École Polytechnique Fédérale de Lausanne

A Double Ratchet implementation for the Android Bluetooth Stack

by Francesco Berla

Master Project Report

Approved by the Examining Committee:

Prof. Dr. sc. ETH Mathias Payer
Thesis Advisor

Prof. Dr. Daniele Antonioli
Thesis Supervisor

EPFL IC IINFCOM HEXHIVE
BC 160 (Bâtiment BC)
Station 14
CH-1015 Lausanne

June 12, 2021

Abstract

The Bluetooth protocol offers very limited security guarantees and many attacks have been performed against it. On the other hand, the Double Ratchet algorithm gives strong security guarantees like future and forward secrecy. This work merges the ubiquity of Bluetooth with the security of Double Ratchet resulting in a more secure communication. The result is an Android library integrated in the Bluetooth stack that will allow to use the Double Ratchet algorithm directly at the link layer.

Contents

Chapter 1

Introduction

In this report we are going to present the work that we done in the Android Bluetooth Stack during our semester project at HexHive.

The Bluetooth protocol is extremely widespread but it carries a series of design flaws and vulnerabilities that affect its security leading to a problematic situation.

In order to improve the security of the Bluetooth protocol we propose to augment it with the Double Ratchet protocol, a cryptography protocol providing strong confidentiality, integrity and authenticity guarantees as well as forward and future secrecy.

In this project we focused on creating a library implementing the Double Ratchet algorithm for the Android Bluetooth stack. This will allow future works to use this library in the desired position on the stack. The design is conceived in a way that it can be used to modify the Bluetooth stack from the Link Layer (bypassing completely the Bluetooth security mechanisms) to the Application Layer (on top of the actual security mechanisms).

In the end, we managed to create an implementation of the Double Ratchet protocol in C++ using the cryptography primitives offered by BoringSSL and the Android ecosystem.

Chapter 2

Background

As mentioned in the Introduction, Bluetooth is an extremely widespread technology with 4.2 billion Bluetooth capable devices shipped in 2019 and 6.2 expected in 2024 [4]. There are two types of Bluetooth: Bluetooth Classic and Bluetooth Low Energy. Bluetooth Classic is the original design of Bluetooth and is used to have persistent connections between two devices with high throughput. Bluetooth Low Energy aims to be used in low powered devices that doesn't need to send large quantities of data and therefore has a lower power consumption and non persistent connections.

Both Bluetooth Classic and Bluetooth Low Energy are considered insecure for today standards and many attacks are possible. Now we will present two of the most important ones.

The first one is the KNOB attack [2]: the KNOB attack consists in lowering the entropy of the key negotiation procedure to 1 byte. A value of entropy of only 1 byte means that the resulting keys are easily bruteforced compromising their security. This allows an attacker to downgrade the security of the communication to a level equivalent of a non encrypted one.

The second attack against Bluetooth that we want to present is the BIAS attack [1]. It can be performed only against Bluetooth Classic but is nevertheless a very powerful technique as it allows to impersonate other devices in order to interject or to falsify communications.

The low security of the Bluetooth protocol and the will to improve the situation in order to mitigate present and future attacks is the main motivation for this work.

Chapter 3

Design

In this chapter we are going to present the design of our work. In the *Overview* section we will discuss all of the high level design choices while in the *Double Ratchet Algorithm* section we will illustrate the inner workings of the Double Ratchet Algorithm.

3.1 Overview

In this section, we are going to discuss all the layers of our design from the choice of the target to the cryptography library used. The first design choice is the target platform. We choose Android as it is the most popular mobile operating system that is currently running on 6.05 billion devices and used by 1.6 billions users worldwide. After choosing the operating system, we had to chose a version: we choose Android 9 because is the last version where you can remount read and write, and therefore modify, the system partition containing the operating system files without flashing completely the operating system to the phone. This decision was made to lower compilation and development times.

We chose to implement the improvements to the Android Bluetooth Stack (Fluoride) in order to be able to modify the link layer. This has the great advantage that we will be able to improve every Bluetooth connection of the phone in a transparent way without needing additional work from the consumers of the Android Bluetooth API in order to improve the security of their application. This allow to greatly reduce the effort required as only one component, the Bluetooth stack, has to modified and all users will benefit from it. This is particularly important as the security aspect is often neglected by Application developers due to financial costs and because many small Applications are developed by single authors that do not necessarily have the knowledge to secure the communications.

