKEs supporting adversarial modification and creation of user-to-device messages, where the user is an active participant in protocol execution.
- we provide the first systematic separation of all variants of adversarial capabilities against user-to-device channels.
- we conduct the first computational analysis of Passkey Entry for Bluetooth BR/EDR, under either Secure Simple Pairing or Secure Connections. This analysis comprehensively covers all protocol variants, where the passkey is initiator-generated, responder-generated, or user-generated.

- we show that basic protocol modifications can improve the security of Passkey Entry in the CYBORG model, even to the point of full CYBORG security. This highlights the usability of the model, as well as the fine-grained clarity it provides to the conditions of protocol security.

The paper organization proceeds as follows: Section 2 introduces the Bluetooth Passkey Entry protocol per specification. The CYBORG model, including freshness conditions to capture various adversarial capabilities, is introduced in Section 3. In Section 4 we analyze all variants of Passkey Entry in the CYBORG model. Finally, in Section 5 we present two modifications of Passkey Entry, requiring different degrees of protocol changes and show, in Section 6 the corresponding security analyses of the modified variants.

2 Passkey Entry Protocol

In this section we present the Passkey Entry protocol adapted to fit within below mathematical definitions and a diagram for reference in Figure 4.

2.1 Relevant Variables

Variables used in Passkey Entry are described as follows:

- $A \in \{0,1\}^{48}$: The Bluetooth device address / identity of device of A.
- btlk: a fixed 16 bit ASCII string label for LK.
- $C_a \in \{0,1\}^{128}$: 128 bit tag from device A, the commitment value.
- DHKey $\in \{0,1\}^{256}$: Ephemeral DH key.
- $E_a \in \{0,1\}^{128}$: 128 bit tag from device A, the check value.
- IOcapA $\in \{0,1\}^{24}$: The IO capability of device A.
- LK $\in \{0,1\}^{128}$: A device's session key, the link key.
- HMAC: a hash-based message authentication code algorithm.
- $N_a \in \{0,1\}^{128}$: A nonce generated from device A.
- P: generator point for elliptic curve employed.
- $PK_a = (PK_{ax}, PK_{ay}) \in \{0,1\}^{256} \times \{0,1\}^{256}$: a bit representation of the ephemeral public key of device A for an elliptic curve DH key agreement. We write PK_{ax} for the x-coordinate and PK_{ay} for the y-coordinate.
- $r \in \{0,\ldots,9\}^6$: A 6-digit, decimal value, the passkey.
- $SK_a \in \{0,1\}^{255}$: Ephemeral secret key of Device A.

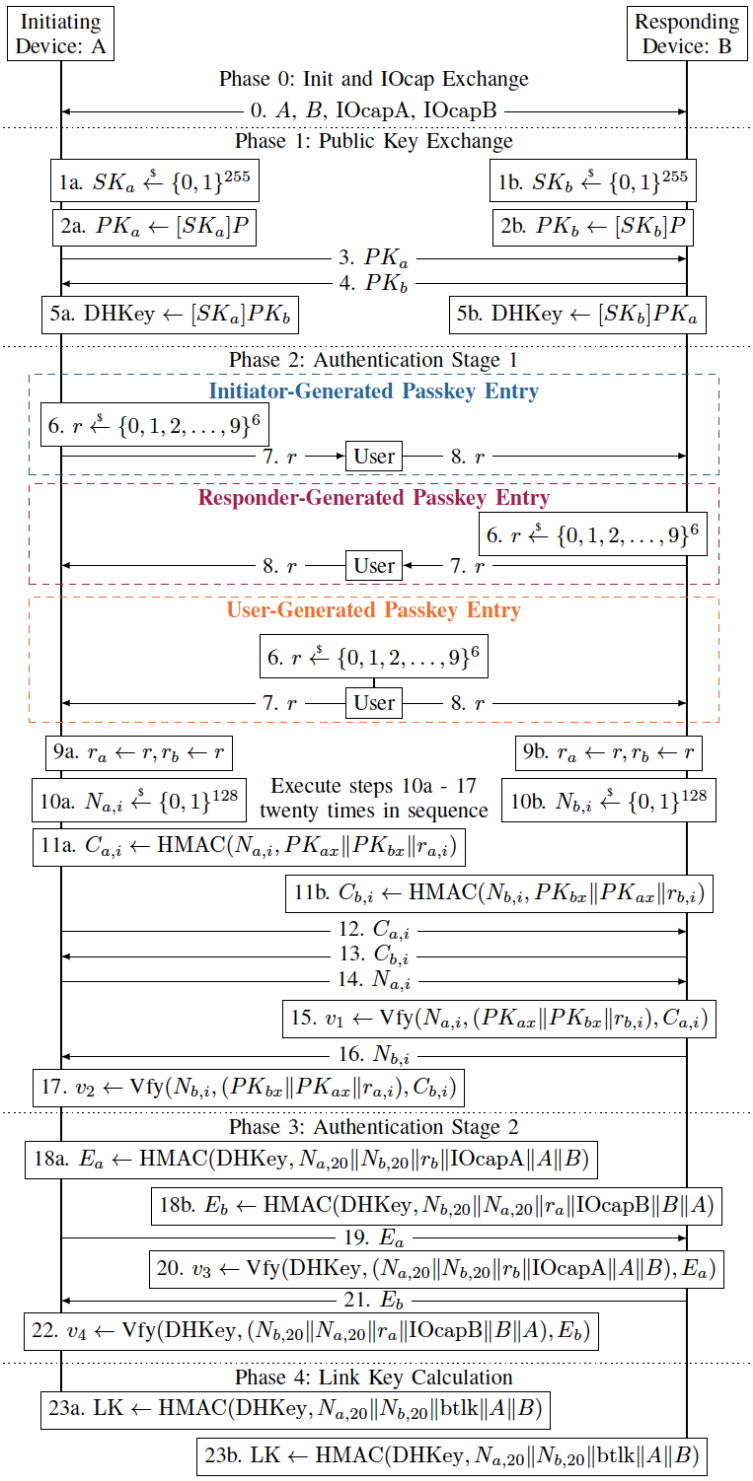

Figure 4: Passkey Entry. Phase 0 takes place before execution of the protocol; A, B, IOcapA, and IOcapB are distributed amongst the devices during this phase. Phase 2, steps 6-8, are version dependent as labeled (for initiator-generated passkey, responder-generated passkey, and user-generated passkey), all other steps are version independent.

Bluetooth's specification requires that N_a be sampled fresh for every execution of Passkey Entry, and recommends the same for SK_a, r. In this analysis we assume SK_a, N_a, and r are fresh values. We impose this restriction not only to simplify analysis but also because Passkey Entry with long-term DH keys allows for potential forward secrecy issues, as only long-term secrets would factor into session key generation, and long term passkeys allows for MitM attacks [31].

We state the actual bit lengths for the relevant variables as required under either BR/EDR Secure Simple Pairing or Secure Connections Passkey Entry. Secure Simple Pairing utilizes the P-192 elliptic curve while Secure Connections uses P-256 [34]. Both modes use SHA-256 [35] for the hash function and HMAC-SHA256 [28] for the MAC. Also note that DHKey is computed as the x-coordinate of $[SK_a]PK_b$ where $[\cdot]\cdot$ is used to symbolize elliptic curve point multiplication. We require PK_{ay} as part of the public key, by way of validation checks to prevent the attack described in [8].

IO capability not only determines the version of Bluetooth AKE (Numeric Comparison, Just Works, Out-of-Band, and Passkey Entry), but also, for Passkey Entry, how the passkey is generated (by the initiating device, the responding device, or the user). Passkey Entry *requires* a display output on no more than one device; in the case of a user-generated passkey, no device display is required. Thus, Passkey Entry can cover a wider variety of device pairings than other Bluetooth pairing modes, such as a wireless keyboard and a computer.

2.2 Protocol Phases

Phase 0: Init and IOcap Exchange.

This phase is not explicitly listed in the Bluetooth v5.2 specification and is not a phase of the Passkey Entry protocol as presented therein; however, it encompasses a variety of steps performed over an insecure channel before pairing commences including the sharing of data essential to the protocol. This includes device discovery for the sharing of identities (A and B), IO capabilities (IOcapA and IOcapB), and initialization of a connection.

Phase 1: Public Key Exchange.

The purpose of Phase 1 is to generate a shared key via an ECDH key agreement. To accomplish this, the public keys of devices A and B are exchanged in steps 2 and 3, and the shared key DHKey is calculated in steps 5a and 5b.

Phase 2: Authentication Stage 1.

Phase 2 handles authentication through the passkey and has three versions.

- Initiator-Generated Passkey Entry (PE-IG). The initiating device randomly samples the passkey r, transmits it to the user, and the user transmits it to the responding device.
- Responder-Generated Passkey Entry (PE-RG). The responding device randomly samples the passkey r, transmits it to the user, and the user transmits it to the initiating device.

- User-Generated Passkey Entry (PE-UG). The user randomly samples the passkey r then sends it to both the initiating and responding devices.

The devices then proceed to authenticate knowledge of r one bit at a time, for 20 bits of r in sequence. The HMAC Vfy algorithm outputs a verification bit, $v \in \{0, 1\}$. If $v_1 = 0$ or $v_2 = 0$, the protocol is aborted.

Phase 3: Authentication Stage 2.

This phase completes entity authentication, binding commitment of the user authenticated passkey to the Phase 1 knowledge shared key, DHKey.

Phase 4: Link Key Calculation.

The purpose of this phase is the calculation of the link key, LK, and completion of this phase concludes Passkey Entry.

3 CYBORG Security Model

In this section we present the CYBORG security model as a synthesis of past security models addressing both cyborg-type protocols [14, 18, 23] and AKEs in general [7, 13, 26, 29].

There are two main methods of identifying communicating sessions within a model: matching conversations (as introduced in [7]) and session identifiers. While matching conversations capture the concept of agreement over the entirety of the communication transcript, session identifiers usually only cover a partial transcript. Due to the usual lightweight nature of cyborg protocols (e.g. in that no certificates are not used) as well as the potential for lightweight cryptographic algorithms used on the subsequent channel (as appropriate for IoT use), detecting any adversarial interference is especially important. Thus, a variant of matching conversations is appropriate to this model.

However, this is problematic when investigating cyborg protocols, since messages sent over the UtD channel do not present an obvious method for inclusion in device transcripts. Therefore, we present a hybridization of matching conversations and session identifiers, termed *session identifiers with user*, a CYBORG analogue of matching conversations. We extract relevant messages sent over both UtD and DtD channels that match between devices.

We extend standard *device-device partnering* with *user-device partnering*. Both forms of partnering play a key role in the construction of the various *freshness* definitions (Definitions 3.8 and 3.10 to 3.11), which we leverage to describe the varying attack scenarios an adversary may mount.

3.1 Participant Model

3.1.1 Sessions

We define a *session* to be a single instance of a protocol and write π_s^A to refer to the s-th session for participant A where $A \in \mathcal{ID} \cup \{U\}$. Let $\mathcal{ID}$ be the set of all possible device identities and U be the identity of the user. We only allow one user identity in keeping with

reasons discussed in Section 3.1.3 and [23]. We set no limit on the number of sessions a single participant can have running at any one time with any other participant.

3.1.2 Devices

We utilize session oracles to capture the participation of a device $A \in \mathcal{ID}$ in a specific session, and describe the internal state of A as a tuple of the following values:

- $\mathsf{skey} \in \mathcal{K} \cup \{\perp\}$. This variable holds device A's completed session key for the session where $\mathcal{K}$ is the associated keyspace.
- $\mathsf{state} \in \{0,1\}^* \cup \{\perp\}$. This variable holds other secret state information.
- $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}, \perp\}$. This variable holds device A's role.
- $\mathsf{pid} \in \mathcal{ID} \setminus \{A\}$. This variable holds the identifier for the partner device.
- $\mathsf{sidu} \in \{0,1\}^* \cup \{\perp\}$. This variable holds the current session identifier.
- $\delta \in \{\mathsf{accept}, \mathsf{reject}, *\}$. This variable holds the specific result of the session either acceptance, rejection, or no decision, respectively.

At the creation of the s-th session for device A, the session oracle π_s^A is initiated to $(\mathsf{skey}, \mathsf{state}, \mathsf{role}, \mathsf{pid}, \mathsf{sidu}, \delta) = (\perp, \perp, \perp, \perp, \perp, *)$. For session acceptance, we require the following:

$$\delta = \mathsf{accept} \iff \mathsf{skey} \neq \perp \ .$$

3.1.3 Users

The user in the CYBORG protocol is assumed to be *honest*, whereby honest means that the user executes its function exactly as described by the protocol specification, and is modeled via session oracles, where each session oracle π_j^U maintains the following state:

- Two device-session pair identifiers $\mathsf{device}_1 = (A, s)$ and $\mathsf{device}_2 = (B, t)$, where $A, B \in \mathcal{ID}$ and $A \neq B$.

3.1.4 Partnering

Session identifiers with user follow the concept of matching conversations, albeit with transcripts requiring only information that both devices hold.

Definition 3.1 (Session Identifiers with User)**.** Let two session oracles, π_s^A and π_t^B, execute an authenticated key exchange protocol, Π, mediated by a user session oracle, π_j^U and let the following tuple of messages be the ordered transcript of all messages sent/received by π_s^A over the course of Π:

$$(msg_1, \ldots, msg_n) \ ,$$

where msg_k, is the k-th message sent/received in sequential order. Then we define the *session identifier*, denoted sidu, as the following subsequence of $(msg_1, \ldots, msg_n)$ pre-appended by an optional msg_0:
for $1 \leq k \leq n$, we append msg_k to $\pi_s^A.\mathsf{sidu}$ if any of the below criteria are met:

1. msg_k is sent by π_s^A to π_t^B, or
2. msg_k is received by π_s^A from π_t^B, or
3. msg_k is sent by π_s^A to π_j^U and π_j^U sends msg_k to π_t^B , or
4. msg_k is received by π_s^A from π_j^U and π_j^U received msg_k from π_t^B , or
5. msg_k is received by π_s^A from π_j^U and π_j^U sends msg_k to π_t^B .

Information exchanged prior to protocol execution for use within may optionally be pre-appended to sidu as a fixed msg_0.

