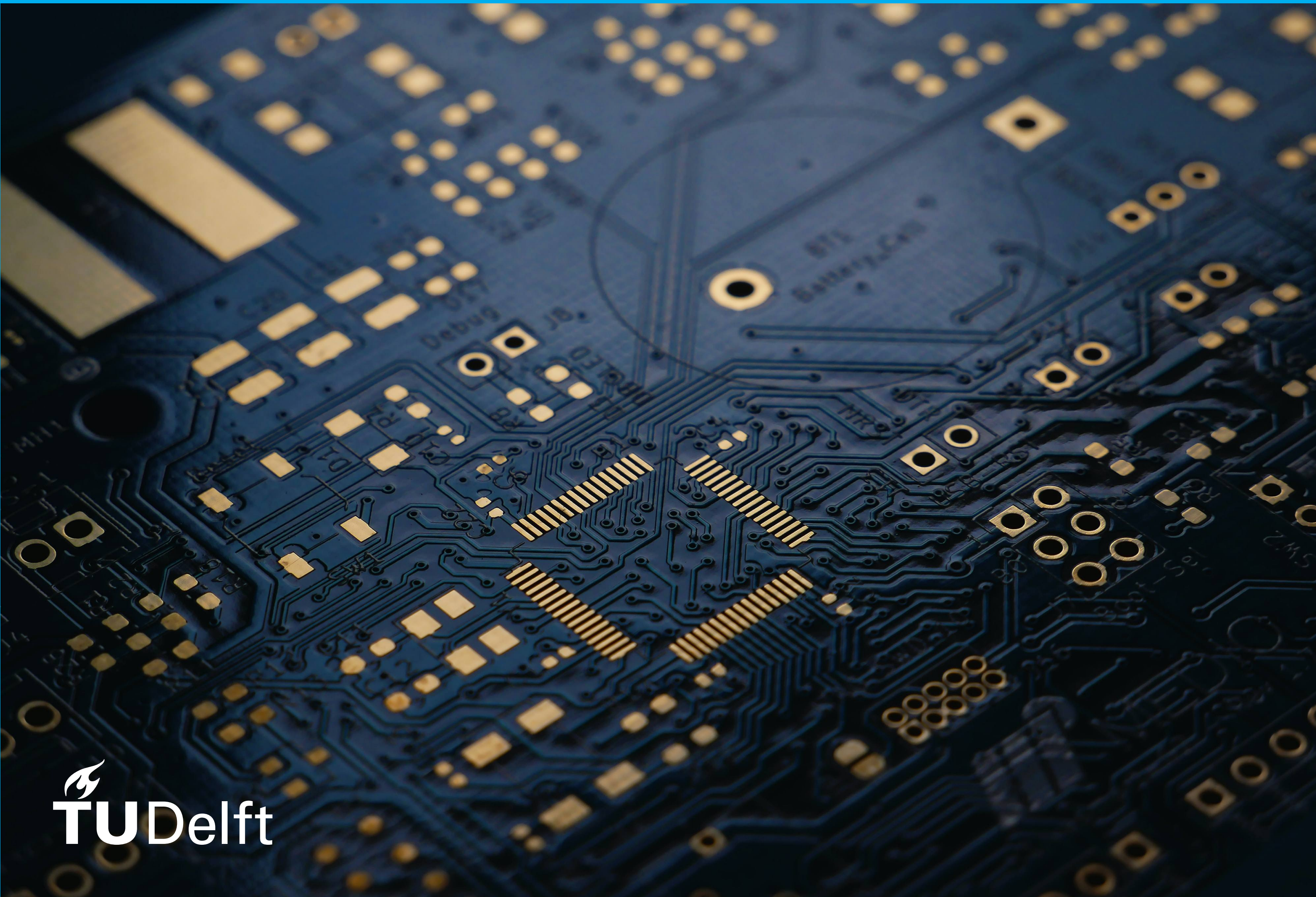
Design and Verification of LUPIn

A Platform for Hardware Attacks on Encrypted USB Drives

R.R. van Wijk



TU Delft

Design and Verification of LUPIn

a Platform for Hardware Attacks on Encrypted USB Drives

by

R.R. van Wijk

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on May 1, 2023 at 14:00.

