

# Encrypted Filesystem for Applications



Master Thesis: Portable multi-key encrypted file storage

Stefan Schindler, BSc

Institut für Netzwerke und Sicherheit & Rust Zürichsee

# Why encrypt in the application?



# Motivation

## For developers:

- Manageable for an end-user
- Independent from the OS
- Sync- or Migrate-able from one device to another
- Easy to use & hard to make mistakes

## For users:

- Normal workflow → fast biometric
- Backup and Migration Path

### Sample App Authentication

Please login to get access

Sample App is using Android biometric authentication



Touch the fingerprint sensor

[Use app password](#)

## Security guarantees?

- Offline attacker, Stolen/Cloud Backup aka. Evil Maid
- Supply chain Hardware or Firmware
- Supply chain Software or Operating System

## What about 0-day exploits?

- Same as a debugger
- Memory encryption is just obfuscation

# Technology Decision

- Use Rust for memory safety
- `#![forbid(unsafe_code)]` ← only safe rust
- Use modern Key Derivation Function Argon2 [3]
- Replace AES with ChaCha20-Poly1305 [2]
- Embeddable for the Digidow.eu project

# Technology Decision - Why replace AES?

- See L2 Cache-timing attacks on AES [1]
- See Micro-architectural exploitation and other hardware attacks<sup>1</sup>
- Replaces S-Boxes with XOR, Integer Addition Modulo  $2^{32}$ , and Integer Shift with Roll

<sup>1</sup><https://github.com/codexlynx/hardware-attacks-state-of-the-art>

# Speed Comparison

From RFC8439

| Chip                       | AES-128-GCM | ChaCha20-Poly1305 |
|----------------------------|-------------|-------------------|
| OMAP 4460                  | 24.1 MB/s   | 75.3 MB/s         |
| Snapdragon S4 Pro          | 41.5 MB/s   | 130.9 MB/s        |
| Sandy Bridge Xeon (AES-NI) | 900 MB/s    | 500 MB/s          |

# Master Key Management





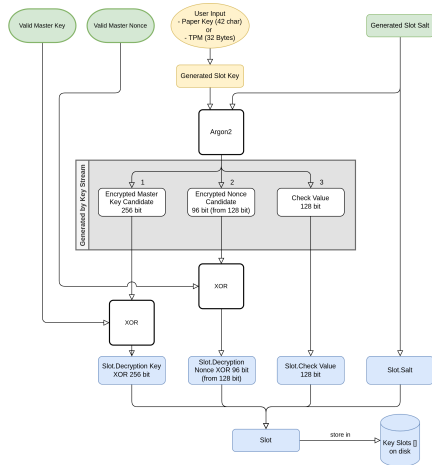
# Technical Decisions

- Reduce the amount of places where the master key resides in memory
- Use the type system to keep the nonce<sup>2</sup> in sync
- Lock the open files so syncing operations don't cause problems
  - Unix: `flock(file, rustix::fs::FlockOperation::LockExclusive)`
  - Windows: `lock_file(file, LOCKFILE_EXCLUSIVE_LOCK)`
- Autosave on `Drop::drop(&mut self)`

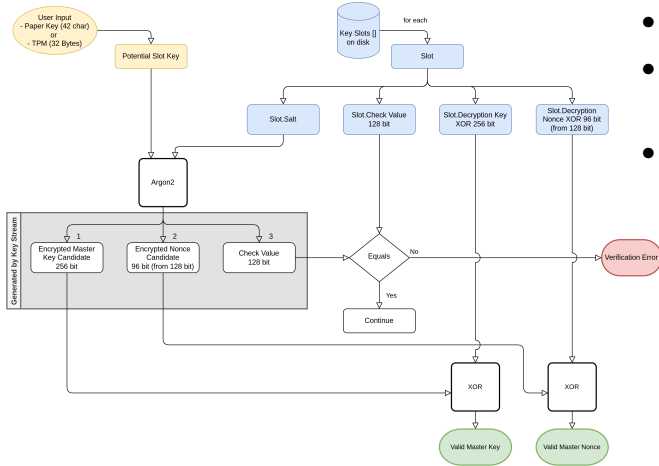
<sup>2</sup>Number used once

# Master Key Creation

- Paper Key for user
- Add n-th key for other devices
- TPM Key for secure hardware
- Master Key only in system memory



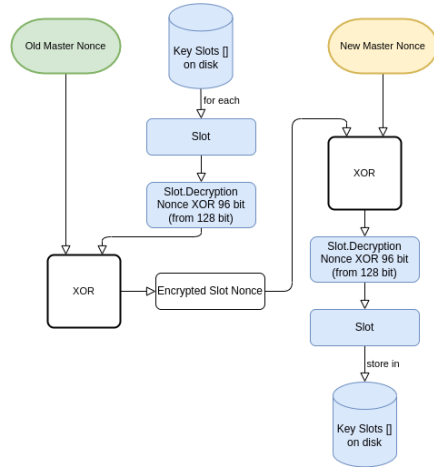
# Per Slot Master Key Decryption



- Try all stored key material
- Nonce is rather predictable, few writes of the whole directory
- “If the seed used to generate the stream is secret then it is **infeasible to compute previously generated bytes** in an Argon2 key stream” Philipp Jovanovic, PhD, Associate Professor in Information Security (University College London)