Remark. The above definition follows closely to matching conversations, with two alterations: 1) we remove information sent between one device and the user which cannot be expected to be held by the other device, and 2) we allow additional information from prior to the protocol run. The second case captures pre-shared data that is common in user-mediated protocols, such as Input/Output capabilities (IOcap) in Bluetooth. While traditional network protocols such as TLS share capabilities during the protocol run, such as available ciphersuites, such mutually held data may be defined externally to the protocol in the lightweight user-mediated setting.

Definition 3.2. We say that a session identifier $\pi_s^A.\mathsf{sidu}$ with length $l \geq 1$ is a *prefix* of $\pi_t^B.\mathsf{sidu}$ if all values in $\pi_s^A.\mathsf{sidu}$ match and are in the same order as the first l messages in $\pi_t^B.\mathsf{sidu}$.

Definition 3.3 (Matching Sessions). We say that a device, A, at session s has *matching session identifiers* with device B at session t if

- $\pi_s^A.\mathsf{sidu} = \pi_t^B.\mathsf{sidu}$ where A receives the last message(s), or
- $\pi_t^B.\mathsf{sidu}$ is a prefix of $\pi_s^A.\mathsf{sidu}$ where A sends the last message(s).

This is a variant of the standard asymmetric definition for (prefix) matching session identifiers because we must account for the instance where a device sends the last message and accepts, but the adversary deletes this message en route to its intended recipient.

Definition 3.4 (Device-Device Partnering). We say sessions π_s^A and π_t^B are *partnered* if $\pi_s^A.\mathsf{pid} = B$, $\pi_t^B.\mathsf{pid} = A$, $\pi_s^A.\mathsf{role} \neq \pi_t^B.\mathsf{role}$, and A and B have matching session identifiers.

Definition 3.5 (User-Device Partnering). If $\pi_j^U.\mathsf{device}_1 = (A, s)$ or $\pi_j^U.\mathsf{device}_2 = (A, s)$, then we say that π_j^U and π_s^A are *partnered*.

User sessions are opaque to device sessions. Consequently, if the user is partnered with two device sessions, we assume that the device sessions always send messages to the correct partnered user session. We also require that a device session cannot be partnered with more than one user session, as can be expected in normal user interactions during device pairing. In the case of devices executing concurrent sessions, the user can still distinguish sessions since we assume that ephemeral passkeys will be used.

3.2 Adversarial Model

3.2.1 Communication Channels.

In this section we look to explicitly define the capabilities of an adversary in regards to messages sent over the two communication channels used: device-to-device (DtD) and user-to-device (UtD).

Definition 3.6 (Device-to-Device Channel). An adversary may read, replay, delete, or modify any message sent between device oracles π_s^A, π_t^B and we call this the *DtD channel*.

Definition 3.7 (User-to-Device Channel, Without Eavesdropping). An adversary may replay or delete any message sent between a device oracle π_s^A and user oracle π_j^U, but may not modify, to include the intended destination, create, nor read any message. We call this the *UtD channel (UtD)*.

Our definition for the DtD channel allows the adversary full control over communications sent between devices. However, we do restrict the adversary's capability to affect communications over the UtD channel. We disallow read, modify, and create capabilities under normal operation, and restrict these to cases of device or user compromise, modeled via adversarial access or queries. Ultimately the difference inherent to how users interact with devices over the UtD channel necessitates stronger attack capabilities not privy to the typical active attacker when compared to the DtD channel. Lastly, we note that allowing eavesdropping over the UtD channel is often protocol dependent as some protocols require secrecy on that channel (e.g. in Passkey Entry or ATM pin codes), while others do not (e.g. Bluetooth Numeric Comparison).

3.2.2 Adversarial Queries.

We now present a list of queries for the adversary to use when interacting with Passkey Entry participants.

- $\mathsf{SendDevice}(\pi_s^A, msg)$. The adversary can use this query to send a message msg to the given session oracle. The session oracle will then act on msg as the protocol specifies and any response will be returned to the adversary. If $msg = (\mathsf{start}, B)$ for $B \in \mathcal{ID}$, a non-protocol specific special initiation message, is the first message the given session oracle has received then it will set $\mathsf{role} = \mathsf{initiator}$ and $\mathsf{pid} = B$ and output the first protocol message. This allows the adversary to initiate a protocol run between two identities. Else, if the first message a device session oracle receives does not consist of the non-protocol specific special initiation message and comes from $B \in \mathcal{ID}$, then the oracle sets $\mathsf{pid} = B$, $\mathsf{role} = \mathsf{responder}$, and executes the protocol as intended in the role of the responder.

- $\mathsf{SendUser}(\pi_j^U, msg)$. The adversary can use this query to send a message msg to the given user session oracle π_j^U. The session oracle will then act on msg as the protocol specifies and any response will be returned to the adversary. If $msg = (\mathsf{start}, (\pi_s^A, \pi_t^B))$ for $A, B \in \mathcal{ID}$, a non-protocol specific special initiation message, is the first message the given session oracle has received, then U first checks if π_s^A or π_t^B were ever part

of a received $msg = (\mathsf{start}, \cdot)$ message for any session $\pi^U_{j'}$. If so, the session outputs $\perp$. Else, the session sets $\mathsf{device}_1 = (A, s)$ and $\mathsf{device}_2 = (B, t)$. Else, if the first message received by π^U_j does not consist of such a start message, the session oracle outputs $\perp$. The session oracle will then execute the protocol in the role of the user as intended with the given device identities.

- $\mathsf{StateReveal}(\pi^A_s)$. The adversary may use this query to obtain access to the session state information state.
- $\mathsf{KeyReveal}(\pi^A_s)$. The adversary may use this query to obtain access to the session key skey.
- $\mathsf{ShowUser}(\pi^A_s)$. This query outputs $\perp$. After this query, the adversary can modify or create any UtD message sent from the given device to the user within π^A_s.
- $\mathsf{ControlUser}(\pi^U_j, A)$. This query outputs $\perp$. After this query, the adversary can modify or create any UtD message sent from the user U to the device A of the current session.
- $\mathsf{Test}(\pi^A_s)$. This query may only be asked once throughout the game. If $\pi^A_s.\delta \neq \mathsf{accept}$, this query returns $\perp$. Else, it samples $\mathsf{b} \xleftarrow{\$} \{0,1\}$, and sets $k \leftarrow \pi^A_s.\mathsf{skey}$ if $\mathsf{b} = 1$ and $k \xleftarrow{\$} \{0,1\}^\lambda$ otherwise. The query outputs k_{b}.

These queries were developed to model compromise of a device or a user in keeping with modeling real-world attacks. The SendDevice, StateReveal, Corrupt, KeyReveal and Test queries are typical variants used to model the adversary's ability to control the experiment and compromise a device's internals via side-channel or malware. SendUser is needed to allow the adversary to initialize user session oracles and also to give him the ability to send messages to the user as desired. The two queries, ShowUser and ControlUser, are included to give the adversary the ability to compromise the UtD channel. ShowUser models the adversary gaining control over a device and inducing it to send messages to the user, via malware-induced pop-ups for example. Meanwhile the ControlUser query gives the adversary the ability to affect messages input to the device from the user, such as from a Ghost Touch Generator or social engineering. The ShowUser and ControlUser queries are also restricted in effect to specific sessions. This is done to capture the idea that adversary compromise of the UtD channel can have a time-sensitive nature.

3.2.3 User and Device Freshness.

Typically, freshness has only been defined with respect to devices. This made sense when only devices engaged in the action of the protocol, but with the advent of cyborg protocols it becomes necessary to define freshness for device and user session oracles. The uncompromised setting enables us to test the baseline security of the protocol, which restricts the adversary's use of both device and user-oriented queries. In the compromised user setting, the adversary gains varying abilities to issue combinations of ShowUser and ControlUser queries. The freshness types ($[iu], [ru], [ui]$ and $[ur]$ are depicted visually in Figure 2). We define a corresponding device freshness that is used to assess key indistinguishability in the security experiment, wherein the adversary may not reveal device secrets; the adversary has much more liberty to reveal device secrets when attempting to break authentication.

Definition 3.8 (Device Freshness). We say that a device session oracle π_s^A is *fresh in the uncompromised setting (*$\mathsf{UncUser}$*-fresh)* and *fresh under compromised user type* $[x]$ *(*$\mathsf{CompUser}_{[x]}$*-fresh)* unless any of the following hold:

- the adversary issues a $\mathsf{StateReveal}(\pi_s^A)$ query, or
- if there exists a session oracle π_t^B partnered with π_s^A and the adversary issues a $\mathsf{StateReveal}(\pi_t^B)$ query, or
- the adversary has issued a $\mathsf{KeyReveal}(\pi_s^A)$ query, or
- if there exists a session oracle, π_t^B, partnered with π_s^A and the adversary issues a $\mathsf{KeyReveal}(\pi_t^B)$ query.

The freshness conditions whereby we restrict adversarial reveals on both the target device session and a partnered device session is in keeping with precedent. The restriction of adversarial queries on a device partnered with the same user oracle is analogous to restricting corruption of the intended partner's long-term keys. Cyborg key exchange protocols rely on the user to authenticate devices in the same vein as a Certificate Authority authenticates devices in typical AKE; thus, we rely on the user's authentication of the intended partner to limit reveals.

Definition 3.9 (User Freshness under No Compromise). We say that a session oracle π_j^U is *fresh in the uncompromised setting (*$\mathsf{UncUser}$*-fresh)* unless any of the following hold:

- the adversary has issued a $\mathsf{ShowUser}(\pi_s^A)$ query before the last UtD message is sent and received between π_j^U and π_s^A according to the protocol, where π_j^U is partnered with π_s^A, or
- the adversary has issued a $\mathsf{ControlUser}(\pi_j^U, A)$ query before the last UtD message is sent and received between π_j^U and π_s^A according to the protocol, where π_j^U is partnered with π_s^A.

We now focus on the freshness scenario under user compromise, which replaces the traditional view of the user as a perfect OOB channel (i.e. an uncompromised user).

Definition 3.10 (User Freshness under Compromise Type *iu* and *ru*). We say that a session oracle π_j^U for U and session j is *fresh under compromised user, type iu (resp. type ru), denoted* $\mathsf{CompUser}_{[iu]}$*-fresh (resp.* $\mathsf{CompUser}_{[ru]}$*-fresh)* unless

- the adversary has issued a $\mathsf{ShowUser}(\pi_s^A)$ query before the last UtD message is sent and received between π_j^U and π_s^A according to the protocol, where π_j^U is partnered with π_s^A, and
  - (if $\mathsf{CompUser}_{[iu]}$:) $\pi_s^A.\mathsf{role} = \mathsf{responder}$
  - (if $\mathsf{CompUser}_{[ru]}$:) $\pi_s^A.\mathsf{role} = \mathsf{initiator}$

  or

- the adversary has issued a $\mathsf{ControlUser}(\pi_j^U, A)$ query before the last UtD message is sent and received between π_j^U and π_s^A according to the protocol, where π_j^U is partnered with π_s^A.

Definition 3.11 (User Freshness under Compromise Type *ui* and *ur*). We say that a session oracle π_j^U for U and session j is *fresh under compromised user, type ui (resp. type ur), denoted* $\mathsf{CompUser}_{[ui]}$*-fresh (resp.* $\mathsf{CompUser}_{[ur]}$*-fresh)* unless

- the adversary has issued a $\mathsf{ShowUser}(\pi_s^A)$ query before the last UtD message is sent and received between π_j^U and π_s^A according to the protocol, where π_j^U is partnered with π_s^A, or
- the adversary has issued a $\mathsf{ControlUser}(\pi_j^U, A)$ query before the last UtD message is sent and received between π_j^U and π_s^A according to the protocol, where π_j^U is partnered with π_s^A, and
  - (if $\mathsf{CompUser}_{[ui]}$:) $\pi_s^A.\mathsf{role} = \mathsf{responder}$
  - (if $\mathsf{CompUser}_{[ur]}$:) $\pi_s^A.\mathsf{role} = \mathsf{initiator}$

We restrict the adversary from making ShowUser and ControlUser queries dependent on the role of the device session, such that only one UtD channel may be compromised in each type (denoted in red/dashes in Figure 2). This decision allows us to focus on the exact circumstances under which a protocol breaks.

Definition 3.12 (User Freshness under Compromise Type Combinations). Let X be a non-empty subset of the set $\{iu, ru, ui, ur\}$; we refer to the set X with the label string x constructed from the elements of X in sequential order. We then say that a session oracle π_j^U for U and session j is termed *fresh under compromised user, type x (*$\mathsf{CompUser}_{[x]}$*-fresh)* unless the adversary issues a single query which breaks $\mathsf{CompUser}_{[x_i]}$-freshness simultaneously for all $x_i \in X$.

Note that a session oracle's freshness must be assessed *per* query under $\mathsf{CompUser}_{[x]}$*-fresh*; said query must break freshness for the session oracle in all the elements of the set X individually. This structuring of $\mathsf{CompUser}_{[x]}$*-fresh* is needed so we can capture more advanced attacks that an adversary may mount. Using only select definitions from among Definitions 3.10 to 3.11 we would be unable to capture the "Tap 'n Ghost" attack as it involves the corruption of the communication channels both from a device to the user and vice versa. However, we can model this using the $\mathsf{CompUser}_{[iu,ui]}$*-fresh*. In a $\mathsf{CompUser}_{[iu,ui]}$*-fresh* environment, the adversary would be allowed to issue both a ShowUser and a ControlUser query so long as both involved the initiating device; and similarly in a $\mathsf{CompUser}_{[ru,ur]}$*-fresh* environment for the responding device. We list out all the types of $\mathsf{CompUser}_{[x]}$*-fresh* that are possible under Definitions 3.10 to 3.12:

$$[iu], [ru], [ui], [ur], [iu, ru], [iu, ui], [iu, ur], [ru, ui], [ru, ur], [ui, ur],$$
$$[iu, ru, ui], [iu, ru, ur], [iu, ui, ur], [ru, ui, ur], [iu, ru, ui, ur] \ .$$

3.3 CYBORG Security Experiment

In light of all previous definitions, we now present the CYBORG security experiment for the reader.

