



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**A CRYPTOGRAPHIC ANALYSIS OF BLUETOOTH'S
HUMAN-MACHINE AUTHENTICATED KEY
EXCHANGE PROTOCOLS**

by

Michael E. Troncoso

September 2020

Thesis Advisor:
Co-Advisor:

Britta Hale
Pantelimon Stanica

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2020		3. REPORT TYPE AND DATES COVERED Master's thesis
4. TITLE AND SUBTITLE A CRYPTOGRAPHIC ANALYSIS OF BLUETOOTH'S HUMAN-MACHINE AUTHENTICATED KEY EXCHANGE PROTOCOLS				5. FUNDING NUMBERS
6. AUTHOR(S) Michael E. Troncoso				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000				8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A				10. SPONSORING / MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.				12b. DISTRIBUTION CODE A
13. ABSTRACT (maximum 200 words) As Bluetooth is firmly ensconced as one of the leading standardizations for wireless communication, it becomes imperative to rigorously quantify its security. To forward this quantification, we conduct a comprehensive analysis of Bluetooth's user-mediated authenticated key exchanges, Numeric Comparison and Passkey Entry, in both the computational and formal cryptographic settings. Due to the reliance on intertwined human and machine functions in the specification of these cyborg protocols, new attack vectors arise for hostile actors to exploit. Consequently, we model a realistic adversary, one not only with access to both the user-to-device interfaces and device-to-device communication channels simultaneously, but also with the capability to compromise device display and input mechanisms. Our analysis shows that while Numeric Comparison and Initiator/Responder-Generated Passkey Entry achieve at least basic levels of security in our model, User-Generated Passkey Entry is insecure in the model. Furthermore, the categories of attacks depicted herein function as a blueprint for the compromise of other protocols with an active user component. To rectify the issues discovered by our analysis, we present the provably secure Dual Passkey Entry protocol with the Secure Hash Modification for addition to the Bluetooth standardization. Dual Passkey Entry demonstrates that full CYBORG security is a realistic and achievable goal with limited change to defined protocols.				
14. SUBJECT TERMS Bluetooth, Authenticated Key Exchange, Cyborg Protocols, secure connections, Secure Simple Pairing Numeric Comparison, Passkey Entry, computational analysis, formal analysis, Tamarin				15. NUMBER OF PAGES 171
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified		18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified		19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified
20. LIMITATION OF ABSTRACT UU				

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**A CRYPTOGRAPHIC ANALYSIS OF BLUETOOTH'S HUMAN-MACHINE
AUTHENTICATED KEY EXCHANGE PROTOCOLS**

Michael E. Troncoso
Captain, United States Marine Corps
BS, U.S. Naval Academy, 2014

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN COMPUTER SCIENCE

and

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

**NAVAL POSTGRADUATE SCHOOL
September 2020**

Approved by: Britta Hale
Advisor

Pantelimon Stanica
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

Wei Kang
Chair, Department of Applied Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

As Bluetooth is firmly ensconced as one of the leading standardizations for wireless communication, it becomes imperative to rigorously quantify its security. To forward this quantification, we conduct a comprehensive analysis of Bluetooth's user-mediated authenticated key exchanges, Numeric Comparison and Passkey Entry, in both the computational and formal cryptographic settings. Due to the reliance on intertwined human and machine functions in the specification of these cyborg protocols, new attack vectors arise for hostile actors to exploit. Consequently, we model a realistic adversary, one not only with access to both the user-to-device interfaces and device-to-device communication channels simultaneously, but also with the capability to compromise device display and input mechanisms. Our analysis shows that while Numeric Comparison and Initiator/Responder-Generated Passkey Entry achieve at least basic levels of security in our model, User-Generated Passkey Entry is insecure in the model. Furthermore, the categories of attacks depicted herein function as a blueprint for the compromise of other protocols with an active user component. To rectify the issues discovered by our analysis, we present the provably secure Dual Passkey Entry protocol with the Secure Hash Modification for addition to the Bluetooth standardization. Dual Passkey Entry demonstrates that full CYBORG security is a realistic and achievable goal with limited change to defined protocols.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1	Introduction	1
2	Literature Review	7
2.1	Cryptographic Analysis	7
2.1.1	Formal Analysis.	7
2.1.2	Computational Analysis.	9
2.2	User Modeling in Cryptographic Analysis	13
2.2.1	Security Ceremonies	13
2.2.2	Formal Analysis of Numeric Comparison with ProVerif	14
2.2.3	Three-Party Possession User-Mediated Authentication Model.	14
2.2.4	Mediated Epoch Three-Party Authentication Model.	15
2.3	Security Analysis of Bluetooth	16
2.3.1	Analysis of Bluetooth SSP.	17
2.3.2	Relevant Attacks on SSP	18
2.3.3	Modeling Choices	23
3	Preliminaries	25
3.1	Hash Functions	25
3.2	Message Authentication Codes	26
3.2.1	Hash-Based Message Authentication Code	26
3.2.2	Strong Unforgeability under Chosen Message Attack	27
3.3	Elliptic Curves	28
3.4	Pseudorandom Functions	31
4	Key Exchange in Bluetooth	33
4.1	SSP Background	33
4.1.1	Device Input/Output Capabilities	33
4.1.2	Out-of-Band	35
4.1.3	Just Works	35
4.1.4	Numeric Comparison.	36
4.1.5	Passkey Entry.	36

4.2	Relevant Variables	37
4.2.1	Keys and Random Values	37
4.2.2	Tags	38
4.2.3	Constants	38
4.2.4	Elliptic Curves	38
4.2.5	Relevant Functions.	39
4.3	Protocol Description	39
4.3.1	Phase 0: Init and IOcap Exchange.	40
4.3.2	Phase 1: Public Key Exchange	41
4.3.3	Phase 2: Authentication Stage 1	41
4.3.4	Phase 3: Authentication Stage 2	44
4.3.5	Phase 4: Link Key Calculation	45
4.4	Correctness	45
4.4.1	Correctness of Numeric Comparison	45
4.4.2	Correctness of Passkey Entry	47
5	CYBORG Model for User-Mediated, Authenticated Key Exchange	51
5.1	Participant Model	51
5.1.1	Sessions	51
5.1.2	Partnering	53
5.2	Adversarial Model.	56
5.2.1	Communication Channels	56
5.2.2	Adversarial Queries	57
5.2.3	User and Device Freshness	59
5.3	CYBORG Security Experiment.	63
6	Results of Analysis under the CYBORG Security Model	67
6.1	Association between User Compromise Types	67
6.2	Numeric Comparison.	68
6.2.1	Numeric Comparison under Strong Session Modeling.	68
6.2.2	Numeric Comparison under Weak Session Modeling	69
6.3	Passkey Entry.	81
6.3.1	Passkey Entry under Strong Session Modeling.	81
6.3.2	Passkey Entry under Weak Session Modeling	84
7	Proposed Modification to Bluetooth Pairing	97

7.1	Secure Hash Modification (SHM).	97
7.1.1	Phase 0: Init and IOcap Exchange.	98
7.1.2	Phase 1: Public Key Exchange.	98
7.1.3	Phase 2: Authentication Stage 1.	98
7.1.4	Phase 3: Authentication Stage 2.	98
7.1.5	Phase 4: Link Key Calculation.	98
7.2	Dual Passkey Entry (DPE).	98
7.2.1	Phase 0: Init and IOcap Exchange.	99
7.2.2	Phase 1: Public Key Exchange.	99
7.2.3	Phase 2: Authentication Stage 1.	99
7.2.4	Phase 3: Authentication Stage 2.	100
7.2.5	Phase 4: Link Key Calculation.	100
8	Results of Analysis of Modified Bluetooth Protocol	103
8.1	SHM Initiator-Generated Passkey	103
8.2	SHM Responder-Generated Passkey	106
8.3	SHM User-Generated Passkey	106
8.4	SHM Dual Passkey	108
9	Formal Analysis in Tamarin	111
9.1	Protocol Model	111
9.1.1	Diffie–Hellman Modeling	112
9.1.2	Restrictions	112
9.2	Adversarial Model	113
9.2.1	Device-to-Device Channel.	113
9.2.2	User to Device Channel.	114
9.3	Formal Security	115
9.3.1	Numeric Comparison.	117
9.3.2	Passkey Entry.	118
9.4	Results of Formal Analysis	119
9.4.1	Modeling Secure Simple Pairing in Tamarin	119
9.4.2	Numeric Comparison.	131
9.4.3	Initiator-Generated Passkey Entry	132
9.4.4	Responder-Generated Passkey Entry	133
9.4.5	User-Generated Passkey Entry	134

10 Conclusions	137
10.1 Computational Analysis.	137
10.2 Formal Analysis.	139
10.3 Concluding Remarks	140
 List of References	 141
 Initial Distribution List	 149

List of Figures

Figure 2.1	<i>BT-Niño-MitM</i> Attack	19
Figure 2.2	Tap ‘n Ghost Attack	22
Figure 3.1	Strong Unforgeability of a MAC	28
Figure 4.1	Communication Chart for Device Discovery	40
Figure 4.2	Message Sequence Chart for the IO Capability Exchange	41
Figure 4.3	Numeric Comparison	43
Figure 4.4	Passkey Entry	49
Figure 5.1	Baseline CompUser_x -freshness	62
Figure 6.1	CYBORG- CompUser_b Attack against Numeric Comparison . . .	77
Figure 6.2	CYBORG- CompUser_c Attack against Numeric Comparison . . .	80
Figure 6.3	Role Confusion Attack against User-Generated Passkey Entry . .	84
Figure 6.4	CYBORG- CompUser_a Attack against Initiator-Generated Passkey Entry.	93
Figure 7.1	Dual Passkey Entry and the Secure Hash Modification.	101
Figure 9.1	Preamble for Translation of Numeric Comparison to Tamarin . .	120
Figure 9.2	Numeric Comparison, Phase 0 Translated to Tamarin	121
Figure 9.3	Numeric Comparison, Phase 1 Translated to Tamarin	122
Figure 9.4	Numeric Comparison, Phase 1 (cont.) and Phase 2 Translated to Tamarin	123
Figure 9.5	Numeric Comparison, Phase 2 (cont.) Translated to Tamarin . . .	124

Figure 9.6	Numeric Comparison, Phase 2 (cont.) and Phase 3 Translated to Tamarin	125
Figure 9.7	Numeric Comparison, Phase 3 (cont.) Translated to Tamarin . . .	126
Figure 9.8	Numeric Comparison, Phase 3 (cont.) and Phase 4 Translated to Tamarin	127
Figure 9.9	Numeric Comparison, Phase 4 (cont.) Translated to Tamarin . . .	128
Figure 9.10	Oracle for Analysis of Numeric Comparison	130
Figure 9.11	Initial Tamarin Output for Numeric Comparison	131
Figure 9.12	Subsequent Tamarin Output for Numeric Comparison	131
Figure 9.13	Tamarin Output for Initiator-Generated Passkey Entry	133
Figure 9.14	Tamarin Output for Responder-Generated Passkey Entry	134
Figure 9.15	Tamarin Output for User-Generated Passkey Entry	134

List of Tables

Table 4.1	Bluetooth IO Capabilities	34
Table 4.2	Default Bluetooth Authentication Methods	35
Table 10.1	Bluetooth’s Achieved CYBORG Security	137
Table 10.2	Bluetooth’s Formal Security	139

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

3-PUMA	Three-Party Possession User Mediated Authentication computational model
AKE	Authenticated Key Exchange
ATM	Automated Teller Machine
BR93	Bellare–Rogaway 1993 computational model
BR/EDR	Basic Rate / Enhanced Data Rate
BT-Nino-MitM	Bluetooth–NoInput/NoOutput–Man-in-the-Middle
CK01	Canetti–Krawczyk 2001 computational model
DDH	Decision Diffie–Hellman
DH	Diffie–Hellman
DoD	Department of Defense
DPE	Dual Passkey Entry
DtD	Device-to-Device
EC-DDH	Elliptic Curve Decision Diffie–Hellman
ECDH	Elliptic Curve Diffie–Hellman
eCK	extended Canetti–Krawczyk computational model
FDR	Failures Divergences Refinement Checker
FIPS	Federal Information Processing Standards
HMAC	Hash-based Message Authentication Code
HTTPS	Hypertext Transfer Protocol Security

IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IO	Input/Output
IoT	Internet of Things
IPsec	Internet Protocol Security
MAC	Message Authentication Code
META	Mediated Epoch Three-Party Authentication computational model
MitM	Man-in-the-Middle
NIST	National Institute of Standards and Technology
NFC	Near-Field Communication
NPS	Naval Postgraduate School
NSA	National Security Agency
OOB	Out-Of-Band
PE	Passkey Entry
PE-IG	Initiator-Generated Passkey Entry
PE-RG	Responder-Generated Passkey Entry
PE-UG	User-Generated Passkey Entry
PIN	Personal Identification Number
PPT	Probabilistic Polynomial Time
PRF	Pseudorandom Function
PRF-ODH	Pseudorandom Function Oracle Diffie–Hellman
PRF-OECDH	Pseudorandom Function Oracle Elliptic Curve Diffie–Hellman

QR	Quick Response
RFC	Request for Comments - IETF published technical and organizational notes pertaining to the Internet.
RFID	Radio Frequency Identifier
SC	Secure Connections
SHA	Secure Hash Algorithm
SHM	Secure Hash Modification
SIG	Special Interest Group
SSP	Secure Simple Pairing
SUF-CMA	Strong Unforgeability under Chosen Message Attack
TAP	Tag-Based Adaptive Ploy
TLS	Transport Layer Security
TTP	Trusted Third Party
USMC	U.S. Marine Corps
USN	U.S. Navy
UtD	User-to-Device
UtD-E	User-to-Device with Eavesdropping

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

Although only my name is transcribed at the beginning of this thesis, it was by no means a solo undertaking. For one, I had the privilege of working with two esteemed professors as my thesis advisors: Dr. Britta Hale and Dr. Pantelimon Stanica. I am indebted to Dr. Hale for her tireless work indoctrinating me to academic research and for her patient teaching methods. The many hours we spent discussing cryptography and breaking protocols proved instrumental to the completion of this thesis. I would also like to pay my special regards to Dr. Stanica for motivating me to pursue a dual degree. Dr. Stanica was the first teacher to pique my interest in cryptography and reignite my passion for learning. I would next like to thank my cohorts in both the Applied Mathematics and Computer Science departments. The bonds we built over these past 27 months learning difficult subjects, studying late, and researching interesting subjects will be something I cherish for years to come. Finally, I would like to express my deepest gratitude to my wife, Ashley. Her steadfast support and humourous crypto-quipps kept me motivated through the finish. Thanks again to everyone for making this such a rewarding experience!

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:

Introduction

Bluetooth pairing has become nearly ubiquitous in the daily lives of Americans since its launch in 1999 where it earned the “Best of Show Technology Award” at COMDEX [1]. Since then, its standardization has now progressed to version 5.2 (v5.2) with multiple modes of operation: Basic Rate / Extended Data Rate (BR/EDR) and Low Energy (LE). Bluetooth has its own suite of protocols to secure communications under the Secure Simple Pairing family, first published in Bluetooth v2.1 and updated to Secure Connections in v4.2, which 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. We term such user-mediated protocols *cyborg* protocols due to the joint reliance on human and machine functions to achieve a greater level of security than simply functioning without authentication.

Version History. Secure Simple Pairing (SSP) was developed in 2007 by the Bluetooth Special Interest Group (SIG) with v2.1 to provide greater security and simplify the pairing process for the user. This was mainly done as a result of the litany of attacks published against Bluetooth’s legacy pairing protocol [2]. Bluetooth v4.1 introduced Secure Connections (SC) which allowed for the FIPS approved elliptic curve P-256, vice only P-192, to be used in the pairing process for Diffie–Hellman key exchange [3]. Although Secure Connections is not required for use, all analysis in this study also applies to Secure Connections. Bluetooth also allows for two modes of operation: Basic Rate / Enhanced Data Rate (BR/EDR) and Low Energy (LE). Secure Simple Pairing in LE is derivative of the same pairing methods used in BR/EDR. However, this analysis cannot be directly applied to Low Energy due to differences in how each mode, BR/EDR and LE, generate the authenticated key at the conclusion of pairing. When we refer to “pairing” between devices, we describe how two devices authenticate each other and agree on a shared key.

Framing the Problem. The human user as a trusted third party is a natural option for settings where certificates cannot be used or reliably updated, such as in the Internet of Things. The sheer number of devices with varying levels of computing power

typically prevents a public key infrastructure from being viable. Thus, public keys used in CYBORG authenticated key exchange (AKE) protocols are typically short-lived and not associated with identities *a priori*. As all values may be ephemeral to the session, the user interaction acts as a trusted third party (TTP) and authentication authority, providing assurance to the respective devices of their communication partner. However, modeling the user presents a problem to analysis as there is no intuitive method for representing their actions while also being faithful to the protocol and previous modeling practices. Moreover, there are questions of how much capability a theoretical adversary should be given to affect communications between a user and a device in attempts to compromise the protocol. Although such *user-to-device* attacks have been published [4]–[6], they remain largely unclassified in contrast to the oft investigated device-to-device attacks. These two issues, modeling the user and classifying user-to-device attacks, demonstrate that prior analysis of CYBORG AKEs is not complete and further investigation is necessary.

Numeric Comparison. Numeric Comparison is a pairing method that requires all devices to be able to display or communicate a six-digit number to the user and accept at least binary inputs from the user. The user’s role in Numeric Comparison is to confirm that two values displayed by the devices to be paired match, and then signal this via an affirmation to the devices. This method also makes no use of a PIN or password, thereby simplifying the pairing process for the user by not requiring any memorization. While Numeric Comparison has been analyzed and proofs of security exist [7]–[10], we will conduct a novel investigation encompassing the full AKE while also modeling user interactions.

Passkey Entry. Passkey Entry [11] relies on the generation and sharing of a random value (a *passkey*) via the user to achieve entity authentication. Investigations into the security of Passkey Entry have been largely ad-hoc [12]–[14], involving exploits in the re-use of random values across multiple executions of the protocol. While these attacks could be rendered moot by requiring ephemeral passkeys in keeping with Bluetooth’s recommendation; this change alone does not provide guarantees of security. Since a systematic analysis of Passkey Entry is lacking, it is still an open question whether Passkey Entry has other unidentified vulnerabilities.

Capturing the User. Furthermore, user communications are regularly treated as out-of-band in the analysis of cryptographic protocols, but recent attack techniques that specifically

target user-to-device communications demonstrate the criticality of including a treatment of user-to-device channels in protocol analysis. Notably the *Tap ‘n Ghost* attack [4], which specifically targets Bluetooth devices, necessitates a treatment of user communications as in-band. Tap ‘n Ghost is notable for demonstrating a break that not only induces control of the device display, but also corrupts a honest user’s inputs to the device without actively attacking device memory. Leading AKE models do not capture user-to-device attacks like Tap ‘n Ghost, due to the sole focus on device-to-device communications.

Cryptographic Modeling Techniques. Cryptographic analysis of protocols mainly falls into two competing schools: formal modeling and computational modeling. Formal modeling has its roots in Dolev and Yao’s seminal paper [15] investigating public key protocols. Meanwhile, computational modeling draws from a similarly pivotal paper from Bellare and Rogaway [16] on authentication and key exchange. To differentiate formal and computational modeling without going into too much detail at this early stage, one major difference is how these schools model cryptographic messages. Under formal modeling, messages are uniquely identified via symbolic expressions from a defined alphabet. Formal modeling also makes the assumption that cryptographic algorithms (such as encryption) behave as they would in an ideal setting (i.e., all encryption is perfect and functions like a one-time pad). Meanwhile, a message is uniquely identified by its bit-string in the computational setting. Cryptographic algorithms are also assumed to be imperfect and are deemed secure by how computationally hard it is for an adversary to break defined security properties. These distinctions has far-reaching consequences in analysis. While both formal and computational models capturing user-to-device communications in the analysis of CYBORG-style authentication protocols have emerged [9], [17], [18], modeling for key exchange is lacking until now.

Contributions. The above paragraphs present three critical gaps: the provable security of Numeric Comparison, the provable security of Passkey Entry, and a suitable model for CYBORG AKEs. We address these gaps, presenting an analysis of Bluetooth Basic Rate / Extended Data Rate (BR/EDR) versions 2.1–5.2 Numeric Comparison and Passkey Entry (abbreviated Numeric Comparison and Passkey Entry respectively) in both the formal and computational settings, which covers both device-to-device and user-to-device communications. This analysis applies to Numeric Comparison and Passkey Entry under both the Secure Simple Pairing or Secure Connections Bluetooth security framework. We introduce

a computational model for AKEs to do this, capturing adversaries capable of exploiting user-to-device transmissions. Abusing notation, we call this the CYBORG model. We also adapt existing infrastructure provided by the formal modeling tool, Tamarin-Prover [19], to capture both device-to-device and user-to-device communications for a formal analysis.

Our specific contributions include the following.

- Provide the first security framework in computational analysis for CYBORG AKEs, where the user is an active participant in protocol execution.
- Propose a method for capturing the user’s role within a protocol using Tamarin-Prover for a formal analysis of CYBORG AKEs.
- Provide the first systematic separation of all variants of adversarial capabilities against user-to-device channels.
- Conduct the first computational analysis of Numeric Comparison (in its entirety) for Bluetooth BR/EDR, under either Secure Simple Pairing or Secure Connections, that addresses both device-to-device and user-to-device communications.
- Conduct the first computational analysis of Passkey Entry (in its entirety) for Bluetooth BR/EDR, under either Secure Simple Pairing or Secure Connections, that addresses both device-to-device and user-to-device communications.
- Demonstrate that Numeric Comparison achieves only the lowest level of CYBORG security in our model.
- Demonstrate that all current versions of Passkey Entry fail to provide security under any CYBORG model variant unless liberties are taken to restrict the information analyzed for authentication.
- Propose the Secure Hash Modification to Passkey Entry and demonstrate that the modified protocol provides security improvements across all passkey generation variants.
- Present Dual Passkey Entry, where the initiator and responder both generate a passkey to be shared via the user, and demonstrate that, composed with the Secure Hash Modification, the new protocol provides the highest-level CYBORG security in our model.
- Conduct the first formal analysis of Passkey Entry (in its entirety) for Bluetooth BR/EDR, under either Secure Simple Pairing or Secure Connections, using Tamarin-Prover that addresses both device-to-device and user-to-device communications.

Outline. This thesis proceeds as follows. In Chapter 2, we survey relevant literature on the above topics and introduce pertinent concepts to the reader. In Chapter 3, we define important mathematical properties applying to hash functions, message authentication codes, elliptic curves, and pseudorandom functions for future use in both computational and formal proofs. In Chapter 4, we summarize Bluetooth’s security terms, definitions, and functions as they pertain to Numeric Comparison and Passkey Entry. We also present the full protocols at this stage. In Chapter 5, we present the CYBORG model for the computational analysis of AKEs. In Chapter 6, we present our computational analysis of Numeric Comparison and Passkey Entry under both weak and strong session modeling within the CYBORG model. In Chapter 7, we propose the Secure Hash Modification and Dual Passkey Entry additions to the Bluetooth specification. In Chapter 8, we prove that the original variants of Passkey Entry achieve an improved degree of CYBORG security in conjunction with the Secure Hash Modification and that Dual Passkey Entry with the Secure Hash Modification achieves the highest classification of CYBORG security. In Chapter 9, we present our techniques for formal modeling CYBORG protocols within Tamarin-Prover and the results of our analysis. In Chapter 10, we summarize the findings of this thesis and suggest avenues of future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:

Literature Review

Our review of applicable research and literature on our subject is split into three main sections. First, we will discuss the various methods employed to analyze cryptographic protocols. These can be organized into two main schools of thought: formal analysis and computational analysis. Second, we will review what other research has been conducted on the user's role in authentication and key exchange. Finally, we will present an overview of research into Bluetooth's security protocols and relevant findings on the various security measures employed by Bluetooth throughout its history. A major contribution of this paper is in the area of user modeling and thus we feel it necessary to highlight the work of others to advance this topic.

2.1 Cryptographic Analysis

In the following section, we present foundational papers for both formal and computational analysis, and further applications of the proposed models. We also briefly discuss automatic tools that implement both methods of analysis and have proven successful in their efforts.

2.1.1 Formal Analysis

One view of analysis that deserves mentioning is formal analysis. Formal analysis (also referred as symbolic analysis) uses the theory of formal language as an analytical framework.

All strings, numbers, and messages are now abstracted to be formal expressions as part of an underlying grammar that includes cryptographic algorithms as idealized functions. Thus, formal analysis sacrifices rigor for simplicity and is limited to only discovering logical faults in cryptographic protocols. Though formal analysis has applications in automated protocol analysis and for showcasing a protocol's logical security, it will not be focused on for this research but will be used to confirm the computational results. Computational analysis will be the focal point due its more rigorous definitions of security and its underlying mathematical framework. Although the preciseness of computational analysis is arguably a detriment to the modeling of user-to-device communications due to its inherent complexity,

accurately accounting for the user is an improvement of such protocol analysis as it more closely models the real world.

Dolev–Yao Model

The modeling method introduced by Dolev and Yao in their 1983 paper laid the groundwork for future work in formal analysis [15]. They had the initial foresight to model encryption and decryption as *idealized functions*. An idealized function is one that operates exactly and perfectly as theoretically defined. For instance, an ideal encryption function would yield no information on the associated plaintext or key to an adversary given access to the ciphertext. Other steps were taken by Dolev and Yao to enable a more formulaic approach to cryptographic analysis. For one, the adversary was modeled as having complete control of communications sent from device to device it is now commonplace to call this type of adversary the “Dolev–Yao adversary”. In addition, all texts and keys are modeled as *formal expressions* vice strings of bits. Formal expressions being a concatenation of symbols from a defined alphabet and stemming from formal language theory [20].

The motivation behind their work was to attempt to model the active adversary in public key protocols: something that had not been actively pursued at the time of their research. Ultimately, this paper was a major step in the continued analysis of cryptographic protocols against ever-stronger adversaries.

Shortly after publication of Dolev and Yao’s 1983 paper, Even and Goldreich extended the model to examine multi-party ping-pong protocols [21]. Another early use of the Dolev–Yao model was in the analysis of the Needham–Schroeder Public Key Authentication Protocol by Lowe [22]. Lowe showed that a Man-in-the-Middle (MitM) attack initiated by an active adversary could break the authentication of the protocol by using one of the participants as a decryption oracle. The paper ultimately led to the recommendation of including identifying information in the plaintext to be encrypted, for which Lowe proved security using the Failures Divergences Refinement Checker (FDR) [23]. FDR was an automatic verification tool initially developed by the Office of Naval Research as a model verification tool for state machines [24]. Although FDR is not currently in use, Lowe’s initial employment of it to analyze the Needham–Schroeder protocol demonstrated the efficacy of automatic verification tools.

Automatic Tools for Formal Verification

One limitation of the Dolev–Yao model is that it can only prove logical security of a protocol since all analysis is performed under idealized conditions. However, the idealization of cryptographic primitives proposed by the model initiated research into the development of automatic tools as shown by FDR [24]. The Interrogator [25] and the NRL Protocol Analyzer [26], both coded using the Prolog language, were two of the first such developments. ProVerif, Scyther, and Tamarin are three more modern automatic verification tools that have seen widespread use in recent formal analyses [27]–[29]. ProVerif is also based off the Prolog language and was initially developed by Blanchet. We examine a specific application of ProVerif to Bluetooth in Section 2.2.2, but it was also used to examine Kerberos [30]. Scyther, originally developed by Cremers, included the novel additions of unbounded verification and guaranteed termination. The flexibility of the program allowed its author to investigate multi-protocol attacks and some of the protocols in the ISO/IEC 11770 and 9798 standards [31]–[33]. Tamarin is a newer entry to the automatic verification market and was developed by a team of researchers to include Basin, Cremers, Dreier, Meier, Sasse, and Schmidt as a successor to Scyther [19]. Tamarin comes with honest modeling of modern key exchange methods to include Diffie–Hellman (DH) key exchange and bilinear mappings; it also allows a user to define his own security properties and adversarial queries for use in analysis. The advances in the field of automatic verification provided by Tamarin enabled the most comprehensive symbolic modeling of the TLS 1.3 protocol at the time by Cremers, Horvat, Scott, and van der Merwe, which provided supporting evidence for the inclusion of the “Finished” messages into the TLS 1.3 handshake [34].

2.1.2 Computational Analysis

The other leading modeling framework when analyzing cryptographic protocols is computational analysis. Under this framework, strings, numbers, keys, and messages are all taken to be bits. Cryptographic functions are then simply algorithms that act on these bits to produce their output. There are three main pieces needed for computational analysis: description of the protocol, adversarial capabilities, and a win condition. We note that although formal analysis also includes these three pieces, its inherent assumptions (like formal expressions and idealized functions) restrict the modeling choices. The first is a description of the protocol and a presentation of its various components. This stage typically involves defining relevant terms specific to the protocol, any assumptions, and a depiction of all

communication flows. Additionally, a method for capturing execution of the protocol and its participants is also presented; recent applications of computational analysis typically employ session oracles that store relevant protocol-specific information for this piece. The second part of a computational model involves defining the adversary, modeled as a Turing machine. We define adversarial capabilities and give him access to oracle queries to control execution of the protocol to be analyzed and learn secret information. The adversary operates opposite a challenger, who is in complete control of the protocol environment and oversees the answering any adversarial queries. The last major piece to a computational analysis is the definition of the security game. At this point, the cryptographer formulates the winning condition(s) for the adversary. One can then demonstrate how well or poorly a protocol limits the ability of the adversary to win. This often leads to a presentation of provable security for the protocol under the given model via reductions to complexity hardness assumptions. The development of computational models thus arises from how one defines these three major pieces and applies them to the analysis of cryptographic protocols.

Bellare-Rogaway Model

Computational analysis has its roots in a foundation-laying paper by Bellare and Rogaway [16]. In their paper, Bellare and Rogaway used the idea of matching conversations, whereby communication partners come to an agreement on the messages exchanged in a protocol instance, to capture what it means for intended communication devices to be *partnered* in execution. Partnering is relevant to computational analysis because it enables the cryptographer to analyze whether two devices that engaged with each other through a protocol and *should* achieve the goal of the protocol (mutual authentication, authenticated key agreement, etc.) *actually* achieve it in the presence of some defined adversary. The writers then used their newly developed tools to analyze the provably secure protocols MAP1 (a mutual authentication protocol), and AKEP1/2 (authenticated key exchange protocols). We will use BR93 as an acronym when referring to this model in future points within this paper.

The BR93 model has seen multiple applications to common protocols and was vital to the development of the Internet Key Exchange (IKE) standard. We direct the reader to Krawczyk's security analysis of SKEME [35], a protocol for secure key exchange developed through the Internet Engineering Task Force's (IETF) work toward the standardization of the Internet Protocol Security (IPSec). IKE was eventually built and standardized in RFC

2409 [36] using ideas and techniques inherent in the SKEME protocol, along with the Oakley [37] and ISAKMP [38] protocols. In addition, Abdalla and Pointcheval used an adapted version of the BR93 model to show that their password-based encrypted key exchange protocols SPAKE1 and SPAKE2, attain provable security [39]. Lindell also used the BR93 as a basis for his security model that analyzed Numeric Comparison v2.1 [13], a paper we give further attention to in Section 2.3.1. Outside of direct application to the analysis of security protocols, the BR93 model is oft cited in cryptography when undertaking a computational analysis proof as the progenitor of more modern models that we now discuss.

Canetti-Krawczyk Model

Canetti and Krawczyk furthered Bellare and Rogaway’s work with the publication of their evolved model in 2001 [40]. The CK01 model, as we will refer to it in this paper, pivoted away from the idea of matching conversations used in BR93 and instead presented the idea of session identifiers. Instead of relying on matching on all messages exchanged between two devices, the cryptographer defines relevant messages or computed values to be checked for matching. A session was defined to be a single instance of a protocol run at a party and they distinguished instances through a tuple of the identity of the participant running the protocol, the identity of its assumed partner in the protocol, a unique session identifier, and role. Two instances can then be identified to be matching if the parties involved believe they are talking with each other, their roles in the protocol are not equal, and their session identifiers are the same. The idea of using session identifiers will become useful in our own computational analysis of Bluetooth as defining what constitutes a matching conversation can prove difficult in user-mediated protocols. Canetti and Krawczyk also furthered the development of adversarial models. One query they included in their adversarial model was the *Session-State Reveal* query; this call would allow an adversary to obtain internal state information, as defined by the protocol, of incomplete sessions. In contrast to queries like *Reveal*, which give access to computed session key(s), or *Corrupt*, which gives access to long-term private key(s), *Session-State Reveal* can give access to any ephemeral information the cryptographer allows in analysis. They also introduced the notion of session expiration as a way to analyze whether a protocol achieves *forward secrecy*, which is when compromise of one session key does not compromise past session keys. This was investigated by the protocol action of deleting any named session key from device memory and analyzing if the adversary could distinguish these *expired* keys from random. Their developments allowed

them to propose the SIG-DH protocol, an authenticated key exchange provably secure in their model.

An instance of the SIGMA protocol, short for "SIGn-and-MAC" and originally proposed by Krawczyk [41], was analyzed under a post-specified version of the CK01 model in [42] as a result of its inclusion in IKE, following the discovery of security vulnerabilities in IKE's original form [43]. Aiello, Bellare, Blaze, Canetti, Ioannidis, Keromytis, and Reingold also used a version of CK01 to analyze the JFK protocol [44], a protocol that also combines signatures and MACs to achieve authenticated key exchange. These applications and others ultimately proved that the CK01 model was a worthy compliment to the BR93 model in the field of computational analysis.

Extended Canetti-Krawczyk

In 2007, LaMacchia, Lauter, and Mityagin published what they termed an extension of the CK01 model, colloquially referred to as the extended Canetti-Krawczyk model (eCK) [45]. In their paper, LaMacchia et al. reasoned that a party only has two possibilities of secrets, ephemeral and long-term. This led them to scrap the previously proposed session-state reveal query for an ephemeral key reveal query in their adversarial model. They reasoned that so long as an adversary does not have both the ephemeral and long-term secrets of a party, then a protocol should remain secure. The NAXOS protocol, which included the novel idea of using the hash of both a party's ephemeral and long-term secrets as a Diffie-Hellman exponent, was born out of their developments in computational analysis and was also proven secure in eCK. The combination of the ephemeral and long-term secrets via a hash function has since been coined the "NAXOS trick" due to its prevalence in protocols proven secure in the eCK model. The eCK model proved to be the first that placed strong emphasis on securing protocols against leaking ephemeral secrets and has seen widespread application to other protocols that rely on the NAXOS trick for security [46], [47]. The NAXOS trick is important because it combines ephemeral and long-term keys in such a way that reveal of any one does not allow for computation of the session key. This combination combines the explicit authentication provided by long-term keys and the freshness provided by the ephemeral keys. The model also sparked a discussion on the *random oracle model*, which assumes that a hash function produces random outputs for distinct inputs. The random oracle model was necessary for the proof of NAXOS under eCK as a result of the NAXOS

trick. A desire to avoid use of the random oracle model led to the production of protocols secure in the eCK model absent an application of the NAXOS trick [48], [49].