# Update Master Key Nonce's

- Requires the Master Key for opening the vault
- Does not require the other Slot Keys
- Is done on every `Write::flush()`
- Very cheap to compute



# Schneeglöggli

Schneeglöggli 21.02.2023

Stefan Schindler



# File Content Encryption and Virtual Filesystem



# File Content Encryption

The based on established code and practices:

- Authenticated Encryption with Associated Data (AEAD) [5]  
ChaCha20-Poly1305 [2] developed by Daniel J. Bernstein
- The ring crate with a safe Rust interface
- C and assembly language code in ring come from BoringSSL [6], that is derived from OpenSSL [4]

# Abstraction Design Choices

The directory:

- HashMap with the clear path as key
- Minimal index → every change requires a full encrypt + write
- Create and access time from underlying filesystem
- Size may be reported incorrectly, AEAD TAG + Padding
- Filesystem Lock during application run

Per file:

- Loading the contents on demand to memory
- Guarding open file contents with RwLock per file
- Automatic flush on scope drop



# Native Integration



# Android Integration - the plan

Build system integration:

- Integrate with gradle
- Build a native toolchain with cmake and cargo

Use the Kotlin runtime for:

- Handle system permissions
- Communicate with finger print reader and KeyStore
- Handle user input for setup and recovery (Paper Key)

# Artistic representation of the working environment

Dramatization incoming

Are you ready?

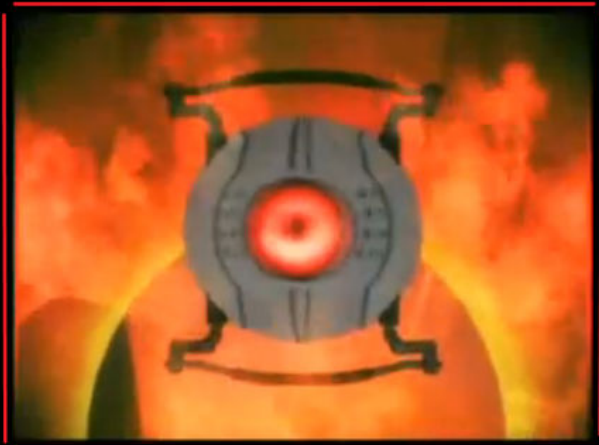
# Artistic representation of the working environment

Source:

[https://www.deviantart.com/](https://www.deviantart.com/aperture--science/art/Android-hell-137246583)

[aperture--science/art/Android-](https://www.deviantart.com/aperture--science/art/Android-hell-137246583)

[hell-137246583](https://www.deviantart.com/aperture--science/art/Android-hell-137246583)



# ANDROID HELL

A real place where you will be sent at the first sign of defiance.



# Current Problems with the Android Ecosystem 1/2

- Open Android Studio
  - Update 2022.1.1 Patch 1
  - Project is broken now
- Start a new default project with Update 2022.1
  - Build fails
  - Find the project setting to upgrade gradle to 7.6
- Find the docs to get finger print sensor presence?
  - No complete sample, just snippets
  - No docs on how to store an actual key in the HSM

## Current Problems with the Android Ecosystem 2/2

- How to generate FFI C include header?
  - javah is replaced by javac -h
  - Does not work with Kotlin
- How to find compiled libraries with gradlew or Android Studio?
  - cmake is completely reconfigured by multiple compile\_commands.json
- How to compile the glue crate? (After reading the error messages)
  - Wast a lot of time building build.rs
  - Install cargo-ndk still pass the NDK path manually

# Linux / Windows Integration

Build system cargo:

- Provide a Rust lib or C98 FFI staticlib interface
- TPM/vTPM communication

Runtime:

- Provide a Rust or C98 FFI interface
- Handle TPM/vTPM communication
- Handle user input for setup and recovery (Paper Key)

# Lets relax

Walensee 21.02.2023

Stefan Schindler





# Encrypted Filesystem for Applications



Master Thesis: Portable multi-key encrypted file storage

Stefan Schindler, BSc

Institut für Netzwerke und Sicherheit & Rust Zürich

# References I

- [1] Daniel J. Bernstein. 2005. Cache-timing attacks on AES.  
<https://cr.yp.to/antiforgery/cachetiming-20050414.pdf>. (2005).
- [2] Daniel J. Bernstein. 2018. ChaCha20-Poly1305 is an authenticated encryption with additional data (AEAD) algorithm, RFC8439.  
<https://datatracker.ietf.org/doc/html/rfc8439>. (2018).
- [3] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. 2017. Argon2: the memory-hard function for password hashing and other applications.  
<https://www.cryptolux.org/index.php/Argon2>. (2017).
- [4] OpenSSL Contributors. 1998-. OpenSSL. <https://www.openssl.org/>. (1998-).

## References II

- [5] Google Engineering. 2021. Authenticated Encryption with Associated Data.  
<https://developers.google.com/tink/aead>. (2021).
- [6] Google Engineering and OpenSSL Contributors. 2014-. BoringSSL.  
<https://boringssl.googlesource.com/boringssl/>. (2014-).



**JOHANNES KEPLER  
UNIVERSITY LINZ**