Definition 3.13. We define the CYBORG-type *security experiment* for a PPT adversarial algorithm $\mathcal{A}$ against a cyborg key exchange protocol Π, and interacting with a challenger via all previously defined adversarial queries in the $\mathsf{EXP}^{\text{CYBORG-type}}_{\Pi,\mathcal{A},\eta_p,\eta_s}$ experiment, where η_p is the maximum number of device participants and η_s is the maximum number of sessions for any participant. We say that the adversary $\mathcal{A}$ wins the experiment with the challenger outputting 1 if any of the following conditions hold for $\mathsf{type} \in \{\mathsf{UncUser}, \mathsf{CompUser}\}$.

1. *Correctness (*correct*)*:
   there exists two type-*fresh* and partnered device oracles π^A_s and π^B_t where:
   - π^A_s and π^B_t are both partnered with the type-*fresh* user oracle π^U_j, and
   - $\pi^A_s.\delta \neq \mathsf{accept}$, or
   - $\pi^B_t.\delta \neq \mathsf{accept}$.
2. *Entity Authentication (*auth*)*:
   there exists a type-*fresh* session oracle π^A_s where:
   - $\pi^A_s.\delta = \mathsf{accept}$ with intended partner $\mathsf{pid} = B$ and
   - π^A_s is partnered with the type-*fresh* user oracle π^U_j, and
   - if π^U_j is also partnered with π^B_t, $\mathcal{A}$ has not issued a $\mathsf{StateReveal}(\pi^B_t)$ query while $\pi^A_s.\delta \neq \mathsf{accept}$, and
   - there does not exist a unique session oracle at B that is partnered with π^A_s.
3. *Key Indistinguishability (*key-ind*)*:
   at some point in the experiment $\mathcal{A}$ issued a $\mathsf{Test}(\pi^A_s)$ query on a type-*fresh* session oracle π^A_s , where
   - $\pi^A_s.\delta = \mathsf{accept}$ with intended partner $\mathsf{pid} = B$, and
   - π^A_s is partnered with a type-*fresh* user oracle π^U_j, and
   - if π^U_j is also partnered with π^B_t, $\mathcal{A}$ has not issued a $\mathsf{StateReveal}(\pi^B_t)$ query while $\pi^A_s.\delta \neq \mathsf{accept}$, and
   - there exists an oracle $\pi^B_{t'}$ partnered with π^A_s, and
   - at some subsequent point in the experiment π^A_s responds with its guess b, where $\Pr[b = \mathsf{b}] \geq 1/2$ where b is the randomly sampled bit from the associated Test query.

Else, the challenger outputs 0. We denote the adversary $\mathcal{A}$ winning the experiment and the challenger outputting 1 as:

$$\mathsf{EXP}^{\text{CYBORG-type}}_{\Pi,\mathcal{A},\eta_p,\eta_s}(\lambda) = 1 \ .$$

We define the advantage for the PPT adversarial algorithm $\mathcal{A}$ in the above experiment to be:

$$\mathrm{Adv}^{\text{CYBORG-type}}_{\Pi,\mathcal{A},\eta_p,\eta_s}(\lambda) := \Pr[\mathsf{EXP}^{\text{CYBORG-type}}_{\Pi,\mathcal{A},\eta_p,\eta_s}(\lambda) = 1] \ .$$

Definition 3.14. If there exists a negligible function $\mathrm{negl}(\lambda)$ such that for all PPT adversaries $\mathcal{A}$ interacting according to the CYBORG-type experiment, it holds that:

$$\mathrm{Adv}^{\text{CYBORG-type}}_{\Pi,\mathcal{A},\eta_p,\eta_s}(\lambda) \leq \mathrm{negl}(\lambda) \ ,$$

then we say that the protocol Π is CYBORG-type-secure.

The above definition presents a change from META in that we require two partnered oracles to imply acceptance, as captured by correct. This is a reasonable assumption when acceptance is not conditioned by a concluding user input.

4 Analysis of Passkey Entry

In this section we present the initial results of our analysis of Passkey Entry under the CYBORG security model and show that all versions of Passkey Entry fail to meet any version of CYBORG security. For Inititiator/Responder-Generated Passkey Entry, this is because the first 19 generated nonces are not authenticated in Phase 3, which leads to an adversarial forgery based off a single bit guess and consequently failure of session matching. Meanwhile, User-Generated Passkey Entry fails to achieve CYBORG security because role agreement is not guaranteed through protocol execution.

We define the session-state information for all versions of Passkey Entry and a given session oracle π^A_s as:

$$\pi^A_s.\mathsf{state} = (SK_a, r, N_{a,1}, \ldots, N_{a,20}) \ ,$$

where all values are as defined in Section 2. We capture all randomly generated information a device considers secret in the session-state. Although nonces are made public in the course of the protocol, they are included here as they are used as secret keys before such disclosure.

Since all versions of Passkey Entry are susceptible to passkey re-use attacks [31], the passkey r must be generated as an ephemeral secret. This presents a unique modeling challenge. If generated by a device, the passkey may be derived from the same source of randomness as other ephemeral keys or nonces. If generated by a user however, this is not the case. In this scenario, it becomes necessary to operate under the assumption that the user has some means of random number generation either via a keyfob or other source.

First we define the session identifier for all versions of Passkey Entry from a correctly executed session involving an initiating device, e.g. A, and responding device, e.g. B, as described in Section 3.1.4:

$$\begin{aligned}\mathsf{sidu} := ((A, B, \mathrm{IOcapA}, \mathrm{IOcapB}), PK_a, PK_b, r, C_{a,1}, \ldots \\ \ldots, C_{b,1}, N_{a,1}, N_{b,1}, C_{a,20}, C_{b,20}, N_{a,20}, N_{b,20}, E_a, E_b).\end{aligned}$$

Note that identities are shown as an example above, and A may be either an initiating device or responding device in the following analysis, specified where appropriate.

Theorem 4.1. *Initiator-Generated and Responder-Generated Passkey Entry are* ***not*** CYBORG-UncUser-secure.

Proof. We prove Theorem 4.1 via a counter-example by describing how an adversary can entice a device to accept maliciously in the Initiator-Generated Passkey Entry case. The case of Responder-Generated Passkey Entry follows similarly.

Let $\mathcal{A}$ be an adversarial algorithm against the CYBORG-UncUser security of Initiator-Generated Passkey Entry. The adversary first issues a $\mathsf{SendDevice}(\pi_s^A, (\mathsf{start}, B))$ to initiate a protocol run between devices A and B, and a $\mathsf{SendUser}(\pi_j^U, (\mathsf{start}, (\pi_s^A, \pi_t^B)))$ query to initiate the associated user oracle. The adversary then allows the protocol to progress through step 8. At this point, $\mathcal{A}$ then makes a guess $r_{e,i}$ for the bit value of $r_{a,i}$ used by the initiating device to construct $C_{a,i}$ where $1 \leq i \leq 19$. $\mathcal{A}$ then randomly samples a new nonce $N_{e,i}$ and calculates a new tag $C_{e,i} = \mathrm{HMAC}(N_{e,i}, PK_{ax} \| PK_{bx} \| r_{e,i})$. When device A attempts to send $C_{a,i}$ to device B in step 12, the adversary replaces this value with $C_{e,i}$, and similarly replaces $N_{a,i}$ with $N_{e,i}$ in step 14. If we have that $r_{a,i} = r_{e,i}$, then device B's verification in step 15 will succeed; $\mathcal{A}$ will then take no further actions and allow the protocol to proceed to completion. This will lead to both π_s^A and π_t^B accepting but they will not have matching session identifiers since they will disagree on the values for $C_{a,i}$ and $N_{a,i}$ and $\mathcal{A}$ succeeds in breaking auth. Since $\mathcal{A}$'s guess of $r_{a,i}$ is correct with probability one-half, we have that

$$\mathrm{Adv}^{\text{CYBORG-UncUser}}_{\text{PE-IG},\mathcal{A},\eta_p,\eta_s}(\lambda) \geq \frac{1}{2} \ .$$

□

In User-Generated Passkey Entry an adversary can mount a "role confusion" attack on the pairing devices (illustrated in Figure 5 and described in the proof of Theorem 4.2 below). This attack leads to both devices accepting although neither actually pairs with the other. Similar attacks have been previously shown on other protocols [5].

Theorem 4.2. *User-Generated Passkey Entry is* ***not*** CYBORG-UncUser-secure.

Proof. Let $\mathcal{A}$ be an adversarial algorithm against the CYBORG-UncUser security of User-Generated Passkey Entry. $\mathcal{A}$ first issues a $\mathsf{SendUser}(\pi_j^U, (\mathsf{start}, (\pi_s^A, \pi_t^B)))$ to initiate the user for a protocol run between devices A and B. $\mathcal{A}$ then issues both a $\mathsf{SendDevice}(\pi_s^A, (\mathsf{start}, B))$ and a $\mathsf{SendDevice}(\pi_t^B, (\mathsf{start}, A))$ query. $\mathcal{A}$ will then function as an intermediary between the session oracles π_s^A and π_t^B, which both run the protocol in the role of the initiating device. After both device A and B exchange their public keys through the adversary, they will be ready to accept the passkey from the user. The user inputs a passkey into both devices according to the protocol (note that the device session role is opaque to the user). $\mathcal{A}$ then simply relays all relevant protocol messages between device A and device B in keeping with the description of User-Generated Passkey Entry. At the conclusion of the above attack, we have that $\mathcal{A}$ has won the CYBORG-UncUser security experiment by breaking auth, since π_s^A and π_t^B have both accepted but there does not exist a paired UncUser-fresh session oracle for either device, due to role disagreement. □

In summary, all versions of Passkey Entry do not meet any level of CYBORG security. Even though Passkey Entry was originally developed to prevent offline dictionary attacks by

using 20 commitments (one commitment for each bit of the passkey), it allows for breaks in auth. Note that the proof in Theorem 4.1 can also be used as a counter-example for User-Generated Passkey Entry insecurity.

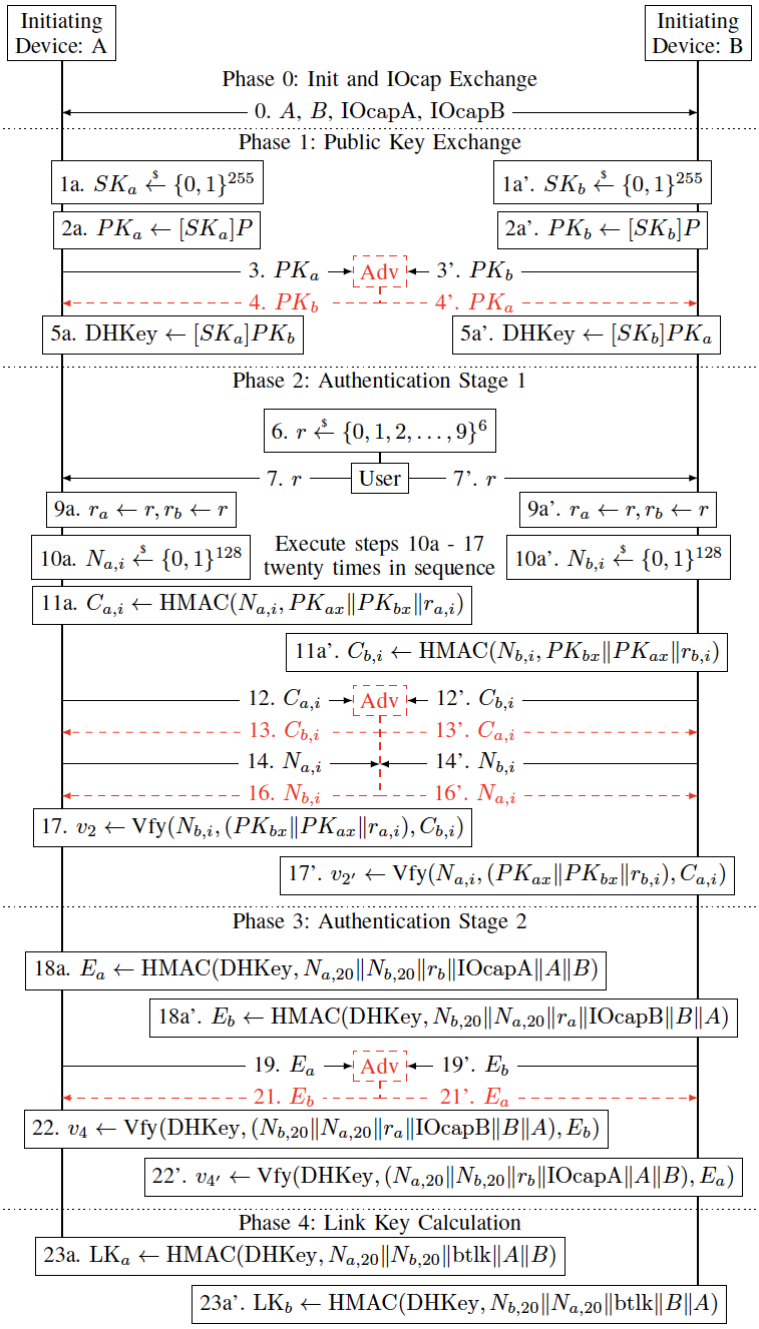

Figure 5: Role confusion attack on User-Generated Passkey Entry that results in two devices, both in the initiator role, accepting. Adversary shown by dashes.

5 Modified Passkey Entry

In this section we present two modified Passkey Entry protocol variants (see Figure 6). With these modifications, Passkey Entry can achieve complete CYBORG security (see analysis in Section 6). We demonstrate these modifications as a proof-of-concept that CYBORG security is achievable with interaction from both devices and full transcript validation.

Modifications were made with the goal of introducing minimal change to the protocol, as well as restricting most changes to Phase 2 in keeping with the requirements for modular construction of Bluetooth protocols.

- **Secure Hash Modification (SHM)**: This modification requires the addition of a collision-resistant hash function, H, with a 128-bit output length (in keeping with the specified nonce length).
  - *Concatenate and hash all of a device's previously generated nonces to form N_a and N_b (steps 8c and 8d). N_a and N_b replace $N_{a,20}$ and $N_{b,20}$ in all further Phases.* This ensures computation of the check values E_a and E_b rely on all generated nonces. It also prevents the bit guessing attack described in the proof of Theorem 4.1.
  - *Include device role in the computation of N_a and N_b.* Each device declares their role and the assumed role of their partner in the hash computation over N_a, N_b using the labels init and resp for the initiator and responder respectively. This protects against the role confusion attack described in the proof of Theorem 4.2.
  - *Include both* IOcap *variables in the computation of E_a and E_b.* This ensures authentication of both IOcap capabilities.
- **Dual Passkey Entry (DPE)**: This modifcation requires the initiator and responder to both possess a numerical display and numerical input capabilities.
  - *Each device generates a passkey and shares this value via the user.* The initiator relays its passkey r_a to the responder through the user, followed similarly by the responder with r_b. This prevents display output and/or input compromise.