Student number:	4261569	
Project duration:	March 1, 2021 – May 1, 2023	
Thesis committee:	Dr. ir. T.G.R.M van Leuken,	TU Delft
	Dr. ir. M. Taouil,	TU Delft
	C. Klaver,	NFI

Abstract

Forensics is the art of gathering evidence, which for electronics amounts to accurately recovering data. Often, this can only be archived with state-of-the-art hacking techniques. However, replicating state-of-the-art research in hardware security can be difficult, due to the large number of components and connections. To counter this, a custom Printed Circuit Board (PCB) is presented, that aids with hardware attacks, and allows them to be executed in a reliable and reproducible way. The PCB is targeted specifically towards hardware encrypted USB drives, and provides accessible ways to break out and interact with the target's electrical components.

In the first part of this thesis, the design and fabrication of the platform, called LUPIn, short for Lawful Unlocking of PIN-protected USB drives, is established. The design integrates commonly-used components and functions, making it suitable for a wide range of different attacks and devices. It also incorporates robust and traceable connections to the target. In the second part, LUPIn is verified by implementing it in a real attack. The target is a PIN-protected USB drive, which contains an IC performing key derivation. Since the debug port is not fully secured, a technique called Cold-Boot Stepping is used. This method is specifically designed to circumvent partially disabled debug ports. To analyse the gathered data, it first must be filtered. This filtering is done using a graph-based algorithm. In one crytographic function, an input parameter is used twice with different XOR masks. By analyzing all the filtered data, it is possible to find masked values, and use those to recover the original input value.

Concluding, a hardware tooling PCB (LUPIn) is successfully designed, assembled and tested. It proves to be a reliable platform for performing hardware attacks against encrypted USB drives. It makes development of hardware attacks simpler and less time-consuming. In the validation of LUPIN, a real-life USB drive is successfully attacked. Thousands of RAM snapshots are collected and an algorithm is developed to filter this data. A single variable can be extracted, but it ultimately proved insufficient to fully crack the target.

Contents

Acronyms

AES	Advanced Encryption Standard
BOR	Brown-Out Reset
CBS	Cold-Boot Stepping
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CVMP	Cryptographic Module Validation Program
DEK	Data-Encrypting Key
EMFI	Electromagnetic Fault Injection
FIFO	first in first out
FIPS	Federal Information Processing Standard
GPIO	General Purpose Input/Output
HF	High frequency
IC	Integrated Circuit
IP	Intellectual Property
KEK	Key-Encrypting Key
MCU	microcontroller (short for Microcontroller Unit)
MST	Minimum Spanning Tree
NIST	National Institute of Standards and Technology
PCB	Printed Circuit Board
PIN	Personal Identification Number
PIO	Programmable Input/Output
PWM	Pulse-Width Modulation
RAM	Random Access Memory
RDP	Read-out Protection
REPL	Read-Evaluate-Print-Loop
STM	STMicroelectronics
SWD	Serial Wire Debug

Acknowledgments

During the course of my research, I have been fortunate to receive support from numerous people. Therefore I want to express my gratitude in this section to everyone that has helped me throughout the past year.

First, I want to thank Dr. Ir. M. Taouil (TU Delft, QCE) and C. Klaver (NFI) for their guidance throughout the entire span of the research. This research would not have been possible without their invaluable help during our weekly meetings.

Next, I would like to thank everyone from the NFI for creating such a pleasant work environment and for all the nice chats at the coffee machine.

And lastly, I want to thank my family and friends for their love and support. They were always there whenever I needed to take my mind off work.

1 Introduction

1.1. Motivation

Security is an age-old problem. Centuries ago only locks were used to guard secrets, while in modern times we are shifting to passwords, private keys and security keys. Similar to the advent of lockpicking, hacking revolves around extracting those secrets from digital locks.

For digital systems there are two aspects that can be hacked: hardware and software. While manufacturers attempt to make their devices secure, white-to-black hat hackers try to break down those defenses, creating an endless game of cat-and-mouse. This results in companies trying to patch exploits as fast as possible, while hackers and researchers are continuously looking for possible security faults. Another involved group are the forensic researchers, which are caught between the other parties. Their goal is gather evidence during investigations. Forensics is the art of recovering data accurately, with the help of either hacking techniques or manufacturers themselves. All three of the groups benefit from cutting edge tools to facilitate development of cutting edge exploits or countermeasures.

Currently, many tools, both home-made and professional, can be used to perform attacks against Integrated Circuits (ICs), like glitching [1] or side-channel attacks[2]. However, building a setup with these tools leads to multiple inefficiencies. In the first place, the lack of standardization leads to wasted time and effort by reinventing the wheel when building a setup. This means that more effort is required to get to already available features and/or performance. Moreover, hardware security research requires multiple tools to be connected together, e.g. an oscilloscope, a glitching tool, a microcontroller and the target. This results in complex setups that are difficult to troubleshoot or reproduce, limiting the possibilities of expanding on other people's research [3, Slide 10][4, Fig. 15].