2.2 User Modeling in Cryptographic Analysis

2.2.1 Security Ceremonies

Introduced by Ellison in 2007 [50], security ceremonies are defined to be network protocols that also include human beings as possible nodes in the network and includes communication avenues for computer devices to users and users to users. Through his definition of ceremonies, Ellison makes the claim that there are no OOB communications; all possible network communications are from device-to-device, user-to-device, or user-to-user. The lack of OOB communication functions as a starting point in an analysis of user interaction in HTTPS and e-mail ceremonies. Ellison's analysis raises concerns about the security of said ceremonies due to their reliance on honest user interaction and the assumption that adversaries cannot affect user-to-device and user-to-user communication.

Carlos, Martina, Price, and Custódio [10] expand on Ellison's original ideas and present a formalization of adversarial capabilities for security ceremonies in keeping with the Dolev-Yao model. Defined capabilities include "Eavesdrop," whereby an adversary can listen on any ceremony communication channel, and "Spoof," whereby an adversary can assume the role of a participant in a ceremony and send messages on his behalf. They go on to employ their new model to analyze Numeric Comparison and describe three possible attack scenarios that compromise the user-to-device and user-to-user communication channels. Their first attack is similar to an attack we propose under the CYBORG security model herein that involves an attacker with the capability to spoof messages from a device to a user. We continue to build on preceding work in ceremonies by applying the ideas of Ellison and Carlos et al. within a computational setting and present adaptations of the Spoof capability as adversarial queries within a computational model. We will also henceforth refer to all communications from device to device as occurring over the Device-to-Device (DtD) channel and all communications from user to device (or device to user) as occurring over the User-to-Device (UtD) channel. We use the term channel to refer to the transmission medium that DtD or UtD communications transverse. In regard to the DtD channel, this is typically via radio waves. For the UtD channel, the communications have greater variance

in transmission medium. Common examples being a message from a device to a user transmitted visually (e.g. reading a device display) or a message from a user to a device transmitted via internal electronic signals (e.g., keyboard or button presses).

2.2.2 Formal Analysis of Numeric Comparison with ProVerif

Although some cryptographic analysis has been done on the Secure Simple Pairing communication protocol [7], [8], [51], the user's role in the protocol is typically ignored. Chang and Shmatikov broke with this practice in a formal analysis of Numeric Comparison [9] and found a previously unknown break in the protocol using ProVerif as a result. The issue arises when the same device is allowed to execute concurrent sessions and the user is unable to associate the comparison values with a specific session. Of note, the current Bluetooth specification places no requirement on implementors to distinguish these values for the user during Numeric Comparison as of the most recent version [11]. They modeled the user as its own process within ProVerif that communicated over secure channels, meaning the adversary could not read or write to channels used by the user. Chang and Shmatikov saw this as a necessary formalization to model the idea of the user looking directly at the screen, but the specification places no restriction on whether the adversary can replay or delete information on the UtD channel. Modeling the user as its own process is mirrored in this work by modeling the user as its own session oracle that interacts with not only other devices in the protocol, but also the adversary.

2.2.3 Three-Party Possession User-Mediated Authentication Model

Hale developed the 3-Party Possession User Mediated Authentication (3-PUMA) model to formalize the user-to-device interactions required in the ISO 9798-6 Mechanism 7a authentication protocol [17]. In this protocol, it was the purpose of the protocol to demonstrate data possession on behalf of the devices. However, the inclusion of an active third-party lead to the development of 3-PUMA as no other computational model could accurately capture user actions in a rigorous fashion. Additionally, Hale distinguished between communications sent over the DtD channel and the UtD channel. This was done to allow the model to differentiate an adversary's capabilities over the two distinct channels. Another important contribution of Hale's paper is initial research into what should contribute to the session identifier of a CYBORG protocol, settling on messages sent between devices and sent from

the user to both devices. Her investigation into the above was a major contributing factor to our own algorithm for the construction of a relevant, user-device transcript; the user-device transcript could then be used to rigorously define matching conversations between devices.

2.2.4 Mediated Epoch Three-Party Authentication Model

Dowling and Hale further expanded computational analysis of user-mediated protocols with an examination of the Signal protocol under their Mediated Epoch Three-party Authentication (META) model. The META model advances analysis of user mediated protocols by allowing the adversary queries that affect UtD communication. This is a logical next step in analysis with the increased prevalence of real-world attacks on the UtD channel described previously. Specifically, they introduced the ShowUser and ControlUser queries, which allow an adversary to modify or create messages from a device to the user and from the user to any device, respectively. They separate adversarial attacks into two threat settings: the compromised device setting (CompDev) and the compromised user (CompUser) setting. The motivation behind this development was to model two types of adversaries. The compromised user setting models an adversary that is unable to learn secret device state information but can affect how the user and device communicate with each other via malware, social engineering, etc. The compromised device setting models the reverse, an adversary capable of learning device state information via a side channel attack or other means, but cannot affect how it communicates with the user.

In our own research, we build upon the groundwork laid by Dowling and Hale with an analysis of Bluetooth and the development of the CYBORG security model. Bluetooth's multiple modes of communication with varying types of interaction with the user lends itself to an analysis similar to the one Dowling and Hale undertook when examining Signal. In META, the ControlUser query enabled the adversary to create and modify messages from the user to any device and any session. However, social engineering attacks tend to only apply over a finite amount of time vice forever. Additionally, Ghost-like attacks are confined spatially by those in contact with the Ghost Touch generator vice all devices. We therefore make the decision to tie the ControlUser query to a single device and session vice any. We see this as a better approximation of defensible real-world attacks, such as where an attacker may have the means to launch a Ghost-like attack on one device's touch screen but not the other.

In addition, the scoping of the `ControlUser` and `ShowUser` queries have consequences when defining user freshness and leads to a natural expansion into four baseline variants of user freshness: compromise of the communication channel from the initiating device to the user (type a), the responding device to the user (type b), the user to the initiating device (type c), and the user to the responding device (type d). We can consider the Tap ‘n Ghost attack as an example this model captures. This attack involves compromise of communications from a single device to a user and from the user to the same device. Thus, we can model this attack as a simultaneous combination of either both a type a and type c compromise (CompUser_{ac}), if the initiating device is being targeted, or a type b and type d compromise (CompUser_{bd}), if the responding device is being targeted. In our model, the action of executing the Tap ‘n Ghost attack corresponds to both a `ShowUser` query and a `ControlUser` query involving the same device, as shown in Figure 2.2.

Though we expanded the `CompUser` environment, we did not apply the `CompDev` environment to this research as investigating the security of modern AKEs when devices are allowed to be fully compromised is a wasted endeavor. This is because *at some point* the device executing the AKE will have to compute a session key. However, in the `CompDev` environment the adversary could simply reveal the device secrets used to compute this session key and render its secrecy moot.

2.3 Security Analysis of Bluetooth

When discussing Bluetooth’s cryptographic protocols we take a two-pronged approach. First, we present work on the provable security of Bluetooth as conducted in a computational setting. In light of this presentation, we then proceed to discuss relevant attacks on Bluetooth that exploit security vulnerabilities. This construction demonstrates that the analysis conducted on Bluetooth has not fully captured its vulnerabilities and the advanced adversaries present in modern scenarios. This thesis intends to capture these additional vulnerabilities with the CYBORG security model that implements an advanced adversary and does not side-step the user’s role in the protocol.

Secure Simple Pairing was first published in version 2.1. Secure Simple Pairing introduced four methods of authenticated key exchange dependent on device input/output (IO) capabilities: Just Works, Out-of-Band, Numeric Comparison, and Passkey Entry. Devices could

then agree on the specific protocol to execute based on the exchange of IO capabilities to ensure they could be executed properly. Secure Simple Pairing was subsequently updated to Secure Connections in version 4.2, which provided functionality for the Federal Information Processing Standards (FIPS) approved elliptic curve P-256, vice only P-192, during pairing. Of these, Numeric Comparison and Passkey Entry both rely on the user playing an active role in an authenticated key exchange (AKE).

2.3.1 Analysis of Bluetooth SSP

Lindell performed the first computational analysis of Bluetooth Numeric Comparison under v2.1 in 2009 using an adapted version of the BR93 model that incorporated the idea of session identifiers from the CK model. In his adaptations, he restricted the session identifier to be only the exchanged public keys during the Diffie–Hellman key exchange. In addition, he did not allow for any leakage of long term, ephemeral, or session-state information by the adversary. Under his model, he also showed that allowing session oracles to run multiple executions of the protocol became meaningless and therefore allowed him to restrict his analysis to only the single execution setting. The security game then encompassed having the adversary win by either non-negligibly guessing the link key or non-negligibly having two session oracles accept without matching session identifiers. This then led to a proof of security reliant upon the DDH assumption for the Diffie–Hellman group used in the protocol, the 2-universality of SHA-256, and the computational binding, non-malleability, and pseudorandomness of HMAC-SHA-256. The proof presents a great first step in the analysis of Numeric Comparison but in keeping with our advancement in knowledge of what constitutes a “secure” protocol it is imperative that we revisit the Bluetooth protocol.

Sun, Sun, and Yang conducted a two-part computational analysis of Numeric Comparison [8], [51]. The first part shares many similarities with Lindell’s original proof of security for v2.1 with a couple exceptions [8]. For one, the analysis allowed for devices to run multiple instances of the pairing protocol sequentially vice only be allowed to run a single instance. Sun et al. focused on the adversary’s capability to distinguish the derived session key from random. Their results showed that v5.0 of Numeric Comparison satisfied a similar level of security as originally shown by Lindell, and was secure under their model given that HMAC-SHA-256 is a collision resistant pseudorandom function and the Diffie–Hellman group used in pairing satisfies the DDH assumption. The second part conducts computational analysis

into the privacy of Bluetooth pairing in the low energy setting [51], where privacy is defined to be whether an adversary can derive device identity. They then show how Bluetooth pairing in the low energy setting leaks device identity if devices use long-term values for their respective secret and public keys; this leads to the natural improvement in the protocol to require these keys to be ephemeral.

2.3.2 Relevant Attacks on SSP

Many published attacks exist against Bluetooth that attempt to circumvent its security guarantees. While the Bluetooth Special Interest Group (SIG), the governing body behind Bluetooth standardization, claims active MitM attacks have at most a one-in-a-million chance at attacking the Numeric Comparison and Passkey Entry protocols, researchers have published methods for greatly increasing an adversary's odds [11]. In this section, a few of the most relevant attacks on Bluetooth pairing are presented with possible countermeasures.

NIST Summary of Security Threats and Vulnerabilities

Before examining some particularly devastating attacks that warrant an in-depth exploration, we begin with a discussion into the National Institute of Standards and Technology's (NIST) "Guide to Bluetooth Security" that summarized the security modes adopted by Bluetooth and possible security threats and vulnerabilities [3]. NIST identified 27 exploitable vulnerabilities in Bluetooth's pairing protocols. Some vulnerabilities of note include static or weakly generated Diffie–Hellman keys and passkeys, presence of downgrade attacks, and the ability to conduct unbounded authentication attempts. Furthermore, two attacks we examine in the forthcoming sections present real-world examples of the issues involved with the reuse of passkeys and allowing devices to downgrade to security modes that provide no protection from an active adversary. Some other focused security threats include *Bluejacking*, a message-sending attack akin to phishing, and fuzzing attacks, a form of side channel attack on a Bluetooth-enabled device's radio. The risks are still great when using Bluetooth and necessitate the need for better understanding of the protocol's computational security. One could conceivably argue that the confluence of the above vulnerabilities means that Bluetooth is insecure. Although this may be true, reaching this conclusion requires a structured analysis not simply a listing of vulnerabilities and attacks. NIST attempts to improve user awareness and mitigate its presented security threats and vulnerabilities with a 37-point

checklist for users to reference when looking to implement countermeasures.

BT-Niño-MitM Attack

Haataja and Toivanen produced possibly the most famous attack on Secure Simple Pairing termed the *BT-Niño-MitM* attack. This was an exploit designed to avoid Numeric Comparison, Passkey Entry, or Out-of-Band pairing altogether by intervening during the IO capability exchange and forcing the devices into the Just Works protocol, which provides no active MitM protection [52]. The design of the attack is displayed in Figure 2.1.

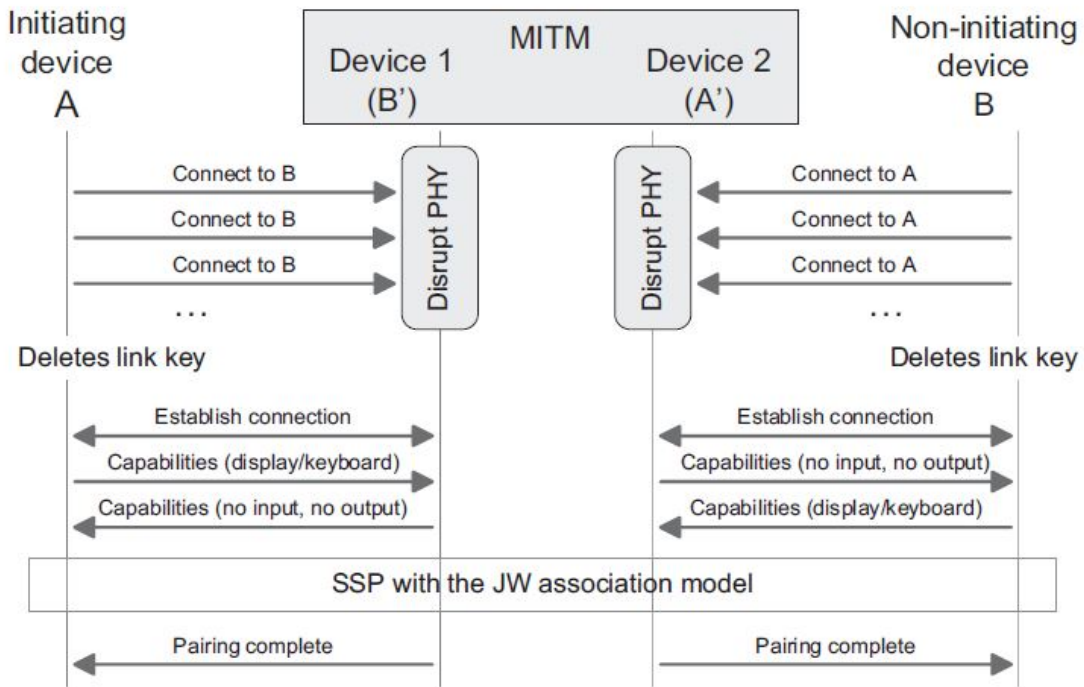


Figure 2.1. Diagram of attack flows for *BT-Niño-MitM* attack. Source: [52].

The physical layer disruption attack is only employed if a pairing already exists among the devices. In such a case, the attacker jams communications between the two devices and forces the user to attempt a new pairing and link key generation. This is done to replace the previously generated link key, which is presumably secure, with a compromised one. From here, the attacker performs a MitM attack during the IO capability exchange to force the devices into the Just Works pairing protocol where the MitM attack can continue unabated

by the user. In terms of application to our own analysis, we rule out the possibility of this attack by assuming honest exchange of pre-protocol information.

The primary solution to this type of attack proposed by Haataja and Toivanen involves a command window for the user to act on following the IO capability exchange, “*The second device has no display and keyboard! Is this true?*” From here, the user can decide whether to proceed with the pairing via a “*Proceed*” command or halt the pairing due to suspected adversary involvement via a “*STOP*” command [52]. This solution places more emphasis on the user to thwart the *BT-Niño-MitM* attack but also still enables Just Works as a possible pairing option, which has certain compatibility benefits. From a usability perspective, the greater dependence on the user in this solution has perhaps prevented widespread adoption in updated Bluetooth specifications. Another unique solution to the problem posed by the *BT-Niño-MitM* attack is to implement a server that ties a device’s radio frequency identifier (RFID) to Bluetooth device identity and can be securely communicated with to ensure malicious devices are not present [53]. It is important to note that the proposal is scaled to be implemented locally and is infeasible at a large scale; however, it could prove viable in a local operational setting for USMC devices.

Re-Used Passkey Attack

As pointed out by NIST in [3], the reuse of secret material in a cryptographic setting can prove devastating to protocols; Lindell was the first to demonstrate that the reuse of the random key in Passkey Entry can lead to a MitM attack on the protocol [13]. The attacker simply passively eavesdrops on a pairing protocol, learns the bits of random password by comparing commitment values with their respective nonces, then mounts a MitM attack using the derived passkey. Barnickle, Wang, and Meyer expanded on Lindell’s results with their development of *Bluetrial* to mount the attack in a feasible amount of time and with a 90% success rate; thereby cementing the reused passkey as a viable vector to exploit in real-world scenarios [12]. Finally, in 2018, Sun, Mu, and Susilo showed that the attack was still viable under v5.0 and proposed a novel solution to the problem [14]. Their recommendation involved combining the user generated random value, termed the passkey, communicated in the beginning of Phase 2 with the Diffie–Hellman key, DHKey, generated at the conclusion of Phase 1 to produce a new random value r^* that they concluded to be secret even under the assumption that the original random value is re-used.

Fixed Coordinate Invalid Curve Attack

Biham and Neumann recently published an attack on Secure Simple Pairing they termed the *Fixed Coordinate Invalid Curve Attack* that exploits the non-authentication of the y -coordinate of the DHKey and allows for either complete denial of service or total compromise of the generated long-term key. This accomplished through a MitM attack during the public key exchange step of Secure Simple Pairing wherein the adversary intercepts the public keys and sets each point's y -coordinate to 0.

Since $PK_{by} = 0$, by definition we have that $2PK_b = O$ where O is the point at infinity for our curve E (see Section 3.3 for preliminary work on elliptic curves). Computation of the DHKey then becomes trivial under the assumption that the initiating device's secret key is even, something that should reasonably happen 50% of the time:

$$\begin{aligned}\text{DHKey} &= [SK_a]PK_b \\ &= (2k)PK_b = (k)2PK_b \quad \text{for some } 0 \leq k < p \\ &= (k)O = O \quad .\end{aligned}$$

An easy-to-implement solution that Biham and Neumann recommend, though surprisingly rare in practice, would be to validate that the received public keys satisfy the elliptic curve equation [54].

Tap 'n Ghost

Maruyama, Wakabayashi, and Mori recently published the first active attack on mobile devices that specifically exploited the touchscreen, which they termed "Tap 'n Ghost" [4]. The attack has multiple requirements in order to be viable as presented in their research: the mobile device to be attacked needs to be an Android smartphone enabled with NFC and the phone needs to be unlocked. To execute the Tap 'n Ghost attack, Maruyama et al. pre-built a table for users to place their mobile devices onto that could execute the "Tap 'n Ghost" attack unbeknownst to the user. The "Tap 'n Ghost" is then a two-pronged attack on a user's device. We display a diagram depicting a high-level view of the attack in Figure 2.2. First the attacker executes a Tag-based Adaptive Ploy (TAP) attack. The TAP attack works by using NFC communication to force a pop-up on a user's device requesting to connect to a Wi-Fi port or Bluetooth device. Next, a Ghost Touch Generator attacks the capacitive

touchscreen of the user device and corrupts user input to force the device to believe the user selected an option to proceed with the connection. Once the adversary has successfully enticed a connection to a malicious device, he then gains full control of the user's device. This is a devastating attack to Bluetooth. Maruyama suggest various software improvements to NFC and touchscreens to help mitigate the efficacy of their attack. Preventing the physical corruption of a table proves to be a much harder problem with no efficient solution proposed by the authors.

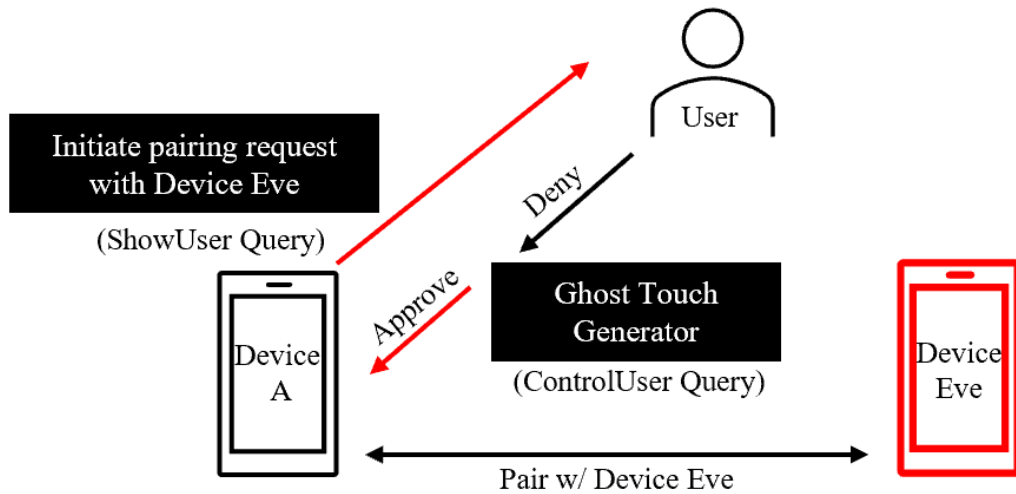


Figure 2.2. Tap ‘n Ghost. Adversary uses a Tag-based Adaptive Ploy to display a message from device A to the user requesting to pair to a malicious device Eve. When the user denies this request, a Ghost Touch Generator corrupts the input and forces acceptance by device A. These actions map to a ShowUser query and ControlUser query, respectively, in the CYBORG model.

The “Tap ‘n Ghost” attack motivates the inclusion of novel adversarial queries when analyzing user-mediated protocols. Maruyama et al. described two specific attacks that separately attacked the two pathways involved in user-to-device communication. The TAP attack enables an adversary to create messages sent from a device to the user and the Ghost attack enables an adversary to modify messages sent from a user to a specific device. This attack provided real-world justification for the creation and description of the ShowUser, to model TAP-like attacks and ControlUser, to model Ghost-like attacks, queries in the CYBORG security model. Additionally, being able to employ the attacks separately warrants multiple

threat scenarios in the compromised user setting to account for all viable combinations of corruption.

2.3.3 Modeling Choices

To account for the above research into Bluetooth vulnerabilities, the following modeling choices were made. In regard to the NIST attacks, we attempted to model a complete adversary with the capability to affect both DtD and UtD communications. We define a typical Dolev–Yao adversary over the DtD channel to capture control over DtD communications and we give the adversary access to the ShowUser and ControlUser queries to model social engineering attacks like Bluejacking. Since much research has been devoted to the *BT–Niño–MitM* attack and viable solutions have been published, we do not devote research to preventing this exploit and assume the adversary actually attempts to break the Numeric Comparison and Passkey Entry protocols. To enable the research of other vulnerabilities against Passkey Entry aside for re-used passkey attacks, and for simplicity’s sake, all ex-changed passkeys are assumed ephemeral in this research. The viability of this assumption is based off the version of Passkey Entry employed. There are three variants of Passkey Entry based off who generates the passkey: the initiating/responding device or the user. For the former, it is reasonable to assume that advanced mobile devices, like smartphones or modern laptops, have access to a pseudorandom number generator for ephemeral passkey generation. However, for the latter, it is necessary to assume that the user has access to some form of secure number generation (e.g., RSA SecureID keyfob). As preventing the *Fixed Coordinate Invalid Curve Attack* is trivial and does not involve the protocol, we place no effort into modeling this attack and assume all devices will validate that received public keys lie on the employed elliptic curve. Finally, much of the modeling choices in regard to the user stem from adequately capturing the Tap ‘n Ghost attack. We refer the reader to Section 5.2 for further discussion on the specifics.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:

Preliminaries

3.1 Hash Functions

A cryptographic hash function, as defined by NIST, is a function that maps arbitrarily-sized bit strings to fixed-sized bit strings and satisfies the following:

1. "it is computationally infeasible to find for a given output, an input which maps to this output, and
2. it is computationally infeasible to find for a given input a second input which maps to the same output" [55].

The above properties are referred to as *preimage resistance* and *second-preimage resistance* respectively. However, the desirable qualities of hash functions are not limited to just these two. Another commonly referenced property of a hash function is *collision resistance*. Rogaway and Shrimpton examine the properties and inter-relationships of preimage resistance, second-preimage resistance, and *collision resistance* in [56]. We give rigorous definitions of the latter two properties below as they are relevant to this work.

Definition 3.1.1. Let H be a hash function and $\mathcal{A}$ a PPT adversary. We defined the second-preimage resistance (sec-pre) of H as such: given $msg \in \{0, 1\}^*$ and hash value such that $H(msg) = hash$, $\mathcal{A}$ cannot find a second-preimage, $msg' \in \{0, 1\}^*$, such that $msg' \neq msg$ and $H(msg') = hash$ with more than negligible probability. We denote $\mathcal{A}$'s advantage in breaking sec-pre of H as $\text{Adv}_{H, \mathcal{A}}^{\text{sec-pre}}$.

Definition 3.1.2. Let H be a hash function and $\mathcal{A}$ a PPT adversary. We defined the collision resistance (col-res) of H as such: $\mathcal{A}$ cannot find two messages $msg_1 \neq msg_2$ such that $H(msg_1) = H(msg_2)$ with more than negligible probability. We denote $\mathcal{A}$'s advantage in breaking col-res of H as $\text{Adv}_{H, \mathcal{A}}^{\text{col-res}}$.

3.2 Message Authentication Codes

The message authentication code (MAC) is a means by which one ensures authenticity and integrity in cryptographic protocols. We provide a mathematical definition of the MAC below, which we adapt from [17].

Definition 3.2.1. We define a message authentication code (MAC) over $\mathcal{M} = \{0, 1\}^*$ (the set of all possible messages), $\mathcal{K} = \{0, 1\}^k$ (the set of all possible keys), $\mathcal{T} = \{0, 1\}^t$ (the set of all possible tags), to be the tuple of algorithms $(\text{Kgn}, \text{MAC}, \text{Vfy})$ 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, $K \in \mathcal{K}$.
- $\text{MAC}(K, \text{msg}) \xrightarrow{\$} \text{tag}$: A probabilistic message authentication algorithm that takes as input a key, $K \in \mathcal{K}$, and message, $\text{msg} \in \mathcal{M}$, and outputs a tag, $\text{tag} \in \mathcal{T}$.
- $\text{Vfy}(K, \text{msg}, \text{tag}) \rightarrow v$: A deterministic verification algorithm that takes as inputs a key, $K \in \mathcal{K}$, a message, $\text{msg} \in \mathcal{M}$, and a tag, $\text{tag} \in \mathcal{T}$ and outputs a verification bit, $v \in \{0, 1\}$.

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

3.2.1 Hash-Based Message Authentication Code

An HMAC, called a hash-based MAC, is one common MAC implementation and is typically written as HMAC-H where “-H” is the specific hash function used (e.g., HMAC-SHA256).

We now direct the readers toward the definition of an HMAC as it was originally described by Bellare, Canetti, and Krawczyk in RFC 2104 [57]. This definition was constructed under the assumption that the hash function used in the HMAC operates iteratively on blocks of data with a fixed length. For example, although SHA-256 can accept any size input and produce a hash, it operates on input in blocks at 32-bytes at time.

Definition 3.2.2. A hash-based message authentication code (HMAC) over $\mathcal{M} = \{0, 1\}^*$ (the set of all possible messages), $\mathcal{K} = \{0, 1\}^k$ (the set of all possible keys), $\mathcal{T} = \{0, 1\}^t$ (the set of all possible tags), is a message authentication code with the below implementation of the MAC algorithm:

$\text{HMAC-H}(K, \text{msg}) = \text{H}((K' \oplus \text{opad}) \parallel \text{H}((K' \oplus \text{ipad}) \parallel \text{msg})) \rightarrow \text{tag}$: A deterministic message authentication algorithm that takes as input $K \in \mathcal{K}$ and $\text{msg} \in \mathcal{M}$ and outputs a tag, $\text{tag} \in \mathcal{T}$ where $\text{opad} = (0x36)^b$, $\text{ipad} = (0x5c)^b$, H is the associated hash algorithm, b is the block size of the hash function, and K' is derived from K as follows:

$$K' = \begin{cases} \text{H}(K) & |K| > b \\ K & \text{otherwise} . \end{cases}$$

The usefulness of the HMAC as a secure MAC construction cannot be understated. Bellare et al. originally proved that an HMAC was a pseudorandom function (PRF) under the assumption that the hash function used in the construction was also a PRF and weakly collision-resistant [57]. Bellare expanded this result to show that the hash function need only be a PRF and the weakly collision-resistant condition was in fact extraneous [58]. This solidified the usefulness of the HMAC construction, and proved that it could be used securely even when constructed with SHA-1 and MD5, which were widely used but provably not weakly collision-resistant hash functions [59], [60].

The hash function used in Secure Simple Pairing as part of the HMAC construction is SHA-256. NIST Special Publication 800-107, which details recommended best practices for the use of approved hash functions and their proven security properties, asserts that SHA-256 has a preimage resistance of 256 bits, a second preimage resistance in the range of 201-256 bits (where the variation is caused by the ratio of message length to input block size), and a collision resistance of 128 bits. These values fall within the expected range for a 256-bit output hash function and highlight that no explicit breaks have been found for this algorithm. Truncating the output of SHA-256, or its associated HMAC, will reduce the security of the function to a function of the length of the truncation [55].

3.2.2 Strong Unforgeability under Chosen Message Attack

In Figure 3.1, 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, or by forging a wholly new message-tag pair that verifies correctly.

$\text{EXP}_{\text{MAC}, \mathcal{A}}^{\text{SUF-CMA}}(\lambda):$ 1: $K \xleftarrow{\$} \text{Kgn}(1^\lambda)$ 2: $\text{win} \leftarrow 0, S \leftarrow \emptyset$ 3: $\mathcal{A}^{\text{MAC}(\cdot), \text{MAC.Vfy}(\cdot)}$ 4: return win	$\text{MAC.Vfy}(msg, tag):$ 1: $v \leftarrow \text{Vfy}(K, msg, tag)$ 2: if $(v = 1) \wedge ((msg, tag) \notin S)$ then 3: $\text{win} \leftarrow 1$ 4: return win from experiment 5: end if 6: return v
$\text{MAC}(msg):$ 1: $tag \leftarrow \text{MAC}(K, msg)$ 2: $S \leftarrow S \cup \{(msg, tag)\}$ 3: return tag	

Figure 3.1. Security experiment for SUF-CMA of a message authentication code algorithm $\text{MAC} = (\text{Kgn}, \text{MAC}, \text{Vfy})$ and adversary $\mathcal{A}$.

We conclude our discussion of SUF-CMA with definitions for adversarial advantage and security as formulated in [17].

Definition 3.2.3. We define the adversarial advantage against the SUF-CMA experiment described in Figure 3.1 for a PPT adversary $\mathcal{A}$ against a message authentication code MAC to be:

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

Definition 3.2.4. 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 $\text{EXP}_{\text{MAC}, \mathcal{A}}^{\text{SUF-CMA}}(\lambda)$ is upper bounded by some negligible function $\text{negl}(\lambda)$:

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

3.3 Elliptic Curves

Definition 3.3.1. An elliptic curve E is defined to be the following set of points over a finite field $\mathbb{F}_q$ of prime order $q > 3$:

$$\{P = (P_x, P_y) = (x, y) \in \mathbb{F}_q \times \mathbb{F}_q : y^2 - x^3 - ax^2 - b \equiv 0 \pmod{q}\} \cup \{O\},$$

where O is the point at infinity and $4a^3 + 27b^2 \not\equiv 0 \pmod{q}$.

This is a generalized definition of elliptic curve as needed for the purposes of this paper and

we direct the interested reader to [61] for a refresher or [62] for an in-depth examination on elliptic curves.

Definition 3.3.2. Point addition (+), is defined as the operation acting on two points $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$ in the elliptic curve E over a finite field $\mathbb{F}_q$ as follows.

- If $P_1 = P_2 = O$, then $P_1 + P_2 = O + O = O$.
- If $P_1 = O$ or $P_2 = O$ (say P_2 without loss of generality), then $P_1 + P_2 = P_1 + O = P_1$.
- Let $\sigma \equiv (y_2 - y_1)(x_2 - x_1)^{-1} \pmod{q}$, if $P_1 \neq P_2$.
- Let $\sigma \equiv (3x_1^2 + a)(2y_1)^{-1} \pmod{q}$, if $P_1 = P_2$ and $y_1 \neq 0$.
- Then we have $P_1 + P_2 = (x, y) = (\sigma^2 - x_1 - x_2, \sigma(x_1 - x) - y_1)$ if σ is defined, or $P_1 + P_2 = O$ otherwise.

We state without proof that the elliptic curve E over a finite field $\mathbb{F}_q$ together with the group operation +, forms a finite abelian group with O as the identity element. We use the notation mP for $m \in \mathbb{F}_q$ and $P \in E$ to signify *point multiplication* in this elliptic curve group, which is defined as repeated point addition of P with itself.

Definition 3.3.3. It is said that a point $P \in E$ has order $n \in \mathbb{F}_q$ if it holds that:

$$nP = O.$$

Definition 3.3.4. The set of points in an elliptic curve E equal to kP for $1 \leq k \leq n$, where $P \in E$ is of order n , forms a cyclic subgroup of E . The point P is referred to as the base point of this cyclic subgroup.

Definition 3.3.5. Let P be a base point in E of order n . The cofactor $h \in \mathbb{F}_q$ of P is defined to be the number such that:

$$|E| = nh.$$

Definition 3.3.6. If base point $P \in E$ has a cofactor of 1, then it is said that P is a generator of E .

We then use the tuple of values (q, a, b, P, n, h) to identify cyclic subgroups of elliptic curves, where the given parameters are as defined above.

Elliptic curves are useful because they allow us to execute Diffie–Hellman key exchange using keys of smaller sizes than needed when operating under standard Galois fields. To be explicit, the NSA operates under the assumption that 256-bit keys when using elliptic curves for Diffie–Hellman exchange achieves security comparable to 2048-bit RSA keys [63]. We depict how two parties may execute elliptic curve Diffie–Hellman (ECDH) to arrive at the shared key K below using the curve (q, a, b, P, n, h) over the field $\mathbb{F}_q$:

$$\begin{aligned}
A &: x \xleftarrow{\$} \mathbb{F}_q, \text{PK}_a \leftarrow xP \\
B &: y \xleftarrow{\$} \mathbb{F}_q, \text{PK}_b \leftarrow yP \\
A &\rightarrow B : \text{PK}_a \\
B &\rightarrow A : \text{PK}_b \\
A &: K \leftarrow x\text{PK}_b \\
B &: K \leftarrow y\text{PK}_a.
\end{aligned}$$

We then have that depending on the security properties of our elliptic curve, and the protocol in which it is employed, solving for K is a computationally infeasible problem. We capture one such formulation of the hardness of breaking ECDH below. In Definition 3.3.7 we present the security experiment for the elliptic curve decision Diffie–Hellman problem (EC-DDH). The basic premise of the typical DDH problem is that given a group G , generator g , and uniformly chosen group elements g^a and g^b , an adversary cannot distinguish g^{ab} from some other uniformly chosen element g^c . We base the experiment off the choice of elliptic curve as to account for known elliptic curves with vulnerabilities that an adversary can exploit and potentially have non-negligible probability of success in the EC-DDH experiment. We direct the reader to [64] and [65] for specific examples of attacks on elliptic curves, that would result in a non-negligible adversarial advantage in the EC-DDH experiment.