Dual Passkey Entry's reliance on device's possessing both numerical displays and keypad entry is more in line with the requirements for Numeric Comparison, which requires both devices to have both numerical displays and binary inputs. Traditional Passkey Entry in comparison requires a display and entry mechanism on respective, instead of both, devices (for Initiator Generated and Responder Generated Passkey Entry), or only requires entry mechanisms on both (for User Generated Passkey Entry). This begs the question of whether simply using Numeric Comparison would be a viable alternative vice a new protocol. Although we do not devote a full analysis to Numeric Comparison under the CYBORG model due to space constraints, observation suggests that Numeric Comparison would not achieve full CYBORG security. In particular, Numeric Comparison requires identical and predictable binary user inputs on both devices (to confirm whether two displayed values match [10]). Under $\mathsf{CompUser}_{[ui]}$ and/or $\mathsf{CompUser}_{[ur]}$ freshness definitions the adversary could modify the user confirmation/denial message to allow a MitM attack to advance regardless of matching in the actual values displayed.

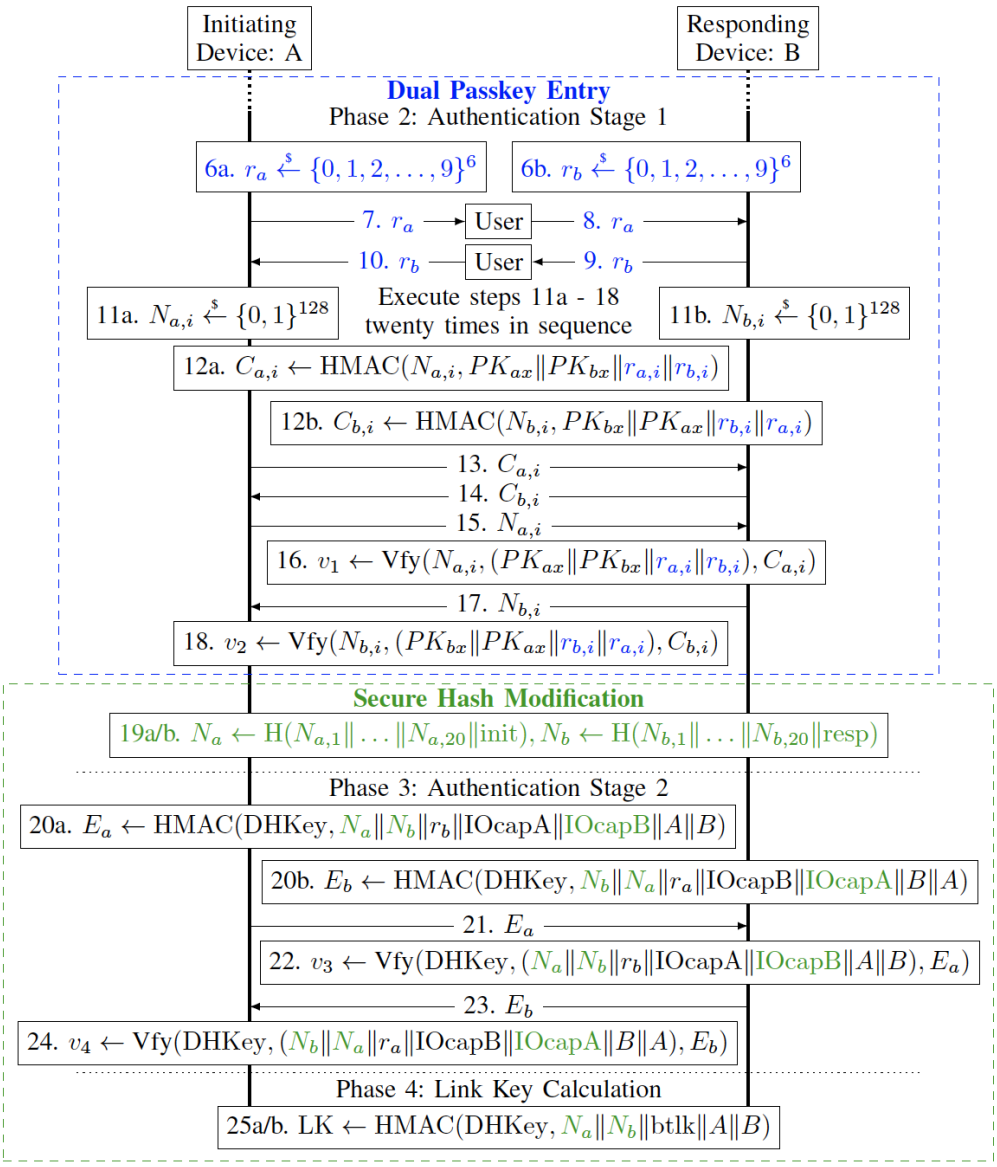

Figure 6: Dual Passkey Entry with Secure Hash Modification. Dual Passkey Entry (DPE) is depicted with the modifications in blue, where both devices generate passkeys.The Secure Hash Modification (SHM) is depicted green and can be used in conjunction with DPE or any version of Passkey Entry. Under this modification all nonces are hashed into the new values N_a, N_b for subsequent use. Devices also declare assumed roles, using the strings init and resp, and both IOcap variables are used in the computation of E_a, E_b.

The SHM falls very closely to techniques used in other protocols and brings Passkey Entry more in line with accepted techniques for protecting against downgrades. For example, TLS 1.3 [38] computes the `Finished` message over exchanged ciphersuites, thereby ensuring that both parties agree to exchanged capabilities. However, while this change adds authenticity against transcript modification during Phase 2 of the protocol, it does not prevent a more direct downgrade; namely, if the IOcap was changed to Just Works, the Passkey Entry

protocol would not be completed at all. In terms of overhead, the SHM notably adds only one hash computation per party. In comparison, Passkey Entry already requires 21 HMAC computations and 21 HMAC verifications per party.

In contrast, DPE demonstrates a straightforward analogue to certificate usage. Passkey Entry is uni-directional in the user receipt and relay of the passkey; one can think of it as similar to a server-only certificate. DPE in comparison is bi-directional, analogous to certificates on both sides. Some cases of lightweight devices without suitable I/O capabilities exist where DPE may not be possible. However, for such cases where mutual authentication is unachievable, the results of Section 4 and shown in Fig. 3 provide particular guidance. Namely, by demonstrating which compromise scenarios are fatal to the protocol in Fig. 3, manufacturers and end users gain insight on the device that is most in need of protection (i.e. dependent on the device that is generating a passkey) and under what conditions.

6 Analysis of the Modified Protocol

The following analysis covers all four versions of Passkey Entry (Initiator-Generated, Responder-Generated, User-Generated, and Dual) with the Secure Hash Modification (signified by the "SHM" prefix) under both security environments defined in the CYBORG model: uncompromised user (UncUser) and compromised user (CompUser). For CompUser, each version of the SHM Passkey protocol is analyzed under each of our four baseline definitions of compromised user scenarios (Definitions 3.10 to 3.11). We operate under the definitions for session-state and the sidu described in Section 4.

Initiator-Generated Passkey

We start with results for the SHM Initiator-Generated Passkey Entry protocol as follows, covering all variants under the CYBORG security model.

Theorem 6.1. *SHM Initiator-Generated Passkey Entry is*

- CYBORG-UncUser-secure *under the* EC-sym-ssPRF-ODH *and* EC-DDH *assumptions, the* sec-pre *of* H*, and the* SUF-CMA *security of* HMAC*.*
- CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$[x] \in \{[ru], [ui], [ru, ui]\} \ .$$

- ***not*** CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$\begin{aligned}[x] \in \{&[iu], [ur], [iu, ru], [iu, ui], [iu, ur], [ru, ur], [ui, ur], \\ &[iu, ru, ui], [iu, ru, ur], [iu, ui, ur], [ru, ui, ur], \\ &[iu, ru, ui, ur]\} \ .\end{aligned}$$

The proofs for the above theorems can be found in Appendix C.1. We provide a proof sketch for the CYBORG-UncUser setting here, which serves as the basis for other proof

variants in the positive results (for negative results, counter-examples are also shown in Appendix C.1).

Proof Sketch. The proof of this theorem involves a series of game hops between an adversarial PPT algorithm $\mathcal{A}$ and the challenger. We denote the adversarial advantage of a specific game as Adv_i, for the i-th game hop.

Game 0. This game is equivalent to the original security experiment. Thus we have $\mathrm{Adv}_0 = \mathrm{Adv}^{\text{CYBORG-UncUser}}_{\text{PE-IG},\mathcal{A},\eta_p,\eta_s}(\lambda)$.

Game 1. In this game we abort if two session oracles ever generate the same ephemeral DH key, SK, of length μ. Thus we have $\mathrm{Adv}_1 \geq \mathrm{Adv}_0 - (\eta_p\eta_s)^2 \cdot 2^{-\mu}$.

Game 2. In this game we abort if we ever have nonce collision, and there are 20 nonces generated each session. Thus we have $\mathrm{Adv}_2 \geq \mathrm{Adv}_1 - 400 \cdot (\eta_p\eta_s)^2 \cdot 2^{-\lambda}$.

Game 3. In this game we abort if a passkey, r, is ever re-used. Thus we have $\mathrm{Adv}_3 \geq \mathrm{Adv}_2 - (\eta_p\eta_s)^2 \cdot 2^{-|r|}$.

Game 4. In this game we the guess the target session oracle, π^A_s, and its partner, π^B_t, uniformly at random, and abort if $\mathcal{A}$ does not attempt to win against this guessed pair. Thus we have $\mathrm{Adv}_4 \geq (\eta_p\eta_s)^{-2} \cdot \mathrm{Adv}_3$.

We then continue by game-hopping dependent on whether $\mathcal{A}$ attempts to win by breaking correct, auth, or key-ind. Thus we have $\mathrm{Adv}_4 = \mathrm{Adv}^{\mathsf{correct}}_4 + \mathrm{Adv}^{\mathsf{auth}}_4 + \mathrm{Adv}^{\mathsf{key\text{-}ind}}_4$.

Advantage against correct**.** This step is straightforward.

Advantage against auth**.**

Game 5. In this game we abort if $\mathcal{A}$ succeeds in forging the DH public keys PK_a, PK_b. To forge public keys, $\mathcal{A}$ must guess all 20 bits of the passkey r. Accounting for π^A_s in the initiator and responder roles, we have $\mathrm{Adv}^{\mathsf{auth}}_5 \geq \mathrm{Adv}^{\mathsf{auth}}_4 - 2^{-(|r|-1)}$.

Game 6. In this game we replace $\text{DHKey} = [SK_a]PK_b = [SK_b]PK_a$ with the uniformly random value $\widetilde{\text{DHKey}}$. By the EC-DDH assumption, $\mathcal{A}$ is unable to distinguish this change. Thus, we have $\mathrm{Adv}^{\mathsf{auth}}_6 \geq \mathrm{Adv}^{\mathsf{auth}}_5 - \mathrm{Adv}^{\mathsf{EC\text{-}DDH}}_{\mathcal{B}_1}(\lambda)$.

We continue the proof by first separating two sub-cases based on the test session's role: initiator (C_1) or responder (C_2). Thus we have $\mathrm{Adv}^{\mathsf{auth}}_6 = \mathrm{Adv}^{\mathsf{auth},C_1}_6 + \mathrm{Adv}^{\mathsf{auth},C_2}_6$.

Case 1: π^A_s.role = initiator.

Game 7. In this game we abort if $\mathcal{A}$ succeeds in forging $E_b = \text{HMAC}(\widetilde{\text{DHKey}}, N_b\|N_a\|r_a\|\text{IOcapB}\|B\|A)$, N_b, B, or IOcapB, and thereby getting A to accept maliciously. We bound this ability by the SUF-CMA security of HMAC. Thus we have $\mathrm{Adv}^{\mathsf{auth},C_1}_7 \geq \mathrm{Adv}^{\mathsf{auth},C_1}_6 - \mathrm{Adv}^{\mathsf{SUF\text{-}CMA}}_{\text{HMAC},\mathcal{B}_2}(\lambda)$.

Game 8. In this game we abort if $\mathcal{A}$ succeeds in forging any of B's nonces or B's role, causing A to accept maliciously. From Game 7 we have that $\mathcal{A}$ does not succeed in forging N_b. This lets us bound this ability by the sec-pre security of H. Thus we have $\mathrm{Adv}^{\mathsf{auth},C_1}_8 \geq \mathrm{Adv}^{\mathsf{auth},C_1}_7 - \mathrm{Adv}^{\mathsf{sec\text{-}pre}}_{\text{H},\mathcal{B}_3}(\lambda)$.

By Game 8 we have matching sidu ($\mathcal{A}$ cannot forge $C_{a,i}, C_{b,i}$ due to the correctness of HMAC), π^A_s.pid $= B$, π^B_t.pid $= A$, π^A_s.role $\neq \pi^B_t$.role. Thus via Definition 3.4 our session oracles are partnered and we have $\mathrm{Adv}^{\mathsf{auth},C_1}_8 = 0$.

Case 2: π^A_s.role = responder. This case follows as in Case 1.

Advantage against key-ind**.**

Game 5. In this game we replace $LK = \text{HMAC}(\text{DHKey}, N_a\|N_b\|\text{btlk}\|A\|B)$ (if π^A_s.role = initiator) with the uniformly random value $\widetilde{LK}$. By the hardness of the EC-sym-ssPRF-ODH

assumption, $\mathcal{A}$ is unable to distinguish this change. Thus we have $\mathrm{Adv}_5^{\mathsf{key\text{-}ind}} \geq \mathrm{Adv}_4^{\mathsf{key\text{-}ind}} - \mathrm{Adv}_{\mathrm{HMAC},\mathcal{B}_4}^{\mathsf{EC\text{-}sym\text{-}ssPRF\text{-}ODH}}(\lambda)$, accounting for π_s^A in either the initiator or responder role.