In this thesis, a custom Printed Circuit Board (PCB) will be presented, that will aid with hardware attacks. The PCB will be targeted specifically towards encrypted USB drives, and will provide accessible ways to break out and interact with the target's electrical components. Additionally, we present a new method to perform an attack called Cold-Boot Stepping (CBS), introduced in [5].

The research outlined in this paper will contribute to efficient and reliable tooling in the field of hardware security. Furthermore, more insight will be gathered on the range of application of the CBS attack.

1.2. State of the Art

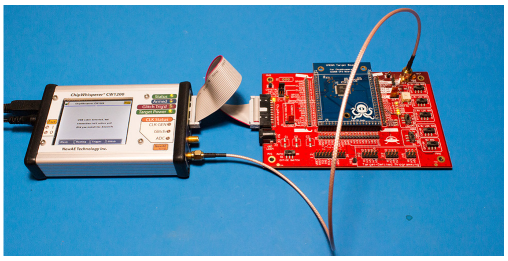
1.2.1. Hardware Attack

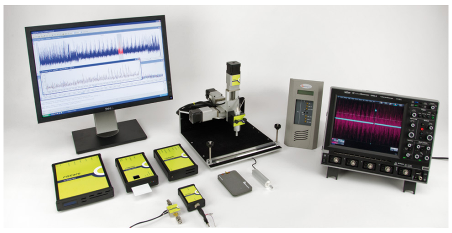
Nowadays, hardware attacks can have a high level of complexity in both setup and execution. An example is [6], where the attackers inject bus commands to the power regulator, in order to cause instability in a secure element IC. The setup consists of a PC, a microcontroller (MCU), the regulator, the secure chip and an oscilloscope. While there are only five active components, the complex interaction makes the setup non-trivial. The PC communicates with the MCU, which is wired to the regulator that controls to power to the secure IC. At the same time, the oscilloscope has to trigger at the right moment. The connections between the MCU and target are made using small wires that are soldered directly onto the regulator. These wires can break easily due their small size and contact area. As a result, the critical parts of the attack are performed over unreliable wires.

1.2.2. Hardware Attack Equipment

With the growing amount of research into hardware attacks, there has been equal development into the production of high-quality hardware security equipment. A global leader in security equipment is Riscure, which started developing professional tools in 2005 [7]. Current offerings include complete integrated setups for a variety of hardware attacks. One of their products is the Riscure Inspector, which can be configured for side-channel analysis or fault injection [8]. Both configuration consist of separate Riscure tools that need to be connected together. Since the tools are all made by the same company, connecting these together is simple. A downside of this system is that it is difficult to integrate a custom tool into the setup.

In 2014, Colin O'Flynn introduced the ChipWhisperer [9], an open-source and modular platform for performing power analysis attacks. Designed as part of his PhD for hardware security, it consists of an FPGA and ADC, both optimized for collecting power traces, which can be used for power analysis. The novelty comes from the fact that the system was low-cost, which made hardware research more approachable. However, the system is designed to be used in synchronous mode, where the target clock is generated by the ChipWhisperer. This design choice is marked by the limited sampling rate of 96 MS/s, which makes it unsuitable for asynchronous applications. O'Flynn has since made improved versions of the original ChipWhisperer, with better performance and improved glitching capabilities [10].


(a) ChipWhisperer Pro connected to the CW308-UFO board. [11]


(b) Riscure Inspector setup. [8]

Another creation of Colin O'Flynn is the CW308-UFO [12] board. It is a PCB with voltage regulators and connectors, designed to connect predesigned targets to the ChipWhisperer tools. It is designed to simplify power side-channel analysis and can be used in educational contexts. The board can be used in combination with an oscilloscope or a ChipWhisperer, where the latter case provides additional features. The UFO board contains a 60-pin connector for target chips. Fifteen different fitting target boards have been designed and are available for purchase. The UFO base provides a coax connector connected to a shunt resistor. This connector is used for monitoring the power consumption of the target. Additionally, a debug connector is present that is wired to the target. Since the system is exclusively designed for side-channel attacks, it does not provide any support for glitching attacks.

