RISC.KEY
PART 1 OF 3

alienflip
https://github.com/auto-cannibal

June 28, 2024

ABSTRACT

The data is created when a key is pressed on a laptop - but then what? In fact, what happens to the data created whenever *any* peripheral on a device is interacted with by an agent?

Signals are produced - then the ***data is stored in memory***, and displayed right back to the agent in control of the device.

How can we ensure that a rogue state cannot use it's wealth to perform ultra-sleuthing [1] on a devices memory to extract sensitive information from keystrokes made on files that were never even saved? Or that a maliciously planted key-logger [2] cannot perform telemetry to a remote hacker?

In this series of papers, a keyboard device is proposed to tackle this exact situation.

This proposal will rely initially on FPGA technology in order to test the custom IO and Cryptography primitives necessary for a full implementation of the device.

1 Approach

The approach will be minimalist to begin with. Basic IO: keys and a display will do for now. These Peripherals should be memory-less, so that they are not a weak link in the devices security stack. For the keys we can use a NxN matrix button pad on PCB, and connect it to the FPGA Input pins. For the display, initially we use a MxM matrix LED on PCB connected to the FPGA output. We are cafeful in both cases to avoid OLED/LCD displays or standard keyboards, as there may be custom microprocessors (hence buffers) within these 3rd party peripherals.

The Input signals produced should be encrypted as soon as they reach the FPGA. The device will first expect a phrase. It will use this for the rest of the active session within volatile memory to decrypt the signals from memory. Once this key has been entered, the FPGA will run a loop continuously searching through the signals (also stored in volatile memory) from the current working session. It will decrypt them, and write them to the LED matrix for the agent to read.

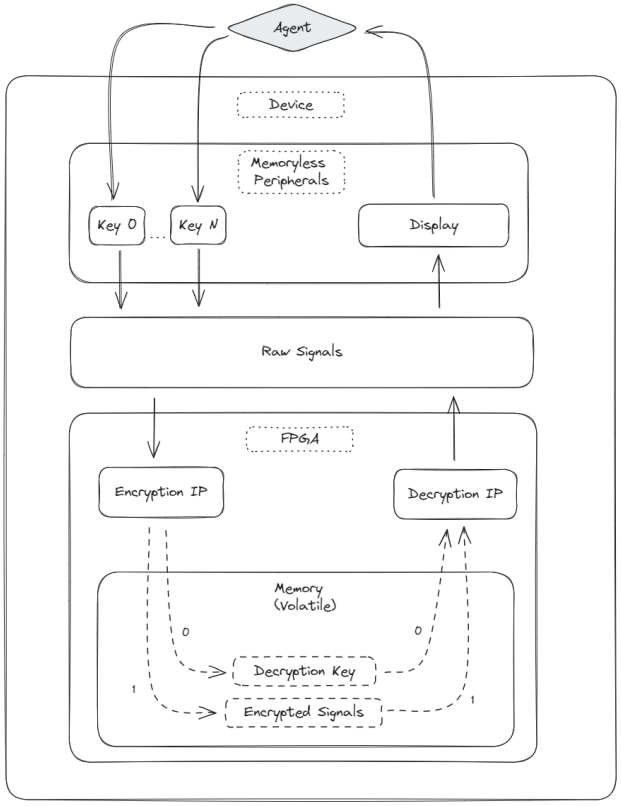

Figure 1: Device Flow Diagram

Storing encrypted data in volatile memory we can rest assured that the sensitive content we are working on will not be retained. When we are done with the message in the current working session, we can optionally send the a final binary signal to flash memory, and then switch off the device. This gives optional persistence to the encrypted data. Note that in this model, to read the data saved to flash we will need to remember **ourselves** the decryption key we stored in memory in the previous session.

The high-level diagram and specification in this paper will be followed in the next paper by a low-level block diagram.

References

[1] https://en.wikipedia.org/wiki/The_Sleuth_Kit

[2] https://en.wikipedia.org/wiki/Keystroke_logging