Since the session key of our test oracle is now uniformly random, we conclude $\mathrm{Adv}_5^{\mathsf{key\text{-}ind}} = 0$.

Combining all game hops and adversarial cases, we have

$$\mathrm{Adv}_{\text{PE-IG},\mathcal{A},\eta_p,\eta_s}^{\text{CYBORG}-\mathsf{UncUser}}(\lambda) \leq (\eta_p\eta_s)^2 \cdot \Big(\frac{1}{2^\mu} + \frac{400}{2^\lambda} + \frac{3}{2^{|r|}} + \mathrm{Adv}_{\mathcal{B}_1}^{\mathsf{EC\text{-}DDH}}(\lambda) + 2 \cdot (\mathrm{Adv}_{\mathrm{HMAC},\mathcal{B}_2}^{\mathsf{SUF\text{-}CMA}}(\lambda) + \mathrm{Adv}_{\mathrm{H},\mathcal{B}_3}^{\mathsf{sec\text{-}pre}}(\lambda) + \mathrm{Adv}_{\mathrm{HMAC},\mathcal{B}_4}^{\mathsf{EC\text{-}sym\text{-}ssPRF\text{-}ODH}}(\lambda))\Big) .$$

□

The above proof demonstrates the CYBORG security of Initiator-Generated Passkey Entry in the UncUser setting, conditioned on the security of the underlying algorithms as well as nonce length, passkey length, etc. Still, per Passkey Entry specification, the passkey has length $|r| = 20$. Consequently, this result demonstrates that the actual security level is below the simple guessing ability for 20 bits.

Since a EC-sym-ssPRF-ODH is a PRF-based security notion (detailed in Appendix A) and as such may not appear as a typical MAC security assumption, we comment here on related work regarding the PRF security of HMAC [28]. HMAC was shown to be a PRF so long as its underlying hash function was a PRF [6]. It has also been shown that HMAC satisfies a strong notion of PRF-ODH security under the random oracle model [11]. Thus the EC-sym-ssPRF-ODH security assumption for HMAC is not out-of-scope.

Responder-Generated Passkey

Analysis results for the SHM Responder-Generated Passkey Entry protocol are as follows, covering all variants under the CYBORG security model. For proof details, see Appendix C.2.

Theorem 6.2. *SHM Responder-Generated Passkey Entry is*

- CYBORG-UncUser-secure *under the* EC-sym-ssPRF-ODH *and* EC-DDH *assumptions, the* sec-pre *of* H*, and the* SUF-CMA *security of* HMAC*.*
- CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$[x] \in \{[iu], [ur], [iu, ur]\} .$$

- ***not*** CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$\begin{aligned}[x] \in \{&[ru], [ui], iu, ru], [iu, ui], [ru, ui], [ru, ur], [ui, ur],\\ &[iu, ru, ui], [iu, ru, ur], [iu, ui, ur], [ru, ui, ur],\\ &[iu, ru, ui, ur]\} .\end{aligned}$$

User-Generated Passkey

Analysis results for the SHM User-Generated Passkey Entry protocol are as follows, covering all variants under the CYBORG security model. For proof details, see Appendix C.3.

Theorem 6.3. *SHM User-Generated Passkey Entry is*

- CYBORG-UncUser-secure *under the* EC-sym-ssPRF-ODH *and* EC-DDH *assumptions, the* sec-pre *of* H*, and the* SUF-CMA *security of* HMAC*.*
- CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$[x] \in \{[iu], [ru], [iu, ru]\}.$$

- ***not*** CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$\begin{aligned}[x] \in \{&[ui], [ur], [iu, ui], [iu, ur], [ru, ui], [ru, ur], [ui, ur],\\ &[iu, ru, ui], [iu, ru, ur], [iu, ui, ur], [ru, ui, ur],\\ &[iu, ru, ui, ur]\}.\end{aligned}$$

Dual Passkey

Analysis results for the SHM Dual Passkey Entry protocol are as follows, covering all variants under the CYBORG security model. For proof details, see Appendix C.4.

Theorem 6.4. *SHM Dual Passkey Entry is* CYBORG-$\mathsf{CompUser}_{[iu,ru,ui,ur]}$-secure *under the* EC-sym-ssPRF-ODH *and* EC-DDH *assumptions, the* sec-pre *of* H*, and the* SUF-CMA *security of* HMAC*.*

Implications

SHM Passkey Entry achieved similar security across all versions, with variations only in the $\mathsf{CompUser}_{[x]}$ setting. The various results give insight into the type of attacks Passkey Entry in its current construction can defend against. Mainly gaining control of the device display used to generate the passkey, or the device input of the passkey receiver is fatal to protocol security. This holds true for all current versions of Passkey Entry and points to the motivation behind DPE, namely that by having both devices generate a passkey, there is always a part of the "whole" passkey that the adversary cannot replace, regardless of the device's initiator/responder role. By Bluetooth specification, secrecy on the UtD channel is essential (reflected in the CYBORG model); thus, while the adversary can forge one r value, it may not read either one.

With both devices generating passkeys, the adversary cannot leverage its corruption queries to gain knowledge of the target session's passkey(s). We note that these security guarantees do not allow for eavesdropping, which is reasonable under Passkey Entry requirements. This shows that one can create protocols achieving a greater of degree of security than current methods with minimal increases in user involvement, and reasonable to device

requirements. With the integrity of at least one of the passkeys ensured (by the fact that each device generates a passkey), devices can successfully authenticate the DH key exchange. As proven in Theorem 6.4, Dual Passkey Entry maintains CYBORG security in spite of the adversary having the full capability to modify UtD messages and without requiring the user to generate random numbers.

7 Conclusion

Human interaction in protocols presents an intriguing challenge for analysis, encompassing the two-sided issues of human-device teaming down to the cryptographic level. Although Passkey Entry in its current construction fails to meet our measure of a secure cyborg key exchange under any variant, we showed how minor modifications improved its capability to achieve a robust level of security. Furthermore, we introduced SHM Dual Passkey Entry, which provably provides defense against the combined and advanced attacks already in existence that exploit corruption of displayed messages to users and user inputs to devices simultaneously.

The Secure Hash Modification presents a minor change that is reflective of normative practice in other real-world protocols, which do not rely on human interaction. We have shown that even such a minor change enables security under some CYBORG variants. The results, summarized in Fig. 3, provide user and manufacturer guidance in the use of an SHM-modified Passkey Entry. In particular, although full CYBORG security cannot be achieved, other variants can be.

Our results are not only relevant in the *security* they establish and design indications for such success, but also in the classes of CYBORG *insecurity* they demonstrate for Passkey Entry. For example, from Fig. 3 it is clear that a range of attacks exist on Passkey Entry in its current form. Notably, as shown in the table, Passkey Entry cannot be protected from the Tap n' Ghost attack even under SHM ($\mathsf{CompUser}_{[iu,ui]}$ and $\mathsf{CompUser}_{[ru,ur]}$). Using this work, any attack that can leverage a device display input/output according to the categories we have shown, gains a blueprint for successful execution.

References

[1] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, "Bias: Bluetooth impersonation attacks," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2020.

[2] D. Antonioli, N. O. Tippenhauer, and K. B. Rasmussen, "The KNOB is broken: Exploiting low entropy in the encryption key negotiation of bluetooth BR/EDR," in *28th USENIX Security Symposium, USENIX Security 2019*, N. Heninger and P. Traynor, Eds. USENIX Association, 2019, pp. 1047–1061.

[3] Apple, "Apple platform security," Available at https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf (2020/07/30), 2020.

[4] J. Barnickel, J. Wang, and U. Meyer, "Implementing an Attack on Bluetooth 2.1+ Secure Simple Pairing in Passkey Entry Mode," in *TrustCom 2012*, 2012, pp. 17–24.

[5] D. A. Basin, C. Cremers, and S. Meier, "Provably repairing the ISO/IEC 9798 standard for entity authentication," *J. Comput. Secur.*, vol. 21, no. 6, pp. 817–846, 2013. [Online]. Available: https://doi.org/10.3233/JCS-130472

[6] M. Bellare, "New proofs for NMAC and HMAC: security without collision-resistance," in *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, ser. Lecture Notes in Computer Science, C. Dwork, Ed., vol. 4117. Springer, 2006, pp. 602–619. [Online]. Available: https://doi.org/10.1007/11818175_36

[7] M. Bellare and P. Rogaway, "Entity Authentication and Key Distribution," in *CRYPTO '93*, 1993, pp. 232–249.

[8] E. Biham and L. Neumann, "Breaking the Bluetooth Pairing - Fixed Coordinate Invalid Curve Attack," Available at http://www.cs.technion.ac.il/~biham/BT/bt-fixed-coordinate-invalid-curve-attack.pdf (2019/08/22), Technion - Israel Institute of Technology, Tech. Rep., July 2018.

[9] C. Bisdikian, "An overview of the bluetooth wireless technology," *IEEE Communications magazine*, vol. 39, no. 12, pp. 86–94, 2001.

[10] *Bluetooth Core Specification*, 5th ed., Bluetooth Special Interest Group (SIG), December 2019.

[11] J. Brendel, M. Fischlin, F. Günther, and C. Janson, "PRF-ODH: Relations, Instantions, and Impossibility Results," Cryptology ePrint Archive, Report 2017/517.

[12] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion." *HotSec*, vol. 11, no. 2011, p. 9, 2011.

[13] R. Canetti and H. Krawczyk, "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels," in *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Proceeding*, 2001, pp. 453–474.

[14] M. C. Carlos, J. E. Martina, G. Price, and R. F. Custódio, "An updated threat model for security ceremonies," in *SAC '13, Coimbra, Portugal, March 18-22*, 2013, pp. 1836–1843.

[15] R. Chang and V. Shmatikov, "Formal Analysis of Authentication in Bluetooth Device Pairing," Available at https://www.cs.cornell.edu/~shmat/shmat_fcs07.pdf (2019/12/14), The University of Texas at Austin, Tech. Rep., July 2007.

[16] D. Damopoulos, G. Kambourakis, and S. Gritzalis, "From keyloggers to touchloggers: Take the rough with the smooth," *Computers & security*, vol. 32, pp. 102–114, 2013.

[17] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A Cryptographic Analysis of the TLS 1.3 Protocol Candidates," Cryptology ePrint Archive, Report 2015/914.

[18] B. Dowling and B. Hale, "There Can Be No Compromise: The Necessity of Ratcheted Authentication in Secure Messaging," Cryptology ePrint Archive, Report 2020/541.

[19] B. Dowling and K. G. Paterson, "A Cryptographic Analysis of the WireGuard Protocol," in *Applied Cryptography and Network Security*. Cham: Springer International Publishing, 2018, pp. 3–21.

[20] C. M. Ellison, "Ceremony design and analysis," *IACR Cryptology ePrint Archive*, p. 399, 2007.

[21] C. Gehrmann, C. J. Mitchell, and K. Nyberg, "Manual authentication for wireless devices," *RSA Cryptobytes*, vol. 7, no. 1, pp. 29–37, 2004.

[22] K. Haataja and P. Toivanen, "Two practical man-in-the-middle attacks on Bluetooth secure simple pairing and countermeasures," *IEEE Trans. Wireless Communications*, vol. 9, no. 1, pp. 384–392, 2010.

[23] B. Hale, "Computationally Modeling User-Mediated Authentication Protocols," *IACR Cryptology ePrint Archive*, p. 1239, 2019.

[24] J. Høegh-Omdal, C. Kaya, and M. Ottensmann, "The strandhogg vulnerability," Available at https://promon.co/security-news/strandhogg/ (2019).

[25] ISO, "Information technology – Security techniques – Key management – Part 4: Mechanisms based on weak secrets." International Organization for Standardization, Tech. Rep., November 2017.

[26] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of TLS-DHE in the standard model," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, 2012, pp. 273–293.

[27] M. Jakobsson, "Method and apparatus for immunizing against offline dictionary attacks," *US Patent Application*, vol. 60, no. 283,996, 2001.

[28] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC:Keyed-Hashing for Message Authentication," *RFC*, vol. 2104, pp. 1–11, 1997.

[29] B. A. LaMacchia, K. E. Lauter, and A. Mityagin, "Stronger Security of Authenticated Key Exchange," in *Provable Security, First International Conference, ProvSec 2007, Proceedings*, 2007, pp. 1–16.

[30] S. Laur and K. Nyberg, "Efficient mutual data authentication using manually authenticated strings," in *International Conference on Cryptology and Network Security*. Springer, 2006, pp. 90–107.

[31] A. Y. Lindell, "Attacks on the Pairing Protocol of Bluetooth v2.1," 2008.

[32] Y. Lindell, "Comparison-Based Key Exchange and the Security of the Numeric Comparison Mode in Bluetooth v2.1," *IACR Cryptology ePrint Archive*, p. 13, 2009.

[33] S. Maruyama, S. Wakabayashi, and T. Mori, "Tap 'n Ghost: A Compilation of Novel Attack Techniques against Smartphone Touchscreens," *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 620–637, 2019.

[34] N. I. of Standards and Technology, "Digital signature standard (shs)," Available at https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (2020/06/01), NIST, Tech. Rep., JulyS 2013.

[35] ——, "Secure hash standard (shs)," Available at https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (2020/06/01), NIST, Tech. Rep., August 2015.

[36] J. Padgette *et al.*, "Guide to Bluetooth Security," Available at https://doi.org/10.6028/NIST.SP.800-121r2 (2019/08/22), National Institute of Standards and Technology, Tech. Rep., May 2017.

[37] T. Peyrin and S. Vaudenay, "The pairing problem with user interaction," in *IFIP International Information Security Conference*. Springer, 2005, pp. 251–265.

[38] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," *RFC*, vol. 8446, pp. 1–160, 2018.

[39] Y. Shaked and A. Wool, "Cracking the bluetooth pin," in *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, 2005, pp. 39–50.

[40] D.-Z. Sun, Y. Mu, and W. Susilo, "Man-in-the-middle attacks on Secure Simple Pairing in Bluetooth standard V5.0 and its countermeasure," *Personal and Ubiquitous Computing*, vol. 22, no. 1, pp. 55–67, 2018.

[41] D. Sun and L. Sun, "On Secure Simple Pairing in Bluetooth Standard v5.0-Part I: Authenticated Link Key Security and Its Home Automation and Entertainment Applications," *Sensors*, vol. 19, no. 5, p. 1158, 2019.