1.2.3. Hacking Encrypted USB Drives

USB drives can get lost or stolen, leading to data leaks. The IronKey corporation was one of the first to develop a transparently encrypted USB drive. It was created after a grant from the United States' Department of Homeland Security, in order to securely store top secret information [13]. Since then, many similar devices has appeared on the market, both for consumers as for business use. Theoretically, such device should not be unlockable to third-parties that Researchers at Google have audited multiple encrypted USB drives for vulnerabilities [14]. The goal of the research was to verify FIPS 140-2[15]. This is a certification process that guarantees a certain level of security for cryptographic modules. The USB drives protect the data using varying techniques, either in software or hardware. With software, the device bundles a program that can unlock the drive. With hardware, there is an unlocking mechanism present on the physical device, e.g. a fingerprint scanner or a keypad. Vulnerabilities were classified according to impact and complexity. The researchers concluded that the FIPS certification while helpful for easier auditing, is not broad enough in scope to prevent -predict/in kaart brengen- novel attacks.

1.2.4. Bypassing Microcontroller Readout Protection

Microcontrollers often contain protection mechanisms against dumping the embedded flash storage. This is to prevent leaking firmware, keys or Intellectual Property (IP) from devices deployed in the field. Researchers have been finding methods to circumvent this security feature. In [5], three different techniques are discussed. One of them, Cold-Boot Stepping (CBS), is discussed in detail in Section 2.2.3. In [16], a bug was found on certain STMicroelectronics (STM) products that allow an attack to completely bypass this protection. STM products can be configured to that a debugger can be connected, but not read any data. However, it was found that a debugger can be used to trigger an exception on the target. The exception itself would then leak a byte of flash data.

1.3. Contributions

There are two objectives for this thesis. The first objective is to design a system that can be used to attack encrypted USB drives. It will provide a test platform that can be used to easily connect peripherals to the target. This fills the current needs of advanced hardware security researchers. The second objective of this thesis is to verify the platform by attacking a off-the-shelf consumer encrypted USB drive. This device will be subject to a number of predetermined attacks. The main contributions of this thesis are:

- **New platform for performing security research against encrypted USB drives**: The platform is designed specifically for USB drives, allowing researchers to rapidly get started. It consists of two PCBs, a main PCB that is designed to fuse with the target and a secondary PCB with a programmable microcontroller for control tasks. It utilizes the UFO board [12] as base. Both boards contain components commonly used for security research, and improve the ability to analyze and interact with USB drives.
- **Implementation of the Cold-Boot Stepping attack for wider range of targets**: The Cold-Boot Stepping attack was introduced in [5]. It was used to bypass the internal flash storage protection in STM microcontrollers, but only applied to a target with known firmware. Additionally, the timing was performed from the reset condition, which is the most accurate and simple. In this research, the attack is extended to work on targets with unknown firmware, where the timing has to be measured from user input.
- **Novel algorithm for filtering memory samples gathered with Cold-Boot Stepping**: The resulting data from the CBS attack proved to have unrealistic properties: samples differ in significantly more bytes than possible given the time window. An algorithm was created to filter the data using hamming distance estimates.

1.4. Outline

There are three parts of this thesis. The first part describes background theory that is applied in the rest this thesis. In the second part, we introduce LUPIn, a platform for performing reliable hardware attacks against hardware encrypted USB sticks. The last part is about verifying LUPIn with an actual attack against an encrypted USB drive. For the attack, the Cold-Boot Stepping (CBS) attack is adjusted for our purpose.

2 Theory

2.1. Hardware Encrypted USB Drives

2.1.1. Definition

A hardware encrypted USB drive is a special type of USB store device that stores the user data encrypted on the internal flash storage. It can only be unlocked using input on the device itself, e.g. a numeric keypad or a fingerprint scanner. Figure 2.1 contains an overview of such a device. The device does not require any custom software on the host PC in order to unlock and use the drive: when unlocked, it appears as a regular USB mass storage device. An example of the keypad for a hardware encrypted USB drive can be seen in Figure 2.2. This device uses a 7 to 15 digit Personal Identification Number (PIN) to secure the contents.

2.1.2. Physical

In [18], many consumer hardware encrypted USB devices are disassembled. The findings are used to form a generic image of a such a device. Regular unencrypted USB drives are limited in dimensions by the USB port, the interface chip and flash storage. Since the size of transistors has shrunk considerably in the past decades, this has lead to USB drives that are barely larger than the USB connector itself [19] On the other hand, encrypted USB drives are larger, due to additional components that are needed. In order for the device to unlock, a key has to be entered on the device itself. This requires a keypad of over 10 buttons. Some devices incorporate a battery so the PIN can be entered before