We focus on the elliptic curve variant of the DDH problem due to Bluetooth’s adoption of elliptic curves during its key exchange phase; they specify the use of the P-192 curve as part of Secure Simple Pairing and the P-256 curve when using Secure Connections [11]. Of note on the security of these curves, both of the above curves were approved by NIST and have been standardized by the FIPS PUB 186-4 as recommended for government use [66]. However, research by Bernstein and Lange [67] dispute the achievable security of the P-256

curve, they did not investigate the P-192 curve. Specifically, the P-256 curve did not pass Bernstein and Lange’s rigidity, ladder, completeness, and indistinguishability tests.

Definition 3.3.7. *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 $a, b < n$, for $c \xleftarrow{\$} \mathbb{F}_q$ and $c < n$, $\mathcal{A}$ cannot distinguish abP from cP with more than negligible probability. We use $\text{Adv}_{\mathcal{A}}^{\text{EC-DDH}}$ to write $\mathcal{A}$ ’s advantage in breaking the EC-DDH assumption.*

3.4 Pseudorandom Functions

A *pseudorandom function* (PRF), as defined in [68], is an any function that is computationally indistinguishable from a *random oracle*. A random oracle is a function whose outputs are truly random. A common pseudorandom function used in cryptography is the HMAC as discussed in Section 3.2. We can also define other security properties a PRF may achieve/contribute toward.

The pseudorandom function oracle Diffie–Hellman (PRF-ODH) assumption is one such example. Informally, this assumption states that even given knowledge to the public share of a DH key, and PRF output(s) involving either the left or right DH key, a PRF output with the full DH key is computationally indistinguishable from random. The PRF-ODH assumption was originally introduced in [69] in an investigation of TLS 1.2, and later modified to allow early adversarial access to both the DH public key shares before a challenge query [70], [71]. Other PRF-ODH assumption variants were analyzed in [72]. We extend the symmetric generic single-single PRF-ODH assumption of [71] to cover ECDH. This construction is the same as presented in [70], [71] but with ECDH vice standard Diffie–Hellman.

Definition 3.4.1. *Let E be an elliptic curve over the field $\mathbb{F}_q$ with generator point P of order n . Let $\text{PRF}_\lambda : E \times \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ be a pseudorandom function with keys in E , input strings in $\{0, 1\}^*$, and output strings in $\{0, 1\}^\lambda$. Let $\mathcal{A}$ be a PPT adversary. We define the symmetric generic single-single PRF Oracle ECDH assumption, EC-sym-ssPRF-ODH, 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 $b \xleftarrow{\$} \{0, 1\}$ uniformly at random and sets $y_0 \leftarrow \text{PRF}_\lambda(abP, x)$ if $b = 0$, 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, OECDH_a and OECDH_b , handled as follows:
 - $\text{OECDH}_a(S, x)$: Challenger returns $\perp$ if $S \notin E$ or if $(S, x) = (bP, x^*)$, otherwise it returns $y \leftarrow \text{PRF}_\lambda(aS, x)$.
 - $\text{OECDH}_b(T, x)$: Challenger returns $\perp$ if $T \notin E$ or if $(T, x) = (aP, x^*)$, otherwise it returns $y \leftarrow \text{PRF}_\lambda(bT, x)$.
5. Eventually, Adversary outputs the bit guess b , and wins the experiment if $b = b$.

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

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

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

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

CHAPTER 4:

Key Exchange in Bluetooth

In this chapter, we introduce to the Secure Simple Pairing (SSP) protocol as described by the Bluetooth SIG Specification 5.1. We also introduce all the necessary functions, sub-protocols, symbols, and variables.

4.1 SSP Background

In this section, we give preliminary descriptions of the four Bluetooth pairing methods within the Secure Simple Pairing suite, and their requirements for use.

4.1.1 Device Input/Output Capabilities

In order for a device to execute one of the SSP authentication protocols it must first meet the minimum device input and output capabilities. Table 4.1 gives a visual overview of all possible input/output (IO) combinations afforded to a Bluetooth device and the resultant four designations of IO capability. We also display the default authentication method used in SSP as a result of the combination of IO capabilities in the two devices executing the protocol in Table 4.2.

- **NoInputNoOutput.** This designation means that the device cannot display or communicate a 6-digit decimal number to the user and can accept no more than simple binary input from the user. Since no method relies on only being able to accept binary input, Bluetooth made the decision to not make a separate IO capability designation for a No Output and Yes / No device. If even one of the devices executing the SSP protocol is NoInputNoOutput then either Just Works or the OOB authentication method must be used. A typical example of a NoInputNoOutput device would be wireless headphones.
- **DisplayOnly.** This designation means that the device can display or communicate a 6-digit decimal number to the user and does not have the ability to accept any kind of input from the user. DisplayOnly devices can execute Passkey Entry, Just Works, or OOB depending on their partner's capabilities. A typical example of a DisplayOnly device would be an RSA SecureID keyfob.

Table 4.1. Table depicting possible device IO capabilities as defined by Bluetooth. We define the various possibilities for input and output capability as follows. “No Input” means the device cannot receive signals from the user. “Yes / No” means the device can accept binary inputs from the user. “Keyboard” means the device has the ability to accept digits 0 through 9 and binary input from the user. “No Output” means the device cannot display or communicate at least a 6-digit decimal number to the user. “Numeric Output” means the device can display or communicate at least a 6-digit decimal number to the user. Bluetooth uses the terminology “display or communicate” to account for the possibility that the numeric output could either be displayed visually to the user or communicated via other means, such as audibly. Adapted from [11].

Table of Possible IO Capability Combinations		Device Output Capability	
		No Output	Numeric Output
Device Input Capability	No Input	NoInputNoOutput	DisplayOnly
	Yes / No	NoInputNoOutput	DisplayYesNo
	Keyboard	KeyboardOnly	KeyboardDisplay

- **DisplayYesNo.** This designation means that the device can display or communicate a 6-digit decimal number to the user and has the ability to accept binary inputs from the user. DisplayYesNo devices can execute any of the possible SSP authentication methods depending on their partner’s capabilities. Typical examples of a DisplayYesNo devices can be seen in Internet of Things (IoT) devices.
- **KeyboardOnly.** This designation means that the device cannot display or communicate a 6-digit decimal number to the user and has the ability to accept digits 0 through 9 and binary input from the user. KeyboardOnly devices can execute Passkey Entry, Just Works, or OOB depending on their partners capabilities. A typical example of a KeyboardOnly devices is a wireless keyboard.
- **KeyboardDisplay.** This designation means that the device can display or communicate a 6-digit decimal number to the user and has the ability to accept digits 0 through 9 from the user. KeyboardDisplay devices can execute any of the possible SSP authentication methods depending on their partner’s capabilities. A typical example of a KeyboardDisplay device is a your typical smartphone.

Table 4.2. Table depicting authentication method employed whether in Bluetooth pairing, whether it be Just Works, Numeric Comparison, or Passkey Entry, based on the combination of device IO capabilities. We differentiate Passkey Entry into three versions based on how the passkey is generated: via the initiating device (Init. Gen'd), responding device (Resp. Gen'd), or the user (User Gen'd). Adapted from [11].

Table of Default Authentication Methods		Initiating Device				
		NoInputNoOutput	DisplayOnly	DisplayYesNo	KeyboardOnly	KeyboardDisplay
Responding Device	NoInputNoOutput	Just Works	Just Works	Just Works	Just Works	Just Works
	DisplayOnly	Just Works	Just Works	Just Works	Resp. Gen'd	Resp. Gen'd
	DisplayYesNo	Just Works	Just Works	Num. Comp.	Resp. Gen'd	Resp. Gen'd Num. Comp.
	KeyboardOnly	Just Works	Init. Gen'd	Init Gen'd	User Gen'd	Init. Gen'd
	KeyboardDisplay	Just Works	Init. Gen'd	Init. Gen'd Num. Comp.	Resp. Gen'd	Init. Gen'd Num. Comp.

4.1.2 Out-of-Band

This protocol is designed for devices that can communicate via a separate communication method that can be used to authenticate devices. Also of note, Out-of-Band can be used by any device regardless of IO capability so long as the two devices have a method for shared out-of-band (OOB) communications. A common method of OOB communication used is near-field communication (NFC). The security guarantees Out-of-Band affords users is completely dependent on the OOB communication method used and whether an adversary has control over the separate medium. Although we choose not to devote analysis to Out-of-Band, analysis of this protocol could progress similarly to that of cyborg protocols. Since Out-of-Band pairing does not rely on certificates or a certificate authority for authentication, an investigation of adversarial capabilities over prominent OOB communication methods, like NFC, could yield an applicable model for analysis similar to our CYBORG model.

4.1.3 Just Works

Just Works is functionally identical to Numeric Comparison but with no user interaction. This means that Just Works provides no active MitM protection. The *BT-Niño-MitM* attack discussed in Section 2.3.2 exploits this weakness by downgrading Numeric Comparison to Just Works for authentication. This protocol is used when one of the devices to be paired is of the NoInputNoOutput classification; one real-world example of Just Works use is pairing involving Bluetooth headphones, which typically have no display capability and can only

accept binary inputs from the user. As extensive research has already shown this protocol is broken, we choose not to devote analysis to Just Works.

4.1.4 Numeric Comparison

Numeric Comparison is a pairing method that requires all devices to be able to display or communicate a six-digit number to the user and have the ability to accept at least binary inputs from the user. This method also makes no use of a PIN or password; thereby, simplifying the pairing process for the user by not requiring any memorization. Motivation behind the development of Numeric Comparison was twofold. Bluetooth SIG wanted a pairing method with MitM protection and they desired to have a method for users to confirm that two devices with non-unique names are communicating with each other. We devote an entire section to a description of Numeric Comparison in Section 4.3 and analysis of its security properties in Section 6.2.

4.1.5 Passkey Entry

Passkey Entry has various requirements based on the devices executing the protocol and refer the reader to Table 4.2 for a detailed description of the pairing possibilities. The main motivation behind the development for Passkey Entry was for the pairing scenario involving a wireless keyboard and a computer. For this combination to achieve MitM protection Numeric Comparison could not be used because wireless keyboards typically do not come with a numeric display capability as standard. However, Passkey Entry would allow for an authenticated pairing to take place whereby the computer generated and displayed a random value to be input to the wireless keyboard by the user. We devote an entire section to a description of Passkey Entry in Section 4.3 and analysis of its security properties in Section 6.3.1. We also propose a new version Passkey Entry, termed Dual Passkey Entry (DPE), and an add-on to original Passkey Entry and DPE, termed Secure Hash Modification (SHM), in Chapter 7. These modifications were developed as a result of our analysis and in an effort to fix security shortfalls discovered in Passkey Entry. We analyze DPE and SHM in Chapter 8.

4.2 Relevant Variables

Various values and variables are used throughout Bluetooth pairing to accomplish the pairing to include keys, tags, nonces, and constants. All presented values will be said to be maintained by a Bluetooth device of identity A . We state the actual bit lengths for the relevant variables as required under either BR/EDR Secure Simple Pairing or Secure Connections Passkey Entry.

4.2.1 Keys and Random Values

- $\text{DHKey} \in \{0, 1\}^{256}$: Referred to as the Diffie–Hellman Key; this is the long-term key generated from the DH key exchange.
- $\text{LK} \in \{0, 1\}^{128}$: Referred to as the Link Key; this is the session key generated as a result of pairing.
- $\text{PK} \in \{0, 1\}^{256}$: This is the bit representation of the long-term public key of device A to be exchanged. This will refer to a point on an elliptic curve. We will write PK_{ix} , to refer to the x -coordinate of device A 's public key.
- $\text{SK} \in \{0, 1\}^{255}$: This is the long-term secret key of Device A to be used in DH key exchange.
- $N_a \in \{0, 1\}^{128}$: A nonce generated from device A .
- $r_a \in \{0, \dots, 9\}^6$: A 6-digit, decimal, random value for device A ; referred to as the passkey. Can be either device or user generated.

Bluetooth's specification requires that N_a be sampled fresh for every execution of Passkey Entry, and recommends the same for SK_a, r . Per the specification, SK should be re-drawn when $S + 3F > 8$ where S is the number of successful pairings and F is the number of failed pairing said device has experienced. Though it is highly recommended that a device change its private key after every pairing, independent of the pairing outcome, this is merely a suggestion and it is reasonable to assume that implementations exist where this does not occur. In this analysis, we assume SK_a, N_a , and r are fresh ephemeral values. We impose this restriction to simplify analysis.

One requirement placed on the actual devices in Bluetooth is random number generation. Bluetooth stipulates that the nonce “should be generated directly from a physical source of randomness or with a good pseudorandom generator seeded with a random value from a

physical source” [11]. Without going into a long discussion on what it means to be a “good” pseudorandom generator, for the purposes of this research it will be assumed that any nonce used in analysis is indistinguishable from a random value.

Also note that DHKey is computed as the x-coordinate of $[SK_a]PK_b$. We require PK_{ay} as part of the public key, by way of curve validation checks to prevent the attack described in [54].

4.2.2 Tags

- $C \in \{0, 1\}^{128}$: 128 bit tag; referred to as the commitment value.
- $E \in \{0, 1\}^{128}$: 128 bit tag; referred to as the check value.
- $V \in \{0, \dots, 9\}^6$: 6 digit tag; referred to as the verification value.

4.2.3 Constants

- “btlk”: A fixed 4-byte ASCII string; the constant value 0x62746c6b and acts as a label in the protocol.
- $IOcapA \in \{0, 1\}^{24}$: The IO capabilities of device A; defined to be constant across sessions for device I.
- $A \in \{0, 1\}^{48}$: The Bluetooth device address of device A, synonymous with the identity of device of A.

4.2.4 Elliptic Curves

The specific curve used in Secure Simple Pairing is P-192 as defined by NIST. We define the curve P-192 below for prime modulus p , order r , generator x-coordinate G_x , generator

y-coordinate G_y , first degree coefficient a , and intercept b .

P-192 :

$$\begin{aligned}p &= 6277101735386680763835789423207666416083908700390324961279, \\r &= 6277101735386680763835789423176059013767194773182842284081, \\a &= -3, \\b &= 64210519 \text{ e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1}, \\G_x &= 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012, \\G_y &= 07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811 [11].\end{aligned}$$

The specific curve used in Secure Connections is P-256 as defined by NIST. We define the curve P-256 below for prime modulus p , order r , generator x-coordinate G_x , generator y-coordinate G_y , first degree coefficient a , and intercept b .

P-256 :

$$\begin{aligned}p &= 6277101735386680763835789423207666416083908700390324961279, \\r &= 6277101735386680763835789423176059013767194773182842284081, \\a &= -3, \\b &= 0x64210519 \text{ e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1}, \\G_x &= 0x188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012, \\G_y &= 0x07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811 [11].\end{aligned}$$

4.2.5 Relevant Functions

Regardless of whether we are operating under Secure Simple Pairing or Secure Connections, both modes use SHA-256 [55] for the hash function and HMAC-SHA256 [57] for the MAC.

4.3 Protocol Description

In this section, we present the Numeric Comparison and Passkey Entry protocols adapted to fit within previous mathematical definitions and diagrams for visualization in Figures 4.3

and 4.4.

4.3.1 Phase 0: Init and IOcap Exchange

This phase is not explicitly listed in the Bluetooth v5.1 specification and is not a Phase of any protocol as presented therein; however, it encompasses a variety of steps performed over an unsecure channel before pairing commences. This includes device discovery, whereby the devices to be paired, *A* and *B*, share Bluetooth device addresses as shown in Figure 4.1, and initialize a connection. Additionally, the IO capabilities, IOcapA and IOcapB, are exchanged during this phase to ensure a compatible authentication method is being used as shown in Figure 4.2.

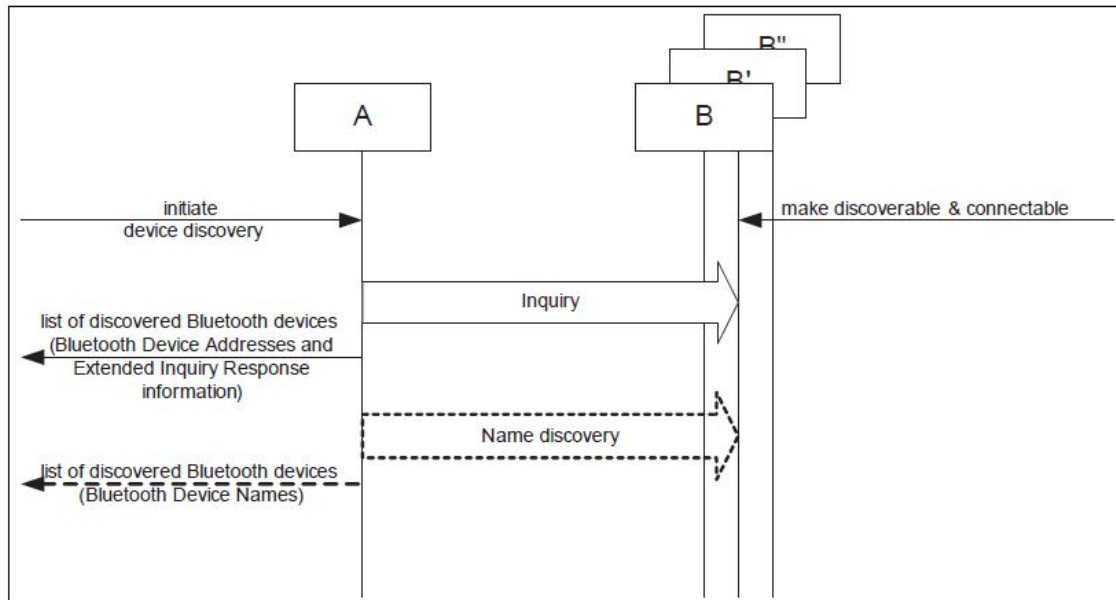


Figure 4.1. Communication chart for Device Discovery that describes the process by which a device learns another device's Bluetooth device address for use in pairing. In the above diagram, we have that Device A broadcasts an inquiry to all discoverable and connectable Bluetooth devices within range, usually around 10m. The second step of Device Discovery involves querying all responders for their display name to be shown to the user. Source: [11].

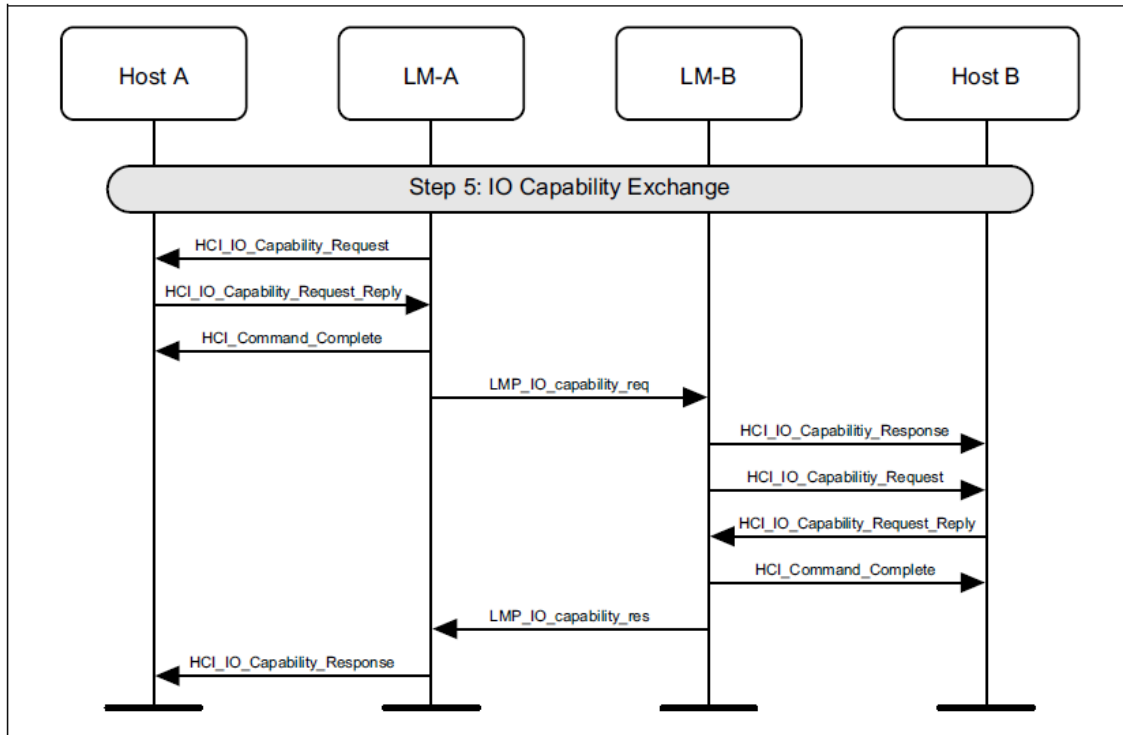


Figure 4.2. Message Sequence Chart for the IO Capability Exchange between an initiating device, Device A, and responding device, Device B. This is a two-communication request/response protocol whereby device A requests and communicates his IO capability to device B, and device B responds with his. Source: [11].

4.3.2 Phase 1: Public Key Exchange

Phase 1 refers to the elliptic curve Diffie–Hellman (ECDH) public key exchange as described in Section 3.3 and calculation of DHKey. The purpose of Phase 1 is to generate a shared key to be used throughout the rest of pairing. To accomplish this, the public keys of devices A and B are exchanged in steps 1a and 1b, and then used to calculate DHKey using elliptic curve multiplication in steps 1c and 1d.

4.3.3 Phase 2: Authentication Stage 1

The purpose of Phase 2 is the first half of authentication, whereby the user functions as a trusted third party (TTP) to ensure devices A and B are actually pairing with each other as expected. We separate the discussion of Phase 2 into descriptions of Numeric Comparison

and Passkey Entry in its various modes.

Numeric Comparison

A visual depiction of Numeric Comparison is given in Figure 4.3. Both the initiating and responding devices generate 128-bit nonces, N_a and N_b respectively, and set device passkeys, r_a and r_b for both devices, to 0 (these numbers are only used in Passkey Entry) in steps 6a through 7b. Then the responding device, B, generates a commitment value, C_b using *hmac* in step 8. The commitment value is then sent to device A in step 9, which responds with its own nonce in step 10. Device B then replies with the nonce used to generate its commitment value in step 11. With the value N_b now known, device A then verifies if $C_b = \text{HMAC}(N_b, PK_{bx} || PK_{ax} || r_b)$ in step 12 as expected. We model the verification using the Vfy algorithm for the HMAC message authentication code with key N_b , message $(PK_{bx} || PK_{ax} || r_b)$, and tag C_b , which will output a verification bit, $v_1 \in \{0, 1\}$, as previously defined. If $v_1 = 0$, the protocol is aborted; otherwise, the verification succeeds and Phase 2 continues with steps 13a and 13b. In these steps, both device A and B generate a verification value, V_a and V_b , respectively, using the hash function H. Both devices then display their verification values for the User to compare; we model this in step 14a and 14b. The User then visually compares the two displayed value to ensure $V_a = V_b$. We model the User interaction with device A and B via steps 15a and 15b. If $V_a = V_b$, the user sends both devices the message OK, to signal to the devices to proceed with pairing; else, the user sends the message FAIL to indicate to abort pairing.

Passkey Entry

Passkey Entry has three options for sharing the passkey, r , among the two devices and user, as shown in Figure 4.4.

- Initiator-Generated (PE-IG). In this version of Passkey Entry, the initiating device randomly samples the passkey, sends it to the user, and the user sends it to the responding device.
- Responder-Generated (PE-RG). In this version of Passkey Entry, the responding device randomly samples the passkey, sends it to the user, and the user sends it to the initiating device.

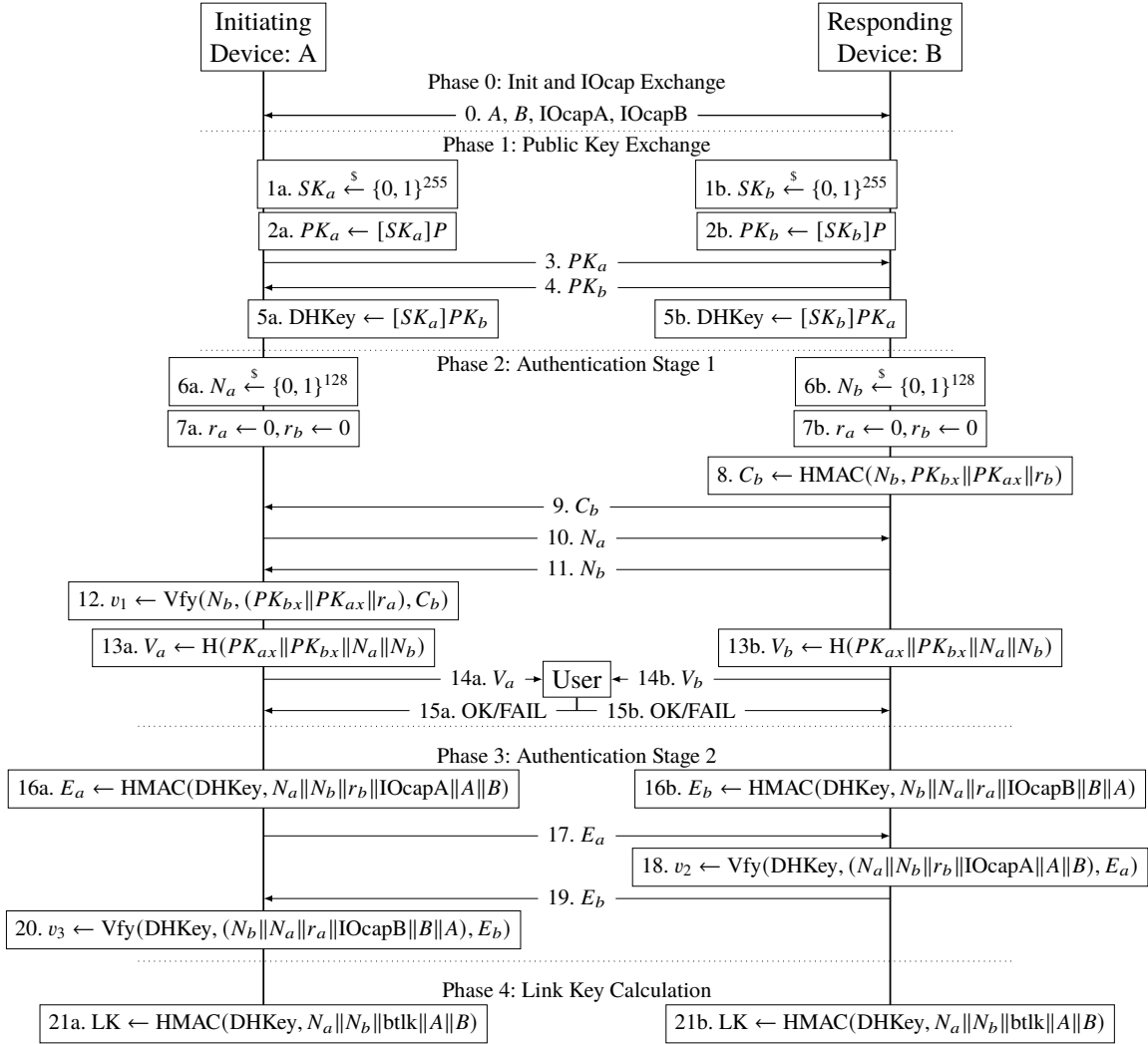


Figure 4.3. Numeric Comparison. Bit sizes shown are reflexive of Secure Connections. Phase 0 takes place before the actual protocol and represents device discovery and the IO capability exchange; IOcapA, IOcapB, A, and B are distributed among the devices during this phase. Phase 2 shows authentication performed in Numeric Comparison mode. Adapted from [11].

- User-Generated (PE-UG). In this version of Passkey Entry, the user selects the passkey then sends it to both the initiating and responding device.

The devices then proceed to verify they agree on each of the twenty digits comprising r in sequence. We use the index i to designate the exact digit the devices are in the process of verifying in r . The index also differentiates the nonce and commitment values, as a fresh

nonce is chosen by each device for every step of the data authentication for r . Both the initiating and responding devices generate nonces, $N_{a,i}$ and $N_{b,i}$ respectively, and compute the respective commitment values, $C_{a,i}$ and $C_{b,i}$, using the message authentication code HMAC in steps 9a through 10b. The devices then proceed to exchange their commitment values in steps 12 and 13. Device A then replies with the associated nonce used to generate its commitment value in step 14 for the current digit of r . With the value $N_{a,i}$ now known, device B then verifies if $C_{a,i} = \text{HMAC}(N_{a,i}, PK_{ax} || PK_{bx} || r_{b,i})$ in step 15 as expected. We model the verification using the Vfy algorithm for the HMAC message authentication code with key $N_{a,i}$, message $(PK_{ax} || PK_{bx} || r_{b,i})$, and tag $C_{a,i}$, which will output a verification bit, $v_1 \in \{0, 1\}$, as previously defined. If $v_1 = 0$, the protocol is aborted; otherwise, the verification succeeds and Phase 2 continues with steps 16 and 17 similarly to steps 14 and 15, but with the roles reversed. Phase 2 concludes upon successful verification of the twentieth digit of r by device A.

Remark. Sampling of fresh nonces and computation of a commitment for every bit of the passkey was done to prevent an adversary from mounting an offline dictionary attack similar to the one presented in [73] against Bluetooth's legacy key exchange prior to Secure Simple Pairing's introduction in v2.1. This is assuming that a passkey may be reused across multiple pairing attempts. In the case where passkeys are not ephemeral, this construction forces an adversary to induce an average of 10 pairing attempts to learn all 20 bits of the passkey through active guessing. The assumption behind this is that knowledge of the passkey is no longer useful once the link key is computed; Bluetooth also recommends that pairing attempts be minimized [11]. Although Passkey Entry complicates the above attack, it is more susceptible to other types of eavesdropping attacks [13].

4.3.4 Phase 3: Authentication Stage 2

The purpose of this phase is to complete device authentication using the shared key DHKey from Phase 1. This phase begins with both devices computing their respective check values, E_a and E_b , for device A and B. First the initiating device sends its check value to the responding device to be verified. If $v_3 = 0$ the responding device aborts the protocol; otherwise, it sends E_b to the initiating device and proceeds to Phase 4. If $v_4 = 0$ the initiating device aborts the protocol; otherwise, it proceeds to Phase 4.

4.3.5 Phase 4: Link Key Calculation

The purpose of this phase is the calculation of the link key, LK. Both devices can perform this calculation in steps 12a and 12b simultaneously using known values. Completion of this phase concludes pairing.

4.4 Correctness

We now present a discussion on the *correctness* for Numeric Comparison and Passkey Entry. When we say correctness, we mean that when two participants execute a run of the protocol as defined in the specification and they agree on all messages they exchanged in execution, then the two devices reach the expected conclusion of computing a matching session key.

4.4.1 Correctness of Numeric Comparison

In order for Numeric Comparison to be correct we must have that both devices generate the same link key, LK, in Phase 4. In order for this to occur we must have that both devices compute the same Diffie–Hellman key, DHKey, in Phase 1; all other values that contribute to the computation of LK are publicly exchanged messages. We also must have that all verification bits, v_i are set to one and the user is sent the same verification values, V_i , in order for the two devices to reach Phase 4.

DHKey. In steps 5a and 5b, both devices perform a similar calculation using elliptic curve multiplication to generate DHKey. Observe the following relationships among the public and private keys of the devices to see that both calculations generate the same value.

$$\begin{aligned} \text{DHKey} &= [SK_a]PK_b \\ &= [SK_a][SK_b](G_x, G_y) \\ &= [SK_b][SK_a](G_x, G_y) \\ &= [SK_b]PK_a \\ &= \text{DHKey} . \end{aligned}$$

v₁. In step 12, the initiating device performs the check $v_1 \leftarrow \text{Vfy}(N_b, (PK_{bx} || PK_{ax} || r_b), C_b)$ where Vfy is the verification algorithm for HMAC. Thus we have that $v_1 = 1$ if and only if:

$$C_b = \text{HMAC}(N_b, PK_{bx} || PK_{ax} || r_b) .$$

Observing step 8 shows that the verification check for v_1 will succeed.

OK. In order for the user to send OK to both devices, we must have that $V_a = V_b$. From a quick observation of steps 13a and 13b we see that this indeed holds since both devices make the same hash function call:

$$V_a = H(PK_{ax} || PK_{bx} || N_a || N_b) = V_b .$$

v₂. In step 18, the responding device performs the check $v_2 \leftarrow \text{Vfy}(\text{DHKey}, (N_a || N_b || r_b || \text{IOcapA} || A || B), E_a)$ where Vfy is the verification algorithm for HMAC. Thus we have that $v_2 = 1$ if and only if:

$$E_a = \text{HMAC}(\text{DHKey}, N_a || N_b || r_b || \text{IOcapA} || A || B) .$$

Observing step 16a shows that the verification check for v_2 will succeed.

v₃. In step 20, the initiating device performs the check $v_3 \leftarrow \text{Vfy}(\text{DHKey}, (N_b || N_a || r_a || \text{IOcapB} || B || A), E_b)$ where Vfy is the verification algorithm for HMAC. Thus we have that $v_3 = 1$ if and only if:

$$E_b = \text{HMAC}(\text{DHKey}, (N_b || N_a || r_a || \text{IOcapB} || B || A) .$$

Observing step 16b shows that the verification check for v_3 will succeed. We have now shown that two devices executing Numeric Comparison will indeed reach Phase 4.

Therefore, both devices will arrive at the same value for the link key, LK, because they both make the same call to HMAC as defined in steps 12a and 12b, and we conclude Numeric Comparison is correct.

4.4.2 Correctness of Passkey Entry

In order for Passkey Entry to be correct we must have that both devices generate the same link key, LK, in Phase 4. In order for this to occur we must have that both devices compute the same Diffie–Hellman key, DHKey, in Phase 1; all other values that contribute to the computation of LK are publicly exchanged messages. We also must have that all verification bits, v_i are set to one in order for the two devices to reach Phase 4. We do not allot explicit discussion to the correctness of the three variants. For the purposes of correctness, all parties will agree on the same passkey values, r_a and r_b , regardless of how they are generated because they are always exchanged among all parties in steps 6 through 8.