The next design choice is the algorithm or protocol to be used to improve the security of the

Bluetooth communications. The decision lead to the choice of the Double Ratchet algorithm [5]. The Double Ratchet algorithm is an algorithm used for secure communications that provides the basic security guarantees (confidentiality, integrity, authenticity) and two additional properties: forward and future secrecy. Forward secrecy grants that past session keys cannot be recovered or computed from the long-term key alone, grating therefore resistance to the leak of the long-term key. Future secrecy is the property that allows future communication to be secure even if a single session key is leaked. This is made possible by the fact that session keys are regenerated every communication round.

After the higher level design decisions, we had to focus on two implementation level aspects: the programming language used and the cryptography library used to implement the Double Ratchet Algorithm.

The Android Bluetooth stack of Android 9 is mainly written in C++ and for that reason we opted for doing our implementation in that language. C++ is a language known for being very good for system level programs and its interoperability with C makes it very versatile in the Android codebase. Rust was another interesting option thanks to its memory safety guarantees and its low level features but at the moment it is not yet well integrated in the Android ecosystem even if many efforts for making this happen are in progress.

The Double Ratchet algorithms requires many cryptographic primitives in order to be implemented, therefore we had to chose a provider for those primitives as writing them ourselves didn't make sense both in terms of development time and in security guarantees. Android, through the libchrome library, ships a cryptographic library called BoringSSL. BoringSSL is a fork from OpenSSL and for this reason shares the majority of the API with it. In order to avoid to inject additional code into the Android stack we decided to use the provided BoringSSL library as the provider of our cryptographic primitives.

3.2 Double Ratchet Algorithm

In this section we are going to present the Double Ratchet Algorithm in more detail. This explanation is useful to show why we chose this Algorithm and how the implementation looks like.

The Double Ratchet Algorithm has three main components: the KDF chains, the symmetric-key ratchet and the Diffie-Hellman ratchet. The KDF function is a cryptographic function with three input parameters: a secret, a random KDF key and input data. The output of a KDF function is data that looks random without knowing the key. A KDF chain is composed by a sequence of KDF operations where part of the output of the previous KDF is the input of the next one, generating every time an additional output key. The resulting chain has three security property:

Resilience The output key appears random without knowing the KDF keys.

Forward secrecy Previous Output keys still appear random to an adversary that discover the KDF key at a specific point in time.

Break-in recovery Future Output keys still appear random to an adversary that discover the KDF key at a specific point in time given that the future inputs use enough entropy.

Every user stores three KDF chains: the root chain, one for sending and another one for receiving.

The symmetric-key ratchet is the second building block of the Double Ratchet. It uses KDF chains presented above with constants input data, with input a chain key and output another chain key and a message key. Every message key is used only for one message that can be deleted after use.

The last main component of the Double Ratchet Algorithm is the Diffie-Hellman Ratchet. The purpose of the Diffie-Hellman Ratchet is to grant Future secrecy by updating chain key with Diffie-Hellman output every round. At every message one of the two parties send its new public key and the receiver updates its ratchet key pair and calculates a new DH output. This procedure is called DH ratchet step. Ratchet steps are used to generate new sending and receiving chain keys creating therefore the future secrecy property mentioned before. The DH output is used as input of the KDF root chain and therefore we computed both the new Receiving and Sending chain keys but we also update the root key every round trip.

The combination of the components mentioned above results in the Double Ratchet algorithm: a symmetric-key ratchet step is made in the sending/receiving chain when a message is sent/received in order to generate the new message key and a DH ratchet step is performed before the symmetric-key ratchet in order to replace the chain keys when a new ratchet public key is received.

Chapter 4

Implementation

In this chapter we are going to present the details of our implementation and the challenges that we faced while working on the project.

The output of this project, as stated in the Design section, is a C++ library that provides the Double Ratchet functionalities. We decided to integrate this code in the Bluetooth stack of Android and specifically in the *stack* folder inside the Bluetooth component. This resulted in a tight integration with the others components of the Bluetooth codebase as the result is a single shared object called *libbluetooth.so* creating an end result that is indistinguishable from an unmodified library.