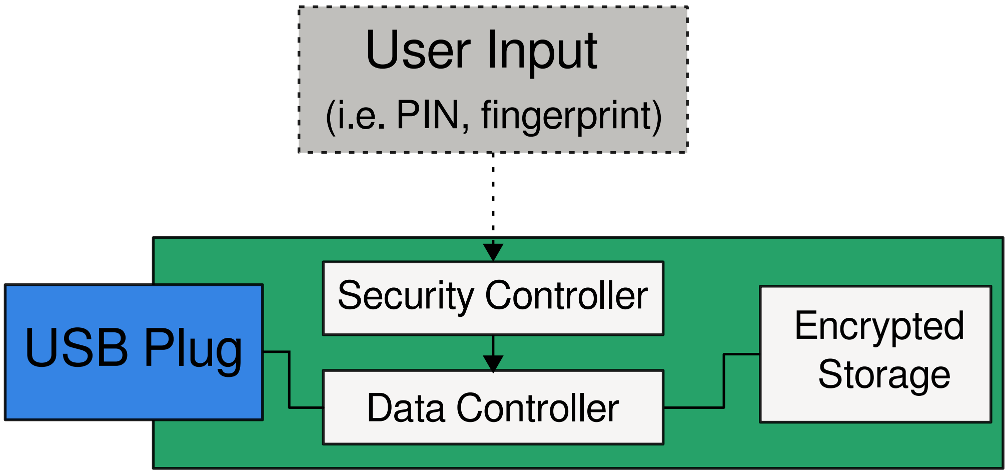


Figure 2.1: Logical overview an encrypted USB drive. The data controller de- and encrypts the data on the fly with a key from the security controller. Both controllers might be separate chips or integrated into one. Adapted from [14, Slide 8]

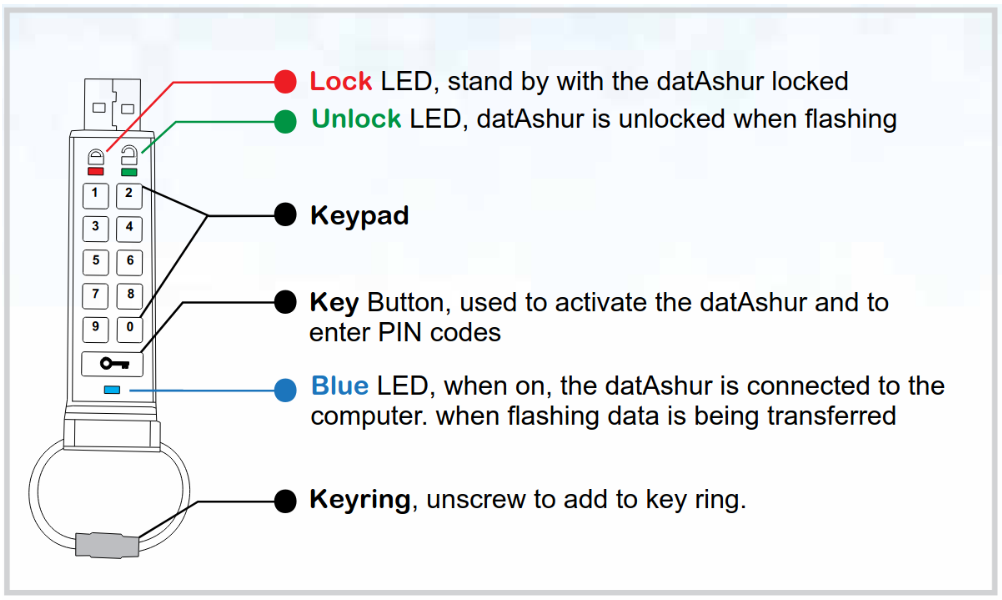

Lock LED, stand by with the datAshur locked
Unlock LED, datAshur is unlocked when flashing
Keypad
Key Button, used to activate the datAshur and to enter PIN codes
Blue LED, when on, the datAshur is connected to the computer. when flashing data is being transferred
Keyring, unscrew to add to key ring.
1
2
3
4
5
6
7
8
9
0

Figure 2.2: Button interface of the iStorage datAshure USB drive[17]

the device is inserted into the USB port. From [18], we can conclude that all products use the same bar shape, where the device is only marginally wider and thicker than the USB-A port.

2.1.3. User Interaction

The standard user flow for unlocking an hardware encrypted USB drive is as follows: The user turns on the device and enters the PIN. The device now checks the validity of the PIN, and unlocks the flash. It can now be inserted into any PC, where it will appear as standard USB storage.

Devices use different ways to provide feedback, e.g. to indicate a correct PIN, or when the PIN can be changed. Some devices use LEDs, while others use a monochrome screen.