DHKey. In steps 5a and 5b both devices perform a similar calculation using elliptic curve multiplication to generate DHKey. Observe the following relationships among the public and private keys of the devices to see that both calculations generate the same value.

$$\begin{aligned}
 \text{DHKey} &= [SK_a]PK_b \\
 &= [SK_a][SK_b](G_x, G_y) \\
 &= [SK_b][SK_a](G_x, G_y) \\
 &= [SK_b]PK_a \\
 &= \text{DHKey} .
 \end{aligned}$$

v₁. In step 15, the responding device performs the check $v_1 \leftarrow \text{Vfy}(N_{a,i}, (PK_{ax} \| PK_{bx} \| r_{a,i}), C_{a,i})$ where Vfy is the verification algorithm for HMAC. Thus we have that $v_1 = 1$ if and only if:

$$C_{a,i} = \text{HMAC}(N_{a,i}, PK_{ax} \| PK_{bx} \| r_{a,i}) .$$

Observing step 11a shows that the verification check for v_1 will succeed.

v₂. In step 17, the initiating device performs the check $v_2 \leftarrow \text{Vfy}(N_{b,i}, (PK_{bx} \| PK_{ax} \| 0), C_{b,i})$ where Vfy is the verification algorithm for HMAC. Thus we have that $v_2 = 1$ if and only if:

$$C_{b,i} = \text{HMAC}(N_{b,i}, PK_{bx} \| PK_{ax} \| 0) .$$

Observing step 11b shows that the verification check for v_2 will succeed.

v₃. In step 20, the responding device performs the check $v_3 \leftarrow \text{Vfy}(\text{DHKey}, N_a \| N_b \| r_b \| \text{IOcapA} \| A \| B, E_a)$ where Vfy is the verification algorithm for HMAC. Thus we have that $v_3 = 1$ if and only if:

$$E_a = \text{HMAC}(\text{DHKey}, N_a \| N_b \| r_b \| \text{IOcapA} \| A \| B) .$$

Observing step 18a shows that the verification check for v_3 will succeed.

v₄. In step 22, the initiating device performs the check $v_4 \leftarrow \text{Vfy}(\text{DHKey}, (N_b \| N_a \| r_a \| \text{IOcapB} \| B \| A), E_b)$ where Vfy is the verification algorithm for HMAC. Thus we have that $v_4 = 1$ if and only if:

$$E_b = \text{HMAC}(\text{DHKey}, N_b \| N_a \| r_a \| \text{IOcapB} \| B \| A) .$$

Observing step 18b shows that the verification check for v_4 will succeed. We have now shown that two devices executing Passkey Entry will indeed reach Phase 4

Therefore, both devices will arrive at the same value for the link key, LK, because they both make the same call to HMAC as defined in steps 12a and 12b, and we conclude Passkey Entry is correct.

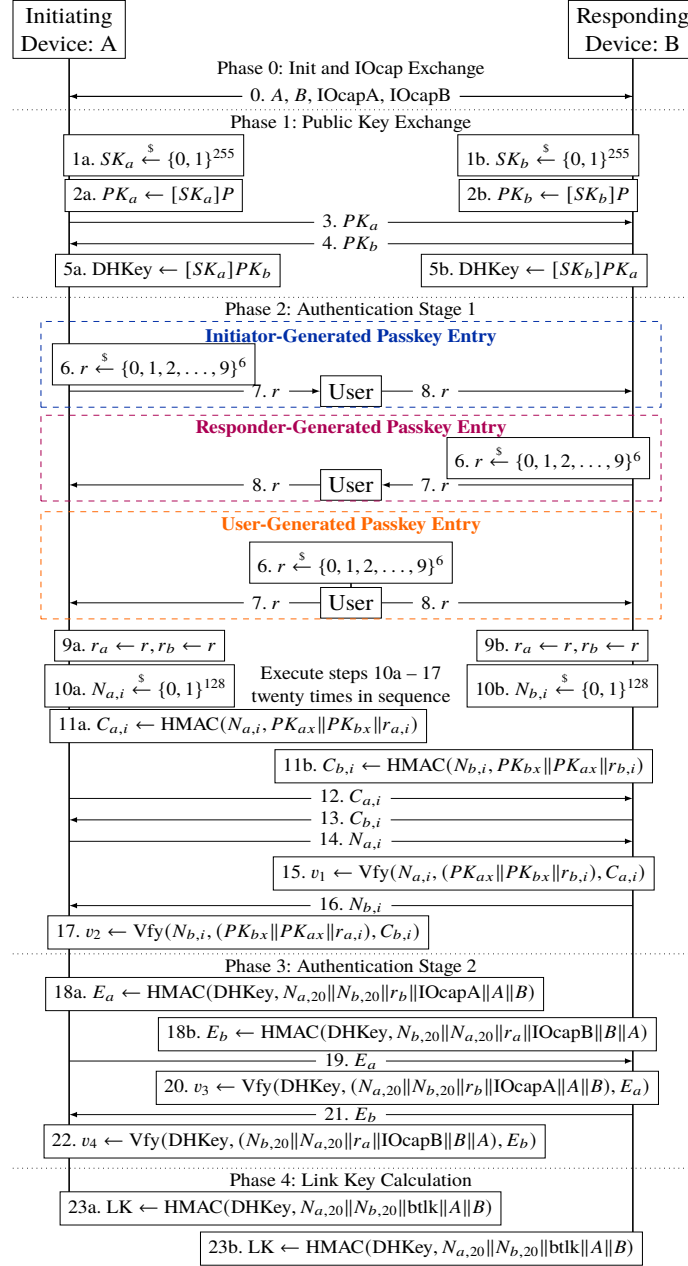


Figure 4.4. Passkey Entry. Phase 0 takes place before execution of the protocol; A , B , IOcapA , and IOcapB are distributed among the devices during this phase. Phase 2, steps 6-8, are version dependent as labeled (for initiator-generated passkey, responder-generated passkey, and user-generated passkey), all other steps are version independent. Adapted from [11]

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5: CYBORG Model for User-Mediated, Authenticated Key Exchange

In this chapter, we present the CYBORG security model as a synthesis of past security models addressing both cyborg-type protocols [10], [17], [18] and AKEs in general [16], [40], [45], [69].

5.1 Participant Model

We begin the presentation of CYBORG by defining the two classes of participants in protocols that CYBORG can handle: devices and users. We also present how session identifiers will be defined in general and we conclude with rigorous definitions of both Device-Device and User-Device partnering.

5.1.1 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 5.1.1 and [17]. We set no limit on the number of sessions a single participant can have running at any one time with any other participant.

Devices

Let $\mathcal{ID}$ be the set of all possible *identities* and $A \in \mathcal{ID}$ to be the identity of device I . We choose to model participants via membership in a set to ensure all device identities are unique. To capture the participation of a device $A \in \mathcal{ID}$ in a specific session, we utilize session oracles and describe their internal state as the tuple of the following values with λ as the security parameter:

- **key** $\in \mathcal{K} \cup \{\perp\}$. This variable holds device A 's completed session key for the session where $\mathcal{K}$ is the associated keyspace.
- **state**. This variable holds other secret state information.

- $\text{role} \in \{\text{initiator}, \text{responder}, \perp\}$. This variable holds device A 's role.
- $\text{pid} \in \mathcal{ID} \setminus \{A\}$. This variable holds the identifier for the partner device.
- $\text{sid} \in \{0, 1\}^* \cup \{\perp\}$. This variable holds the current session identifier.
- $\delta \in \{\text{accept}, \text{reject}, *\}$. This variable holds the specific result of the session either acceptance, rejection, or no decision, respectively.

All variables are assumed to be undefined upon initialization of a session, we model this through the use of logical falsum, $\perp$. At the creation of the s -th session for device A , the session oracle π_s^A is initiated to $(\text{skey}, \text{state}, \text{role}, \text{pid}, \text{sid}, \delta) = (\perp, \perp, \perp, \perp, \perp, *)$. For session acceptance, we require the following:

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

For session π_s^A , we choose to remove A from the set of possible pids. This is done to prevent trivial attacks. Role is included in the session state in order to differentiate sessions involving the same two identities but executing different roles in the protocol.

Users

Any user that participates 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. Since any user is said to be honest within CYBORG, nothing is gained from allowing for multiple users in a protocol since they would all perform the same function. As a result of this, we choose to model all possible users as the singular entity U vice as a member of a larger set; this construction and logic matches with that presented in [17], [18]. Similar to our definition for devices, users are also modeled via session oracles with session state:

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

Initialization of the user session oracles is described in Section 5.2.2. We require the user to be able to distinguish the fact that he is communicating with two separate devices, but we do not require him to distinguish the role these devices play in the protocol. The assumption is that this information is not guaranteed knowledge to the user but is instead protocol dependent. The ISO/IEC 9798-6 7a mechanism [74] is an example of a cyborg protocol

where the user has knowledge of the role each device plays, and Numeric Comparison is a counter-example where the user does not have knowledge of device role.

5.1.2 Partnering

In order to capture what it means for two participants in a protocol to partner, it becomes necessary to differentiate sessions from each other; we do this via session identifiers. Ultimately, the decision to use session identifiers vice matching conversations is based off the fact that devices in a user-mediated protocol tend not to engage in a strict matching conversation; we see this in Numeric Comparison, where two devices display values to be compared by the user. This constitutes a point of discontinuity in the conversation between the devices executing the protocol whereby both devices send a message that does not match with an identical message received by the corresponding device. Although the user's function is to ensure these two values match, the above correspondence would fail to be classified as a matching conversation under Bellare and Rogaway's definition. The presence of the user as a third party is not captured in the original matching conversation model, which has proven to be too rigid of a definition to encompass user-mediated protocols.

We therefore have two possibilities to rectify this discontinuity: *weak session identifiers* and *strong session identifiers*. Weak session identifiers are protocol dependent and follow the CK01 model, which places the onus on the analyst to define a protocol specific session identifier based on relevant exchanged messages or computed values. Strong session identifiers are constructed as a transcript of mutually held messages; messages a device can reasonably surmise its partner device also has explicit knowledge of in any given session. This follows the concept of matching conversations, albeit with transcripts requiring only information that both devices hold. We give an explicit definition of strong session identifiers below.

Definition 5.1.1 (Strong Session Identifiers). *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 :*

$$(msg_1, \dots, msg_n),$$

where msg_k , is the k -th message sent/received in sequential order. Then we define the session identifier, denoted sid , as the following subsequence of $(msg_1, \dots, msg_n)$ pre-appended by

an optional msg_0 :

for $1 \leq k \leq n$, we append msg_k to $\pi_s^A.sid$ 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 sid as a fixed msg_0 .

The above definition follows closely to matching conversations, with two discrepancies. For one, we do not include messages sent from one device directly to the user that are never relayed to another device. This captures displayed information devices seek to confirm match via the user, such as the comparison values in Numeric Comparison or Signal. We also allow for the inclusion of any pre-exchanged messages the cryptographer deems necessary be appended to the beginning of the transcript. This case captures pre-shared data that is common in user-mediated protocols, such as IO capabilities in Bluetooth or ciphersuites in TLS.

We term this version of session identifier *strong* because it is both independent of the protocol being examined and allows one to test if any adversarial presence in protocol execution that may be overlooked by the weak version. This latter point has a profound impact on the provable security of a protocol. To demonstrate the importance of the distinction between strong and weak session identifiers, we examine the security of Passkey Entry under both weak (Section 6.3.2) and strong session identifiers (Section 6.3.1).

We conclude this section with three more important definitions. First, we present what it means for one session identifier to be a prefix of another and for two devices to hold matching session identifiers. We then present our rigorous definitions of DtD and UtD pairing. We present no definition of pairing among users, which is not possible because we only allow for one user identity.

Definition 5.1.2. We say that a session identifier sid_s^A with length $l \geq 1$ is a prefix of sid_t^B if

all values in sid_s^A match and are in the same order as the first l messages in sid_t^B .

Definition 5.1.3. We say that a device, A , at session s has matching session identifiers with device B at session t if we have that either:

- $\text{sid}_s^A = \text{sid}_t^B$ where A receives the last message in the protocol, or
- sid_t^B is a prefix of sid_s^A where A sends the last message in the protocol.

Definition 5.1.4 (Device-Device Partnering). We say sessions π_s^A and π_t^B are partnered if $\pi_s^A.\text{pid} = B$, $\pi_t^B.\text{pid} = A$, $\pi_s^A.\text{role} \neq \pi_t^B.\text{role}$, and A and B have matching session identifiers.

We adapt the definitions for prefix and matching session identifier from [69], where Jager, Kohlar, Schäge, and Schwenk employed similar definitions in an examination of TLS 1.3 to also account for the last message sent/received being dropped. We use this asymmetric definition for 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. Bellare and Rogaway [16] first recognized that the last message sent/received in a protocol had to be handled differently in analysis to prevent this kind of trivial win. In such a scenario, the combination of the above definitions would still allow for these two session oracles to be paired. This becomes important in our definition of the CYBORG security experiment to prevent a trivial win for an adversary that simply drops the last protocol message.

Definition 5.1.5 (User-Device Partnering). If $\pi_j^U.\text{device}_1 = (A, s)$ or $\pi_j^U.\text{device}_2 = (A, s)$, 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. We find this to be a reasonable modeling choice in keeping with the idea that a user can distinguish one pairing instance from another, such as executing pairing sessions over the passage of time or in different real-world settings.

5.2 Adversarial Model

The adversarial model within CYBORG was developed to provide maximum capability without allowing the protocol to be trivially broken. The main challenge in this construction stems from the capabilities an adversary should be given over the UtD channel. On one hand, giving the adversary full control of the channel at all times would be an unrealistic modeling of UtD communications. On the other hand, acting as if the UtD channel assumes away risk for the purposes of rendering a tidy proof. We have thus attempted to establish a middle ground between these two possibilities and follow in the wake of past work in three-party protocols where a human user plays an active role in the protocol.

5.2.1 Communication Channels

In this section we look to explicitly define the capabilities of an adversary in regard to messages sent over the DtD and UtD communication channels.

Definition 5.2.1. *An adversary may read, replay, delete, or modify any message sent between devices $A, B \in \mathcal{ID}$ and we call this the DtD channel.*

This definition for the DtD channel allows our analysis to operate under the assumption that an adversary has full control of the DtD communication channel. We do, however, slightly restrict the adversary over the UtD channel and designate two possibilities of UtD channels for use as required by protocol specification.

Definition 5.2.2 (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).*

Definition 5.2.3 (User-to-Device Channel, With Eavesdropping). *An adversary may replay, delete, or read 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, nor create any message. We call this the UtD channel with eavesdropping (UtD-E).*

Our current definitions assume an authenticated channel for devices to communicate with their user. This is captured by the fact that the adversary cannot modify a message's

content, its sender, nor its receiver in both the UtD and UtD-E channels. We see such a modification attack as not realistic for the basic adversary without device corruption due to the physical nature of UtD communications. Replay or deletion can be accomplished via faulty hardware in the IO mechanisms of a device that do not necessarily allow an attacker to interpret or modify the inputs. The read capability is addressed separately to account for protocols that require secrecy on the UtD channel (e.g., in Passkey Entry or ATM pin codes), from others that do not (e.g., Bluetooth Numeric Comparison or Signal). We give definitions for two channels, UtD and UtD-E, to model these different adversarial capabilities. The UtD-E channel specifically allows an adversary to read messages over the UtD channel, presumably via shoulder-surfing attacks or device compromise via keylogging or touchlogging. Meanwhile, the UtD channel does afford the adversary a read capability. We employ the UtD channel in our analysis of Passkey Entry (see Section 6.3 and Chapter 8) and the UtD-E channel in our analysis of Numeric Comparison (see Section 6.2).

5.2.2 Adversarial Queries

We now present a list of queries for the adversary to use when interacting with SSP participants. These queries not only allow an adversary to start a protocol run, but also to gain session specific information he would not otherwise be privy to learning. These queries were developed to model real-world compromise of a device or a user to ensure the analysis is as practical as possible. The `SendDevice` and `SendUser` queries are needed to allow an adversary to begin protocol runs between devices of his choosing or to send messages to a device or the user as desired. The `StateReveal`, `Corrupt`, and `KeyReveal` queries are included to give an adversary the ability to compromise a device and gain knowledge of its secrets. The two queries, `ShowUser` and `ControlUser`, are included to give an adversary the ability to compromise the UtD channel. The final query `Test`, is included to give the adversary the capability to test whether or not he can distinguish a derived session key from random.

`SendDevice` (π_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 $msg = (\text{start}, B)$ 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

role = initiator and pid = B and output the first protocol message. 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 pid = B , role = responder, and executes the protocol as intended in the role of the responder device. This allows the adversary to initiate a protocol run between two identities and to send legitimate messages to a device session oracle.

SendUser (π_s^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 $msg = (\text{start}, (\pi_s^A, \pi_t^B))$ 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 $msg = (\text{start}, \cdot)$ message for any session $\pi_{j'}^U$. If so, the session outputs $\perp$. Else, the session sets device₁ = (A, s) and device₂ = (B, t) . Else, if the first message received by π_j^U 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. This allows an adversary to initiate the user for a specific protocol run and to send legitimate messages to a user session oracle.

StateReveal (π_s^A)

The adversary may use this query to obtain access to the session state information state. This query is included to model compromise of a device's memory at the exact instance the query is called.

KeyReveal (π_s^A)

The adversary may use this query to obtain access to the session key skey. This query is used to model the leakage of a session key, either in its use in a channel or through device compromise.

ShowUser (π_s^A)

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_s^A . This query allows an adversary to gain

control of one-half of the UtD channel: communications from a device to a user. Following this query, the user will still perform in an honest fashion. This is done to model an adversary's ability to compromise a device and manipulate messages displayed to a user within a single session, which is why it requires a specific device session oracle as an input, such as with a Tag-based Adaptive Ploy.

ControlUser (π_s^U, 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. This query allows an adversary to gain control of the other half of the UtD channel: communications from a user to a device. Following this query, the user may no longer perform his role in an honest fashion when interacting with device A and is assumed to be an extension of the adversary. The ControlUser query models an adversary's ability to compromise a user and manipulate his actions via either social engineering or an attack on the device's input capability, such as a Ghost Touch Generator. As is often the case with social engineering or corruption of a device's touch screen, there is a time sensitive nature to the user corruption that is captured by the restriction of this corruption to a single session.

Test (π_s^A)

This query may only be asked once throughout the game. If $\pi_s^A.\delta \neq \text{accept}$, this query returns $\perp$. Else, it samples $b \xleftarrow{s} \{0, 1\}$, and sets $k \leftarrow \pi_s^A.\text{skey}$ if $b = 1$ and $k \xleftarrow{s} \{0, 1\}^\lambda$ otherwise. The query outputs k_b . This query simulates the adversarial capability of testing whether or not he can distinguish the session key from random.

5.2.3 User and Device Freshness

With all adversarial capabilities laid out, it now becomes important to define session freshness. This becomes important in the following section when we define security under the CYBORG experiment. Assume, for instance, that an adversary reveals all secret information and gains full control of the UtD channel. We would be hard pressed to find any protocol that provides any sort of guarantees under these trying conditions. Thus, we use freshness to capture the exact sessions an adversary is allowed to attack. Though this places somewhat

of a restriction on his capabilities, this is necessary to generate applicable results in our analysis.

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. We define device and user freshness over two settings: uncompromised user and compromised user. 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.

Definition 5.2.4 (Device Freshness). *We say that a device session oracle π_s^A is fresh in the uncompromised setting (UncUser-fresh) and fresh under compromised user type i (CompUser $_i$ -fresh) unless any of the following hold:*

- *the adversary issues a StateReveal(π_s^A) query, or*
- *if there exists a session oracle π_t^B partnered with π_s^A and the adversary issues a StateReveal(π_t^B) query, or*
- *the adversary has issued a KeyReveal(π_s^A) query, or*
- *if there exists a session oracle, π_t^B , partnered with π_s^A and the adversary issues a KeyReveal(π_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 5.2.5 (User Freshness under No Compromise). *We say that a session oracle π_j^U is fresh in the uncompromised setting (UncUser-fresh) unless any of the following hold:*

- *the adversary has issued a ShowUser(π_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 $\text{ControlUser}(\pi_j^U, A)$ 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 5.2.6. We claim that a session oracle π_j^U for U and session j is termed fresh-under-compromised user, type a ($\text{CompUser}_a\text{-fresh}$) unless either of the following hold:

- the adversary has issued a $\text{ShowUser}(\pi_s^A)$ 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 $\pi_s^A.\text{role} = \text{responder}$, or
- the adversary has issued a $\text{ControlUser}(\pi_j^U, A)$ 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 5.2.7. We claim that a session oracle π_j^U for U and session j is termed fresh-under-compromised user, type b ($\text{CompUser}_b\text{-fresh}$) unless either of the following hold:

- the adversary has issued a $\text{ShowUser}(\pi_s^A)$ 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 $\pi_s^A.\text{role} = \text{initiator}$, or
- the adversary has issued a $\text{ControlUser}(\pi_j^U, A)$ 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 5.2.8. We claim that a session oracle π_j^U for U and session j is termed fresh-under-compromised user, type c ($\text{CompUser}_c\text{-fresh}$) unless either of the following hold:

- the adversary has issued a $\text{ShowUser}(\pi_s^A)$ 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 $\text{ControlUser}(\pi_j^U, A)$ 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 $\pi_s^A.\text{role} = \text{responder}$.

Definition 5.2.9. We claim that a session oracle π_j^U for U and session j is termed fresh-under-compromised user, type d ($\text{CompUser}_d\text{-fresh}$) unless either of the following hold:

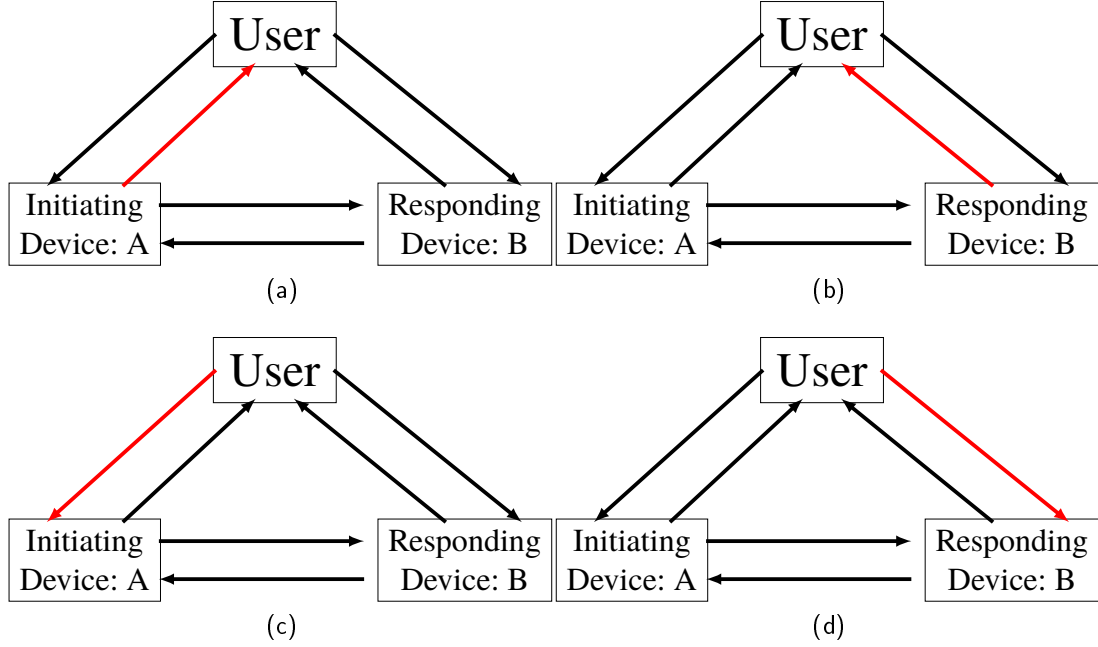


Figure 5.1. Visual depiction of the baseline four possibilities of CompUser-freshness where the adversary is allowed to corrupt only one direction of the UtD channel at a time, denoted by red coloring. We use the terms CompUser_a , CompUser_b , CompUser_c , and CompUser_d to describe the attack scenarios represented by Figure 5.1a, Figure 5.1b, Figure 5.1c, and Figure 5.1d, respectively.

- *the adversary has issued a $\text{ShowUser}(\pi_s^A)$ 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 $\text{ControlUser}(\pi_j^U, A)$ 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 $\pi_s^A.\text{role} = \text{initiator}$.*

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 in Figure 5.1). This decision allows us to focus on the exact circumstances under which a protocol breaks. We provide a visualization of the four baseline types of user compromise the CYBORG model captures in Figure 5.1.

We expand upon the baseline definitions presented in Definitions 5.2.6 to 5.2.9 by allowing for all combinations of these types.

Definition 5.2.10 (User Freshness under Compromise Type Combinations). *Let X be a non-empty subset of the set $\{a, b, c, d\}$; we refer to the set X with the label string x constructed from the elements of X in lexicographical order. We then say that a session oracle π_j^U for U and session j is termed fresh under compromised user, type x ($\text{CompUser}_x\text{-fresh}$) unless the adversary issues a single query which breaks $\text{CompUser}_{x_i}\text{-freshness}$ simultaneously for all $x_i \in X$.*

Note that a session oracle's freshness must be assessed *per* query under $\text{CompUser}_x\text{-fresh}$; said query must break freshness for the session oracle in all the elements of the set X individually. This structuring of $\text{CompUser}_x\text{-fresh}$ is needed so we can capture more advanced attacks that an adversary may mount. With simply Definitions 5.2.6 to 5.2.9 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 $\text{CompUser}_{ac}\text{-fresh}$ environment. In a $\text{CompUser}_{ac}\text{-fresh}$ 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 $\text{CompUser}_{bd}\text{-fresh}$ environment for the responding device. We list out all the types of $\text{CompUser}_x\text{-fresh}$ that are possible under Definitions 5.2.6 to 5.2.10: $a, b, c, d, ab, ac, ad, bc, bd, cd, abc, abd, acd, bcd, abcd$.

5.3 CYBORG Security Experiment

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

Definition 5.3.1. *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 $\text{EXP}_{\Pi, \mathcal{A}, \eta_p, \eta_s}^{\text{CYBORG-type}}$ 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 $\text{type} \in \{\text{UncUser}, \text{CompUser}\}$.*

1. Correctness (correct):

there exists two type -fresh and partnered device oracles π_s^A and π_t^B where:

- π_s^A and π_t^B are both partnered with the type-fresh user oracle π_j^U , and
- $\pi_s^A.\delta \neq \text{accept}$, or
- $\pi_t^B.\delta \neq \text{accept}$.

2. Entity Authentication (auth):

there exists a type-fresh session oracle π_s^A where:

- $\pi_s^A.\delta = \text{accept}$ with intended partner $\text{pid} = B$ and
- π_s^A is partnered with the type-fresh user oracle π_j^U , and
- if π_j^U is also partnered with π_t^B , $\mathcal{A}$ has not issued a $\text{StateReveal}(\pi_t^B)$ query while $\pi_s^A.\delta \neq \text{accept}$, and
- there does not exist a unique session oracle at B that is partnered with π_s^A .

3. Key Indistinguishability (key-ind):

at some point in the experiment $\mathcal{A}$ issued a $\text{Test}(\pi_s^A)$ query on a type-fresh session oracle π_s^A , where

- $\pi_s^A.\delta = \text{accept}$ with intended partner $\text{pid} = B$, and
- π_s^A is partnered with a type-fresh user oracle π_j^U , and
- if π_j^U is also partnered with π_t^B , $\mathcal{A}$ has not issued a $\text{StateReveal}(\pi_t^B)$ query while $\pi_s^A.\delta \neq \text{accept}$, and
- there exists an oracle π_t^B partnered with π_s^A , and
- at some subsequent point in the experiment π_s^A responds with its guess b , where $\Pr[b = \mathbf{b}] \geq 1/2$ where $\mathbf{b}$ is the randomly sampled bit from the associated Test query.

Else, the challenger outputs 0. If the adversary wins the experiment via the first condition then we say $\mathcal{A}$ breaks the correctness (correct) of Π . If the adversary wins the experiment via the second condition then we say $\mathcal{A}$ breaks entity authentication (auth) in Π . If the adversary wins the experiment via the third condition then we say $\mathcal{A}$ breaks key indistinguishability (key-ind) in Π . We denote the adversary $\mathcal{A}$ winning the experiment and the challenger outputting 1 as follows:

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

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

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

Definition 5.3.2. If there exists a negligible function $\text{negl}(\lambda)$ such that for all PPT adver-

series $\mathcal{A}$ interacting according to the CYBORG-type experiment, it holds that:

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

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`. However, forcing `correct` under `CompUser` is protocol dependent, and for CYBORG protocols like ISO/IEC 9798-6 Mechanism 7a and Signal this requirement is unachievable. In the ISO/IEC 9798-6 Mechanism 7a protocol the final exchanged messages in the protocol involve the user sending OK to both devices; the devices do not accept until they receive this message. In `CompUser` the adversary would be able to change this message to FAIL at will and possibly break the correctness of the protocol. META handles this as a trivial win for the adversary and, therefore, excludes breaking `correct` under `CompUser` from the model.

We also break from META in that we do not include any formulation of the compromised device setting. Recall that under the compromised device setting an adversary may reveal any device's secrets at will. This setting is not applicable to CYBORG AKEs because at some point a device will have to compute the session key, but under `CompDev` the adversary may reveal all device secrets immediately before this computation and trivially break key-ind.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:

Results of Analysis under the CYBORG Security Model

6.1 Association between User Compromise Types

Theorem 6.1.2 provides a framework for conducting our analysis of Numeric Comparison and Passkey Entry in the compromised user setting. We begin by starting from our baseline definitions of $\text{CompUser}_x\text{-fresh}$ within Definitions 5.2.6 to 5.2.9 and advance by incrementally introducing more capabilities. Conversely, if we prove that a protocol is $\text{CompUser}_{abcd}\text{-secure}$, then we know that it is secure for any CompUser_x variant.

Lemma 6.1.1. *Let X and Y be non-empty subsets of $\{a, b, c, d\}$ such that $X \subseteq Y$ with labels x and y , respectively. If a session oracle π_j^U is $\text{CompUser}_x\text{-fresh}$, then it is also $\text{CompUser}_y\text{-fresh}$.*

Proof. Let Π be a CYBORG key exchange protocol, and let $\mathcal{A}$ be a PPT adversary against the CYBORG- CompUser_x security of Π . If some user session oracle, π_j^U , is $\text{CompUser}_x\text{-fresh}$ then we must have that $\mathcal{A}$ never issued a single query that broke $\text{CompUser}_{x_i}\text{-freshness}$ simultaneously for all $x_i \in X$ over the course of the CYBORG- CompUser_x security experiment by Definition 5.2.10. This means that for every query issued by $\mathcal{A}$, π_j^U must have met the definition for $\text{CompUser}_{x_i}\text{-fresh}$ for some $x_i \in X$. Since $X \subseteq Y$, then we also have that π_j^U must have met the definition for $\text{CompUser}_{y_i}\text{-fresh}$ for some $y_i \in Y$ for every query issued by $\mathcal{A}$. Thus, $\mathcal{A}$ never issued a single query that broke $\text{CompUser}_{y_i}\text{-freshness}$ simultaneously for all $y_i \in Y$ and we have that π_j^U is also $\text{CompUser}_y\text{-fresh}$. $\square$

Theorem 6.1.2. *Let X and Y be non-empty subsets of $\{a, b, c, d\}$ 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- $\text{CompUser}_x\text{-secure}$, then it is not CYBORG- $\text{CompUser}_y\text{-secure}$.*

Proof. Let Π be a CYBORG key exchange protocol, and let $\mathcal{A}$ be a PPT adversary that breaks the CYBORG- CompUser_x security of Π . We then construct a second adversary, $\mathcal{B}$,

against the CYBORG-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 6.1.1, and Definition 5.2.10, we have CompUser_y freshness and the success of $\mathcal{B}$. Therefore, we have that

$$\text{Adv}_{\Pi, \mathcal{A}, \eta_p, \eta_s}^{\text{CYBORG-CompUser}_x}(\lambda) \leq \text{Adv}_{\Pi, \mathcal{B}, \eta_p, \eta_s}^{\text{CYBORG-CompUser}_y}(\lambda) .$$

□

6.2 Numeric Comparison

In this section, we present the initial results of our analysis of Numeric Comparison under the CYBORG security model. We define the session-state information for all versions of Numeric Comparison and a given session oracle π_s^A as:

$$\pi_s^A.\text{state} = (SK_a, N_a) ,$$

where all values are as defined in Chapter 4. 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.

6.2.1 Numeric Comparison under Strong Session Modeling

We define the session identifier for Numeric Comparison from a correctly executed session involving initiating device A and responding device B as:

$$\text{sid}_{strong} := ((A, B, \text{IOcapA}, \text{IOcapB}), PK_a, PK_b, C_b, N_a, N_b, \text{OK}, E_a, E_b) .$$

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.

Numeric Comparison does not meet any level of CYBORG security when analyzed under a strong session identifier. This is due to the fact that the initiating device does not authenticate receipt of IOcapB to the responding device, as demonstrated in Theorem 6.2.1.

Theorem 6.2.1. *Numeric Comparison is not CYBORG-UncUser-secure under strong session modeling.*

Proof. Let $\mathcal{A}$ be an adversarial algorithm against the CYBORG-UncUser security of the Initiator-Generated Passkey Entry protocol. The adversary first issues a $\text{SendDevice}(\pi_s^A, (\text{start}, B))$ to initiate a protocol run between devices A and B, and a $\text{SendUser}(\pi_j^U, (\text{start}, (\pi_s^A, \pi_t^B)))$ query to initiate the associated user oracle. When device B attempts to send IOcapB to device A, $\mathcal{A}$ modifies this message to a different value, IOcapE, that still enables pairing over Numeric Comparison to occur. $\mathcal{A}$ then allows the devices to progress through the protocol to step 20. Since device will attempt to verify E_b with the incorrect IOcap variable, the v_3 verification check will fail and device A will abort the protocol. However, device B will continue to step 21b and generate a link key. Thus, device B will accept but there will not exist a partnered session oracle at A and $\mathcal{A}$ wins the CYBORG-UncUser security experiment by breaking auth. $\square$

Although Theorem 6.2.1 presents a break in security under the CYBORG model, it is a trivial one because the purpose of the IOcap variable is to ensure that the two devices execute an applicable version of Secure Simple Pairing. In the above case, even in light of IOcapB being forged, both devices still execute Numeric Comparison as they intended. It is difficult to envision a real-world application where an adversary would want to modify this value but still allow the devices to pair exactly as they intended. The more applicable version of this kind of attack is the *BT-Niño-MitM* downgrade attack. Nevertheless, we discuss a modification in Chapter 7 that ensures authentication of both IOcap variables in Phase 3 to shore up this vulnerability.

6.2.2 Numeric Comparison under Weak Session Modeling

As a result of Numeric Comparison failing to achieve CYBORG security under strong session identifiers, we execute an analysis under weak session modeling. This enables further investigation of Numeric Comparison when the trivial attack described in Theorem 6.2.1 is excluded.

We define the session identifier for Numeric Comparison from a correctly executed session

involving initiating device A and responding device B as:

$$\text{sid}_{\text{weak}} := ((A, B, \text{IOcapA}), PK_a, PK_b, C_b, N_a, N_b, \text{OK}, E_a, E_b) .$$

This should come as an expected definition of the Numeric Comparison session identifier as it is constructed as described in Section 5.1.2 and it includes all information, outside of the constant “btlk,” used in the construction of the session key LK. The only information excluded, which would be included in the strong session identifier for Numeric Comparison, is the responder’s IO capability variable, IOcapB.