[42] S. Vaudenay, "Secure communications over insecure channels based on short authenticated strings," in *Annual International Cryptology Conference*. Springer, 2005, pp. 309–326.

A Security Assumptions

We present relevant security properties of hash functions and hash-based message authentication codes that will prove consequential to our analysis of Passkey Entry. We also introduce relevant Diffie-Hellman (DH) problems to our analysis.

Definition A.1. A *message authentication code (*MAC*)* over ($\mathcal{M} = \{0,1\}^{\mathrm{m}}$, $\mathcal{K} = \{0,1\}^{\mathrm{k}}$, $\mathcal{T} = \{0,1\}^{\mathrm{t}}$) where $\mathcal{M}$ is the set of all possible messages, $\mathcal{K}$ is the set of all possible keys, and $\mathcal{T}$ is the set of all possible tags, to be the tuple of algorithms $(\mathrm{Kgn}, \mathrm{MAC}, \mathrm{Vfy})$ defined as follows:

- $\text{Kgn}(1^\lambda) \xrightarrow{\$} K$: A probabilistic key generation algorithm that takes as input 1^λ where λ is the security parameter, and outputs a key, $K \in \mathcal{K}$.
- $\text{MAC}(K, msg) \xrightarrow{\$} tag$: A deterministic hash-based message authentication algorithm that takes as input a key $K \in \mathcal{K}$ and a message $msg \in \mathcal{M}$, and outputs a tag, $tag \in \mathcal{T}$.
- $\text{Vfy}(K, msg, tag) \to v$: A deterministic verification algorithm that takes as inputs a key $K \in \mathcal{K}$, a message $msg \in \mathcal{M}$, and a tag $tag \in \mathcal{T}$, and outputs a verification bit, $v \in \{0, 1\}$.

We also require for the correctness of a message authentication code MAC that $\text{Vfy}(K, msg, tag) = 1$ iff $\text{MAC}(K, msg) = tag$, for all $msg \in \mathcal{M}$, $K \in \mathcal{K}$, and $tag \in \mathcal{T}$.

Second-Preimage Resistance

Definition A.2. Let H be a hash function and $\mathcal{A}$ a PPT adversary. We define the *Second-Preimage Resistance (*sec-pre*) of* H as such: given $msg \in \{0,1\}^*$ and *hash* value such that $\text{H}(msg) = hash$, $\mathcal{A}$ cannot find a second-preimage, $msg' \in \{0,1\}^*$, such that $msg' \neq msg$ and $\text{H}(msg') = hash$ with more than negligible probability. We denote $\mathcal{A}$'s advantage in breaking sec-pre of H as $\text{Adv}^{\text{sec-pre}}_{\text{H},\mathcal{A}}$.

SUF-CMA of a MAC

In Figure 7 we display the security game for strong unforgeability under chosen message attack (SUF-CMA) in algorithmic notation. This experiment models an attacker's ability to break the unforgeability of a MAC by forging a new message or a new tag of a known message-tag pair that verifies correctly.

$\underline{\text{EXP}^{\text{SUF-CMA}}_{\text{MAC},\mathcal{A}}(\lambda)}$:
1: $K \xleftarrow{\$} \text{Kgn}(1^\lambda)$
2: $\texttt{win} \leftarrow 0, S \leftarrow \emptyset$
3: $\mathcal{A}^{\text{MAC}(\cdot),\text{MAC.Vfy}(\cdot)}()$
4: **return** $\texttt{win}$

$\underline{\text{MAC}(msg)}$:
1: $tag \leftarrow \text{MAC}(K, msg)$
2: $S \leftarrow S \cup \{(msg, tag)\}$
3: **return** tag

$\underline{\text{MAC.Vfy}(msg, tag)}$:
1: $v \leftarrow \text{Vfy}(K, msg, tag)$
2: **if** $(v = 1) \wedge ((msg, tag) \notin S)$ **then**
3: $\quad \texttt{win} \leftarrow 1$
4: $\quad$ **return** $\texttt{win}$ from exp.
5: **end if**
6: **return** v

Figure 7: Security experiment for strong unforgeability under chosen message attack (SUF-CMA) of a message authentication code algorithm $\text{MAC} = (\text{Kgn}, \text{MAC}, \text{Vfy})$ and adversary $\mathcal{A}$.

Definition A.3. We define the *adversarial advantage against the* SUF-CMA *experiment* described in Figure 7 for a PPT adversary $\mathcal{A}$ against a message authentication code MAC to be:

$$\text{Adv}^{\text{SUF-CMA}}_{\text{MAC},\mathcal{A}}(\lambda) := \Pr[\text{EXP}^{\text{SUF-CMA}}_{\text{MAC},\mathcal{A}}(\lambda) = 1] \ .$$

Definition A.4. We say that some message authentication code MAC is SUF-CMA-secure if the advantage for any PPT adversary $\mathcal{A}$ interacting according to the experiment $\mathsf{EXP}^{\mathsf{SUF\text{-}CMA}}_{\mathrm{MAC},\mathcal{A}}(\lambda)$ is upper bounded by some negligible function $\mathrm{negl}(\lambda)$:

$$\mathrm{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathrm{MAC},\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda) \ .$$

EC-DDH Assumption

Definition A.5. Let E be an elliptic curve over the field $\mathbb{F}_q$ with generator point P of order n. Let $\mathcal{A}$ be a PPT adversary. We state the *Elliptic Curve Decisional DH (*EC-DDH*) assumption* as such: given access to E, P, and knowledge of aP and bP for $a, b \xleftarrow{\$} \mathbb{F}_q$ and $a, b < n$, $\mathcal{A}$ cannot distinguish abP from cP, for $c \xleftarrow{\$} \mathbb{F}_q$ and $c < n$, with more than negligible probability. We use $\mathrm{Adv}^{\mathsf{EC\text{-}DDH}}_{\mathcal{A}}$ to write $\mathcal{A}$'s advantage in breaking the EC-DDH assumption.

EC-sym-ssPRF-ODH Assumption

The PRF-ODH was originally introduced in [26], and later modified [17,19]. Other PRF-ODH assumption variants were analyzed in [11], with the sym-ssPRF-ODH assumption of [19] being a variant thereof. We present to the sym-ssPRF-ODH assumption of [19] below, but for Elliptic Curve (EC) Diffie–Hellman vs. standard Diffie–Hellman.

Definition A.6. Let E be an elliptic curve over the field $\mathbb{F}_q$ with generator point P of order n. Let $\mathrm{PRF}_\lambda : E \times \{0,1\}^* \to \{0,1\}^\lambda$ be a psuedorandom function with keys in E, input strings in $\{0,1\}^*$, and output strings in $\{0,1\}^\lambda$. Let $\mathcal{A}$ be a PPT adversary. *Elliptic Curve symmetric generic single-single PRF Oracle DH assumption* (EC-sym-ssPRF-ODH) is defined as follows:

1. The challenger samples $a, b \xleftarrow{\$} \mathbb{F}_q$ uniformly at random with $a, b < n$, computes aP and bP, and provides (E, P, aP, bP) to $\mathcal{A}$.
2. Eventually, $\mathcal{A}$ issues the challenge query $x^* \leftarrow \{0,1\}^*$.
3. The challenger samples $\mathsf{b} \xleftarrow{\$} \{0,1\}$ uniformly at random and sets $y_0 \leftarrow \mathrm{PRF}_\lambda(abP, x)$ if $\mathsf{b} = 0$, and $y_1 \leftarrow \{0,1\}^\lambda$ otherwise. The challenger returns y_b to $\mathcal{A}$.
4. $\mathcal{A}$ may issue a single query to the oracles, $\mathsf{EC\text{-}ODH}_a$ and $\mathsf{EC\text{-}ODH}_b$, handled as follows:
   - $\mathsf{EC\text{-}ODH}_a(S, x)$: Challenger returns $\perp$ if $S \notin E$ or if $(S, x) = (bP, x^*)$, otherwise it returns $y \leftarrow \mathrm{PRF}_\lambda(aS, x)$.
   - $\mathsf{EC\text{-}ODH}_b(T, x)$: Challenger returns $\perp$ if $T \notin E$ or if $(T, x) = (aP, x^*)$, otherwise it returns $y \leftarrow \mathrm{PRF}_\lambda(bT, x)$.
5. Eventually, $\mathcal{A}$ outputs the bit guess b, and wins the experiment if $b = \mathsf{b}$.

We define the adversarial advantage in the EC-sym-ssPRF-ODH experiment as

$$\mathrm{Adv}^{\mathsf{EC\text{-}sym\text{-}ssPRF\text{-}ODH}}_{\mathrm{PRF}_\lambda,\mathcal{A}}(\lambda) := \Pr[b = \mathsf{b}] - \frac{1}{2} \ ,$$

and we say that the EC-sym-ssPRF-ODH *assumption holds* if

$$\mathrm{Adv}^{\mathsf{EC\text{-}sym\text{-}ssPRF\text{-}ODH}}_{\mathrm{PRF}_\lambda,\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda) \ .$$

B Association between User Compromise Types

We begin by presenting Theorem B.2, which will prove useful as a framework in future proofs within the compromised user setting.

Lemma B.1. *Let X and Y be non-empty subsets of $\{iu, ru, ui, ur\}$ such that $X \subseteq Y$ with labels x and y respectively. If a session oracle π_j^U is* $\mathsf{CompUser}_{[x]}$*-fresh, then it is also* $\mathsf{CompUser}_{[y]}$*-fresh.*

Proof. Let Π be a cyborg key exchange protocol, and let $\mathcal{A}$ be a PPT adversary against the CYBORG-$\mathsf{CompUser}_{[x]}$ security of Π. If some user session oracle, π_j^U, is $\mathsf{CompUser}_{[x]}$*-fresh* then we must have that $\mathcal{A}$ never issued a single query that broke $\mathsf{CompUser}_{[x_i]}$-freshness simultaneously for all $x_i \in X$ over the course of the CYBORG-$\mathsf{CompUser}_{[x]}$ security experiment by Definition 3.12. This means that for every query issued by $\mathcal{A}$, π_j^U must have met the definition for $\mathsf{CompUser}_{[x_i]}$*-fresh* for some $x_i \in X$. Since $X \subseteq Y$, then we also have that π_j^U must have met the definition for $\mathsf{CompUser}_{[y_i]}$*-fresh* for some $y_i \in Y$ for every query issued by $\mathcal{A}$. Thus, $\mathcal{A}$ never issued a single query that broke $\mathsf{CompUser}_{[y_i]}$-freshness simultaneously for all $y_i \in Y$ and we have that π_j^U is also $\mathsf{CompUser}_{[y]}$*-fresh*. □

Theorem B.2. *Let X and Y be non-empty subsets of $\{iu, ru, ui, ur\}$ such that $X \subseteq Y$ with labels x and y respectively, and let Π be a cyborg key exchange protocol. If Π is not* CYBORG-$\mathsf{CompUser}_{[x]}$-secure*, then it is not* CYBORG-$\mathsf{CompUser}_{[y]}$-secure.

Proof. Let Π be a cyborg key exchange protocol, and let $\mathcal{A}$ be a PPT adversary that breaks the CYBORG-$\mathsf{CompUser}_{[x]}$ security of Π. We then construct a second adversary, $\mathcal{B}$, against the CYBORG-$\mathsf{CompUser}_{[y]}$ security experiment. The challenger starts the experiment and forwards the protocol flows of Π to $\mathcal{A}$ and uses $\mathcal{A}$'s responses. By the success of $\mathcal{A}$, Lemma B.1, and Definition 3.12, we have $\mathsf{CompUser}_{[y]}$ freshness and the success of $\mathcal{B}$. Therefore we have that

$$\mathrm{Adv}^{\text{CYBORG-}\mathsf{CompUser}_{[x]}}_{\Pi,\mathcal{A},\eta_p,\eta_s}(\lambda) \leq \mathrm{Adv}^{\text{CYBORG-}\mathsf{CompUser}_{[y]}}_{\Pi,\mathcal{B},\eta_p,\eta_s}(\lambda) \ .$$

□

C SHM Passkey Entry Proofs

We restate Theorem 6.1–Theorem 6.3 and provide their proofs.

Using Theorem B.2, we advance analysis incrementally introducing more allowable combinations of user compromise until we reach an environment where the protocol breaks. At such a point, all subsequent definitions of the compromised user setting where the break persists can then be addressed as a corollary.

C.1 SHM Initiator-Generated Passkey Proofs

Theorem (6.1 part 1). *SHM Initiator-Generated Passkey Entry is* CYBORG-UncUser-secure *under the* EC-sym-ssPRF-ODH *and* EC-DDH *assumptions, the* sec-pre *of* H*, and the* SUF-CMA *security of* HMAC.

Proof. The proof of this theorem involves a series of game hops between an adversarial PPT algorithm $\mathcal{A}$ and the challenger. We denote the adversarial advantage of a specific game as Adv_i, for the i-th game hop.

Game 0 This game is equivalent to the original security experiment:

$$\mathrm{Adv}_0 = \mathrm{Adv}^{\text{CYBORG-UncUser}}_{\text{PE-IG},\mathcal{A},\eta_p,\eta_s}(\lambda) \ ,$$

where λ is the security parameter, η_p is a bound on the number of participants, and η_s is a bound on the number of sessions a participant can run.

Game 1 This game is equivalent to the previous except we raise the event `abort`, end the experiment, and output zero if there ever exists two session oracles that generate the same ephemeral key, SK, in Phase 1. If session keys ever repeat, then $\mathcal{A}$ could execute a StateReveal query on the second session to recover SK and compute DHKey. We have that:

$$\mathrm{Adv}_1 \geq \mathrm{Adv}_0 - \frac{(\eta_p\eta_s)^2}{2^\mu} \ .$$

where μ is the length of SK.