2.1.4. Hardware

Keyboard Scan Matrix

The keys to enter the PIN are commonly designed using a row/column keyboard scan matrix, in order to save GPIO pins. For a $n \times m$ matrix, there are n output rows, and m input columns. While scanning for keypresses, the controller brings one row to a high voltage, while leaving the other low. In the next step, it reads all column states. If any column has the same voltage as the row, the two must be shorted by a pressed key. The cycle then repeats with the next row.

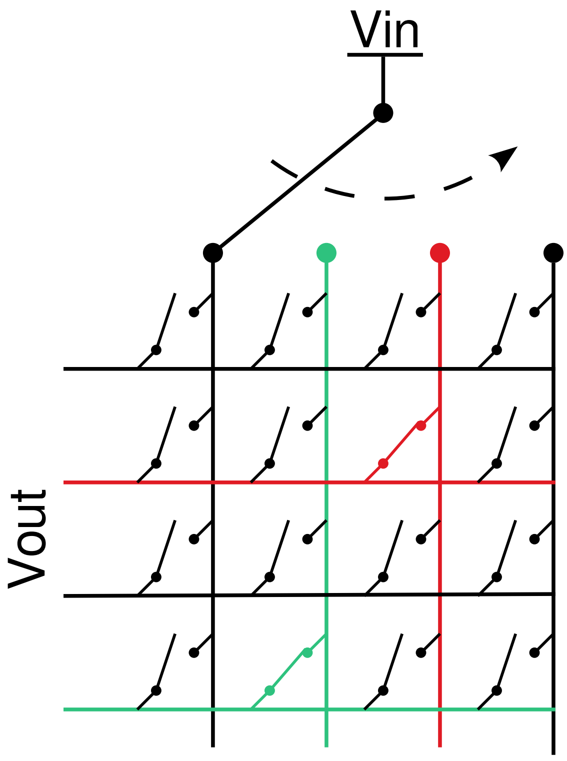


Figure 2.3: Keyboard matrix scanning explained. The red and green keys are pressed and short the corresponding lines. V_{in} is cycled through the columns, while the rows at V_{out} are read.

Storage

As found in [18] the devices use regular flash chips, from large manufacturers like Samsung or Micron. The data is written in encrypted form to the storage Integrated Circuits (ICs).

USB Controller

When the device is unlocked, the stored data has to be transferred from or to flash, while performing transparent de-/encryption. The USB controller handles these tasks, by connecting to both USB and flash storage. A third channel is needed to gather the encryption keys, which is commonly connected to the crypto controller. The Phison PS2251 variants are commonly used[18]. Unfortunately, no public documentation is available for this component.

Crypto Controller

The crypto controller is responsible for performing the PIN check and unlocking the device. It can be an off-the-shelf microcontroller, or a specialized secure element. Since the device performs the cryptographic operations to transform the PIN into a secret key, it is the most interesting target from a security perspective.

2.1.5. Software

Cryptography

From a high level perspective, the crypto controller has one input and one output: the user enters the secret key, passphrase or PIN, and the controller outputs the Data-Encrypting Key (DEK). A naive way of doing this operation is by storing both the secret and DEK in flash. The unlocking process checks whether the secret matches the one in flash, and copies the DEK to the output. This is insecure since both the PIN and DEK can be read by attackers with physical access to the controller. If the DEK can be derived from the PIN by performing one-way transformations, there would be no need to store the PIN. For instance, securely hashing the PIN results in a fixed-size, high entropy byte buffer, suitable for a DEK. To avoid storage of the DEK in flash, we encrypt it using a block cipher (e.g. Advanced Encryption Standard (AES)), using the transformed PIN as key. This intermediate key is called the Key-Encrypting Key (KEK). Since the block cipher itself has no notion of correct or incorrect keys, additional data can be added that will be used to verify the authenticity. In this scheme only encrypted data needs to be stored, which should be secure.

Attack Aim

If attacking a hardware encrypted USB drive, the goal is to recover the data. Either the PIN or DEK is required for this. There might be several prerequisites to get to the DEK, like a salt or AES data.

2.2. Hardware Attacks

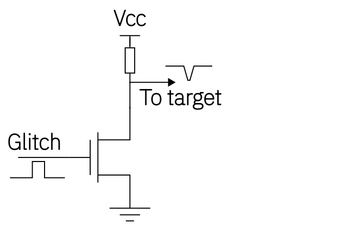
2.2.1. Fault Injection

With fault injection (glitching), external components modify internal states of a chip, in an effort to change the IC behavior. For instance, if the power supply or clock circuitry to the chip is manipulated, the target might misbehave in a way beneficial to the attack.