Theorem 6.2.2. *Numeric Comparison is CYBORG-UncUser-secure under the EC-DDH and EC-sym-ssPRF-ODH assumptions, the col-res security of H, and the SUF-CMA of HMAC under weak session modeling.*

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:

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

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

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

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 captures trivial collisions in the computation of the verification values V_a, V_b . Each session oracle computes one nonce in each session, therefore, we have that

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

Game 3. This game is equivalent to the previous security experiment except we guess the session oracles executing the protocol, the test session π_s^A and its partner π_t^B , and abort if $\mathcal{A}$ does not try to win against this guessed pair. Thus,

$$\text{Adv}_3 \geq \frac{1}{(\eta_p \eta_s)^2} \cdot \text{Adv}_2.$$

We continue by separating out cases. In particular, separately addressing adversarial win by correct (Case 1), auth (Case 2), or key-ind (Case 3):

$$\text{Adv}_3 = \text{Adv}_3^{\text{correct}} + \text{Adv}_3^{\text{auth}} + \text{Adv}_3^{\text{key-ind}}.$$

Case 1. Since session oracles with matching session identifiers are guaranteed to accept by the correctness of Numeric Comparison (Section 4.4.1), we have that:

$$\text{Adv}_3^{\text{correct}} = 0.$$

Case 2. Game 4. We continue with $\text{Adv}_3^{\text{auth}}$. 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 `StateReveal`(π_t^B) query while $\pi_s^A \neq \text{accept}$. We raise the event `abort`, end the experiment, and output zero if $\mathcal{A}$ succeeds in replacing PK_a, PK_b, N_a , and/or N_b .

Suppose that the adversary succeeds in this replacement, then we can construct another adversary, $\mathcal{B}_1$, against the collision resistance of H . Per the col-res experiment, $\mathcal{B}_1$ need only find two distinct messages $msg_1 \neq msg_2$ such that $H(msg_1) = H(msg_2)$. By our assumption, $\mathcal{A}$ succeeds in replacing PK_a, PK_b, N_a , and/or N_b (call these new values PK'_a, PK'_b, N'_a, N'_b); thus, the computed V_a and V_b verify correctly ($V_a = V_b$) when sent to the user. As the messages used to compute V_a and V_b do not match, since the devices do not agree on at least one of the public keys or nonces, $\mathcal{A}$ may forward the modified messages used by devices A and B , $msg_1 = PK_a || PK'_b || N_a || N'_b$ and $msg_2 = PK'_a || PK_b || N'_a || N_b$ respectively, to $\mathcal{B}_1$, who succeeds in breaking the col-res of H . Thus we have

$$\text{Adv}_4^{\text{auth}} \geq \text{Adv}_3^{\text{auth}} - \text{Adv}_{\mathcal{B}_1}^{\text{col-res}}(\lambda) .$$

Following this game we also know that the adversary may not forge C_b via the correctness of HMAC as the message and key used to compute C_b are now fixed.

Game 5. We replace DHKey with a uniformly random value $\widetilde{\text{DHKey}} = cP$ 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, $\mathcal{B}_2$, 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 $\widetilde{\text{DHKey}} = cP$, which is used for both partners. Algorithms $\mathcal{B}_2$ receives as input (E, P, aP, bP, cP) where $aP = PK_a$, $bP = PK_b$, and $cP = \widetilde{\text{DHKey}}$. $\mathcal{B}_2$ can simulate all flows for $\mathcal{A}$ correctly. Thus, the view of $\mathcal{A}$ when interacting with $\mathcal{B}_2$ is identical in Game 4 and Game 5 apart from the use of a random $\widetilde{\text{DHKey}}$. The EC-DDH assumption therefore implies that

$$\text{Adv}_5^{\text{auth}} \geq \text{Adv}_4^{\text{auth}} - \text{Adv}_{\mathcal{B}_2}^{\text{EC-DDH}}(\lambda) .$$

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

$$\text{Adv}_5^{\text{auth}} = \text{Adv}_5^{\text{auth}, C_1} + \text{Adv}_5^{\text{auth}, C_2} .$$

Sub-case 1: $\pi_s^A.\text{role} = \text{initiator}$.

Game 6. This game is equivalent to the previous security experiment except we raise the event `abort`, end the experiment, and output zero if $\mathcal{A}$ succeeds in forging E_b , or B , or some combination thereof.

We construct a new $\mathcal{B}_3$, against the SUF-CMA security of HMAC. The challenger sets the MAC key to $\overline{\text{DHKey}}$ as per the SUF-CMA experiment, and $\mathcal{B}_3$ uses the oracle MAC to compute tags. $\mathcal{B}_3$ calls $\text{MAC}(N_b \| N_a \| r_a \| \text{IOcapB} \| B \| A)$, which returns the tag E_b (note that r , N_b , and N_a are all public at this point in the protocol). $\mathcal{B}_3$ then forwards this response to $\mathcal{A}$. $\mathcal{A}$ gives the message-tag pair $(N_b \| N_a \| r_a \| \text{IOcapB} \| B \| A, E_b)$ to $\mathcal{A}$. If $\mathcal{A}$ is able to forge a new tag, call it E_{win} , leading to malicious acceptance, or a new message, call it msg_{win} , by forging B , such that $\text{MAC.Vfy}(\overline{\text{DHKey}}, msg_{win}, E_b) = 1$, then $\mathcal{B}_3$ can win the SUF-CMA experiment with the winning message-tag pair. Thus we have

$$\text{Adv}_6^{\text{auth}, C_1} \geq \text{Adv}_5^{\text{auth}, C_1} - \text{Adv}_{\text{MAC}, \mathcal{B}_3}^{\text{SUF-CMA}}(\lambda) .$$

Thus we have matching `sid`, $\pi_s^A.\text{pid} = B$, $\pi_t^B.\text{pid} = A$. The adversary can only succeed in breaking `auth` by breaking role agreement among our test and partner sessions. However, this is impossible since the responder, and only the responder, always computes and sends a commitment value, C_b , to the initiator as part of Numeric Comparison. Thus we have

$$\text{Adv}_7^{\text{auth}, C_1} = 0 .$$

Sub-case 2: $\pi_s^A.\text{role} = \text{responder}$. This case follows similarly to Sub-case 1, but we abort if $\mathcal{A}$ succeeds in forging E_b , B , and/or `IOcapB` causing A to accept maliciously in the responder role.

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

$$\begin{aligned} \text{Adv}^{\text{auth}} &\leq \text{Adv}_{\mathcal{B}_1}^{\text{col-res}}(\lambda) + \text{Adv}_{\mathcal{B}_2}^{\text{EC-DDH}}(\lambda) \\ &\quad + 2 \cdot \text{Adv}_{\text{MAC}, \mathcal{B}_3}^{\text{SUF-CMA}}(\lambda) . \end{aligned}$$

Case 3. Game 4. We now bound $\text{Adv}_3^{\text{key-ind}}$. 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 `StateReveal`(π_t^B) query while $\pi_s^A \neq \text{accept}$.

Suppose that $\mathcal{A}$ can distinguish between the games. 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 $x = (N_a \| N_b \| \text{btlk} \| A \| B)$ 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 $b = 1$ 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 $LK_b = \text{HMAC}(\text{DHKey}, N_a \| N_b \| \text{btlk} \| A \| B)$, 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 $LK_b \xleftarrow{\$} \{0, 1\}^\lambda$. Thus by the success of $\mathcal{A}$ in distinguishing Game 4 and Game 5, we have the success of $\mathcal{B}_4$:

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

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

$$\text{Adv}_4^{\text{key-ind}} = 0.$$

Combining all previous probability statements we arrive at the final security reduction:

$$\begin{aligned} \text{Adv}_{\text{NC}, \mathcal{A}, \eta_p, \eta_s}^{\text{CYBORG-UncUser}}(\lambda) &\leq (\eta_p \eta_s)^2 \cdot \left(\frac{1}{2^\mu} + \frac{1}{2^\lambda} + \right. \\ &\quad \text{Adv}_{\mathcal{B}_1}^{\text{col-res}}(\lambda) + \\ &\quad \text{Adv}_{\mathcal{B}_2}^{\text{EC-DDH}}(\lambda) + 2 \cdot \text{Adv}_{\text{MAC}, \mathcal{B}_3}^{\text{SUF-CMA}}(\lambda) + \\ &\quad \left. \text{Adv}_{\text{MAC}, \mathcal{B}_4}^{\text{EC-sym-ssPRF-ODH}}(\lambda) \right). \end{aligned}$$

□

From Chapter 4, we know that μ is based off 255-bit keys and λ is based off 128-bit keys, both within NIST's standards of security [55]. We also know that the elliptic curve used will be P-256 under Secure Connections; though not proven, no inherent computational weaknesses have been found to forgo its use [66]. Additionally, the HMAC and Hash function used in Numeric Comparison are HMAC-SHA256 and SHA-256. Both HMAC-SHA256 and SHA-256 are NIST approved algorithms and no consequential breaks in their security have been presented [55]. We also have that the HMAC construction was proven to meet Brendel, Fischlin, Günther, and Janson's highest version of PRF-ODH security in [72]. To address the $(\eta_p, \eta_s)^2$ term, Bluetooth recommends that pairing be executed as few times as possible. In addition, the hash values V_a and V_b are truncated down to 6 digits (about 20 bits) for the user. Although this bit-length is *not* in keeping with NIST standards, Bluetooth considers this length acceptable [11]. Thus, we have that our col-res reduction is acceptable by Bluetooth standards and all other reductions are computationally hard.

Lemma 6.2.3. *Numeric Comparison is not CYBORG-CompUser_b-secure under strong session modeling.*

Proof. We prove Lemma 6.2.3 via a counterexample by describing how an adversary may mount a successful MitM attack on the pairing devices. This leads to both devices accepting without pairing with each other. Additionally, the adversary gains full knowledge of the derived link keys generated by both devices. Let $\mathcal{A}$ be the adversarial algorithm against the CYBORG-CompUser_b security of Numeric Comparison. The Adversary first issues a SendDevice($\pi_s^A, (\text{start}, B)$) and a SendUser($\pi_j^U, (\text{start}, (\pi_s^A, \pi_t^B))$) query to initiate protocol participants. $\mathcal{A}$ then issues a ShowUser(π_t^B) query and proceeds with a MitM attack as shown in Figure 6.1. We describe the steps of the attack as follows:

1. $\mathcal{A}$ executes Phase 1 of the protocol with Device A in the guise of B and similarly with B; whereby, he exchanges his public key PK_e with each of the devices.
2. $\mathcal{A}$ and Device A generate DHKey_a, while $\mathcal{A}$ and Device B generate DHKey_b.
3. $\mathcal{A}$ generates a nonce value N_e and sets both r_a and r_b to 0. Similarly, devices A and B generate the nonces N_a and N_b , respectively, and both set r_a and r_b to 0.

4. $\mathcal{A}$ sends a commitment value $C_e = \text{HMAC}(N_e, PK_{ex} \| PK_{ax} \| 0)$ to A and receives a commitment value $C_b = \text{HMAC}(N_b, PK_{bx} \| PK_{ex} \| r_b)$ from B.
5. $\mathcal{A}$ exchanges generated nonces with both A and B.
6. A verifies that $\text{Vfy}(N_e, PK_{ex} \| PK_{ax} \| r_a, C_e) = 1$, which verifies correctly as a result of the definition of MAC. Similarly for $\mathcal{A}$ in regard to C_b .
7. $\mathcal{A}$ and A generate the confirmation values $V_e = V_a = H(PK_{ax} \| PK_{ex} \| N_a \| N_e)$ and B generates the confirmation value $V_b = H(PK_{ex} \| PK_{bx} \| N_e \| N_b)$. A and B then send their confirmation values, V_a, V_b , to U.
8. Since $\mathcal{A}$ issued a $\text{ShowUser}(\pi_t^B)$ query, he can create messages from B to U in this run of the protocol. He then deletes the message, V_b , from B to U and instead sends the message V_e from B to U. Since $V_e = V_a$ an honest user will confirm these values are equal and send an OK message to both A and B.
9. A generates the check value $E_a = \text{HMAC}(\text{DHKey}_a, N_a \| N_e \| r_b \| \text{IOcapA} \| A \| B)$, $\mathcal{A}$ generates the check values $E_e = \text{HMAC}(\text{DHKey}_b, N_e \| N_b \| r_b \| \text{IOcapA} \| A \| B)$ and $E_{e'} = \text{HMAC}(\text{DHKey}_a, N_e \| N_a \| r_a \| \text{IOcapB} \| B \| A)$, and B generates the check value $E_b = \text{HMAC}(\text{DHKey}_b, N_e \| N_b \| r_a \| \text{IOcapB} \| B \| A)$. A then sends E_a to $\mathcal{A}$ and $\mathcal{A}$ sends E_e to B.
10. B will then verify that $\text{Vfy}(\text{DHKey}_b, (N_e \| N_b \| r_b \| \text{IOcapA} \| A \| B), E_e) = 1$. This check will pass because B honestly executes the protocol with $\mathcal{A}$ under the belief he is A. Similarly for $\mathcal{A}$'s verification of E_a . B then sends E_b to $\mathcal{A}$ and $\mathcal{A}$ sends $E_{e'}$ to A.
11. A will then verify that $\text{Vfy}(\text{DHKey}_a, (N_e \| N_a \| r_a \| \text{IOcapB} \| B \| A), E_{e'}) = 1$. This check will pass because A honestly executes the protocol with $\mathcal{A}$ under the belief he is B. Similarly for $\mathcal{A}$'s verification of E_b .
12. The attack ends with A and $\mathcal{A}$ generating the shared key $\text{LK}_a = \text{HMAC}(\text{DHKey}_a, N_a \| N_e \| \text{btlk} \| A \| B)$, and B and $\mathcal{A}$ generating the shared key $\text{LK}_b = \text{HMAC}(\text{DHKey}_b, N_e \| N_b \| \text{btlk} \| A \| B)$. Both A and B then accept under the assumption that they have paired with each other.

At the conclusion of the above attack we have that $\mathcal{A}$ has won the CYBORG-CompUser_b security experiment by breaking auth since π_s^A is CYBORG-CompUser_b -fresh and has accepted, π_j^U is CompUser_a -fresh, but there does not exist a partnered session oracle for π_s^A . $\square$

We envision an adversary executing the attack presented in Lemma 6.2.3 by embedding malware into a device that enables one to modify device display outputs. Similar in function to the exploits in the Strandhogg Vulnerability and a TAP-like attack, hackers have already demonstrated the capability to modify device outputs as desired to display corrupted messages. A potential mitigating factor to the feasibility of this attack is that the displayed value cannot be pre-set into the embedded malware because it relies on device A's public key (a value we assume to be ephemeral). However, since the displayed value is only a single hash function with 768 bytes of input, we suspect that this computation can be done in a fraction of a millisecond on modern computers [75].

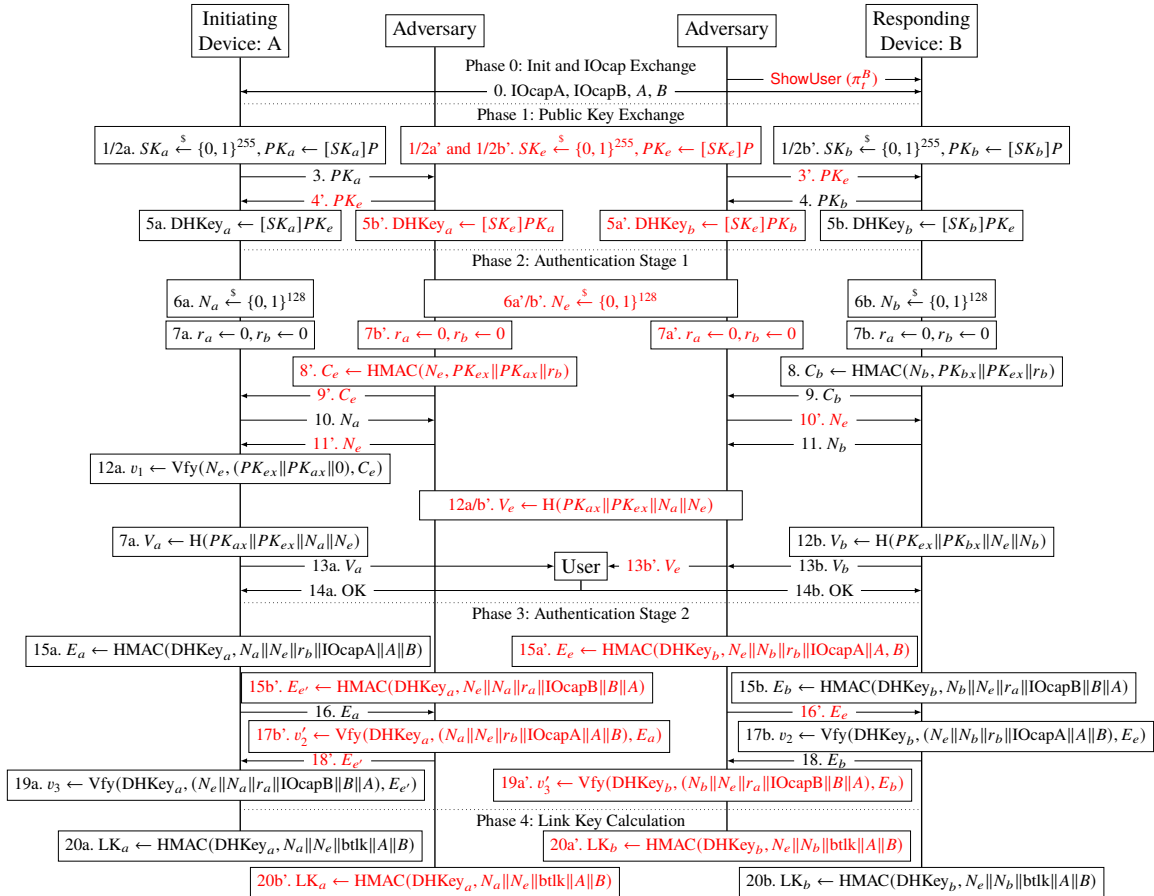


Figure 6.1. Depiction of adversarial attack (singular actions shown in red) against Numeric Comparison under the CYBORG-CompUser_b security experiment; the attack runs similarly to the BT-Niño-MitM attack.

Lemma 6.2.4. *Numeric Comparison is not CYBORG-CompUser_a-secure under strong*

session modeling.

Proof. This proof runs similarly to the one described in Lemma 6.2.3 with the exception that $\mathcal{A}$ issues a ShowUser (π_s^A) query, computes $V_e = V_b = H(PK_{ex} \| PK_{bx} \| N_e \| N_b)$ in Phase 2, and replaces the message sent from A to U with V_e . An honest user will still verify that $V_e = V_b$, send OK to both A and B, and allow the attack to proceed. At the conclusion of the attack we will have a similar result as in Lemma 6.2.3. $\square$

Lemma 6.2.5. *Numeric Comparison is not CYBORG-CompUser_c-secure under strong session modeling.*

Proof. We prove this by describing how an adversary may mount a successful MitM attack on the initiating device. This leads to the initiating device accepting without pairing with the responder. Additionally, the adversary gains full knowledge of its derived link key. Let $\mathcal{A}$ be the adversarial algorithm against the CYBORG-CompUser_c security of Numeric Comparison. The Adversary first issues a SendDevice($\pi_s^A, (\text{start}, B)$) and a SendUser($\pi_j^U, (\text{start}, (\pi_s^A, \pi_t^B))$) query to initiate protocol participants. $\mathcal{A}$ then issues a ControlUser (π_j^U, A) query and proceeds with a MitM attack as shown in Figure 6.2. We describe the steps of the attack as follows:

1. $\mathcal{A}$ executes Phase 1 of the protocol with Device A in the guise of B and similarly with B; whereby, he exchanges his public key PK_e with each of the devices.
2. $\mathcal{A}$ and A then generate DHKey_a, and $\mathcal{A}$ and Device B generate DHKey_b.
3. $\mathcal{A}$ generates a nonce value N_e and sets both random values to 0.
4. $\mathcal{A}$ sends a commitment value $C_e = \text{HMAC}(N_e, PK_{ex} \| PK_{ax} \| 0)$ to A and receives a commitment value $C_b = \text{HMAC}(N_b, PK_{bx} \| PK_{ex} \| 0)$ from B.
5. $\mathcal{A}$ exchanges generated nonces with both A and B, and visa versa.
6. A verifies that $\text{Vfy}(N_e, PK_{ex} \| PK_{ax} \| 0, C_e) = 1$. This check will pass as result of honest execution of the protocol with the agreed upon values from Phase 1.
7. A generates the confirmation value $V_a = H(PK_{ax} \| PK_{ex} \| N_a \| N_e)$ and B generates the confirmation value $V_b = H(PK_{ex} \| PK_{bx} \| N_e \| N_b)$. A and B then send their confirmation values, V_a, V_b , to U.
8. We will then have that $V_a \neq V_b$, U will attempt to send FAIL messages to both devices A and B. Since $\mathcal{A}$ issued a ControlUser (π_j^U, A) query, he can create messages from

U to A in this run of the protocol. $\mathcal{A}$ will then delete U's original message, and then send OK to device A. $\mathcal{A}$ will subsequently only continue the protocol run with device A as device B will reject as a result of receiving the FAIL message.

9. A generates the check value $E_a = \text{HMAC}(\text{DHKey}_a, N_a \| N_e \| r_b \| \text{IOcapA} \| A \| B)$ and $\mathcal{A}$ generates the check values $E_e = \text{HMAC}(\text{DHKey}_a, N_e \| N_a \| r_a \| \text{IOcapB} \| B \| A)$. A then sends E_a to $\mathcal{A}$ for verification.
10. $\mathcal{A}$ will verify that $\text{Vfy}(\text{DHKey}_a, (N_a \| N_e \| r_b \| \text{IOcapA} \| A \| B), E_a) = 1$. This check will pass because A honestly executes the protocol with $\mathcal{A}$ under the belief he is B. $\mathcal{A}$ will then send E_e to A.
11. A will then verify that $\text{Vfy}(\text{DHKey}_a, (N_e \| N_a \| r_a \| \text{IOcapB} \| B \| A), E_e) = 1$. This check will pass because A honestly executes the protocol with $\mathcal{A}$ under the belief he is B.
12. The attack ends with A and $\mathcal{A}$ generating the shared key $\text{LK}_a = \text{HMAC}(\text{DHKey}_a, N_a \| N_e \| \text{btlk} \| A \| B)$. A then accepts under the assumption that it paired with device B.

At the conclusion of the above attack we have that $\mathcal{A}$ has won the CYBORG-CompUser_c security experiment by breaking auth since π_s^A is CYBORG-CompUser_c -fresh and has accepted, π_j^U is CompUser_a -fresh, but there does not exist a partnered session oracle for π_s^A . $\square$

We envision an adversary executing the attack presented in Lemma 6.2.5 by embedding malware into a device that enables one to modify device inputs. A Ghost touch generator [4] would certainly enable this type attack by corrupting the user's press of the OK button on a touchscreen-enabled device. An adversary could also minimize his possible workload executing a CompUser_c -type attack against Numeric Comparison by pre-setting a device to always accept regardless of user input.

Lemma 6.2.6. *Numeric Comparison is not CYBORG-CompUser_d -secure under strong session modeling.*

Proof. This proof runs similarly to the one described in Lemma 6.2.5 with the exception that $\mathcal{A}$ issues a $\text{ControlUser}(\pi_j^U, B)$ query, replaces the FAIL message sent from U to B with OK, and continues the attack with B instead of A. At the conclusion device B will accept but there will not exist a partnered session oracle for B at device A. $\square$

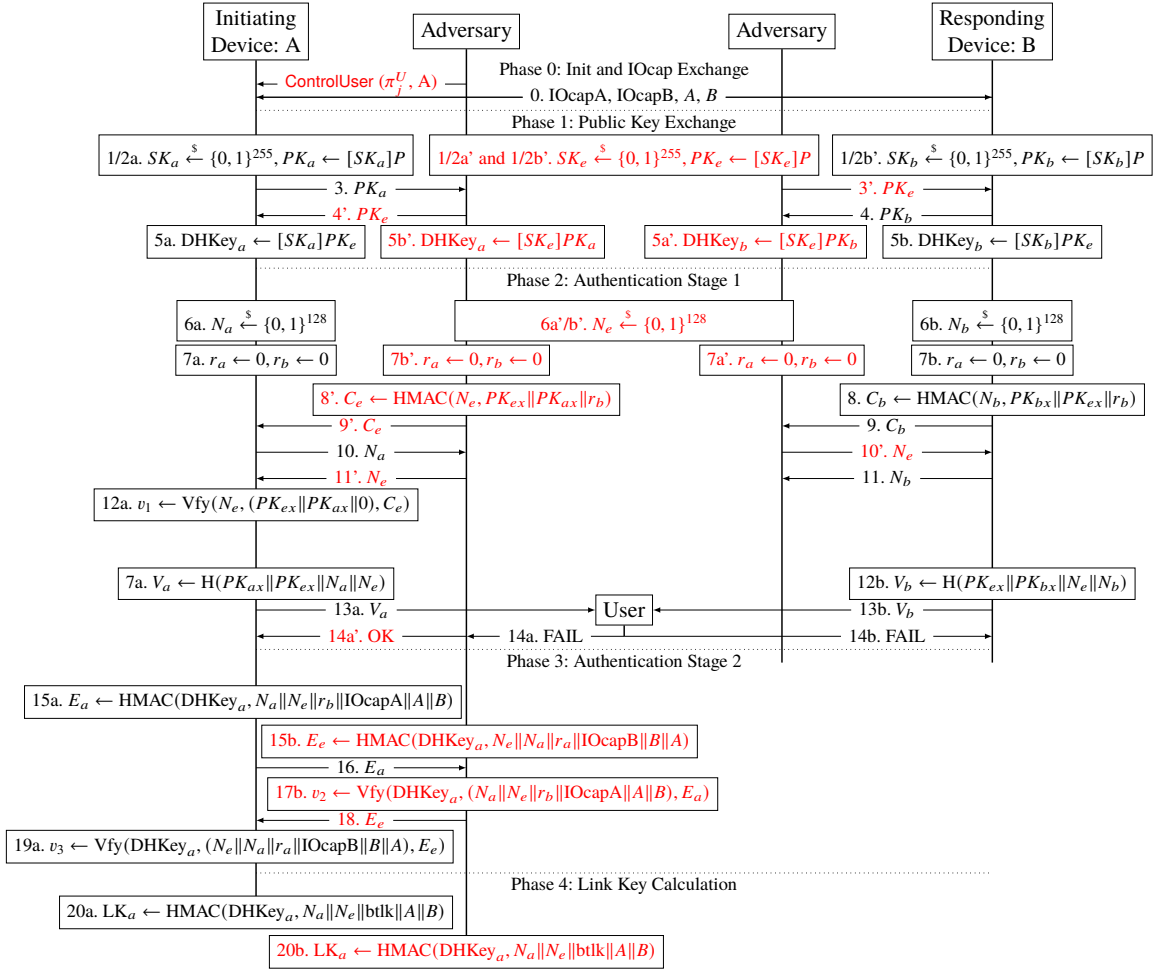


Figure 6.2. Depiction of adversarial attack (singular actions shown in red) against Numeric Comparison under the CYBORG-CompUser_c security experiment; similar to the CYBORG-CompUser_b attack but utilizes a corruption of the user's communication to the device.

Theorem 6.2.7. *Secure Simple Pairing with Numeric Comparison is not CYBORG-CompUser_x secure for*

$$x \in \{a, b, c, d, ab, ac, ad, bc, bd, cd, abc, abd, acd, bcd, abcd\}$$

under strong session modeling.

Proof. By Lemmas 6.2.3 to 6.2.6 we have that Numeric Comparison is not CYBORG-

CompUser_x secure for $x \in \{a, b, c, d\}$. Since one of a, b, c , or d must be an element of every non-empty subset of $\{a, b, c, d\}$, our desired result follows directly from Theorem 6.1.2. $\square$

The results described in this section highlight the weakness of Numeric Comparison under compromised user attacks. Although Numeric Comparison utilizes all four communication pathways between the device and the user, if the adversary has the capability to compromise messages sent over at least one of these pathways, then he gains an avenue to defeat the protocol. We would also like to note that although all of these counterexamples were conducted in the UtD-E setting, they would also hold in the UtD setting as none of our presented attacks rely on reading the actual messages sent between devices and the user in Numeric Comparison.

6.3 Passkey Entry

In this section we present the initial results of our analysis of Passkey Entry under the CYBORG security model. We define the session-state information for all versions of Passkey Entry and a given session oracle π_s^A as:

$$\pi_s^A.\text{state} = (SK_a, r, N_{a,1}, \dots, N_{a,20}) ,$$

where all values are as defined in Chapter 4. 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 [13], 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.

6.3.1 Passkey Entry under Strong Session Modeling

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 5.1.2:

$$\begin{aligned} \text{sid}_{\text{strong}} := & ((A, B, \text{IOcapA}, \text{IOcapB}), PK_a, PK_b, r, C_{a,1}, \dots \\ & \dots, C_{b,1}, N_{a,1}, N_{b,1}, C_{a,20}, C_{b,20}, N_{a,20}, N_{b,20}, E_a, E_b). \end{aligned}$$

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.

All versions of Passkey Entry do not meet any level of CYBORG security when analyzed under a strong session identifier. Even though Passkey Entry complicates the attack discussed in Section 4.3.3 by using 20 commitments, it allows for breaks in auth. We present attacks that succeed with probability $1/2$ against all versions of Passkey Entry, and an attack that succeeds with probability 1 against User-Generated Passkey Entry depicted in Figure 6.3. Ultimately, the decision to rely on a single bit of security when computing the commitments in Phase 2 is too weak an assumption in the computational setting.

Theorem 6.3.1. *Initiator-Generated Passkey Entry is not CYBORG-UncUser-secure under strong session modeling.*

Proof. We prove Theorem 6.3.1 via a counterexample by describing how an adversary can entice a device to accept maliciously.

Let $\mathcal{A}$ be an adversarial algorithm against the CYBORG-UncUser security of the Initiator-Generated Passkey Entry protocol. The adversary first issues a $\text{SendDevice}(\pi_s^A, (\text{start}, B))$ to initiate a protocol run between devices A and B , and a $\text{SendUser}(\pi_j^U, (\text{start}, (\pi_s^A, \pi_t^B)))$ 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 $C_{e,i} = \text{HMAC}(N_{e,i}, PK_{ax} || PK_{bx} || r_{e,i})$. 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

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

□

Theorem 6.3.2. *Responder-Generated Passkey Entry is not CYBORG-UncUser-secure under strong session modeling.*

Theorem 6.3.2 follows similarly to Theorem 6.3.1.

Theorem 6.3.3. *User-Generated Passkey Entry is not CYBORG-UncUser-secure under strong session modeling.*

Proof. We prove Theorem 6.3.3 via a counterexample by describing how an adversary may mount a successful “role confusion” attack on the pairing devices. This attack leads to both devices accepting but neither actually pairs with the other.

Let $\mathcal{A}$ be an adversarial algorithm against the CYBORG-UncUser security of the User-Generated Passkey Entry protocol. $\mathcal{A}$ first issues a $\text{SendUser}(\pi_j^U, (\text{start}, (\pi_s^A, \pi_t^B)))$ to initiate the user for a protocol run between devices A and B. $\mathcal{A}$ then issues both a $\text{SendDevice}(\pi_s^A, (\text{start}, B))$ and a $\text{SendDevice}(\pi_t^B, (\text{start}, A))$ 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. □

The real-world feasibility of the attacks presented in Theorems 6.3.1 and 6.3.3 are not equal. Theorem 6.3.1 is a realistic attack for any Dolev-Yao type adversary as the only requirement is the ability to modify messages sent between devices and execute bit guesses. On the other hand, Theorem 6.3.3 has questionable real-world application. For one, both

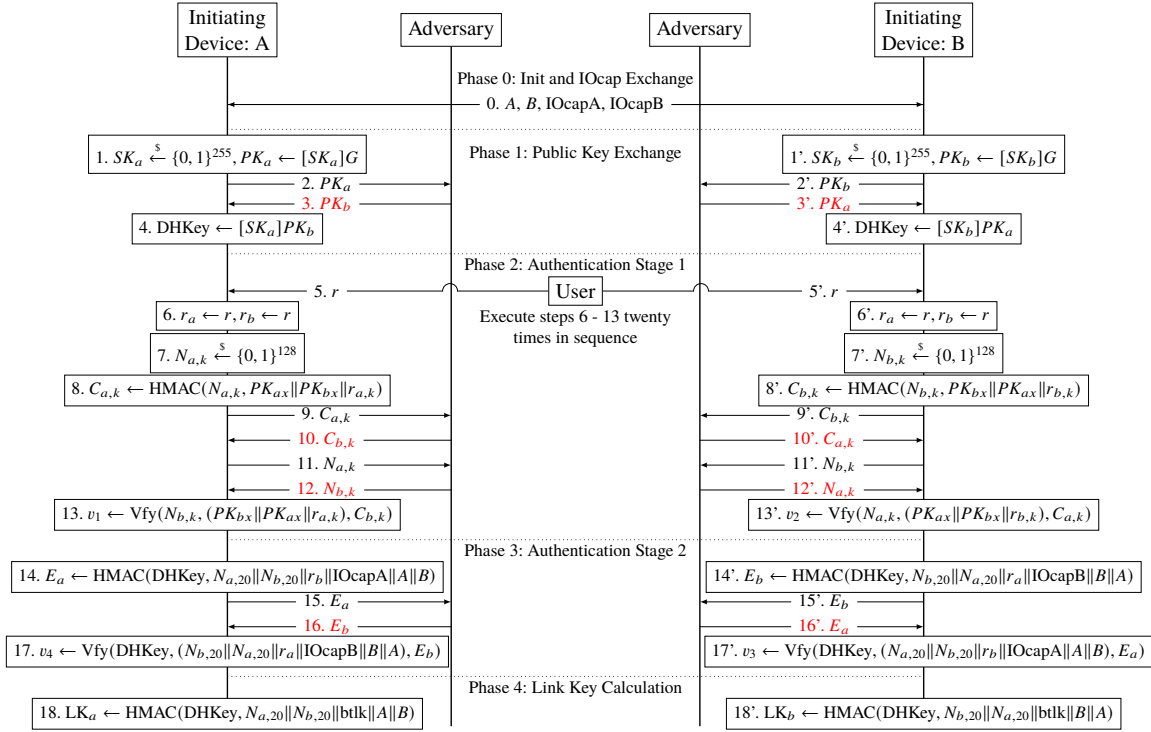


Figure 6.3. Depiction of adversarial attack (singular actions for the adversary shown in red) against User-Generated Passkey Entry. Adversary relays messages between two devices that both think they are the initiator.

devices must simultaneously attempt to pair with each other for this attack to progress. This is possible if a user attempted to initiate a Bluetooth pairing on two devices, vice only initiating the connection on one. However, it is difficult to pinpoint what an adversary *gains* from executing the Theorem 6.3.3 attack outside of simply denial of service. On an unrelated note, this attack cannot progress if both devices are in the responder role because this would force the adversary to perform the initial computation for $C_{a,i}$ involving the unknown passkey.

6.3.2 Passkey Entry under Weak Session Modeling

As a result of all versions of Passkey Entry failing to achieve CYBORG security under strong session identifiers, we execute an analysis under weak session modeling. This enables further investigation of Passkey Entry when the relevant attack described in Theorem 6.3.1 and the trivial attack described in Theorem 6.3.3 are excluded.