Game 2 This game is equivalent to the previous except we raise the event `abort`, end the experiment, and output zero if there ever exists a nonce collision in the experiment. This prevents trivial guesses of passkey bits and lets us assume all nonces are generated fresh. There are 20 nonces generated in each session, therefore, we have that:

$$\mathrm{Adv}_2 \geq \mathrm{Adv}_1 - \frac{400(\eta_p\eta_s)^2}{2^\lambda} \ .$$

Game 3 This game is equivalent to the previous security experiment except we raise the event `abort`, end the experiment, and output zero if a passkey is ever reused. Since the passkey is inherently revealed during the completion of the Passkey Entry protocol, re-use of this value would allow $\mathcal{A}$ to break auth or key-ind with probability 1. Since only one passkey is generated each session we have that:

$$\mathrm{Adv}_3 \geq \mathrm{Adv}_2 - \frac{(\eta_p\eta_s)^2}{2^{|r|}} \ ,$$

where $|r|$ is the length of the passkey r.

Game 4 This game is equivalent to the previous security experiment except we guess the session oracles executing the protocol, the test session π^A_s and its partner π^B_t, and abort if $\mathcal{A}$ does not try to win against this guessed pair. Thus,

$$\mathrm{Adv}_4 \geq \frac{1}{(\eta_p\eta_s)^2} \cdot \mathrm{Adv}_3 \ .$$

We then continue by case dependency on if $\mathcal{A}$ attempts to win by breaking correct, auth, or key-ind.

$$\mathrm{Adv}_4 = \mathrm{Adv}^{\mathsf{correct}}_4 + \mathrm{Adv}^{\mathsf{auth}}_4 + \mathrm{Adv}^{\mathsf{key\text{-}ind}}_4 \ .$$

Advantage against correct Since session oracles with matching session identifiers are guaranteed to accept by the correctness of Passkey Entry, we have that:

$$\mathrm{Adv}_4^{\mathsf{correct}} = 0 \ .$$

Advantage against auth

Game 5 We continue with $\mathrm{Adv}_4^{\mathsf{auth}}$. Since we have the requirement that π_s^A remains fresh, we will abort the experiment if $\mathcal{A}$ issues a StateReveal, KeyReveal, ShowUser, or ControlUser queries such that π_s^A or the partnered user session, π_j^U, are no longer UncUser-*fresh*. If π_j^U is also partnered with π_t^B, we abort if $\mathcal{A}$ issues a $\mathsf{StateReveal}(\pi_t^B)$ query while $\pi_s^A \neq \mathsf{accept}$. We raise the event `abort`, end the experiment, and output zero if $\mathcal{A}$ succeeds in replacing PK_b.

In order to replace PK_b, and therefore get π_s^A to accept maliciously, $\mathcal{A}$ must guess all $|r|$ bits of the passkey r, allowing it to recalculate the commitment under a nonce key of its choice. Accounting for π_s^A in either an initiator or responder role, we have

$$\mathrm{Adv}_5^{\mathsf{auth}} \geq \mathrm{Adv}_4^{\mathsf{auth}} - \frac{1}{2^{|r|-1}} \ .$$

Game 6 We replace DHKey with a uniformly random value $\widetilde{\mathrm{DHKey}} = cP$ for $c \xleftarrow{\$} \mathbb{F}_q$ where $\mathbb{F}_q$ is the finite field over which we define our elliptic curve E and P is a generator for E. Suppose that the adversary can distinguish between this and the previous game. Then we can construct a new adversary, $\mathcal{B}_1$, solving the DDH problem, as from the previous game hop we have that Diffie-Hellman shares $aP = PK_a$, and $bP = PK_b$ have been exchanged.

The challenger proceeds as before, but replaces the DHKey with a uniformly random value $\widetilde{\mathrm{DHKey}} = cP$, which is used for both partners. Algorithm $\mathcal{B}_1$ receives as input (E, P, aP, bP, cP) where $aP = PK_a$, $bP = PK_b$, and $cP = \widetilde{\mathrm{DHKey}}$. The EC-DDH assumption therefore implies

$$\mathrm{Adv}_6^{\mathsf{auth}} \geq \mathrm{Adv}_5^{\mathsf{auth}} - \mathrm{Adv}_{\mathcal{B}_1}^{\mathsf{EC\text{-}DDH}}(\lambda) \ .$$

We continue the proof by separating two sub-cases based on the test session's role: initiator (C_1) or responder (C_2):

$$\mathrm{Adv}_6^{\mathsf{auth}} = \mathrm{Adv}_6^{\mathsf{auth},C_1} + \mathrm{Adv}_6^{\mathsf{auth},C_2} \ .$$

Case 1 $\pi_s^A.\mathsf{role} = \mathsf{initiator}$.

Game 7 This game is equivalent to the previous security experiment except we raise the event `abort`, end the experiment, and output zero if the adversary succeeds in forging the tag E_b or any of N_b, IOcapB, or B in the message used to compute E_b.

We bound the abort condition by constructing the adversary, $\mathcal{B}_2$, against the SUF-CMA security of HMAC. The challenger sets the MAC key to $\widetilde{\mathrm{DHKey}}$, and $\mathcal{B}_2$ uses the oracle MAC to compute tags. $\mathcal{B}_2$ calls $\mathsf{MAC}(N_b \| N_a \| r_a \| \mathrm{IOcapB} \| \mathrm{IOcapA} \| B \| A)$, which returns the tag E_b (r, $N_{b,i}$, and $N_{a,i}$ are all public at this point in the protocol). $\mathcal{B}_2$ then gives the message-tag pair $(N_b \| N_a \| r_a \| \mathrm{IOcapB} \| \mathrm{IOcapA} \| B \| A, E_b)$ to $\mathcal{A}$. If $\mathcal{A}$ is able to forge a new tag, call it E_{win},

such that $\mathsf{MAC.Vfy}(N_b \| N_a \| r_a \| \text{IOcapB} \| \text{IOcapA} \| B \| A, E_{win}) = 1$, or a new message, call it msg_{win}, by forging at least one of N_b, IOcapB, and B, such that $\mathsf{MAC.Vfy}(msg_{win}, E_b) = 1$, then $\mathcal{B}_2$ can win the SUF-CMA experiment with the winning message-tag pair. Thus we have

$$\text{Adv}_7^{\mathsf{auth},C_1} \geq \text{Adv}_6^{\mathsf{auth},C_1} - \text{Adv}_{\text{HMAC},\mathcal{B}_2}^{\mathsf{SUF\text{-}CMA}}(\lambda) \ .$$

Game 8 This game is equivalent to the previous security experiment except we raise the event `abort`, end the experiment, and output zero if the adversary succeeds in forging any of B's nonces or B's role, causing A to accept maliciously.

From Game 7 we have that $\mathcal{A}$ does not succeed in forging N_b. Thus if $\mathcal{A}$ succeeds in forging any nonce $N_{b,i}$ or B's role, we use the success of $\mathcal{A}$ to construct a new adversary, $\mathcal{B}_3$, against the second-preimage resistance of the hash function. Per the sec-pre experiment, $\mathcal{B}_3$ is given the message-hash pair $(N_{b,1} \| \ldots \| N_{b,20} \| \mathsf{role}_b, N_b)$ by $\mathcal{A}$ (note that all $N_{b,i}$ nonces are public at this point in the protocol), where the nonces are as sampled by π_t^B and $N_b = \text{H}(N_{b,1} \| \ldots \| N_{b,20} \| \mathsf{role}_b)$. By our `abort` condition, $\mathcal{A}$ is able to forge at least one nonce $N'_{b,i}$ or B's role. $\mathcal{B}_3$ uses the new sequence $N'_{b,1} \| \ldots \| N'_{b,20} \| \mathsf{role}'_b$ as its guess at a second-preimage for N_b. By the success of $\mathcal{A}$ we have that $H(N'_{b,1} \| \ldots \| N'_{b,20} \| \mathsf{role}'_b) = N_b$. Thus we have

$$\text{Adv}_8^{\mathsf{auth},C_1} \geq \text{Adv}_7^{\mathsf{auth},C_1} - \text{Adv}_{\text{H},\mathcal{B}_3}^{\mathsf{sec\text{-}pre}}(\lambda) \ .$$

By Game 8, the adversary can only succeed in breaking auth by forging a commitment value, $C_{a,i}$ or $C_{b,i}$. However, the correctness of HMAC ensures $C_{a,i}$ or $C_{b,i}$ against forgery for all i since the messages and keys used to compute them are fixed from previous game hops. We therefore have matching sidu, $\pi_s^A.\mathsf{pid} = B$, $\pi_t^B.\mathsf{pid} = A$, $\pi_s^A.\mathsf{role} \neq \pi_t^B.\mathsf{role}$ telling us our session oracles are partnered via Definition 3.4. Thus,

$$\text{Adv}_8^{\mathsf{auth},C_1} = 0 \ .$$

Case 2 $\pi_s^A.\mathsf{role} = \mathsf{responder}$. This case follows similarly to Case 1.

Combining our previous probability statements for the two sub-cases above we have:

$$\text{Adv}^{\mathsf{auth}} \leq \frac{1}{2^{|r|-1}} + \text{Adv}_{\mathcal{B}_1}^{\mathsf{EC\text{-}DDH}}(\lambda) + 2 \cdot (\text{Adv}_{\text{HMAC},\mathcal{B}_2}^{\mathsf{SUF\text{-}CMA}}(\lambda) + \text{Adv}_{\text{H},\mathcal{B}_3}^{\mathsf{sec\text{-}pre}}(\lambda)) \ .$$

Advantage against key-ind In this case we assume that

Game 5 We now bound $\text{Adv}_4^{\mathsf{key\text{-}ind}}$. Since we have the requirement that π_s^A remains fresh, we will abort the experiment if $\mathcal{A}$ issues a StateReveal, KeyReveal, ShowUser, or ControlUser queries such that π_s^A or the partnered user session, π_j^U, are no longer UncUser-*fresh*. If π_j^U is also partnered with π_t^B, we abort if $\mathcal{A}$ issues a $\mathsf{StateReveal}(\pi_t^B)$ query while $\pi_s^A \neq \mathsf{accept}$.

Suppose that $\mathcal{A}$ can correctly distinguish the provided key as real or random. Then we can construct a new adversary, $\mathcal{B}_4$, solving the EC-sym-ssPRF-ODH problem as follows. $\mathcal{B}_4$ receives as input (E, P, aP, bP) where P is a generator of our elliptic curve group, $aP = PK_a$, and $bP = PK_b$. $\mathcal{B}_4$ then issues the challenge query $x = (N_a \| N_b \| \text{btlk} \| A \| B)$ (if $\pi_s^A.\mathsf{role} = \mathsf{initiator}$ and $x = (N_b \| N_a \| \text{btlk} \| B \| A)$ otherwise) also using values as chosen by our test and partner oracles. The challenger then randomly samples b and sets

$LK \leftarrow \text{HMAC}(\text{DHKey}, N_a \| N_b \| \text{btlk} \| A \| B)$ if $\mathsf{b} = 1$ and $LK \xleftarrow{\$} \{0,1\}^\lambda$ otherwise. The challenger then returns LK_b to $\mathcal{B}_4$. The challenger also allots one-time access to the left and right HMAC oracles for computation of E_a and E_b. At this point, $\mathcal{B}_4$ can simulate all other flows between our test session and partner session. If we have that $LK_\mathsf{b} = \text{HMAC}(\text{DHKey}, N_a \| N_b \| \text{btlk} \| A \| B)$, then the view of $\mathcal{A}$ when interacting with this game is identical to Game 4. Similarly, the view of $\mathcal{A}$ when interacting with this game is identical to Game 5 if $LK_\mathsf{b} \xleftarrow{\$} \{0,1\}^\lambda$. Thus by the success of $\mathcal{A}$ in distinguishing Game 4 and Game 5, we have the success of $\mathcal{B}_4$. With π_s^A as either the initiator or responder, we have:

$$\text{Adv}_5^{\mathsf{key\text{-}ind}} \geq \text{Adv}_4^{\mathsf{key\text{-}ind}} - \text{Adv}_{\text{HMAC},\mathcal{B}_4}^{\mathsf{EC\text{-}sym\text{-}ssPRF\text{-}ODH}}(\lambda) \ .$$

Since the session key of our test oracle is now uniformly random, we also conclude:

$$\text{Adv}_5^{\mathsf{key\text{-}ind}} = 0 \ .$$

We now combine all probability statements to arrive at our final security reduction:

$$\text{Adv}_{\text{PE-IG},\mathcal{A},\eta_p,\eta_s}^{\text{CYBORG}-\mathsf{UncUser}}(\lambda) \leq (\eta_p \eta_s)^2 \cdot \Big(\frac{1}{2^\mu} + \frac{400}{2^\lambda} + \frac{3}{2^{|r|}} + \text{Adv}_{\mathcal{B}_1}^{\mathsf{EC\text{-}DDH}}(\lambda) + 2 \cdot (\text{Adv}_{\text{HMAC},\mathcal{B}_2}^{\mathsf{SUF\text{-}CMA}}(\lambda) + \text{Adv}_{\text{H},\mathcal{B}_3}^{\mathsf{sec\text{-}pre}}(\lambda)) + \text{Adv}_{\text{HMAC},\mathcal{B}_4}^{\mathsf{EC\text{-}sym\text{-}ssPRF\text{-}ODH}}(\lambda) \Big) \ .$$

□

Per the Passkey Entry specification, $|r| = 20$.

Theorem (6.1 part 2). *SHM Initiator-Generated Passkey Entry is* CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$x \in \{[ru], [ui]\, [ru, ui]\} \ .$$

Proof. This proof follows from a triviality. In all three of the above listed CYBORG-$\mathsf{CompUser}_{[x]}$ security environments, the adversary gains the capability to issue queries that allow him to compromise a UtD channel(s) that is/are not employed in Initiator-Generated Passkey Entry. Therefore, these settings reduce to the CYBORG-$\mathsf{UncUser}$ setting, which was proven secure in Theorem 6.1. □

Lemma C.1. *SHM Initiator-Generated Passkey Entry is not* CYBORG-$\mathsf{CompUser}_{[iu]}$-secure.

Proof. We proceed via counter-example. Let $\mathcal{A}$ be an adversarial algorithm against the CYBORG-$\mathsf{CompUser}_{[iu]}$ security of the Initiator-Generated Passkey Entry protocol. $\mathcal{A}$ first issues a $\mathsf{SendDevice}(\pi_s^A, (\mathsf{start}, B))$ query and a $\mathsf{SendUser}(\pi_j^U, (\mathsf{start}, (\pi_s^A, \pi_t^B)))$ query to initiate protocol participants. $\mathcal{A}$ then issues a $\mathsf{ShowUser}(\pi_s^A)$ query and proceeds with a MitM attack as follows:

- Phase 0 proceeds as normal.
- In Phase 1: $\mathcal{A}$ impersonates π_s^A to π_t^B.
- In Phase 2:

- A generates the passkey and sends it to the user.
- Since $\mathcal{A}$ issued a $\mathsf{ShowUser}(\pi_s^A)$ query, he may modify the value r shown to the user. We denote this new passkey r_e.
- The user forwards r_e to π_t^B per the protocol specification.