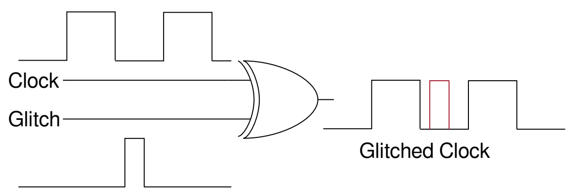
Processors work by executing instructions. While programmers typically assume integrated circuits work digital, these are still analog components. If the voltage to the processor is lowered while executing an instruction, there is a chance the transistors will not fully switch, thus not executing the instruction as intended. There are three main methods of glitching: the voltage glitching, clock glitching, or glitching using electromagnetic waves. Each will be briefly discussed in the following sections.

Voltage Glitching

Voltage glitching is one of the simpler types of attacks. By dropping the voltage for a brief period, transistor might not operate properly during a clock cycle [20]. This can lead to e.g. skipped instruction or branch instructions that invert the condition. A very simple implementation is by using a single MOSFET that is connected between the supply voltage and ground. Enabling this transistor for a brief period will cause the voltage to dip. This method is called crowbar glitching. More advanced methods exist, where the voltage will follow a arbitrary waveform[21]. These methods require more sophisticated equipment to execute. Fault injection attacks can be difficult to reproduce, since a small change in the setup can change capacitance and inductance characteristics, which in turn change the shape of the waveform.



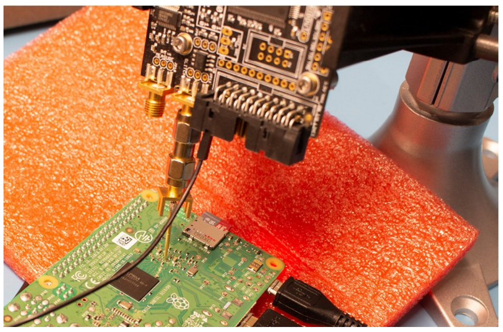
(a) Example implementation of crowbar glitching

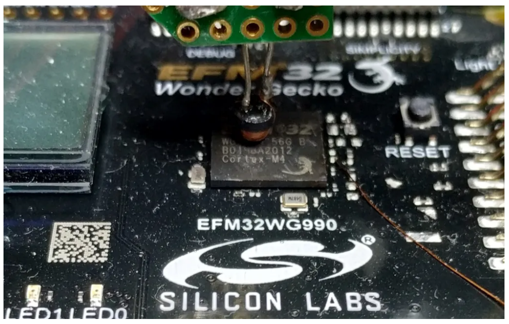


(b) Example implementation of clock glitching

Clock Glitching

As opposed to voltage glitching, with clock glitching the clock signal of the device is disturbed. Clocked ICs are designed for some range frequencies. By introducing intermittent additional clock cycles, the frequency is briefly changed beyond the standard operating limits. This can cause e.g. flip-flops to not fully store the input state, or transistors to never fully switch on. The major downside to this method is that it is not very applicable on ICs that have an internal oscillator. It only works when the clock signal is accessible.


(a) A setup for voltage glitching a Raspberry Pi 3 (bottom) using a ChipWhisperer (top). Image by [22]


(b) Electromagnetic Fault Injection (EMFI) setup by [1], build to bypass a debug lock

Electromagnetic Fault Injection

In Electromagnetic Fault Injection (EMFI), a localized electromagnetic field is applied to the target [23], [24], using a small antenna. The energy is dissipated in the chip, which can cause transistors to switch or malfunction. There are two main techniques: using a short and powerful pulse or a continuous harmonic wave. The strategy depends on the target of the attack. EMFI is non-invasive, which makes the attack quick to set up. On the other hand, the required hardware is complex, as it requires generating high-voltage and specialized coils. Additionally, the solution space is large with six variables: the antenna's x, y and z coordinates next to timing offset, glitch power and glitch length.

Other types
Glitching can be performed using other energy sources, e.g. using lasers or temperature. However, these types of attacks require very invasive setups and a are not considered for the subject of this thesis.

2.2.2. Side-channel analysis

A side-channel attack refers to an attack were the secret data can be recovered by measuring unintended external variables. This includes execution time [25], cache presence [26] or audio waves emitted by capacitors [27]. In this paper, only power analysis-based side-channel attacks are discussed. With these types of attacks, power or EM traces are captured while the target is performing sensitive tasks. The idea is that the data being is processed can be correlated with the power consumption or EM emissions [28]. It was discovered back in the second World War, when Bell Telephone engineers found that the relays in encrypted teletypes would radiate strong electromagnetic fields when printing a letter [29]. The fields could be read over long ranges, and used to recover the plaintext. To perform a side-channel power analysis, one needs to collect samples of the target's power consumption. This can be done by measuring the current using a oscilloscope or the electromagnetic radiation using an antenna. If the system is vulnerable, the resulting traces will reveal some sort of correlation between in the secret data and the measured power. Current high-end processors run at frequencies above 100 MHz. The traces need capture multiple samples during a single clock cycle, requiring in high-bandwidth ADCs. Oscilloscopes can be used, or specialized equipment. Power analysis attacks are difficult to counter as it exploits an inherit property of switching transistors.