Now we consider a weak session identifier for Passkey Entry for a correctly executed session involving initiating device, e.g., A , and responding device, e.g., B , as:

$$\text{sid}_{\text{weak}} := ((A, B, \text{IOcap}A), PK_a, PK_b, N_{a,20}, N_{b,20}, r, E_a, E_b) .$$

Again, 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. We leave our definition of session-state unchanged. Under this weaker session-identifier, Initiator-Generated and Responder-Generated Passkey Entry can be proved to be a CYBORG-UncUser-secure protocol.

Although our proof in Theorem 6.3.4 presents Initiator and Responder-Generated Passkey Entry as CYBORG-UncUser-secure, it does so under the premise of a misleading session identifier. Instead of including all messages that ought to match in the session identifier, we only include those messages that are involved in the computation of the session key LK at the conclusion of the protocol. The attacks presented in the previous section against Initiator and Responder-Generated Passkey Entry still remain, but are veiled due to modeling choices. In addition to setting a poor precedent of cherry-picking the values for a session identifier to manufacture a tidy proof, this also hides possible security vulnerabilities from users and implementors. It is important to not mask possible security vulnerabilities in modeling, but instead to relay as much information as possible. We thus include this examination to highlight how definition of a protocol's session identifier has far-reaching effects on its provable security. This is also useful for proving the exact messages within a protocol that come with authentication guarantees.

Initiator-Generated

Theorem 6.3.4. *Initiator-Generated Passkey Entry is CYBORG-UncUser-secure under the EC-DDH and EC-sym-ssPRF-ODH assumptions, and the SUF-CMA of HMAC under weak session modeling.*

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:

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

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:

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

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:

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

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:

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

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_s^A and its partner π_t^B , and abort if $\mathcal{A}$ does not try to win against this guessed pair. Thus,

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

We continue by separating out cases. In particular, separately addressing adversarial win by correct (Case 1), auth (Case 2), or key-ind (Case 3):

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

Case 1. Since session oracles with matching session identifiers are guaranteed to accept by the correctness of Passkey Entry (Section 4.4.2), we have that:

$$\text{Adv}_4^{\text{correct}} = 0 .$$

Case 2. Game 5. We continue with $\text{Adv}_4^{\text{auth}}$. 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 `StateReveal`(π_t^B) query while $\pi_s^A \neq \text{accept}$. We raise the event `abort`, end the experiment, and output zero if $\mathcal{A}$ succeeds in replacing PK_a or PK_b .

In order to replace PK_a or PK_b , $\mathcal{A}$ must either 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,

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

Game 6. We replace `DHKey` with a uniformly random value $\widetilde{\text{DHKey}} = cP$ 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 $\widetilde{\text{DHKey}} = cP$, which is used for both partners. Algorithms $\mathcal{B}_1$ receives as input (E, P, aP, bP, cP) where $aP = PK_a$, $bP = PK_b$, and $cP = \widetilde{\text{DHKey}}$. $\mathcal{B}_2$ can simulate all flows for $\mathcal{A}$ correctly. Thus, the view of $\mathcal{A}$ when interacting with $\mathcal{B}_2$ is identical in Game 5 and Game 6 apart from the use of a random $\widetilde{\text{DHKey}}$. The EC-DDH assumption therefore implies that

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

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

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

Sub-case 1: $\pi_s^A.\text{role} = \text{initiator}$.

Game 7. This game is equivalent to the previous security experiment except we raise the event `abort`, end the experiment, and output zero if $\mathcal{A}$ succeeds in forging E_b , $N_{b,20}$, or B , or some combination thereof. Note that while $\mathcal{A}$ may generate a new nonce $N_{b,20}$ and commitment $C_{b,20}$ which it passes to A earlier in the protocol, $N_{b,20}$ is covered by the HMAC algorithm run on $\widetilde{\text{DHKey}}$, so $N_{b,20}$ will only be accepted if the adversary can also forge the HMAC.

We construct a new $\mathcal{B}_2$, against the SUF-CMA security of MAC. The challenger sets the MAC key to $\widetilde{\text{DHKey}}$, and $\mathcal{B}_2$ uses the oracle MAC to compute tags. $\mathcal{B}_2$ calls $\text{MAC}(N_{b,20} \| N_{a,20} \| r_a \| \text{IOcapB} \| B \| A)$, which returns the tag E_b (note that r , $N_{b,20}$, and $N_{a,20}$ are all public at this point in the protocol). $\mathcal{B}_2$ then forwards this response to $\mathcal{A}$. gives the message-tag pair $(N_{b,20} \| N_{a,20} \| r_a \| \text{IOcapB} \| B \| A, E_b)$ to $\mathcal{A}$. If $\mathcal{A}$ is able to forge a new tag, call it E_{win} , leading to malicious acceptance, or a new message, call it msg_{win} , by forging some combination of $N_{b,20}$, and/or B , such that $\text{MAC.Vfy}(\widetilde{\text{DHKey}}, \text{msg}_{\text{win}}, E_b) = 1$, then

$\mathcal{B}_2$ can win the SUF-CMA experiment with the winning message-tag pair Thus we have

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

Thus we have matching sid, $\pi_s^A.\text{pid} = B$, $\pi_t^B.\text{pid} = A$. The adversary can only succeed in breaking auth by breaking role agreement among our test and partner sessions. However, this is impossible since the initiator always sends the passkey r to the user and the responder always receives the passkey r from the user as part of Initiator-Generated Passkey Entry. Thus we have

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

Sub-case 2: $\pi_s^A.\text{role} = \text{responder}$. This case follows similarly to Sub-case 1, but we abort if $\mathcal{A}$ succeeds in forging E_b , $N_{b,20}$, B , and/or IOcapB causing A to accept maliciously in the responder role.

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

$$\begin{aligned} \text{Adv}^{\text{auth}} &\leq \frac{1}{2^{|r|-1}} + \text{Adv}_{\mathcal{B}_1}^{\text{EC-DDH}}(\lambda) \\ &\quad + 2 \cdot \text{Adv}_{\text{MAC}, \mathcal{B}_2}^{\text{SUF-CMA}}(\lambda) . \end{aligned}$$

Case 3. Game 5. We now bound $\text{Adv}_4^{\text{key-ind}}$. 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 StateReveal(π_t^B) query while $\pi_s^A \neq \text{accept}$.

Suppose that $\mathcal{A}$ can distinguish between the games. Then we can construct a new adversary, $\mathcal{B}_3$, solving the EC-sym-ssPRF-ODH problem as follows. $\mathcal{B}_3$ receives as input (E, P, aP, bP) where P is a generator of our elliptic curve group, $aP = PK_a$, and $bP = PK_b$. $\mathcal{B}_3$ then issues the challenge query $x = (N_{a,20} \| N_{b,20} \| \text{btlk} \| A \| B)$ also using values as chosen by our test and partner oracles. The challenger then randomly samples b and sets $LK \leftarrow \text{HMAC}(\text{DHKey}, N_{a,20} \| N_{b,20} \| \text{btlk} \| A \| B)$ if $b = 1$ and $LK \xleftarrow{\$} \{0, 1\}^\lambda$ otherwise. The challenger then returns LK_b to $\mathcal{B}_3$. 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, $\mathcal{B}_3$ can simulate all other flows between our test session and partner session. If we have that $LK_b = \text{HMAC}(\text{DHKey}, N_{a,20} || N_{b,20} || \text{btlk} || A || B)$, 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 $LK_b \xleftarrow{\$} \{0, 1\}^\lambda$. Thus by the success of $\mathcal{A}$ in distinguishing Game 4 and Game 5, we have the success of $\mathcal{B}_3$:

$$\text{Adv}_5^{\text{key-ind}} \geq \text{Adv}_4^{\text{key-ind}} - \text{Adv}_{\text{MAC}, \mathcal{B}_3}^{\text{EC-sym-ssPRF-ODH}}(\lambda).$$

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

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

Combining all previous probability statements we arrive at the final security reduction:

$$\begin{aligned} \text{Adv}_{\text{PE-IG}, \mathcal{A}, \eta_p, \eta_s}^{\text{CYBORG-UncUser}}(\lambda) &\leq (\eta_p \eta_s)^2 \cdot \left(\frac{1}{2^\mu} + \frac{400}{2^\lambda} + \frac{3}{2^{|r|}} + \right. \\ &\quad \left. \text{Adv}_{\mathcal{B}_1}^{\text{EC-DDH}}(\lambda) + 2 \cdot \text{Adv}_{\text{MAC}, \mathcal{B}_2}^{\text{SUF-CMA}}(\lambda) + \right. \\ &\quad \left. \text{Adv}_{\text{MAC}, \mathcal{B}_3}^{\text{EC-sym-ssPRF-ODH}}(\lambda) \right). \end{aligned}$$

□

The discussion following Theorem 6.2.2 holds for the security of Initiator-Generated Passkey Entry with the exception of $|r|$, which is only 20 bits in length. A key size of 20 bits is well below NIST approved key size but is acceptable by Bluetooth standards [11]. Additionally, 20-bit passkeys provides closer to 18 bits of security in the final security reduction. The strength of the passkey could always be increased by generating larger decimal values or possibly allowing for alphanumeric passkeys when able. With only 20 bits of security, it is paramount that implementors heed Bluetooth's recommendation of minimizing pairing attempts.

Theorem 6.3.5. *Initiator-Generated Passkey Entry is CYBORG-CompUser_x-secure for*

$$x \in \{b, c, bc\}$$

under weak session modeling.

Proof. This proof follows from a triviality. In all three of the above listed CYBORG-CompUser_x security environments, the adversary gains the capability to issue queries that allow him to compromise a UtD channel that are not employed in Initiator-Generated Passkey Entry. Therefore, these settings reduce to the CYBORG-UncUser setting, which was proven secure in Theorem 6.3.4. $\square$

Lemma 6.3.6. *Initiator-Generated Passkey Entry is not CYBORG-CompUser_a-secure under weak session modeling.*

Proof. We proceed via counterexample. Let $\mathcal{A}$ be an adversarial algorithm against the CYBORG-CompUser_a security of the Initiator-Generated Passkey Entry protocol. $\mathcal{A}$ first issues a SendDevice(π_s^A , (start, B)) query and a SendUser(π_j^U , (start, (π_s^A , π_t^B))) query to initiate protocol participants. $\mathcal{A}$ then issues a ShowUser(π_s^A) query and proceeds with the MitM attack as shown in Figure 6.4. We describe the steps of the attack as follows:

1. $\mathcal{A}$ executes Phase 1 of the protocol with Device B in the guise of A and similarly with B; whereby, he exchanges his public key PK_e with B.
2. $\mathcal{A}$ and Device B generate DHKey_b.
3. Device A selects the passkey, r and sends it to the user.
4. Since $\mathcal{A}$ issued a ShowUser(π_s^A) query, he modifies r to a new, known passkey r_e . This new value is then forwarded to the user.
5. User sends the value r_e to device B.
6. $\mathcal{A}$ and device B set their values for r_a and r_b to r_e .
7. Repeat steps 8 through 12 for values $1 \leq k \leq 20$.
8. Device B generates a nonce value $N_{b,k}$ and $\mathcal{A}$ generates a nonce value $N_{e,k}$.
9. $\mathcal{A}$ sends a commitment value $C_{e,k} = \text{HMAC}(N_{e,k}, PK_{ex} || PK_{bx} || r_{a,k} || \text{init})$ to B and receives a commitment value $C_{b,k} = \text{HMAC}(N_{b,k}, PK_{bx} || PK_{ex} || r_{b,k} || \text{resp})$ from B.
10. $\mathcal{A}$ sends its nonce $N_{e,k}$ to device B for verification.
11. B verifies that $\text{Vfy}(N_{e,k}, PK_{ex} || PK_{bx} || r_{a,k} || \text{init}, C_{e,k}) = 1$. B then sends its nonce $N_{b,k}$ to $\mathcal{A}$ for verification.
12. $\mathcal{A}$ verifies that $\text{Vfy}(N_{b,k}, PK_{bx} || PK_{ex} || r_{b,k}, C_{e,k}) = 1$.

13. $\mathcal{A}$ generates the check value $E_e = \text{HMAC}(\text{DHKey}_b, N_{a,20} \| N_{b,20} \| r_b \| \text{IOcapA} \| A \| B)$, $\mathcal{A}$ and sends it to B.
14. B generates the check value $E_b = \text{HMAC}(\text{DHKey}_b, N_{b,20} \| N_{a,20} \| r_a \| \text{IOcapB} \| B \| A)$.
15. B will then verify that $\text{Vfy}(\text{DHKey}_b, (N_{a,20} \| N_{b,20} \| r_b \| \text{IOcapA} \| A \| B), E_e) = 1$. This check will pass because B honestly executes the protocol with $\mathcal{A}$ under the belief he is A. B then sends E_b to $\mathcal{A}$.
16. $\mathcal{A}$ will then verify that $\text{Vfy}(\text{DHKey}_b, (N_{b,20} \| N_{a,20} \| r_a \| \text{IOcapB} \| B \| A), E_b) = 1$.
17. The attack ends with B and $\mathcal{A}$ generating the shared key $\text{LK}_b = \text{HMAC}(\text{DHKey}_b, N_{a,20} \| N_{b,20} \| \text{btlk} \| A \| B)$. B then accepts under the assumption that it just paired with A.

At the conclusion of the above attack we have that $\mathcal{A}$ has won the CYBORG-CompUser_a security experiment by breaking auth since π_t^B has accepted, π_t^B is CompUser_a -fresh, π_j^U is CompUser_a -fresh, but there does not exist a paired device session oracle.

□

Lemma 6.3.7. *Initiator-Generated Passkey Entry is not CYBORG-CompUser_d -secure.*

Proof. This proof runs similarly to the one described in Lemma 6.3.6 with a few changes. Instead of issuing a $\text{ShowUser}(\pi_s^A)$ query, $\mathcal{A}$ issues a $\text{ControlUser}(\pi_j^U, B)$ query. This allows him to modify the passkey of r to r_e on input to π_t^B . The rest of the attack can progress as previously described.

□

Theorem. *Initiator-Generated Passkey Entry is not CYBORG-CompUser_x -secure for*

$$x \in \{a, d, ab, ac, ad, bd, cd, abc, abd, acd, bcd, abcd\}$$

under weak session modeling.

Proof. By Lemmas 6.3.6 and 6.3.7 we have that Initiator-Generated Passkey Entry is not CYBORG-CompUser_x for $x \in \{a, d\}$. Therefore, we have that Initiator-Generated Passkey Entry is not CYBORG-CompUser_x secure for $X \subseteq \{a, b, c, d\}$ where at least one of a or d is an element of X , by applying Theorem 6.1.2.

□

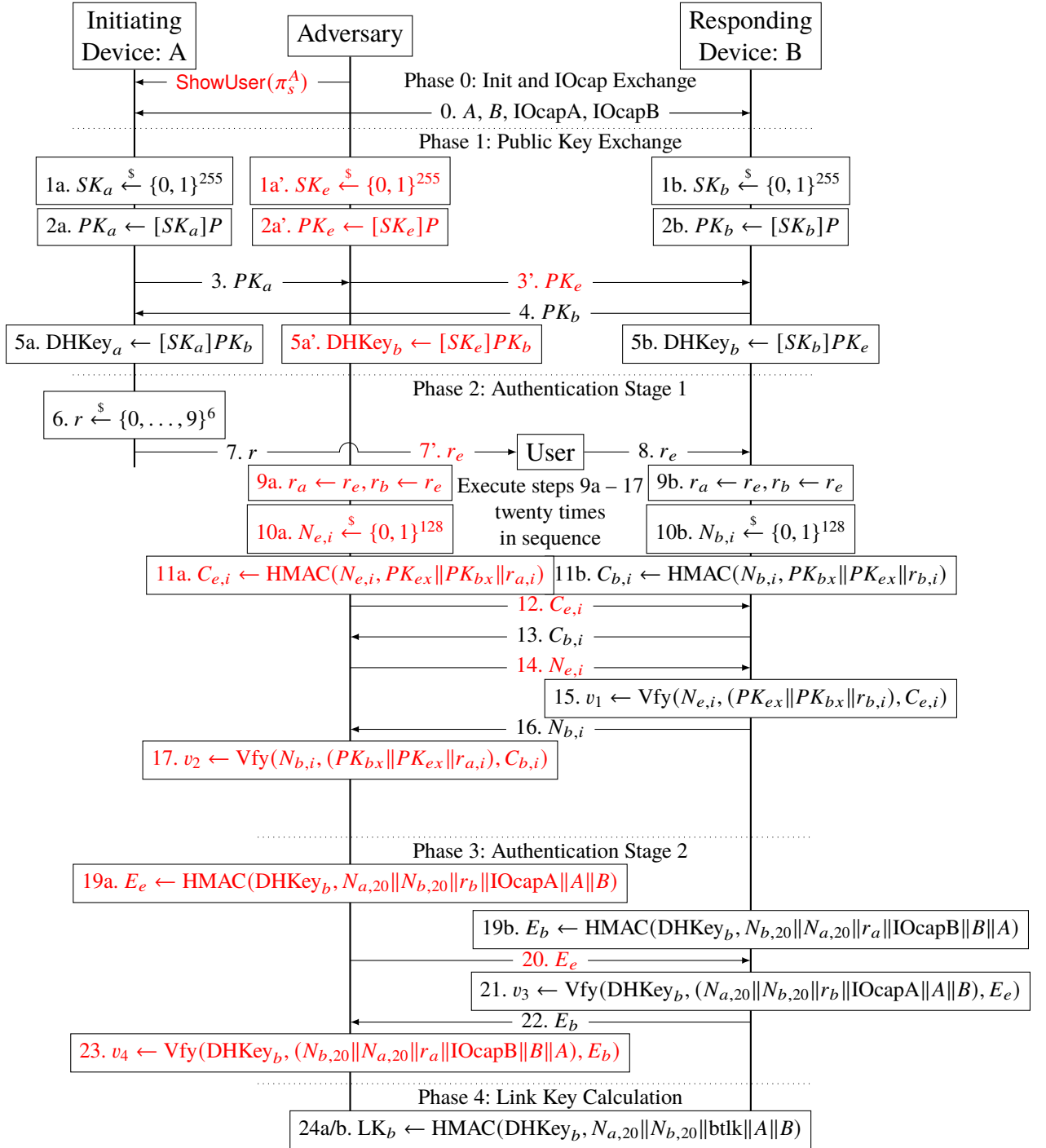


Figure 6.4. Depiction of adversarial attack (singular actions shown in red) against Initiator-Generated Passkey Entry under the CYBORG-CompUser_a security experiment, discussed in Lemma 6.3.6. Any instance where the communication flow “jumps” over a device line is done to illustrate adversarial inability to read the message.

Responder-Generated

Theorem 6.3.8. *Responder-Generated Passkey Entry is CYBORG-UncUser-secure under the EC-DDH and EC-sym-ssPRF-ODH assumptions, and the SUF-CMA of HMAC under weak session modeling.*

This proof follows similarly to Theorem 6.3.4.

Theorem 6.3.9. *Responder-Generated Passkey Entry is CYBORG-CompUser_x-secure for*

$$x \in \{a, d, ad\}$$

under weak session modeling.

Proof. This proof follows from a triviality. In all three of the above listed CYBORG-CompUser_x security environments, the adversary may compromise UtD channels that are not employed in Responder-Generated Passkey Entry. Therefore, these settings reduce to the CYBORG-UncUser setting, which was proven secure in Theorem 6.3.8. $\square$

Lemma 6.3.10. *Responder-Generated Passkey Entry is not CYBORG-CompUser_b-secure under weak session modeling.*

Proof. This proof runs similarly to the one described for Lemma 6.3.6, but with A in the responder role and B as initiator. $\square$

Lemma 6.3.11. *Responder-Generated Passkey Entry is not CYBORG-CompUser_c-secure under weak session modeling.*

Proof. This proof runs similarly to the one described in Lemma 6.3.6 with a few changes. Instead of issuing a ShowUser(π_s^A) query, the adversary $\mathcal{A}$ issues a ControlUser(π_j^U, A) query. This allows him to modify the passkey of r to r_e on input to π_s^A . $\square$

Theorem 6.3.12. *Responder-Generated Passkey Entry is not CYBORG-CompUser_x-secure for*

$$x \in \{b, c, ab, ac, bc, bd, cd, abc, abd, acd, bcd, abcd\}$$

under weak session modeling.

Proof. By Lemmas 6.3.10 and 6.3.11 we have that Responder-Generated Passkey Entry is not CYBORG-CompUser _{x} for $x \in \{b, c\}$. Therefore, we have that Responder-Generated Passkey Entry is not CYBORG-CompUser _{x} secure for $X \subseteq \{a, b, c, d\}$ where at least one of b or c is an element of X by applying Theorem 6.1.2. $\square$

User-Generated

Theorem 6.3.13. *User-Generated Passkey Entry is not CYBORG-UncUser-secure under weak session modeling.*

This proof follows similarly to Theorem 6.3.3.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 7:

Proposed Modification to Bluetooth Pairing

In this chapter we present two modified Passkey Entry protocol variants, see Figure 7.1, with the following adaptations. The first modification is the Secure Hash Modification (SHM), which is a modification constructed for the current variants of Passkey Entry to rectify the vulnerabilities discussed in Chapter 6. The second is Dual Passkey Entry (DPE), a wholly new variant of passkey entry developed to achieve greater security in under CYBORG – CompUser_x environments. In Chapter 8 we will show that, with these modifications, Passkey Entry can achieve complete CYBORG security under a strong session identifier and the UtD without eavesdropping channel.

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.

7.1 Secure Hash Modification (SHM)

Motivation for the Secure Hash Modification stemmed from the investigation of Passkey Entry security under weak and strong session identifiers. Recall that all versions of Passkey Entry were provably insecure under strong session modeling due to not authenticating all generated nonces and not authenticating the responding device’s IO capability. Additionally, User-Generated Passkey Entry was susceptible to a role confusion attack. Secure Hash Modification rectifies all of these faults with minimal changes to existing Passkey Entry architecture. First, it requires devices to compute two hashes involving all of the initiating and responding devices nonces, $N_{a,i}, N_{b,i}$. Also included in the computation of these hashes are text strings for device role to ensure explicit confirmation of assumed role. Finally, both IO capability variables, IOcapA, IOcapB, are included in the computation of E_a, E_b . By concluding the protocol with an authentication of the entire transcript by both devices, the Secure Hash Modification mirrors techniques used in TLS 1.3 (i.e., the computation of the Finished message) to prevent downgrade attacks. Of note, 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) and does not prevent more direct downgrade attacks like

BT–Niño–MitM.

7.1.1 Phase 0: Init and IOcap Exchange.

No changes.

7.1.2 Phase 1: Public Key Exchange.

No changes.

7.1.3 Phase 2: Authentication Stage 1.

Each device declares their role in the computation of the commitment values $C_{a,i}, C_{b,i}$ using the labels “init” and “resp” for the initiator and responder respectively. At the conclusion of Phase 2, all of a device’s previously generated nonces are concatenated and hashed into N_a and N_b in steps 8c and 8d. N_a and N_b replace $N_{a,20}$ and $N_{b,20}$ in all further Phases. No other changes.

7.1.4 Phase 3: Authentication Stage 2.

Modified such that previously computed values for N_a and N_b are used in the computation of check values E_a and E_b . Both IOcap variables are included in the computation of E_a and E_b . No other changes.

7.1.5 Phase 4: Link Key Calculation.

Modified such that previously computed values for N_a and N_b are used in computation of the link key LK . No other changes.

7.2 Dual Passkey Entry (DPE)

Motivation for Dual Passkey Entry stems from a desire to achieve a greater level of CYBORG security. All current versions of Bluetooth pairing are still susceptible to Tap ‘n Ghost-like attacks. Dual Passkey Entry on the other hand, proposes a method for protecting against advanced UtD attacks by having both devices generate and exchange passkeys. The observant reader of Chapter 6, specifically the analysis of Passkey Entry under CompUser_x security, may have been able to guess this construction from all of our adversary’s actions

when attacking Passkey Entry. The adversary never attacks the device that generates the passkey. Thus, the modification proposed by Dual Passkey Entry in regard to the original variants of Passkey Entry, is similar to the idea of bi-directional certificates vs uni-directional certificates. In this sense, the passkey shared by the generator in original Passkey Entry is akin to a server-only certificate shared with a client for authentication (the user performs the function of the certificate authority). Dual Passkey Entry, on the other hand, requires that the “client” respond with a passkey of its own for bi-directional authentication.

This modification requires the initiator and responder to both possess a numerical display and numerical input capabilities, a KeyboardDisplay type device as described by the Bluetooth classification from Section 4.1. 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. 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. Note that Numeric Comparison only achieves the lowest level of CYBORG security (UncUser – secure from Theorem 6.2.2), we show in Theorem 8.4.1 that Dual Passkey Entry achieves the strongest level of CYBORG security. Although we do not rule out any possible modification of Numeric Comparison to improve it’s security, attempting to devise said modification was not investigated in this thesis.

7.2.1 Phase 0: Init and IOcap Exchange.

No changes.

7.2.2 Phase 1: Public Key Exchange.

No changes.

7.2.3 Phase 2: Authentication Stage 1.

Both the initiating and responding device compute and exchange passkeys, r_a and r_b respectively. Both passkeys are then included in the computation and verification of the commitment values C_a and C_b .

7.2.4 Phase 3: Authentication Stage 2.

No changes.

7.2.5 Phase 4: Link Key Calculation.

No changes

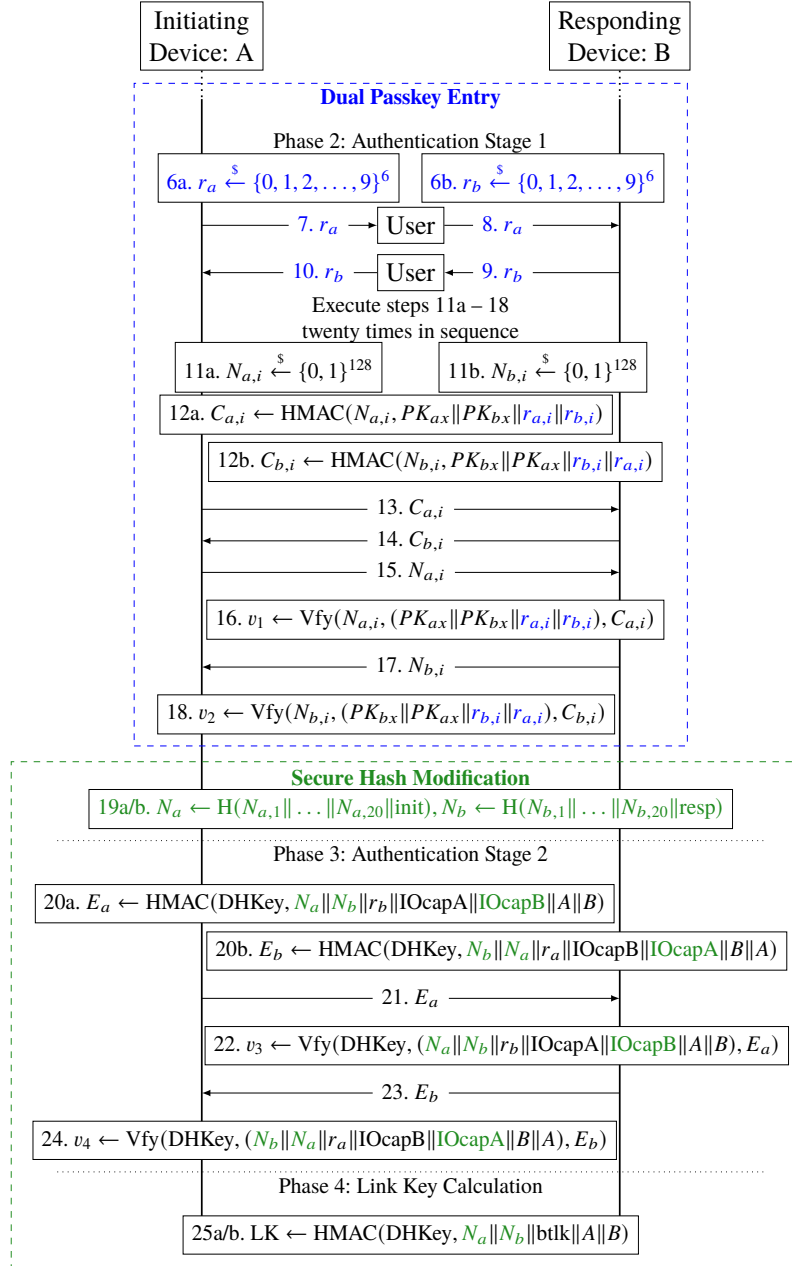


Figure 7.1. Dual Passkey Entry with Secure Hash Modification. **Dual Passkey Entry (DPE)** is depicted with the modifications in blue, where both devices generate passkeys. The **Secure Hash Modification (SHM)** is depicted green and can be used in conjunction with DPE or any version of Passkey Entry. 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 .

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 8:

Results of Analysis of Modified Bluetooth Protocol

In this section we present our analysis of the three original variants of Passkey Entry when operated with the Secure Hash Modification under strong session modeling. We refer to these “modified” versions as SHM Initiator/Responder/User-Generated Passkey Entry. Using Theorem 6.1.2, 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.

We prove SHM Dual Passkey Entry’s security in the CompUser_{abcd} environment under strong session modeling, which via Theorem 6.1.2 demonstrates its total CYBORG security across all variants. All proofs operate under the assumption that the adversary is confined via the UtD without eavesdropping channel, as required for execution of Passkey Entry, and definitions for strong session identifiers and state from Section 6.3.1.

8.1 SHM Initiator-Generated Passkey

Theorem 8.1.1. *SHM Initiator-Generated Passkey Entry is CYBORG-UncUser-secure under the EC-sym-ssPRF-ODH and EC-DDH assumptions, the sec-pre security of H, and the SUF-CMA security of HMAC under strong session modeling.*

Proof. The proof of this theorem follows closely to that of Theorem 6.3.4, but with the following alterations.

Case 2. Sub-case 1: $\pi_s^A.\text{role} = \text{initiator}$.

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 $\overline{\text{DHKey}}$, and $\mathcal{B}_2$ uses the oracle

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

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

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 8 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 $(N_{b,1} \| \dots \| N_{b,20} \| \text{role}_b, N_b)$ 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 $N_b = H(N_{b,1} \| \dots \| N_{b,20} \| \text{role}_b)$. 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 $N'_{b,1} \| \dots \| N'_{b,20} \| \text{role}'_b$ as its guess at a second preimage for N_b . By the success of $\mathcal{A}$ we have that $H(N'_{b,1} \| \dots \| N'_{b,20} \| \text{role}'_b) = N_b$. Thus we have

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

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 `sid`, $\pi_s^A.\text{pid} = B$, $\pi_t^B.\text{pid} = A$, $\pi_s^A.\text{role} \neq \pi_t^B.\text{role}$ telling us our session oracles are partnered via Definition 5.1.4 Thus we have

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

Sub-case 2: $\pi_s^A.\text{role} = \text{responder}$. This case follows similarly to Sub-case 1.

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

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

This leads to our final security reduction:

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

□

Theorem 8.1.2. *SHM Initiator-Generated Passkey Entry is CYBORG-CompUser_x-secure for*

$$x \in \{b, c, bc\}$$

under strong session modeling.

Proof. This proof runs similarly to the one described in Theorem 6.3.5. □

Theorem 8.1.3. *SHM Initiator-Generated Passkey Entry is not CYBORG-CompUser_x-secure for*

$$x \in \{a, d, ab, ac, ad, bd, cd, abc, abd, acd, bcd, abcd\}$$

under strong session modeling.

Proof. This proof runs similarly to the one described in Section 6.3.2. □

8.2 SHM Responder-Generated Passkey

Theorem 8.2.1. *SHM Responder-Generated Passkey Entry is CYBORG-UncUser-secure under the EC-sym-ssPRF-ODH and EC-DDH assumptions, the sec-pre security of H, and the SUF-CMA security of HMAC under strong session modeling.*

Proof. This proof runs similarly to the one described in Theorem 8.1.1. □

Theorem 8.2.2. *SHM Responder-Generated Passkey Entry is CYBORG-CompUser_x-secure for*

$$x \in \{a, d, ad\}$$

under strong session modeling.

Proof. This proof runs similarly to the one described in Theorem 6.3.9. □

Theorem 8.2.3. *SHM Responder-Generated Passkey Entry is not CYBORG-CompUser_x-secure for*

$$x \in \{b, c, ab, ac, bc, bd, cd, abc, abd, acd, bcd, abcd\}$$

under strong session modeling.

Proof. This proof runs similarly to the one described in Theorem 6.3.12. □

8.3 SHM User-Generated Passkey

Theorem 8.3.1. *SHM User-Generated Passkey Entry is CYBORG-UncUser-secure under the EC-sym-ssPRF-ODH and EC-DDH assumptions, the sec-pre security of H, and the SUF-CMA security of HMAC under strong session modeling.*

Proof. This proof follows similarly to Theorem 8.1.1. 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-UncUser environment. □

Theorem 8.3.2. *SHM User-Generated Passkey Entry is CYBORG-CompUser_x-secure for*

$$x \in \{a, b, ab\}$$

under strong session modeling.

Proof. This proof follows from a triviality. In all three of the above listed CYBORG-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-UncUser setting, which was proven secure in Theorem 8.2.1. $\square$

Lemma 8.3.3. *SHM User-Generated Passkey Entry is not CYBORG-CompUser_c-secure under strong session modeling.*

Proof. This proof runs similarly to the one described in Lemma 6.3.11. The adversary $\mathcal{A}$ issues a ControlUser(π_j^U, A) query, which allows him to modify the passkey of r to r_e on input to π_s^A . The adversary may then proceed to MitM communications with π_s^A and break auth. $\square$

Lemma 8.3.4. *SHM User-Generated Passkey Entry is not CYBORG-CompUser_d-secure under strong session modeling.*

Proof. This proof runs similarly to the one described in Lemma 6.3.7. The adversary $\mathcal{A}$ issues a ControlUser(π_j^U, B) query, which allows him to modify the passkey of r to r_e on input to π_t^B . The adversary may then proceed to MitM communications with π_t^B and break auth. $\square$

Theorem 8.3.5. *User-Generated Passkey Entry is not CYBORG-CompUser_x-secure for*

$$x \in \{c, d, ac, ad, bc, bd, cd, abc, abd, acd, bcd, abcd\}$$

under strong session modeling.

Proof. By Lemmas 8.3.3 and 8.3.4 we have that SHM User-Generated Passkey Entry is not CYBORG-CompUser_x for $x \in \{c, d\}$. Therefore, we have that SHM User-Generated Passkey Entry is not CYBORG-CompUser_x-secure for $X \subseteq \{a, b, c, d\}$ where at least one of c or d is an element of X , by applying Theorem 6.1.2. $\square$

8.4 SHM Dual Passkey

The proof for SHM Dual Passkey follows from two key ideas. First we rely on the contrapositive of Theorem 6.1.2. From Theorem 6.1.2, we know that if SHM Dual Passkey is $\text{CYBORG-CompUser}_{abcd}$ -secure, it is also secure in all other CYBORG security experiments. Secondly, we need only show that the integrity of at least one of the exchanged passkeys in SHM Dual Passkey remains uncompromised in the CompUser_{abcd} environment. Specifically, with both devices generating passkeys, the adversary can neither gain knowledge nor forge the target session's passkey. The adversary cannot observe the exchange of passkeys between the devices and user because by definition they are exchanged over the UtD without eavesdropping channel. The adversary cannot forge the target session's passkey because none of the allowed queries in the CompUser_{abcd} allow for this capability. This ensures at least one passkey remains secret from the adversary throughout execution of SHM Dual Passkey. We then have that all other reductions can proceed similarly to our proof of SHM Initiator-Generated Passkey Entry in the UncUser setting (Theorem 8.1.1), which also rely on the security of a single passkey.