In implementing our library we tried to follow as close as possible the Double Ratchet documentation, this has two major benefits. The first one is that there is a lower probability of making mistakes and the code is more easily testable using the specification. The second benefit is that following the standard algorithm we favor future contributions as everyone who knows, or reads, the Double Ratchet specification, will be able to understand and modify the program. The resulting library exposes the complete Double Ratchet API allowing future works to modify the control flow of the Bluetooth communication process and substitute the standard Bluetooth security mechanisms with the Double Ratchet primitives in a simple and concise way.

The Double Ratchet protocol gives a certain degree of freedom in the choice of cryptographic primitives used. For the generation of Diffie-Hellman key pairs we decided to use the *Curve25519* elliptic curve, similarly, for calculating the Diffie-Hellman step we used the *X25519* function. The specification of the functions *KDF_RK*, *KDF_CK* for calculating the Double Ratchet steps and the *HKDF* key expansion function allows to chose between *SHA-256* and *SHA-512*. For all of those three implementations we choose to use *SHA-256*. We had to implement our version of *HKDF* as the one included in BoringSSL was not available from the Fluoride stack.

Before being able to modify the Android Bluetooth stack and inject our code we had to have a modifiable version of it. The first problem has been compiling Fluoride as the official

documentation is outdated and incorrect. After finding out how to compile the Bluetooth Stack we faced many problem in creating a compiled version for the phones that we had available without compiling and flashing the whole AOSP. This lead to us to change the target phone to a Google Pixel 2 that is officially supported. The next problem, always related to the impossibility of flashing a debug version of Android to our phone, was the impossibility to remount the system partition to read-write in Android 11. After some research, we discovered that after Android 9 it is impossible to do it and therefore we had to flash our phone to Android 9 and start working on that version of Fluoride. The last major problem that we encountered was a common one in security development: the lack of documentation on the OpenSSL/BoringSSL API resulting in a slower development.

Chapter 5

Evaluation

We managed to extend the Android Bluetooth stack with a Double Ratchet library. The resulting *libbluetooth.so* library is fully functional and performs identical compared to the original one when substituted in a Google Pixel 2 phone running Android 9 revision 46. The development has been carried on using Test Driven Development and following really closely the Double Ratchet specification. This allows us the have a high degree of confidence in the correctness of our implementation.

Chapter 6

Related Work

Many efforts have been done during the years in order to improve the security of the Bluetooth protocol like *Dabinder* [3]. Nevertheless they always fix only a subset of the Bluetooth protocol or a specific user case without trying to improve the security of all Bluetooth communications. Our work tries to elevate the security of all communications done through Bluetooth on an Android device. The usage of Double Ratchet algorithm presented here could be applied to others platforms like embedded devices enhancing the security level of even more Bluetooth enabled devices.

Chapter 7

Conclusion

In this work we successfully extended the Fluoride Android Bluetooth Stack with a Double Ratchet library that can be used to secure the communications at the link layer. With this work we prove that is possible to modify the Android Bluetooth stack in order to improve its security properties. The resulting *libbluetooth.so* library is the first step towards the usage of the Double Ratchet as security mechanism for the Bluetooth protocol at the link layer in Android. The adoption of Double Ratchet instead of the standard Bluetooth protocol security mechanism will bring massive security improvements thanks to the stronger properties guaranteed by the Double Ratchet.

Future work should focus on modifying the pairing and communications mechanisms inside the Fluoride stack to use the provided Double Ratchet API instead and evaluate the possible downsides in terms of usability, power consumption and latency.

This project chose to focus on the Android platform but the same idea of altering the Operating System Bluetooth Stack with the addition of Double Ratchet could be applied to other platforms such as embedded devices.

Bibliography

[1] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. “Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy”. In: *ACM Transactions on Privacy and Security (TOPS)* 23.3 (2020), pp. 1–28. DOI: 10.1145/3394497.

[2] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. “The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation of Bluetooth BR/EDR.” In: *Proceedings of the USENIX Security Symposium*. Aug. 2019.

[3] Muhammad Naveed, Xiao-yong Zhou, Soteris Demetriou, XiaoFeng Wang, and Carl A Gunter. “Inside Job: Understanding and Mitigating the Threat of External Device Mis-Binding on Android.” In: *NDSS*. 2014.

[4] Shawn O’Dea. *Android - Statistics & Facts*. May 2021. URL: https://www.statista.com/topics/876/android/.

[5] Trevor Perrin and Moxie Marlinspike. *The Double Ratchet Algorithm*. Nov. 2016. URL: https://signal.org/docs/specifications/doubleratchet/.