- Phase 3 and 4 proceed according to the protocol, with $\mathcal{A}$ impersonating π_s^A.
- π_t^B sets $\pi_t^B.\delta = \mathsf{accept}$.

At the conclusion of the above attack we have that $\mathcal{A}$ is PPT algorithm winning the CYBORG-$\mathsf{CompUser}_{[iu]}$ security experiment by breaking auth. □

Lemma C.2. *SHM Initiator-Generated Passkey Entry is not* CYBORG-$\mathsf{CompUser}_{[ur]}$-secure.

Proof. This proof runs similarly to the one described in Lemma C.1 with a few changes. Instead of issuing a $\mathsf{ShowUser}(\pi_s^A)$ query, $\mathcal{A}$ issues a $\mathsf{ControlUser}(\pi_j^U, B)$ query. This allows him to modify the passkey of r to r_e on input to π_t^B. □

Theorem (6.1 part 3). *SHM Initiator-Generated Passkey Entry is not* CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$\begin{aligned}[x] \in \{&[iu], [ur], [iu, ru], [iu, ui], [iu, ur], [ru, ur], [ui, ur],\\ &[iu, ru, ui], [iu, ru, ur], [iu, ui, ur], [ru, ui, ur],\\ &[iu, ru, ui, ur]\} \ .\end{aligned}$$

Proof. By Lemmas C.1 and C.2 we have that Initiator-Generated Passkey Entry is not CYBORG-$\mathsf{CompUser}_{[x]}$ for $x \in \{[iu], [ur]\}$. Therefore, we have that Initiator-Generated Passkey Entry is not CYBORG-$\mathsf{CompUser}_{[x]}$ secure for $X \subseteq \{iu, ru, ui, ur\}$ where at least one of iu or ur is an element of X, by applying Theorem B.2. □

C.2 SHM Responder-Generated Passkey Proofs

Theorem (6.2 part 1). *SHM Responder-Generated Passkey Entry is* CYBORG-$\mathsf{UncUser}$-secure *under the* $\mathsf{EC\text{-}sym\text{-}ssPRF\text{-}ODH}$ *and* $\mathsf{EC\text{-}DDH}$ *assumptions, the second pre-image resistance of* H*, and the* $\mathsf{SUF\text{-}CMA}$ *security of* HMAC.

Proof. This proof follows similarly to Theorem 6.1. This is due to the fact that the adversary is unable to exploit how the passkey is exchanged between the devices, regardless of which device generated the value, in the CYBORG-$\mathsf{UncUser}$ environment. □

Theorem (6.2 part 2). *SHM Responder-Generated Passkey Entry is* CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$x \in \{[iu], [ur], [iu, ur]\} \ .$$

Proof. This proof follows from a triviality. In all three of the above listed CYBORG-$\mathsf{CompUser}_{[x]}$ security environments, the adversary may compromise UtD channels that are not employed in SHM Responder-Generated Passkey Entry. Therefore, these settings reduce to the CYBORG-$\mathsf{UncUser}$ setting, which was proven secure in Theorem 6.2. □

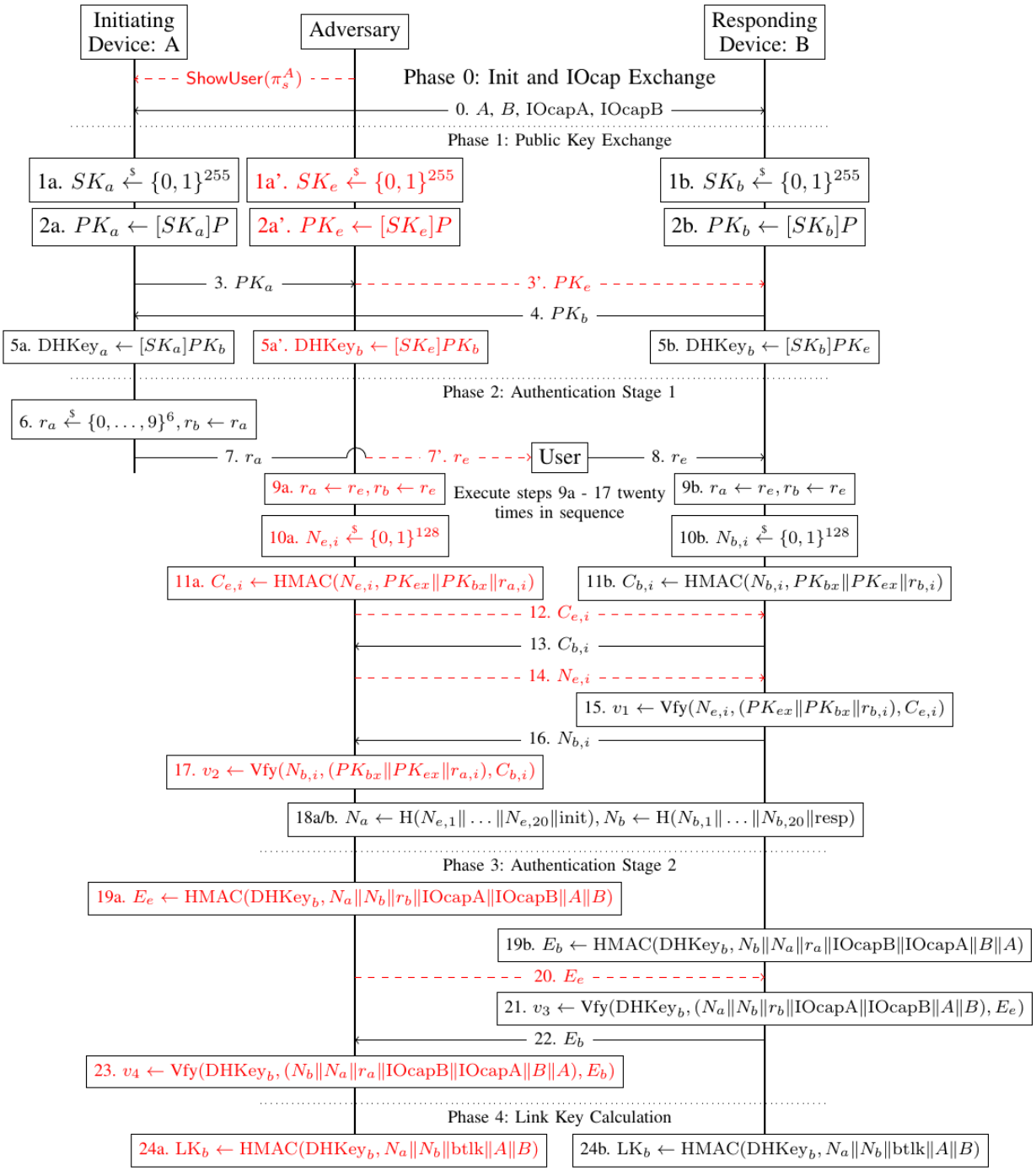

Figure 8: Depiction of adversarial attack (singular actions shown in red) against SHM Initiator-Generated Passkey Entry under the CYBORG-CompUser$_{[iu]}$ security experiment, discussed in Lemma C.1. Similar vulnerabilities also affect Responder-Generated and User-Generated Passkey Entry. Any instance where the communication flow "jumps" over a device line is done to illustrate adversarial inability to read the message.

Lemma C.3. *SHM Responder-Generated Passkey Entry is not* CYBORG-$\mathsf{CompUser}_{[ru]}$-secure.

Proof. This proof runs similarly to the one described for Lemma C.1, but with A in the responder role and B as initiator. □

Lemma C.4. *SHM Responder-Generated Passkey Entry is not* CYBORG-$\mathsf{CompUser}_{[ui]}$-secure.

Proof. This proof runs similarly to the one described in Lemma C.1 with a few changes. Instead of issuing a $\mathsf{ShowUser}(\pi_s^A)$ query, the adversary $\mathcal{A}$ issues a $\mathsf{ControlUser}(\pi_j^U, A)$ query. This allows him to modify the passkey of r to r_e on input to π_s^A. □

Theorem (6.2 part 3). *SHM Responder-Generated Passkey Entry is not* CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$\begin{aligned}[x] \in \{&[ru], [ui], [iu, ru], [iu, ui], [ru, ui], [ru, ur], [ui, ur],\\ &[iu, ru, ui], [iu, ru, ur], [iu, ui, ur], [ru, ui, ur],\\ &[iu, ru, ui, ur]\} \ .\end{aligned}$$

Proof. By Lemmas C.3 and C.4 we have that SHM Responder-Generated Passkey Entry is not CYBORG-$\mathsf{CompUser}_{[x]}$ for $[x] \in \{[ru], [ui]\}$. Therefore, we have that SHM Responder-Generated Passkey Entry is not CYBORG-$\mathsf{CompUser}_{[x]}$ secure for $X \subseteq \{iu, ru, ui, ur\}$ where at least one of ru or ui is an element of X by applying Theorem B.2. □

C.3 SHM User-Generated Passkey Proofs

Theorem (6.3 part 1). *SHM User-Generated Passkey Entry is* CYBORG-$\mathsf{UncUser}$-secure *under the* $\mathsf{EC\text{-}sym\text{-}ssPRF\text{-}ODH}$ *and* $\mathsf{EC\text{-}DDH}$ *assumptions, the second pre-image resistance of* H*, and the* $\mathsf{SUF\text{-}CMA}$ *security of* HMAC.

Proof. This proof follows similarly to Theorem 6.1. This is due to the fact that the adversary is unable to exploit how the passkey is exchanged between the devices, regardless of which device generated the value, in the CYBORG-$\mathsf{UncUser}$ environment. □

Theorem (6.3 part 2). *SHM User-Generated Passkey Entry is* CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$[x] \in \{[iu], [ru], [iu, ru]\}.$$

Proof. This proof follows from a triviality. In all three of the above listed CYBORG-$\mathsf{CompUser}_{[x]}$ security environments, the adversary may compromise UtD channels that are not employed in SHM User-Generated Passkey Entry. Therefore, these settings reduce to the CYBORG-$\mathsf{UncUser}$ setting, which was proven secure in Theorem 6.2. □

Lemma C.5. *SHM User-Generated Passkey Entry is not* CYBORG-$\mathsf{CompUser}_{[ui]}$-secure.

Proof. This proof runs similarly to the one described in Lemma C.4. □

Lemma C.6. *SHM User-Generated Passkey Entry is not* CYBORG-$\mathsf{CompUser}_{[ur]}$-secure.

Proof. This proof runs similarly to the one described in Lemma C.2. □

Theorem (6.3 part 3). *User-Generated Passkey Entry is not* CYBORG-$\mathsf{CompUser}_{[x]}$-secure *for*

$$\begin{aligned}[x] \in \{&[ui], [ur], [iu, ui], [iu, ur], [ru, ui], [ru, ur], [ui, ur],\\ &[iu, ru, ui], [iu, ru, ur], [iu, ui, ur], [ru, ui, ur],\\ &[iu, ru, ui, ur]\}.\end{aligned}$$

Proof. By Lemmas C.5 and C.6 we have that SHM User-Generated Passkey Entry is not CYBORG-$\mathsf{CompUser}_{[x]}$ for $[x] \in \{[ui], [ur]\}$. Therefore, we have that SHM User-Generated Passkey Entry is not CYBORG-$\mathsf{CompUser}_{[x]}$-secure for $X \subseteq \{iu, ru, ui, ur\}$ where at least one of ui or ur is an element of X, by applying Theorem B.2. □

C.4 SHM Dual Passkey Proofs

Theorem (6.4). *SHM Dual Passkey Entry is* CYBORG-$\mathsf{CompUser}_{[iu,ru,ui,ur]}$-secure *under the* EC-sym-ssPRF-ODH *and* EC-DDH *assumptions, the second pre-image resistance of* H*, and the* SUF-CMA *security of* HMAC.

Proof. The proof of this theorem follows closely to that of Theorem 6.1 part 1, but with the following alterations. Following the proof of Theorem 6.1, let Adv_i denote the i-th game hop in the same series of games between an adversarial PPT algorithm $\mathcal{A}$ and the challenger.

Advantage against auth

Game 5 We continue with $\mathrm{Adv}_4^{\mathsf{auth}}$. Since we have the requirement that π_s^A remains fresh, we will abort the experiment if $\mathcal{A}$ issues $\mathsf{StateReveal}()$ or $\mathsf{KeyReveal}$ queries such that π_s^A is no longer $\mathsf{CompUser}_{[iu,ru,ui,ur]}$-*fresh*. This implies that $\mathcal{A}$ may issue $\mathsf{ControlUser}$ queries involving the partnered user oracle π_j^U as desired, since we are under $\mathsf{CompUser}_{[iu,ru,ui,ur]}$ environment π_j^U remains fresh regardless of adversarial action. Similarly $\mathcal{A}$ may issue $\mathsf{ShowUser}$ queries at will. Thus $\mathcal{A}$ is only limited on the UtD channel by its inability to read valid values r_a and r_b.

If π_j^U is also partnered with π_t^B, we abort if $\mathcal{A}$ issues a $\mathsf{StateReveal}(\pi_t^B)$ query while $\pi_s^A \neq \mathsf{accept}$. We raise the event `abort`, end the experiment, and output zero if $\mathcal{A}$ succeeds in replacing PK_b.

Note $\mathcal{A}$ may issue either a $\mathsf{ShowUser}(\pi_t^B)$ or $\mathsf{ControlUser}(\pi_j^U, B)$ query to forge the passkey r_b of the would-be partner. In order to replace PK_b, the $\mathcal{A}$ must guess all $|r|$ bits of the passkey r_a, allowing it to recalculate the commitment under a nonce key of its choice. Accounting for π_s^A in either the initiator and responder role, we have

$$\mathrm{Adv}_5^{\mathsf{auth}} \geq \mathrm{Adv}_4^{\mathsf{auth}} - \frac{1}{2^{|r|-1}} \ .$$

□