Theorem 8.4.1. *SHM Dual Passkey Entry is $\text{CYBORG-CompUser}_{abcd}$ -secure under the EC-sym-ssPRF-ODH and EC-DDH assumptions, the sec-pre security of H, and the SUF-CMA security of HMAC under strong session modeling.*

Proof. The proof of this theorem follows closely to that of Theorem 8.1.1, but with the following alterations. Following the proof of Theorem 8.1.1, 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.

Case 2. Game 5. We continue with $\text{Adv}_4^{\text{auth}}$. Since we have the requirement that π_s^A remains fresh, we will abort the experiment if $\mathcal{A}$ issues a `StateReveal()` or `KeyReveal` queries such that π_s^A is no longer CompUser_{abcd} -fresh. This implies that $\mathcal{A}$ may issue `ControlUser` queries involving the partnered user oracle π_j^U as desired, since we are under CompUser_{abcd} environment π_j^U remains fresh regardless of adversarial action. Similarly $\mathcal{A}$ may issue `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 `StateReveal`(π_t^B) query while $\pi_s^A \neq \text{accept}$. 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 $\text{ShowUser}(\pi_t^B)$ or $\text{ControlUser}(\pi_j^U, B)$ 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

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

□

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 9:

Formal Analysis in Tamarin

The formal analysis of Secure Simple Pairing was conducted using the Tamarin-Prover, a cryptographic protocol verification tool developed by Schmidt, Meier, Cremers, and Basin [76] for a symbolic verification of the NAXOS protocol. Tamarin provides extensive control to the user to tailor his analysis to the specific needs; it is this flexibility that has allowed Tamarin to accomplish symbolic verification of a wide range of protocols [77], [78].

9.1 Protocol Model

We now provide insights into how the Tamarin tool was employed in the verification of Numeric Comparison and Passkey Entry, but it is assumed the reader has a baseline knowledge of Tamarin’s operation and how it leverages the multiset rewriting systems. We give a brief introduction into Tamarin’s working below, but for further reading, we direct the interested party to the Tamarin Prover GitHub page with links to a user manual and teaching materials [19].

Modeling a protocol in Tamarin typically consists of a series of *rules* to model protocol behavior. A rule consists of three parts: *premise*, *action facts*, and *conclusion*. The premise lists out all *facts* that must be consumed for the execution of the rule. Tamarin maintains a listing of all facts, starting from an empty set, in what is termed the *state*. The conclusion represents the facts added to Tamarin’s state at the conclusion of the rule. This allows us to model execution of a protocol as a back and forth between consuming facts/information in the state and adding more facts/information to the state. Action facts are the third piece of a rule. They capture properties and are maintained by Tamarin in the verification of pre-defined properties, termed *lemmas*. Action facts are not stored in state and cannot be consumed in the premise of a rule. Instead, action facts exist within the specific *trace* Tamarin is executing. When assessing a protocol, the modeler defines lemmas based on generalizations of the action facts for Tamarin to either prove false, via a counterexample, or verify. Each attempt by Tamarin to either prove or falsify a lemma is called a trace. The Tamarin heuristic is written such that it will attempt to find falsification traces for the lemma, and once all possible avenues are exhausted then the lemma is deemed verified.

9.1.1 Diffie–Hellman Modeling

One useful aspect of Tamarin is a built-in message theory that enables modeling of Diffie–Hellman groups. Recall that symbolic modeling () is based off the theory of formal language and expressions. Therefore, there is no built-in theoretical basis for mathematical group operations as with computational modeling. Thus, we must explicitly symbols and functions to represent Diffie–Hellman operations in our model.

Tamarin predefines the inverse function and multiplicative group identity, `inv` and `1` respectively, along with symbols to represent the group operations of exponentiation and multiplication, `^` and `*`, respectively. These symbols are related using a natural equations list that captures relevant Diffie–Hellman group identities and properties. Assume x, y, z below are all elements of a Diffie–Hellman group and a, b are integers:

$$\begin{aligned} (x \wedge a) \wedge b &= x \wedge (a * b) , & x * y &= y * x , & x * 1 &= x , \\ x \wedge 1 &= x , & (x * y) * z &= x * (y * z) , & x * \text{inv}(x) &= 1 . \end{aligned}$$

The above equations allow one to model Diffie–Hellman key exchange as actual group operations with associativity and commutativity vice with functions as was required in Scyther and ProVerif. This is necessary for our analysis as the participants in Numeric Comparison and Passkey Entry create a shared Diffie–Hellman Key in Phase 1 that is then employed in Phases 3 and 4.

9.1.2 Restrictions

Tamarin allows for the construction of restrictions for the purpose of disallowing certain traces from being examined during automated analysis. This prevents investigation of unrealistic traces and saves computational time. Unlike lemmas, which Tamarin attempts to falsify or prove, restrictions must always be satisfied within the trace. If at any point in the trace a restriction is broken, the trace is abandoned and that attempt at falsification is deemed incorrect. This analysis utilizes three specific restrictions from the Tamarin-prover manual [79]:

$$\text{restriction Equality: "All } x \ y \ \#i. \text{ Eq}(x,y) \ @i \ ==> \ x = y" ,$$

restriction Inequality: "All $x \#i. \text{Neq}(x,x) @i \implies F$ ",

restriction OnlyOnceV: "All $\#i \#j x. \text{OnlyOnceV}(x) @i \ \& \ \text{OnlyOnceV}(x) @j \implies \#i = \#j$ ".

The above statements can all be read similarly to a statement in formal logic. Equality states that for all strings, x and y , and instance i , if we declare $\text{Eq}(x,y)$ at instance i , then string x equals string y . When $\text{Eq}(x,y)$ is employed as part of a rule in Tamarin, the trace will only continue if string x equals string y . This allows us to model any verification checks and ensure a device will only continue executing its role in the protocol if the check passes.

Inequality states that for all strings, x , and instance i , if we declare $\text{Neq}(x,x)$ at instance i , then we have a contradiction. When $\text{Neq}(x,y)$ is employed as part of a rule in Tamarin, the trace will only continue if string x does not equal string y . The Inequality restriction, allows us to model that the user faithfully executes the protocol into different devices and only communicates with other devices. We make no attempt to model communication between users nor multiple users in the protocol. Since this analysis is performed under the assumption that the user will faithfully execute his role, this restriction is necessary. It also allows us to prevent devices from attempting to pair with themselves, similarly to the CYBORG computational model.

OnlyOnceV states that for all strings, x , and instances i and j , if we declare $\text{OnlyOnceV}(x)$ at instance i and $\text{OnlyOnceV}(x)$ at instance j , then we must have that instance i equals instance j . When OnlyOnceV is employed as part of a rule in Tamarin, the trace will only continue if $\text{OnlyOnceV}(x)$ only appears at a single instance throughout. We use the OnlyOnceV restriction to ensure the user does not reuse passkeys over the course of a protocol run and that a device only associates a single IO Capability value with itself. It is also useful when one needs to force certain rules to only execute once when performing debugging and correctness checks of the encoded protocol.

9.2 Adversarial Model

9.2.1 Device-to-Device Channel

The Tamarin adversary has full control to modify, create, replay, or delete any message sent over the DtD channel in keeping with Definition 5.2.1 from the CYBORG model.

Tamarin uses the built-in facts `In` and `Out` to model communications received and sent over this channel. These are unique facts that are not consumed in a one-to-one correspondence from the state. Instead `Out` is used to add expressions from a rule's conclusion to the state maintained by Tamarin. These expressions can then be employed to fulfill a subsequent rule's `In` fact. The Tamarin adversary has full control to fulfill a rules `In` facts as it pleases in its attempts to falsify a lemma.

9.2.2 User to Device Channel

To model adversarial capabilities over the UtD channel in Tamarin, we adopt the recommendations from the Tamarin manual to create authenticated and secure channels, similar to Definition 5.2.2 and Definition 5.2.3 [79]. Passkey Entry operates under the strict assumption that the passkey shared between devices over the UtD Channel remains secret from the adversary. We capture this fact by writing two unique rules to create a specific UtD channel for use in verification that provides confidentiality (adversary cannot read any messages sent over this channel) and authentication (adversary cannot modify content nor sender/receiver of messages). We then have that the below two rules in conjunction express our definition of the UtD channel (similar to Definition 5.2.2) in Tamarin:

rule ChanOut_UtD: [Out_UtD(\$A,\$B,x)]-[ChanOut_UtD(\$A,\$B,x)]->[!UtD(\$A,\$B,x)] ,

rule ChanIn_UtD: [!UtD(\$A,\$B,x)]-[ChanIn_UtD(\$A,\$B,x)]->[In_UtD(\$A,\$B,x)] .

The rule ChanOut_UtD models sending a message over the UtD channel. When a device or the user wants to send a message over the UtD channel, it first includes a `Out_UtD($A,$B,x)` rule for public device identities, A (the sender) and B (the recipient) and message, x, in its conclusion. This fact is then consumed by the ChanOut_UtD rule to create the *persistent* `!UtD` fact, which is used to bind the public sender/receiver to the specific message. This prevents the Tamarin adversary from modifying the sender/receiver and the content of the communication. When a fact in the conclusion is pre-fixed with an exclamation point, it can be consumed as many times as desired; thus, it is made to be a persistent fact. The persistence allows this message to be replayed, or consumed by multiple times. The qualifier, `$`, is used to signify that this value is publicly known by all and can be freely used in any rule or by the Tamarin adversary. Additionally, since the two rules do not contain an `Out` fact, the Tamarin

adversary has no ability to read the sent message (i.e., no expressions are added to its state, only entire facts). When a device or the user wants to receive a message over the UtD channel, it places a $\text{In_UtD}(\$A, \$B, x)$ fact in its premise. Then so long as a corresponding $\text{Out_UtD}(\$A, \$B, x)$ fact exists in the state, a $\text{In_UtD}(\$A, \$B, x)$ fact can be generated from ChanIn_UtD to satisfy the original rule.

Numeric Comparison does not list a strict requirement for secrecy in its specification; however, we still disallow the adversary from modifying messages sent over the UtD channel. As a result, we slightly modify the ChanOut_UtD rule for use in the analysis of Numeric Comparison with the inclusion of an $\text{Out}(<\$A, \$B, x>)$ fact, detailed below. The left and right arrows are used to signify that it is a single message vice three. This modification allows the Tamarin adversary to read sender/receiver and the message itself, via the contents of the Out fact, for all communications sent over the UtD-E channel.

$$\text{rule ChanOut_UtD: } [\text{Out_UtD}(\$A, \$B, x)] - [\text{ChanOut_UtD}(\$A, \$B, x)] -> \\ [!\text{UtD}(\$A, \$B, x), \text{Out}(<\$A, \$B, x>)] ,$$

$$\text{rule ChanIn_UtD: } [!\text{UtD}(\$A, \$B, x)] - [\text{ChanIn_UtD}(\$A, \$B, x)] -> [\text{In_UtD}(\$A, \$B, x)] .$$

This allows us to model the UtD-E channel (similar to Definition 5.2.3) for analysis of Numeric Comparison.

9.3 Formal Security

This section presents various definitions and goals for authenticated key exchange protocols as Lowe originally formulated them [80] for use in analyzing authentication in a formal setting. We adapt his definitions for our current context and notation for use within the Tamarin formal analysis tool as suggested by the Tamarin-Prover Manual [79]. We differ at points from those instantiated in the Tamarin-Prover Manual in variable names, which are irrelevant. Our definitions also differ because we do not allow the adversary to reveal *any* device secrets (like we do in our computational model with the `StateReveal` and/or `KeyReveal` queries for non-target sessions). This choice was made to simplify analysis and still allows our definitions to match those originally provided by Lowe. Definitions in this section operate under the assumption that all protocol participant identities and exchanged messages are unique strings drawn from the set Σ^* where Σ is the protocol's alphabet. We

use this terminology to be consistent with the Dolev–Yao security model [15], which forms the theoretical basis of our formal analysis into Numeric Comparison and Passkey Entry. All exchanged messages in a protocol are modeled as occurring in a specific sequence and are each assigned a sequence number starting from 0 from the set of non-negative integers, $\mathbb{Z}^+$. This is done so we can apply an ordering to any claims or messages sent within a protocol and performed at a specific point relative to another message sent/received.

lemma aliveness: "All a b t #i. Commit(a, b, t) @i ==>
(Ex role_b #j. Create(b, role_b) @j)",

lemma weak_agreement: "All a b t1 #i. Commit(a, b, t1) @i ==>
(Ex t2 #j. Running(b, a, t2) @j & j < i)",

lemma noninjective_agreement: "All a b t #i.
Commit(a,b,t) @i ==>
(Ex #j. Running(b,a,t) @j
& j < i)",

lemma injective_agreement: "All a b t #i. Commit(a,b,t) @i ==>
(Ex #j. Running(b,a,t) @j & j < i
& not (Ex a2 b2 #i2.
Commit(a2,b2,t) @i2
& not (#i2 = #i)))",

lemma secrecy: "All val #i. Secret(val) @i ==> not (Ex #j. K(val)@j)".

In the above definitions we have that the variable strings a, b, a2, b2 represent the identities of devices executing the protocol, role_a, role_b represent a participant's role in the protocol, t, t1, t2, val are either messages exchanged or computed in conjunction with the execution

of the protocol, and $i, i2, j$ are sequencing numbers. All strings referenced above can take any value and are merely variables; the above constructions are similar to the writing of formulas in formal logic.

The Commit and Running Tamarin action facts are utilized in the same way Lowe originally did. $\text{Commit}(a, b, t)@i$ means that participant a previously received the message t , and a believes, at instance i , that b previously sent message t . Note that instance i is used to mark the point at which a commits to who sent message t , not the instance when a received t . $\text{Running}(a, b, t)@i$ means that participant a believes participant b sent message t at instance i . Although the Running action fact can occur at any point in the protocol, we only employ the Commit action fact once a device has completed its role in the protocol. The Create fact is instantiated at the beginning of a protocol session to signify that a participant sees himself, b , in the role of $\text{role_}b$ within the session and captures the requirement of “previously running the protocol” [80].

Note the strict hierarchy among weak_agreement , $\text{noninjective_agreement}$, and $\text{injective_agreement}$. Specifically, a protocol that achieves $\text{injective_agreement}$ necessarily achieves $\text{noninjective_agreement}$, and a protocol that achieves $\text{noninjective_agreement}$ necessarily achieves weak_agreement . This is to be expected and echoes Lowe’s own analysis [80].

All of the above lemmas, with the exception of secrecy, are translations of Lowe’s properties to Tamarin. The secrecy lemma states that for all strings, val , claimed to be secret, $\text{Secret}(\text{val})$, at instance i , it is not that case that the Tamarin adversary learns val , $K(\text{val})$, at some instance j .

We now address specific strings that we examined with the above lemmas. All strings were examined from the perspective of both the initiating and responding devices.

9.3.1 Numeric Comparison

- DHKey. Upon computation of the Diffie–Hellman key DHKey in a protocol session we used the action fact $\text{Running}(\$A, \$B, \langle \text{'role_}\$B', \text{'role_}\$A', \text{DHKey} \rangle)$ to capture that identity A believes it is running DHKey with B and the specific roles it believes each is performing in the session. Upon completion of Phase 4, we then used the action

fact $\text{Commit}(\$A, \$B, \langle \text{'role_\$A'}, \text{'role_\$B'}, \text{DHKey} \rangle)$ to capture that identity A commits to DHKey with B and the specific roles each performed in the session. Referring back to our above lemmas, say the Commit fact in $\text{noninjective_agreement}$ for example, we see that $\$A$ is a specific instantiation of the variable a , similarly for $\$B$ and b , and similarly for $\langle \text{'role_\$B'}, \text{'role_\$A'}, \text{DHKey} \rangle$ and t . Thus, if Tamarin is able to find a trace where there does not exist a $\text{Running}(b, a, t)$ fact that was declared at an instance before the Commit fact, then Tamarin found a trace that falsified the lemma. Since the entire message of t must match in the lemma, this allows us to investigate both role agreement and agreement of DHKey. We also used the action fact $\text{Secret}(\text{DHKey})$ upon completion of Phase 4 to signify that a device believes the value to be secret at that point. Note that DHKey is qualified with a dollar sign because it is not assumed to be a public value.

- N_a . Upon the generation of identity A's nonce N_a , we used the action fact $\text{Running}(\$A, \$B, \langle \text{'role_\$B'}, \text{'role_\$A'}, N_a \rangle)$ to capture that identity A believes it is running N_a with B and the specific roles it believes each is performing in the session. Upon completion of Phase 4, we then used the action fact $\text{Commit}(\$A, \$B, \langle \text{'role_\$A'}, \text{'role_\$B'}, N_b \rangle)$ to capture that identity A commits to N_b with B and the specific roles each performed in the session.
- LK. Upon completion of Phase 4 we used the action fact $\text{Secret}(\text{LK})$ to signify that a device believes the derived link key LK to be secret at that point.

9.3.2 Passkey Entry

- DHKey. Similar to Numeric Comparison.
- r_a . Upon either generation or receipt of the passkey r_a in a protocol session we used the action fact $\text{Running}(\$A, \$B, \langle \text{'role_\$B'}, \text{'role_\$A'}, r_a \rangle)$ to capture that identity A believes it is running r_a with B and the specific roles it believes each is performing in the session. Upon completion of Phase 4, we then used the action fact $\text{Commit}(\$A, \$B, \langle \text{'role_\$A'}, \text{'role_\$B'}, r_a \rangle)$ to capture that identity A commits to r_a with B and the specific roles each performed in the session. We also used the action fact $\text{Secret}(r_a)$ upon completion of Phase 4 to signify that a device believes the value to be secret at that point.
- $N_{a,i}$. Similar to Numeric Comparison.
- LK. Similar to Numeric Comparison.

9.4 Results of Formal Analysis

9.4.1 Modeling Secure Simple Pairing in Tamarin

Utilizing the Tamarin language multiset writing rules and previously discussed facets in this chapter, we translated Numeric Comparison and Passkey Entry as described in Section 4.3 into Tamarin theory files. We included two specific rules to capture Phase 0 to ensure the devices executing the protocol agree on each other's intended identities and IO capabilities. We modeled the user's identity as the string constant 'User' to capture the assumption that all users are equivalent. In Passkey Entry all 82 communication flows of Phase 2 were modeled in Tamarin; this was accomplished by defining the exchanged random value as a tuple of 20 strings. Although this greatly increased the computational overhead involved in verification, it allowed for a faithful model of the actual protocol in Tamarin. The modeling of all other phases presented no unique challenges in translations not already addressed. Due to length concerns, the full write-up of the Numeric Comparison and Passkey Entry protocols as translated can be found at NPS GitLab: <https://gitlab.nps.edu/michael.troncoso/thesis/-/tree/master/inputs/app/theories>. We present our formalization of Numeric Comparison in the following paragraphs to highlight our modeling choices and to provide an example for reading our models of the Secure Simple Pairing protocols.

Figure 9.1 represents the preamble to our theory file, where we declare the name of the file (SSP_NC_SID) and our defined functions (hmac/2, h/1), which model our MAC and hash function as defined in Section 4.2 respectively. We also define the UtD-E channel in this section as discussed in Section 9.2.

Figure 9.2 depicts our translation of Phase 0. The rule Get_IOcap ensures each device has a only a single IO capability variable associated with it. The persistent out fact !IOcap(\$A, \$IOcap), associates a public device identity, \$A, with a public IO capability, \$IOcapA. This acts a device registration and prevents the Tamarin adversary from assigning multiple, unique IO capability variables to a single device when proving lemmas. SSP_A_IO and SSP_B_IO then model the rest of Phase 0. In SSP_A_IO, we first require the device to declare its IO capability variable, and then send this value to a second device \$B with the out fact Out(<\$A, \$B, IOcapA>). Out facts will appear regularly throughout our formalization of Numeric Comparison and will consistently be used to model sending a message over the DtD channel. The Create action facts associate roles with each device and the Inequality

Figure 9.1. Preamble for Translation of Numeric Comparison to Tamarin.

```

theory SSP_NC_SID
begin

functions: hmac/2
functions: h/1

builtins: diffie-hellman

/* UtD Channel rules:
   Adversary cannot modify a message or its sender/receiver
   on the User to Device channel.
   Adversary can read/delete/replay messages sent on this channel.
*/
rule ChanOut_UtD:
  [   Out_UtD( $A, $B, x )
  ]
  --[ ChanOut_UtD( $A, $B, x )
  ]->
  [   !UtD( $A, $B, x )
    ,   Out( <$A, $B, x> )
  ]

rule ChanIn_UtD:
  [   !UtD( $A, $B, x )
  ]
  --[ ChanIn_UtD( $A, $B, x )
  ]->
  [   In_UtD( $A, $B, x )
  ]

```

action facts ensure the devices are not the User. The out fact $\text{State_A_IO}(\langle \$A, \text{IOcapA} \rangle, \$B)$ models the information stored in device $\$A$'s memory. Facts starting with “State” will appear regularly throughout our formalization of Numeric Comparison and will consistently be used to track device memory. SSP_B_IO then proceeds similarly with the addition that it receives A 's message from the previous rule, modeled with $\text{In}(\langle A, B, \text{IOcapA} \rangle)$ and we require A and B to be different devices ($\text{Neq}(A, B)$) in keeping with the CYBORG model. In facts will appear regularly throughout our formalization of Numeric Comparison and will consistently be used to model receiving a message over the DtD channel. Note that “//” represent comment lines applied to lines giving some description or lines useful in debugging only.

Figure 9.2. Numeric Comparison, Phase 0.

```
//Force a device to only have one IOcap
rule Get_IOcap:
  []
  --[ OnlyOnceV( $A )
  ]->
  [   !IOcap( $A, $IOcap )
  ]

rule SSP_A_IO:
  [   !IOcap( $A, IOcapA )
  ]
  --[ Create( $A, 'Initiating' )
    ,   Create( $B, 'NonInitiating' )
  //   ,   OnlyOnceV( 'Initiating' )
    ,   Neq( $A, 'User' )
    ,   Neq( $B, 'User' )
  ]->
  [   Out( <$A, $B, IOcapA> )
    ,   State_A_IO( <$A, IOcapA>, $B )
  ]

rule SSP_B_IO:
  [   In( <A, B, IOcapA> )
    ,   !IOcap( B, IOcapB )
  ]
  --[ Recv( B, A, IOcapA )
    ,   Send( B, A, IOcapB )
  //   ,   OnlyOnceV( 'NonInitiating' )
    ,   Neq( A, B )
  ]->
  [   Out( <B, A, IOcapB> )
    ,   State_B_IO( <A, IOcapA>, <B, IOcapB> )
  ]
```

Figure 9.3 depicts steps 1a, 2a, and 3 from Phase 1 for Numeric Comparison. The device uses the fact $\text{Fr}(\sim\text{SKa})$ to draw a fresh secret key, signified by the $\sim$ qualifier, for use in the subsequent Diffie–Hellman exchange. The $\sim$ symbol in front of a string stands for freshness, and means that said string is ephemeral to the trace that Tamarin is currently executing. We use the let-in construction to equate the string PKa with the string $'g' \wedge \sim\text{SKa}$. As shown, device A receives device B’s IO capability variable, IOcapB , sends out its public key, PKa ,

Figure 9.3. Numeric Comparison, Phase 1.

```
// A --PKa-> B
rule SSP_NC_A_0:
  let
    PKa = 'g'^~SKa
  in
    [ Fr( ~SKa )
    , In( <B, A, IOcapB> )
    , State_A_IO( <A, IOcapA>, B )
    ]
  --[ Recv( A, B, IOcapB )
  , Send( A, B, PKa )
  ]->
  [ Out( <A, B, PKa> )
  , State_A_0( <A, IOcapA>, <B, IOcapB>, PKa, ~SKa )
  ]
```

to B, and updates its memory. The action facts Recv and Send do not play a role in analysis but are useful for debugging.

Figure 9.4 depicts steps 1b, 2b, 3, 4, 5b, 6b, 7b, 8, and 9 from Phase 1 and the beginning part of Phase 2 for Numeric Comparison. Device B receives device A's public key, and responds with its own similarly to the previous rule. In addition, device B computes its Diffie-Hellman key, DHKey, as $PKa \wedge \sim SKb$. Since device B is not currently waiting on any information from device A and has all the information it requires to proceed to Phase 2, it proceeds with the computation of its commitment value Cb. B draws a fresh nonce, $\sim Nb$, and sets its passkeys, rb, ra, to the constant string '0'. It then uses the declared function hmac to compute Cb using $\sim Nb$ as the key and $\langle PKb, PKa, rb \rangle$ as the message. Device B concludes this rule by sending out its own public key, its commitment value, and updating its memory. This rule also contains the first instance of the Running action fact as discussed in Section 9.3 because we want to verify authentication of both device B's nonce and Diffie-Hellman key.

Rule SSP_NC_A_1 in Figure 9.5 proceeds similarly to rule SSP_NC_B_0 from Figure 9.4 in its modeling of steps 4, 5a, 6a, 7a, 9, and 10. Device A receives device B's public key, PKb, and commitment value, Cb, computes its Diffie-Hellman key, $DHKey = PKb \wedge \sim SKa$, and responds with its own fresh nonce $\sim Na$. Rule SSP_NC_B_1 models steps 10,

Figure 9.4. Numeric Comparison, Phase 1 (cont.) and Phase 2.

```

// B <--PKa-- A
// B: DHKey <- PKa ^ SKb
// B --PKb--> A
// B: Cb <- hmac( ~Nb, <PKb, PKa, rb> )
// B --Cb--> A
rule SSP_NC_B_0:
  let
    PKb = 'g'^~SKb
    DHKey = PKa ^ ~SKb
    rb = '0'
    ra = rb
    Cb = hmac( ~Nb, <PKb, PKa, rb> )
  in
  [ Fr( ~SKb )
  , Fr( ~Nb )
  , In( <A, B, PKa> )
  , State_B_IO( <A, IOcapA>, <B, IOcapB> )
  ]
  --[ Recv( B, A, PKa )
  , Send( B, A, PKb )
  , Running( B, A, <'Initiating', 'NonInitiating', DHKey> )
  , Running( B, A, <'Initiating', 'NonInitiating', ~Nb> )
  ]->
  [ Out( <B, A, PKb> )
  , Out( <B, A, Cb> )
  , State_B_0( <A, IOcapA>, <B, IOcapB>, PKa, PKb, DHKey, ~Nb, ra, rb )
  ]

```

11, 13b, and 14b. Device B receives device A’s nonce, Na, and uses this value to compute its verification value, $Vb = h(\langle PKa, PKb, Na, \sim Nb \rangle)$. Device B then sends Vb to the user using the out fact $Out_UtD(B, 'User', \langle Vb, \sim Nb \rangle)$.

Though not referred to in the Bluetooth specification [11], we include $\sim Nb$ in this message as simulated metadata to ensure the Tamarin adversary does not attempt cross-session replay attacks involving UtD messages, which are disallowed in our CYBORG model. Since all nonces in Numeric Comparison are guaranteed to be ephemeral, a device can use it as a session identifier to ensure any received “OK” message from the user is applicable to the currently executing session. This is a calculated modeling choice to prevent Tamarin from pursuing un-realistic traces that would lead to falsification of lemmas. To minimize the

Figure 9.5. Numeric Comparison, Phase 2 (cont.).

```

// A <--PKb-- B
// A: DHKey <- PKb ^ SKa
// A <--Cb-- B
// A --Na--> B
rule SSP_NC_A_1:
  let
    DHKey = PKb^~SKa
    ra = '0'
    rb = ra
  in
  [ State_A_0( <A, IOcapA>, <B, IOcapB>, PKa, ~SKa )
  , In( <B, A, PKb> )
  , In( <B, A, Cb> )
  , Fr( ~Na )
  ]
  --[ Recv( A, B, PKb )
  , Recv( A, B, Cb )
  , Send( A, B, ~Na )
  , Running( A, B, <'NonInitiating', 'Initiating', DHKey> )
  , Running( A, B, <'NonInitiating', 'Initiating', ~Na> )
  ]->
  [ Out( <A, B, ~Na> )
  , State_A_1( <A, IOcapA>, <B, IOcapB>, PKa, PKb, DHKey, ra, rb, Cb, ~Na )
  ]

// B <--Na-- A
// B --Nb--> A
rule SSP_NC_B_1:
  let
    Vb = h(<PKa, PKb, Na, ~Nb>)
  in
  [ In( <A, B, Na> )
  , State_B_0( <A, IOcapA>, <B, IOcapB>, PKa, PKb, DHKey, ~Nb, ra, rb )
  ]
  --[ Recv( B, A, Na )
  , Send( B, A, ~Nb )
  , Send_UtD(B, 'User', <Vb, ~Nb>)
  ]->
  [ Out( <B, A, ~Nb> )
  , Out_UtD( B, 'User', <Vb, ~Nb> )
  , State_B_1( <A, IOcapA>, <B, IOcapB>, DHKey, PKa, PKb, ~Nb, ra, rb, Na )
  ]

```

Figure 9.6. Numeric Comparison, Phase 2 (cont.) and Phase 3.

```
// A <--Nb-- B
// A: Cb == hmac(Nb, <PKb, PKa, ra>)
// A --OK--> B
// A --Va--> User
rule SSP_NC_A_2:
  let
    Va = h(<PKa, PKb, ~Na, Nb>)
  in
  [ In( <B, A, Nb> )
  , State_A_1( <A, IOcapA>, <B, IOcapB>, PKa, PKb, DHKey, ra, rb, Cb, ~Na )
  ]
  --[ Recv( A, B, Nb )
  , Send_UtD( A, 'User', <Va, ~Na> )
  , Eq(Cb, hmac(Nb, <PKb, PKa, ra>) )
  ]->
  [ Out_UtD( A, 'User', <Va, ~Na> )
  , State_A_2( <A, IOcapA>, <B, IOcapB>, PKa, PKb, DHKey, ra, rb, ~Na, Nb )
  ]

// User <--Va-- A
// User <--Vb-- B
// User: Va == Vb
// User --OK--> A
// User --OK--> B
rule SSP_NC_User:
  [ In_UtD( A, 'User', <Va, sidA> )
  , In_UtD( B, 'User', <Vb, sidB> )
  ]
  --[ Recv_UtD( 'User', A, <Va, sidA> )
  , Recv_UtD( 'User', B, <Vb, sidB> )
  , Neq( A, B )
  , Neq( A, 'User')
  , Neq( B, 'User')
  , Eq( Va, Vb )
  , Send_UtD( 'User', A, <'OK', sidA> )
  , Send_UtD( 'User', B, <'OK', sidB> )
  // , OnlyOnceV('User')
  ]->
  [ Out_UtD( 'User', A, <'OK', sidA> )
  , Out_UtD( 'User', B, <'OK', sidB> )
  ]
```

chance that other attacks were also prevented, we ensured the user did not interact with the ephemeral nonce. As shown in Figure 9.6, the nonces $\sim Na$ and $\sim Nb$, symbolized by

the variable strings `sidA` and `sidB` respectively, are merely forwarded through with their associated identity after the user verifies `Va` and `Vb` match. Nevertheless, we cannot say with certainty that this choice did not also prevent other attacks aside from the UtD cross-session replay attack. However, after much trial and error, including the nonces in the UtD messages proved to be the most sensible method of preventing erroneous falsification and allowing analysis to continue with minimal detriment.

Rule `SSP_NC_A_2` follows similarly to rule `SSP_NC_B_1` and models steps 12, 13a, and 14a. Note that we use the action fact `Eq(Cb, hmac(Nb, <PKb, PKa, ra>))` to model device A's computation in step 12. Rule `SSP_NC_User` models the user's role in Numeric Comparison. He receives both of the verification values, `Va`, `Vb`, compares them for equality, `Eq(Va, Vb)`, and inputs 'OK' into both devices. Note that the user performs no actions involving `sidA`, `sidB`, the metadata strings device A and B's nonces. Thus, our modeling choice of including the nonces does not fundamentally alter the user's role in the protocol. The action facts `Recv_UtD` and `Send_UtD` do not play a role in analysis but are useful for debugging.

Figure 9.7. Numeric Comparison, Phase 3 (cont.).

```
// A <--OK-- User
// A: Ea <- hmac(DHKey, <~Na, Nb, 0, $IOcapA, $A, $B>)
// A --Ea--> B
rule SSP_NC_A_3:
  let
    Ea = hmac( DHKey, <~Na, Nb, rb, IOcapA, A, B> )
  in
  [ In_UtD('User', A, <'OK', sidA> )
    , State_A_2( <A, IOcapA>, <B, IOcapB>, PKa, PKb, DHKey, ra, rb, ~Na, Nb )
  ]
  --[ Recv_UtD( A, 'User', <'OK', sidA> )
    , Eq( sidA, ~Na )
    , Send( A, B, Ea )
  ]->
  [ Out( <A, B, Ea> )
    , State_A_3( <A, IOcapA>, <B, IOcapB>, PKa, PKb, DHKey, ra, rb, ~Na, Nb )
  ]
```

With Figure 9.7 and rule `SSP_NC_A_3`, we model steps 15a, 16a, and 17. Device A receives the “OK” message from the user, computes its check value, `Ea`, and sends `Ea` to

device B. Also observe that device A performs the check $\text{Eq}(\text{sidA}, \sim\text{Na})$. This ensures that the “OK” received from the user applies to the correct session. Since the device has confirmation that the “OK” message applies to the session it expects, we can rule out cross-session attacks over the UtD-E channel with minimal change to the protocol specification.

Figure 9.8. Numeric Comparison, Phase 3 (cont.) and Phase 4.

```
// B <--OK-- User
// B: Eb <- hmac(DHKey, <~Nb, Na, 0, $IOcapB, $B, $A>)
// B <--Ea-- A
// B: Ea == hmac(DHKey, <~Na, Nb, 0, $IOcapA, $A, $B>)
// B --Eb--> A
// B: LK <- hmac(DHKey, <Na, Nb, 'btlk', $A, $B>)
rule SSP_NC_B_2:
  let
    Eb = hmac( DHKey, <~Nb, Na, ra, IOcapB, B, A> )
    LK = hmac( DHKey, <Na, ~Nb, 'btlk', A, B> )
  in
  [ In_UtD('User', B, <'OK', sidB>)
  , In( <A, B, Ea> )
  , State_B_1( <A, IOcapA>, <B, IOcapB>, DHKey, PKa, PKb, ~Nb, ra, rb, Na )
  ]
  --[ Recv_UtD( B, 'User', <'OK', sidB> )
  , Eq( sidB, ~Nb )
  , Recv( B, A, Ea )
  , Send( B, A, Eb )
  , Eq( Ea, hmac( DHKey, <Na, ~Nb, rb, IOcapA, A, B> ) )
  , Secret( LK )
  , Secret ( DHKey )
  , Commit( B, A, <'NonInitiating', 'Initiating', DHKey> )
  , Commit( B, A, <'NonInitiating', 'Initiating', Na> )
  , SessionKey( B, A, LK )
  ]->
  [ Out( <B, A, Eb> )
  ]
```

Rule `SSP_NC_B_2` is the concluding rule for device B and models steps 15b, 16b, 17, 18, 19, and 21b. In this rule, device B executes similar actions to those performed by device A in rule `SSP_NC_A_3`. Additionally, device B verifies the received check value, $\text{Eq}(\text{Ea}, \text{hmac}(\text{DHKey}, \langle \text{Na}, \sim\text{Nb}, \text{rb}, \text{IOcapA}, \text{A}, \text{B} \rangle))$. If this check passes, device B sends its own check value Eb, commits to the Diffie-Hellman key, DHKey, and commits to device A’s nonce, Na. Device B also computes the link key, LK, and declares this value and DHKey

to be secret.

