Trust Dies in Darkness: Shedding Light on Samsung's TrustZone Keymaster Design

Alon Shakevsky, Eyal Ronen, and Avishai Wool, *Tel-Aviv University*

https://www.usenix.org/conference/usenixsecurity22/presentation/shakevsky

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

Keybuster is a research tool that allows to interact with the Keymaster TA (Trusted Application) on Samsung devices that run Android. Keybuster implements a Keymaster client - based on the `libkeymaster_helper.so` library from Samsung's Keymaster HAL.

Keybuster requires sufficient permissions (root and SELinux context) to access TZ drivers. To achieve this, we rooted our device using Magisk and used the strong context that it provides. Keybuster requires special hardware - an Samsung Galaxy smartphone (S9 and newer models) with a Trusted Execution Environment (TEE). The binary can be downloaded from GitHub releases or built using Android NDK. To reproduce our attacks, only minimal software requirements are required (e.g., adb/ssh, openssl and python3).

Keybuster allows to reproduce the attacks that we describe in the paper - the IV reuse attack and the downgrade attack. The GitHub repository contains detailed step-by-step instruction on how to recreate both attacks. Additionally, it allows researchers to freely explore the Keymaster TA without input validation or filtering. Samsung validated both attacks using Keybuster and assigned a High severity CVE to each issue.

In essence, the proof-of-concept attacks utilize Keybuster to demonstrate private key material extraction of hardware-protected keys that were encrypted by the TEE.

A.2 Artifact check-list (meta-information)

- **Compilation: Android NDK (alternatively, the binary can be downloaded from GitHub releases)**
- **Binary: keybuster binary for Android, included in GitHub releases**
- **Run-time environment: Android specific, requires a sufficiently strong context (e.g., rooted device)**
- **Hardware: Rooted Samsung Galaxy device (e.g. available over adb/SSH)**
- **Security, privacy, and ethical concerns: The vulnerabilities were responsibly disclosed to Samsung and they issued patches. Running on a rooted Android device (that is connected to WiFi) over SSH might be dangerous.**
- **Output: Console output. GitHub repository includes expected output.**
- **Experiments: Follow instructions in GitHub repository to run the proof-of-concept scripts**
- **How much disk space required (approximately)?: Minimal**
- **How much time is needed to prepare workflow (approximately)?: < 2 minutes**
- **How much time is needed to complete experiments (approximately)?: < 2 minutes**
- **Publicly available (explicitly provide evolving version reference)?: Will be made available on `https://github.com/shakevsky/keybuster`**
- **Code licenses (if publicly available)?: Apache-2.0 License**
- **Archived (explicitly provide DOI or stable reference)?: Will be made available on `https://github.com/shakevsky/keybuster/tree/v0.1.0`**

A.3 Description

A.3.1 How to access

The stable artifact will be made available on `https://github.com/shakevsky/keybuster/tree/v0.1.0`.

A.3.2 Hardware dependencies

To reproduce our attacks with Keybuster it is required to have a Samsung device (S9 and newer models) with a sufficiently strong context, e.g., by rooting a device or by having a development device from Samsung. We can try to make such device available over SSH. Unpacking the artifact requires very little space (only to download the binary or compile the sources).

A.3.3 Software dependencies

Keybuster is a binary that can be run on a rooted Samsung device. To access such a device we've used adb, and we can try to make a vulnerable device available over SSH. To reproduce the IV reuse attack, no additional software is required. To reproduce the downgrade attack (e.g., against a simplified Secure Key Import server) python3 and openssh can be used (although they only emphasize the point - running the proof of concept script should be enough).

A.3.4 Security, privacy, and ethical concerns

The vulnerabilities were responsibly disclosed to Samsung and they issued patches. We have some concerns over giving SSH access to a rooted Android device over the internet, as it can potentially compromise the home WiFi network of one of the authors.

A.4 Installation

Assuming that we'll provide remote SSH access, we can upload all the necessary files to the device so that no further setup is required and reviewers can simply run the proof-of-concept scripts. Otherwise, the GitHub repository includes detailed steps of how to reproduce the attacks (which bash commands to run).

A.5 Evaluation and expected results

Full details on reproducing the proof-of-concept attacks, as well as expected outputs, will be made available in the GitHub repository.

The main claims of our paper include:

- We show that the hardware protection in Samsung Galaxy S9 devices is vulnerable to an IV reuse attack on AES-GCM, allowing the extraction of protected key material.

- We show a downgrade attack on Samsung Galaxy S10, S20, and S21 devices, making them vulnerable to our IV reuse attack.
- We evaluate the impact of our attacks and describe how to exploit them to misuse the Keystore key attestation to bypass FIDO2 WebAuthn login and compromise Google's Secure Key Import.

The proof-of-concept attacks that use Keybuster support the claims:

- The IV reuse PoC shows that we are able to fully recover key material from hardware-protected keys that were encrypted by the TEE.
- The downgrade attack PoC shows that we are able to force even the latest devices (S10, S20, and S21) to generate keys that are vulnerable to IV reuse.
- The README.md of the downgrade attack also includes information about how an attacker can use the downgrade attack to target higher level cryptographic protocols such as Secure Key Import (we included python3 code that emulates the server and show how a successful attack breaks the security of the keys) and WebAuthn (we included a GDB script that we've successfully used against the StrongKey FIDO2 WebAuthn server, as described in the paper).

The expected outputs and results are shown in the proof-of-concept README files in the GitHub repository.

A.6 Notes

A.7 Version

Based on the LaTeX template for Artifact Evaluation V20220119.