Figure 9.9. Numeric Comparison, Phase 4 (cont.).

```
// A <--Eb-- B
// A: Eb == hmac(DHKey, <~Na, Nb, 0, $IOcapA, $A, $B>)
// A --OK--> B
// A: LK <- hmac(DHKey, <~Na, Nb, 0, 'btlk', $A, $B>)
rule SSP_NC_A_4:
  let
    LK = hmac( DHKey, <~Na, Nb, 'btlk', A, B> )
  in
  [ In( <B, A, Eb> )
  , State_A_3( <A, IOcapA>, <B, IOcapB>, PKa, PKb, DHKey, ra, rb, ~Na, Nb )
  ]
  --[ Recv( A, B, Eb )
  , Secret( LK )
  , Secret( DHKey )
  , Commit( A, B, <'Initiating', 'NonInitiating', DHKey> )
  , Commit( A, B, <'Initiating', 'NonInitiating', Nb> )
  , Eq(Eb, hmac( DHKey, <Nb, ~Na, ra, IOcapB, B, A> ) )
  , SessionKey( A, B, LK )
  ]->
  []
```

Rule SSP_NC_A_4 is final rule in our formalization of Numeric Comparison and models steps 19, 20, and 21. Device A’s steps in this rule are similar to the phase 4 steps of device B in rule SSP_NC_B_2.

Though not displayed above, to assist with the verification of the various protocols in this section, we wrote three *oracle* programs to guide the Tamarin heuristic in its analysis as discussed in the Tamarin-Prover manual [79]. These programs are termed oracles because they act as a black box for the Tamarin heuristic to call when proving lemmas and provide direction for its choices. All three oracle programs can be found at NPS GitLab due to length considerations: <https://gitlab.nps.edu/michael.troncoso/thesis/-/tree/master/inputs/app/oracles>. The python program “ssp-nc-oracle.py” was employed in the analysis of the Numeric Comparison protocol, the python program “ssp-pe-oracle.py” was employed in the analysis of the Initiator/Responder-Generated Passkey Entry protocols, and the python program “ssp-pe-ug-oracle.py” was employed in the analysis of the User-Generated Passkey Entry protocol.

To assist with automatic proving, the noninjective agreement lemma was modified slightly

to the following for use in all Passkey Entry protocols.

```
lemma noninjective_agreement: "All a b role_a role_b val #i.
    Commit(a,b,< role_a, role_b, val >) @i ==>
    (Ex #j. Running(b,a,< role_a, role_b, val >) @j
    & j < i)",
```

Attempts to write oracles for the Passkey Entry protocols without this modification proved unsuccessful. The modification being that we instantiated the commitment string variable, t , from the original definition described in Section 9.3, as the three-variable string, $\langle \text{role_a}, \text{role_b}, \text{val} \rangle$, for use in these protocols. This change forces Tamarin to instantiate val in the action fact, which allowed us to direct our oracles based on val and provided a greater degree of control. The choice to instantiate t as $\langle \text{role_a}, \text{role_b}, \text{val} \rangle$ stems from the fact that all Commit messages in our Passkey Entry protocol theory files are of this general form. Thus, t is guaranteed to be of the form $\langle \text{role_a}, \text{role_b}, \text{val} \rangle$ in every Commit action fact Tamarin examines. Therefore, this change does not alter the correctness of our analysis and is still in keeping with Lowe’s definitions.

We now walk through the Numeric Comparison oracle program as a example for the reader. The oracle is essentially a python program that tells the Tamarin heuristic which fact to attempt to satisfy in attempts to falsify a lemma. We read all facts that need to be satisfied in the trace via the call `lines = sys.stdin.readlines()`. We then iterate through the lines, identified by the preceding number `num`, and proceed to rank them, from **11** through **18**, based on their contents. We determined the rank of a line based on the probability, arbitrarily quantified through trial and error, that the line will lead to a contradiction.

Take for instance the rule classifying the **11** rank: “**~SK**” in line and “**In_UtD**” in line. This basically captures an instance where the Tamarin adversary is attempting to send a Diffie–Hellman exponent over the UtD channel, which is something that should never occur in Numeric Comparison based off the specification. Since we suspect this to lead to a contradiction quickly, we direct the Tamarin heuristic to attempt to satisfy this fact. The **16** ranking is similar in this facet because it is a line where a Diffie–Hellman exponent is involved in a group operation outside of the MAC. The next three ranks, **12** through **14**, all involve the Tamarin adversary attempting to gain knowledge of something by trivial

Figure 9.10. Oracle for Analysis of Numeric Comparison.

```
#!/usr/bin/python
import sys

lines = sys.stdin.readlines()
l1 = []
l2 = []
l3 = []
l4 = []
l5 = []
l6 = []
l7 = []
l8 = []

for line in lines:
    num = line.split(':')[0]

    if "~SK" in line and "In_UtD" in line:
        l1.append(num)
    elif "KU( ~N" in line and "^" in line:
        l2.append(num)
    elif "KU( ~SK" in line:
        l3.append(num)
    elif "KU( ~r" in line:
        l4.append(num)
    elif "In_UtD" in line:
        l5.append(num)
    elif "~SK" in line and "*" in line and "hmac" not in line:
        l6.append(num)
    elif "State" in line:
        l7.append(num)
    else:
        l8.append(num)

ranked = l1 + l2 + l3 + l4 + l5 + l6 + l7 + l8
for i in ranked:
    print(i)
```

means: a nonce involved in a Diffie–Hellman exponentiation, a Diffie–Hellman exponent itself, or a passkey. The **l5** and **l7** lines tells Tamarin to try to satisfy UtD channel facts and device memory requirements first, which typically trigger higher ranked facts in the oracle in subsequent oracle calls. The final rank, **l8**, is a default rank when no triggers are present and directs Tamarin to follow its default programming.

All numbers are then combined into a singular list and returned to Tamarin for action by printing to **stdout**. Tamarin will then act on the first fact in the returned list and attempt to satisfy it. The inclusion of oracles for the Secure Simple Pairing protocols, like the one discussed above, allowed for analysis to proceed in a more timely and efficient manner.

9.4.2 Numeric Comparison

The initial summary of results of the analysis of Numeric Comparison with the Tamarin Prover are depicted in Figure 9.11.

```
=====
summary of summaries:

analyzed: ssp-nc.spthy

secrecy (all-traces): falsified - found trace (16 steps)
aliveness (all-traces): falsified - found trace (16 steps)
weak_agreement (all-traces): falsified - found trace (16 steps)
noninjective_agreement (all-traces): falsified - found trace (16 steps)
injectiveagreement (all-traces): falsified - found trace (16 steps)
=====
```

Figure 9.11. Initial Tamarin prover output depicting falsification of all analyzed lemmas for Numeric Comparison.

```
=====
summary of summaries:

analyzed: ssp-nc-sid.spthy

secrecy (all-traces): verified (100 steps)
aliveness (all-traces): verified (74 steps)
weak_agreement (all-traces): verified (120 steps)
noninjective_agreement (all-traces): verified (120 steps)
injectiveagreement (all-traces): verified (1376 steps)
=====
```

Figure 9.12. Subsequent Tamarin prover output depicting verification of all security properties after removing the adversary's ability to mount cross-session attacks.

Under our security definitions, Tamarin found falsification traces for secrecy, aliveness, weak agreement, noninjective agreement, and injective agreement. The steps listed above in Figs. 9.11 and 9.12 count the number of facts Tamarin had to satisfy before either a falsification trace was found or all possible traces were deemed to lead to contradictions. Further examination of the protocol in Tamarin’s interactive mode revealed that the protocol is susceptible to a cross-session replay attack whereby the adversary replays the “OK” message from a correctly executed session with an incorrectly executed session. This attack is similar to the one discovered by the ProVerif analysis in [9]. Ours differs in that it includes all steps of the protocol. These attacks are able to succeed because there exists no session specific information in the user’s “OK” message to each device. This causes the devices to be unable to associate “OK” messages with specific sessions. To add relevance to this attack, one possible execution method would be if an adversary forces a device to replay the same comparison value to be verified and incites multiple “OK” messages from the user. Another could be a compromised display that replays a user’s inputs to the device unbeknownst to the user. Ultimately, since the device is unable to associate each “OK” message with a specific session, it mistakenly assumes a different session has been verified and continues pairing with the adversary.

Nonetheless, the power to replay UtD messages across sessions is denied within the CYBORG model as described in Section 5.2. To prevent the Tamarin adversary from pursuing this line of attack, a device’s session specific nonce is also included in messages to/from the user as was discussed previously in our walk-through of the Numeric Comparison formalization. Since the nonce is fresh for each session, it acts as a pre-built session identifier that can aid in deciphering what session the user has verified. Upon this fix, Tamarin verified aliveness, weak agreement, non-injective agreement, injective agreement, and secrecy as shown in Figure 9.12.

9.4.3 Initiator-Generated Passkey Entry

The summary of results of the analysis of Initiator-Generated Passkey Entry with the Tamarin Prover are depicted in Figure 9.13. Tamarin verified all of the security lemmas with injective agreement taking the largest time investment with 11,127 steps. In comparison to the verification of Numeric Comparison, which took 1376 steps to verify injective agreement, this is noticeably longer. The difference in step count is because of the difference in code

length. Since all 20 bits of the passkey have to be verified sequentially per specification, Passkey Entry requires 87 message exchanges while Numeric Comparison only requires 12. Thus, Passkey Entry affords 55 more opportunities for the Tamarin adversary to affect communications, which greatly increases verification time. These results highlight the

```
=====
summary of summaries:
```

```
analyzed: ssp-pe-ig.spthy
```

```

secrecy (all-traces): analysis verified (66 steps)
aliveness (all-traces): analysis verified (266 steps)
weak_agreement (all-traces): analysis verified (266 steps)
noninjective_agreement (all-traces): verified (582 steps)
injectiveagreement (all-traces): verified (11127 steps)
=====
```

Figure 9.13. Tamarin prover output depicting verification of all security properties for Initiator-Generated Passkey Entry.

difference in abstraction between the CYBORG computational model and the Tamarin formal model. Mainly that, each bit of the passkey is labeled fresh and the Tamarin adversary has no capacity to guess its value. Thus, the formal analysis arrives at the conclusion that the nonces cannot be attacked, while the computational analysis arrives at the guessing attack discussed in Theorem 6.3.1. Another difference is that Tamarin’s analysis is of a lemma is a binary: either the lemma is verified or falsified in some number of steps. Meanwhile, in computational analysis we receive a probability of security and must assess whether the probability is negligible enough to claim security.

9.4.4 Responder-Generated Passkey Entry

The summary of results of the analysis of Responder-Generated Passkey Entry with the Tamarin Prover are depicted in Figure 9.14 and showcase similar results in comparison to Initiator-Generated Passkey Entry.

```

=====
summary of summaries:

analyzed: ssp-pe-ng.spthy

output:          ssp-pe-ng-sec.txt
processing time: 432.8962563s
secrecy (all-traces): verified (73 steps)
aliveness (all-traces): verified (156 steps)
weak_agreement (all-traces): verified (266 steps)
noninjective_agreement (all-traces): verified (494 steps)
injectiveagreement (all-traces): verified (8034 steps)
=====

```

Figure 9.14. Tamarin prover output depicting verification of all security properties for Responder-Generated Passkey Entry.

9.4.5 User-Generated Passkey Entry

The summary of results of the analysis of User-Generated Passkey Entry with the Tamarin Prover are depicted in Figure 9.15.

```

=====
summary of summaries:

analyzed: ssp-pe-ug.spthy

output:          ssp-pe-ug-sec.txt
processing time: 19489.4259367s
secrecy (all-traces): verified (696 steps)
aliveness (all-traces): verified (704 steps)
weak_agreement (all-traces): verified (704 steps)
noninjective_agreement (all-traces): falsified (90 steps)
=====

```

Figure 9.15. Tamarin prover output depicting falsification of noninjective agreement and injective agreement for User-Generated Passkey Entry. Since noninjective agreement was falsified, we did not devote further analysis to injective agreement.

Under our security definitions, Tamarin verified that User-Generated Passkey Entry satisfied

aliveness, weak agreement, and secrecy on the data elements defined under Section 9.3. However, Tamarin found falsification traces for both noninjective agreement and injective agreement. Further examination of the protocol in Tamarin's interactive mode revealed that the Tamarin adversary found the same attack described in Theorem 6.3.3 and depicted in Figure 6.3 to break noninjective agreement. Specifically, Tamarin found a trace where one device committed to its role, its partner's role and a message, but no other device had ever been running this same message with the first device and in the believed role. This situation occurred because although the two devices agreed on the content of the message, they did not agree on their respective roles in the protocol.

The efficacy of this attack was previously discussed following its presentation in Theorem 6.3.3. Though the attack is certainly executable in theory, in practice it would require two devices to attempt to pair with other simultaneously and both in the role of the initiator, which is an improbable occurrence. The Secure Hash Modification proposed in Chapter 7 was developed to rectify this issue, but was not analyzed in Tamarin and is left for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 10:

Conclusions

10.1 Computational Analysis

Table 10.1. Table depicting levels of CYBORG security in the uncompromised (UncUser) and compromised user (CompUser_x) settings achieved by the Numeric Comparison and Passkey Entry protocols under weak and strong session modeling, as well as Secure Hash Modification (SHM) Passkey Entry and SHM Dual Passkey Entry. SHM Dual Passkey Entry is provably secure under all variants of the CYBORG security model. If a variant of CompUser_x is not depicted, it is because only SHM Dual Passkey Entry achieved security under said variant. ✓ depicts proven secure in this work, ✓* depicts provably secure by implication, and ✗ depicts insecure.

<i>Bluetooth BR/EDR Secure Simple Pairing/Secure Connections v2.1–5.2</i>													
Num. Comp.		Init-Gen'd				Resp-Gen'd				User-Gen'd			Dual
Original		Original		SHM		Original		SHM		Original		SHM	SHM
Weak	Strong	Weak	Strong	Strong	Weak	Strong	Strong	Weak	Strong	Weak	Strong	Strong	Strong
CYBORG Security Framework	UncUser	✓	✗	✓	✗	✓	✓	✗	✓	✗	✗	✓	✓*
	CompUser _a	✗	✗	✗	✗	✗	✓*	✗	✓*	✗	✗	✓*	✓*
	CompUser _b	✗	✗	✓*	✗	✓*	✗	✗	✗	✗	✗	✓*	✓*
	CompUser _c	✗	✗	✓*	✗	✓*	✗	✗	✗	✗	✗	✗	✓*
	CompUser _d	✗	✗	✗	✗	✗	✓*	✗	✓*	✗	✗	✗	✓*
	CompUser _{ab}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓*	✓*
	CompUser _{ad}	✗	✗	✗	✗	✗	✓*	✗	✓*	✗	✗	✗	✓*
	CompUser _{bc}	✗	✗	✓*	✗	✓*	✗	✗	✗	✗	✗	✗	✓*
	CompUser _{ac}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓*
	CompUser _{bd}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓*
	CompUser _{abcd}	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Comparison of Results The results of our computational analysis are summarized below in Table 10.1. Numeric Comparison and Passkey Entry achieved middling results in the CYBORG model, with only Numeric Comparison achieving a semblance of security under strong session modeling. SHM Passkey Entry achieved similar security across all versions, with variations only in the CompUser_x setting. The various results give insight into the type of attacks Numeric Comparison and Passkey Entry in their current construction can defend against. In regard to Numeric Comparison, gaining control of any one of a device's display or input capabilities will lead to a break in the protocol. The examination of Passkey

Entry (with a weak session identifier) and SHM Passkey Entry herein reveals that gaining control of the device display used to generate the passkey, or the device input of the passkey receiver, also allows for breaks in the protocol. Furthermore, none of the current Bluetooth protocols are secure against Tap ‘n Ghost-like attacks (CompUser_{ac} and CompUser_{bd}).

The UtD weaknesses present in Passkey Entry point to the motivation behind Dual Passkey Entry, namely to have both devices generate a passkey. This function ensures some measure of secrecy, which the adversary can neither affect nor learn, is maintained throughout execution of Dual Passkey Entry. Although the CompUser_{abcd} environment allows the adversary to change all passkeys being transferred among Passkey Entry participants, the absence of eavesdropping in the UtD channel (as required for execution of Passkey Entry per [11]) prevents the adversary from learning the original passkeys before the conclusion of Phase 2. Furthermore, since both devices generate passkeys, the adversary cannot leverage its corruption queries to gain knowledge of the target session’s passkey. With the integrity of at least one of the passkeys ensured by the user transfer and native generation, devices can successfully authenticate the DH key exchange. We therefore show that under this strong guarantee, whereby the user functions as a medium for the secret transfer of device communications, one can create protocols achieving a greater degree of security than current methods with minimal increases in user involvement. As proven in Theorem 8.4.1, Dual Passkey Entry with the Secure Hash Modification maintains CYBORG security in spite of the adversary having the full capability to modify UtD messages and without user-generated random numbers.

Future Work We note that the security of Dual Passkey Entry with the Secure Hash Modification is reliant upon the assumption that it is executed in the UtD without eavesdropping environment, as required by Bluetooth specification for all variants of Passkey Entry [11]. This lends itself to an investigation into the possible security guarantees a protocol can achieve in the CYBORG model under an adversarial setting that allows for eavesdropping on the UtD channel. Past work on Signal [18] presented Modified Device-to-User Signal Authentication (MoDUSA) as a solution to this problem and proved it secure under an equivalent instantiation of the CompUser_a and CompUser_b environments. Although Signal authentication in its current construction did not demonstrate this, MoDUSA achieved it by relying on ratcheted keys to compute epoch-specific QR and numeric comparison codes for user verification. This highlights that the techniques for the future advancement of cyborg

protocols are currently in existence and further investigation is warranted. One protocol of note that could be the subject of such an investigation would be Numeric Comparison. Although this thesis did not rectify the security issues present in Numeric Comparison, the door is open for novel proposals and proofs. The cyborg protocols standardized under ISO/IEC 9798-6 also present a possibility for future investigation as security analysis of this protocol family has been limited [17], [81].

10.2 Formal Analysis

Table 10.2. Table depicting formal security lemmas satisfied by the Numeric Comparison and Passkey Entry protocols. ✓ depicts verified by Tamarin-Prover and ✗ depicts falsified.

		<i>Bluetooth BR/EDR Secure Simple Pairing/Secure Connections v2.1–5.2</i>			
		Numeric Comp.	Initiator-Gen'd	Responder-Gen'd	User-Gen'd
Formal Lemma	Secrecy	✓	✓	✓	✓
	Aliveness	✓	✓	✓	✓
	Weak Agree.	✓	✓	✓	✓
	Noninject. Agree	✓	✓	✓	✗
	Inject. Agree	✓	✓	✓	✗

Comparison of Results The entirety of the formal analysis depicted in Table 10.2 is only comparable to computational analysis in the CYBORG-UncUser setting, as shown in row one of Table 10.1, under weak session modeling. The combination of secrecy and injective agreement analyzed in Tamarin is similar to the key-ind and auth requirements under the CYBORG-UncUser security experiment. Aliveness, weak agreement, and noninjective agreement investigate weaker authentication guarantees not investigated computationally. Our Tamarin analysis is incomparable to our computational analysis under strong session modeling because we only analyzed specific values (nonces, passkeys, and DH keys) within our formal lemmas, vice every message exchanged. Under these narrow parameters, all examined protocols achieved similar security levels in both the symbolic and computational analysis. Numeric Comparison and Initiator/Responder-Generated Passkey Entry's Tamarin results were in keeping with those proved computationally in Chapter 6. User-Generated Passkey Entry meanwhile, was still found to be susceptible to the role confusion attack depicted in Figure 6.3.

Future Work For our formal analysis, we did not give the adversary the capability to reveal device secrets or compromise the UtD channel as we did in the computational setting. Including these capabilities in a future analysis is possible by re-working the formal security lemmas as described in [79]. However, the biggest detriment to including additional adversarial capabilities would be computing power and time. The current analysis was conducted using a 2018 Microsoft Surface Book 2, with an eighth generation Intel Quad-Core i7 processor and 16GB of RAM [82]. Assuming that allotting the adversary improved capabilities to affect communications will increase lemma verification/falsification time for Tamarin, we recommend a dedicated server with greatly improved processing power for an extended analysis.

10.3 Concluding Remarks

We have provided a comprehensive cryptographic investigation of cyborg protocols in Bluetooth. Although previous investigations had been conducted on both Numeric Comparison and Passkey Entry, ours was executed with a more faithful abstraction of the user than had been used in previous analysis. Abstractions are a useful tool in cryptography, but care must be taken not to over-generalize fundamental protocol functions. Just as it would be a mischaracterization to simply abstract queues and stacks as linked lists, so too is it a mischaracterization to abstract the user from cyborg protocols. Cyborg protocols are equally dependent on both users and devices for their functions, and any modeling decisions for analysis must be faithful to their relationship.

List of References

- [1] R. Triggs, “A quick history of bluetooth,” Available at <https://www.androidauthority.com/history-bluetooth-explained-846345/> (2020/07/10), March 2018.
- [2] N. B.-N. I. Minar and M. Tarique, “Bluetooth security threats and solutions: a survey,” *International Journal of Distributed and Parallel Systems*, vol. 3, no. 1, p. 127, 2012.
- [3] J. Padgette, J. Bahr, M. Batra, M. Holtmann, R. Smithbey, L. Chen, and K. Scarfone, “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.
- [4] 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.
- [5] J. Høegh-Omdal, C. Kaya, and M. Ottensmann, “The strandhogg vulnerability,” Available at <https://promon.co/security-news/strandhogg/> (2019).
- [6] L. Cai and H. Chen, “Touchlogger: Inferring keystrokes on touch screen from smartphone motion.” *HotSec*, vol. 11, no. 2011, p. 9, 2011.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] *Bluetooth Core Specification*, 5th ed., Bluetooth Special Interest Group (SIG), December 2019.

- [12] 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.
- [13] A. Y. Lindell, “Attacks on the Pairing Protocol of Bluetooth v2.1,” 2008.
- [14] 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.
- [15] D. Dolev and A. C. Yao, “On the Security of Public Key Protocols,” *IEEE Trans. Information Theory*, vol. 29, no. 2, pp. 198–207, 1983.
- [16] M. Bellare and P. Rogaway, “Entity Authentication and Key Distribution,” in *CRYPTO ’93*, 1993, pp. 232–249.
- [17] B. Hale, “Computationally Modeling User-Mediated Authentication Protocols,” *IACR Cryptology ePrint Archive*, p. 1239, 2019.
- [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] D. Basin, C. Cremers, J. Dreier, S. Meier, R. Sasse, and B. Schmidt, “Tamarin prover,” Available at <https://tamarin-prover.github.io/> (2019/12/14).
- [20] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to automata theory, languages, and computation, 3rd Edition* (Pearson international edition). Addison-Wesley, 2007.
- [21] S. Even and O. Goldreich, “On the Security of Multi-Party Ping-Pong Protocols,” in *24th Annual Symposium on Foundations of Computer Science*, 1983, pp. 34–39.
- [22] G. Lowe, “An Attack on the Needham-Schroeder Public-Key Authentication Protocol,” *Inf. Process. Lett.*, vol. 56, no. 3, pp. 131–133, 1995.
- [23] G. Lowe, “Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR,” *Software - Concepts and Tools*, vol. 17, no. 3, pp. 93–102, 1996.
- [24] P. Gardiner, M. Goldsmith, J. Hulance, D. Jackson, B. Roscoe, B. Scattergood, and P. Armstrong, “Failures-Divergence Refinement: FDR2 User Manual,” Available at <https://www.cs.ox.ac.uk/projects/concurrency-tools/fdr-2.94-html-manual/fdr2manual.html> (2020/01/10), Oxford University Computing Laboratory, Tech. Rep., 2012.
- [25] J. K. Millen, “The Interrogator A Tool for Cryptographic Protocol Security,” in *1984 IEEE Symposium on Security and Privacy*, 1984, pp. 134–134.

- [26] C. Meadows, “Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer,” in *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No. 99CB36344)*. IEEE, 1999, pp. 216–231.
- [27] C. J. F. Cremers, “The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols,” in *Computer Aided Verification, 20th International Conference, CAV 2008, Proceedings*, 2008, pp. 414–418.
- [28] B. Blanchet, “An Efficient Cryptographic Protocol Verifier Based on Prolog Rules,” in *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001)*, 2001, pp. 82–96.
- [29] B. Schmidt, S. Meier, C. J. F. Cremers, and D. A. Basin, “Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties,” in *25th IEEE Computer Security Foundations Symposium, CSF 2012*, 2012, pp. 78–94.
- [30] B. Blanchet, A. D. Jaggard, A. Scedrov, and J. Tsay, “Computationally sound mechanized proofs for basic and public-key Kerberos,” in *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008*, 2008, pp. 87–99.
- [31] C. Cremers, “Feasibility of multi-protocol attacks,” in *First International Conference on Availability, Reliability and Security (ARES’06)*. IEEE, 2006, pp. 8–pp.
- [32] C. Cremers and M. Horvat, “Improving the ISO/IEC 11770 standard for key management techniques,” *Int. J. Inf. Sec.*, vol. 15, no. 6, pp. 659–673, 2016.
- [33] D. A. Basin, C. Cremers, and S. Meier, “Provably repairing the ISO/IEC 9798 standard for entity authentication,” *Journal of Computer Security*, vol. 21, no. 6, pp. 817–846, 2013.
- [34] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe, “Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication,” in *IEEE Symposium on Security and Privacy, SP 2016*, 2016, pp. 470–485.
- [35] H. Krawczyk, “Skeme: A versatile secure key exchange mechanism for internet,” in *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*. IEEE, 1996, pp. 114–127.
- [36] D. Harkins and D. Carrel, “The internet key exchange (IKE),” *RFC*, vol. 2409, pp. 1–41, 1998. Available: <https://doi.org/10.17487/RFC2409>
- [37] H. K. Orman, “The OAKLEY key determination protocol,” *RFC*, vol. 2412, pp. 1–55, 1998. Available: <https://doi.org/10.17487/RFC2412>

- [38] D. Maughan, M. Schertler, M. Schneider, and J. Turner, “Internet security association and key management protocol (ISAKMP),” *RFC*, vol. 2408, pp. 1–86, 1998. Available: <https://doi.org/10.17487/RFC2408>
- [39] M. Abdalla and D. Pointcheval, “Simple password-based encrypted key exchange protocols,” in *Cryptographers’ track at the RSA conference*. Springer, 2005, pp. 191–208.
- [40] 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.
- [41] H. Krawczyk, “SIGMA: the ‘sign-and-mac’ approach to authenticated diffie-hellman and its use in the ike-protocols,” in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings* (Lecture Notes in Computer Science), D. Boneh, Ed. Springer, 2003, vol. 2729, pp. 400–425. Available: https://doi.org/10.1007/978-3-540-45146-4_24
- [42] R. Canetti and H. Krawczyk, “Security analysis of ike’s signature-based key-exchange protocol,” in *CRYPTO 2002, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, 2002, pp. 143–161.
- [43] C. A. Meadows, “Analysis of the internet key exchange protocol using the NRL protocol analyzer,” in *1999 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 9-12, 1999*. IEEE Computer Society, 1999, pp. 216–231. Available: <https://doi.org/10.1109/SECPRI.1999.766916>
- [44] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold, “Just fast keying: Key agreement in a hostile internet,” *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 2, pp. 242–273, 2004. Available: <https://doi.org/10.1145/996943.996946>
- [45] 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.
- [46] B. Ustaoglu, “Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS,” *Des. Codes Cryptogr.*, vol. 46, no. 3, pp. 329–342, 2008. Available: <https://doi.org/10.1007/s10623-007-9159-1>
- [47] J. Lee and C. S. Park, “An efficient authenticated key exchange protocol with a tight security reduction,” *IACR Cryptology ePrint Archive*, vol. 2008, p. 345, 2008.

- [48] T. Okamoto, “Authenticated key exchange and key encapsulation in the standard model,” in *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings* (Lecture Notes in Computer Science), K. Kurosawa, Ed. Springer, 2007, vol. 4833, pp. 474–484. Available: https://doi.org/10.1007/978-3-540-76900-2_29
- [49] M. Kim, A. Fujioka, and B. Ustaoglu, “Strongly secure authenticated key exchange without naxos’ approach,” in *Advances in Information and Computer Security, 4th International Workshop on Security, IWSEC 2009, Toyama, Japan, October 28-30, 2009, Proceedings* (Lecture Notes in Computer Science), T. Takagi and M. Mambo, Eds. Springer, 2009, vol. 5824, pp. 174–191. Available: https://doi.org/10.1007/978-3-642-04846-3_12
- [50] C. M. Ellison, “Ceremony design and analysis,” *IACR Cryptology ePrint Archive*, p. 399, 2007.
- [51] D. Sun, L. Sun, and Y. Yang, “On Secure Simple Pairing in Bluetooth Standard v5.0-Part II: Privacy Analysis and Enhancement for Low Energy,” *Sensors*, vol. 19, no. 15, p. 3259, 2019.
- [52] 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.
- [53] S. Pasanen, K. Haataja, N. Päävinen, and P. J. Toivanen, “New Efficient RF Fingerprint-Based Security Solution for Bluetooth Secure Simple Pairing,” in *43rd Hawaii International International Conference on Systems Science (HICSS-43 2010), Proceedings*, 2010, pp. 1–8.
- [54] 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.
- [55] N. I. of Standards and Technology, “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.
- [56] P. Rogaway and T. Shrimpton, “Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance,” in *Fast Software Encryption, 11th International Workshop, FSE, 2004*, pp. 371–388.

- [57] H. Krawczyk, M. Bellare, and R. Canetti, “HMAC:Keyed-Hashing for Message Authentication,” *RFC*, vol. 2104, pp. 1–11, 1997.
- [58] 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* (Lecture Notes in Computer Science), C. Dwork, Ed. Springer, 2006, vol. 4117, pp. 602–619. Available: https://doi.org/10.1007/11818175_36
- [59] X. Wang, Y. L. Yin, and H. Yu, “Finding collisions in the full SHA-1,” in *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings* (Lecture Notes in Computer Science), V. Shoup, Ed. Springer, 2005, vol. 3621, pp. 17–36. Available: https://doi.org/10.1007/11535218_2
- [60] X. Wang and H. Yu, “How to break MD5 and other hash functions,” in *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings* (Lecture Notes in Computer Science), R. Cramer, Ed. Springer, 2005, vol. 3494, pp. 19–35. Available: https://doi.org/10.1007/11426639_2
- [61] N. Koblitz, A. Menezes, and S. Vanstone, “The state of elliptic curve cryptography,” *Designs, codes and cryptography*, vol. 19, no. 2-3, pp. 173–193, 2000.
- [62] J. H. Silverman, *The arithmetic of elliptic curves*. Springer Science & Business Media, 2009, vol. 106.
- [63] N. S. A. S. Service, “Commercial national security algorithm suite and quantum computing faq,” Available at <https://cryptome.org/2016/01/CNSA-Suite-and-Quantum-Computing-FAQ.pdf> (2020/07/10).
- [64] M. J. J. Jr., A. Menezes, and A. Stein, “Solving Elliptic Curve Discrete Logarithm Problems Using Weil Descent,” *IACR Cryptology ePrint Archive*, vol. 2001, p. 41, 2001.
- [65] A. Menezes, T. Okamoto, and S. A. Vanstone, “Reducing elliptic curve logarithms to logarithms in a finite field,” *IEEE Trans. Inf. Theory*, vol. 39, no. 5, pp. 1639–1646, 1993.
- [66] 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.
- [67] D. J. Bernstein and T. Lange, “Safecurves: choosing safe curves for elliptic-curve cryptography,” Available at <https://safecurves.cr.yp.to> (2019/11/24).

- [68] O. Goldreich, S. Goldwasser, and S. Micali, “How to construct random functions,” in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 241–264.
- [69] 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.
- [70] 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.
- [71] 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.
- [72] J. Brendel, M. Fischlin, F. Günther, and C. Janson, “PRF-ODH: Relations, Instantiations, and Impossibility Results,” Cryptology ePrint Archive, Report 2017/517.
- [73] 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.
- [74] B. Hale, “User-Mediated Authentication Protocols and Unforgeability in Key Collision,” in *Provable Security - 12th International Conference, ProvSec 2018, Proceedings*, 2018, pp. 387–396.
- [75] W. Dai, “Crypto++ 5.6.0 benchmarks,” Available at <https://www.cryptopp.com/benchmarks.html> (2020/08/05), March 2009.
- [76] B. Schmidt, S. Meier, C. J. F. Cremers, and D. A. Basin, “Automated analysis of diffie-hellman protocols and advanced security properties,” in *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, S. Chong, Ed. IEEE Computer Society, 2012, pp. 78–94. Available: <https://doi.org/10.1109/CSF.2012.25>
- [77] C. Cremers, “Symbolic security analysis using the tamarin prover,” in *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, D. Stewart and G. Weissenbacher, Eds. IEEE, 2017, p. 5. Available: <https://doi.org/10.23919/FMCAD.2017.8102229>
- [78] D. A. Basin, C. J. F. Cremers, T. H. Kim, A. Perrig, R. Sasse, and P. Szalachowski, “ARPKI: attack resilient public-key infrastructure,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ*,

USA, November 3-7, 2014, G. Ahn, M. Yung, and N. Li, Eds. ACM, 2014, pp. 382–393. Available: <https://doi.org/10.1145/2660267.2660298>

- [79] *Tamarin-Prover Manual: Security Protocol Analysis in the Symbolic Model*, Available at <https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf> (2019/11/23), The Tamarin Team, June 2019.
- [80] G. Lowe, “A hierarchy of authentication specification,” in *10th Computer Security Foundations Workshop (CSFW ’97), June 10-12, 1997, Rockport, Massachusetts, USA*. IEEE Computer Society, 1997, pp. 31–44. Available: <https://doi.org/10.1109/CSFW.1997.596782>
- [81] S. Delaune, S. Kremer, and L. Robin, “Formal verification of protocols based on short authenticated strings,” in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 2017, pp. 130–143.
- [82] Microsoft, “Surface book 2 tech specs,” Available at <https://www.microsoft.com/en-us/surface/devices/surface-book-2/tech-specs> (2020/08/21), August 2020.

Initial Distribution List

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California