2.2.3. Cold-Boot Stepping

Background
Cold-Boot Stepping (CBS) was developed to bypass the Read-out Protection (RDP) mechanism present in the STM32F0 series of microcontrollers (MCUs) made by STMicroelectronics (STM) [5][30, Sec. 3.3]. This mechanism makes the internal flash unreadable, in order to prevent reading and reverse-engineering the firmware. There are three levels of RDP: at level 0, no protection is enabled, and the debugging port functions fully. At level 1, debuggers can still interact with the microcontroller, but reading the internal storage will throw an exception. Additionally, the Central Processing Unit (CPU) core is halted when a debugger is connected and will only restart after the whole MCU is reset. At RDP level 2, the debugging port is disabled, meaning debugger tools cannot connect to the MCU at all. This attack targets RDP level 1. Note that the RDP subsystem is present in all ARM STM microcontrollers.

CBS requires a accurate way of timing the microcontroller time spend executing the firmware, which is archived by used an external reset. The STM32F0 contains a nRST line, which under regular operation is pulled high by the system itself. Figure 2.6 shows the internal reset circuit for the STM32F0 series. If the line is kept at a low voltage, the system will be kept in a reset state, where processor is stalled.

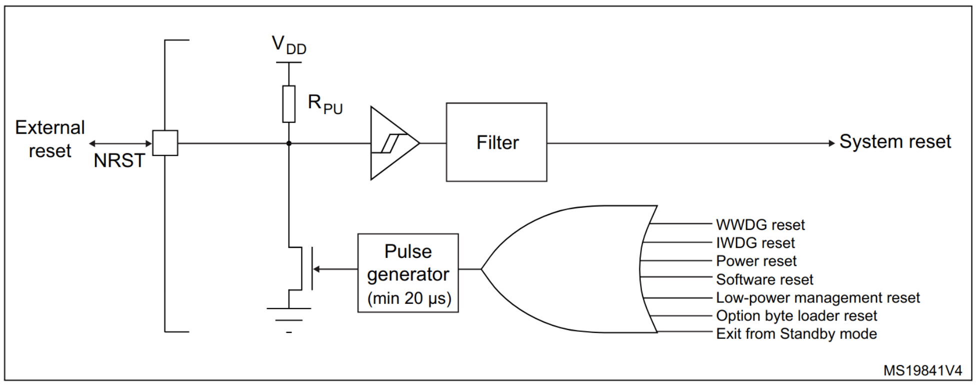


Figure 2.6: Simplified diagram of the reset circuit present in the STM32F0 series MCUs [30, Fig. 9]

It is possible connect a debugger to the system while under reset. Like most ARM microcontrollers, the Serial Wire Debug (SWD) protocol is used [31, Sec. 5.2]. The debug subsystem will be enabled after a handshake is send to the target, but it requires the system to get out of reset before any data can be read.

Attack

At RDP level 1, only Random Access Memory (RAM) and registers can be read using a debugger, not the internal flash storage. However, with certain operations, contents of the flash might leak to RAM. As an example, some firmware check the integrity of the flash with a checksum. In the paper, a vulnerable application was crafted that performed a Cyclic Redundancy Check (CRC) check over the entire flash region, with the goal of reconstructing the flash data with the intermediate CRC values.
The challenge with this approach is that the processor will halt itself after a debugger has been attached, with no way to continue executing. In order to monitor the system

1. Reset the system: Using both power and reset line, the target is reset into the startup state
2. Run the system for some time t: Put nRST into a high state
3. Stop the system after t has expired: Pull nRST to 0V
4. Dump the memory contents with a debugger
5. Increase t by stepsize T
6. Go to step 1

The result of this operation is a large number of RAM dumps that describe the system state from $t = 0$ to $t = n \cdot T$, where n is the number of steps. This gives an image of the RAM contents while the system is executing.

In [5], a proof of concept is shown with a specially prepared target. The target firmware does a CRC checksum of the flash memory upon boot. With CBS, the addresses used by the checksum algorithm could be monitored, allowing the attackers to determine the input.

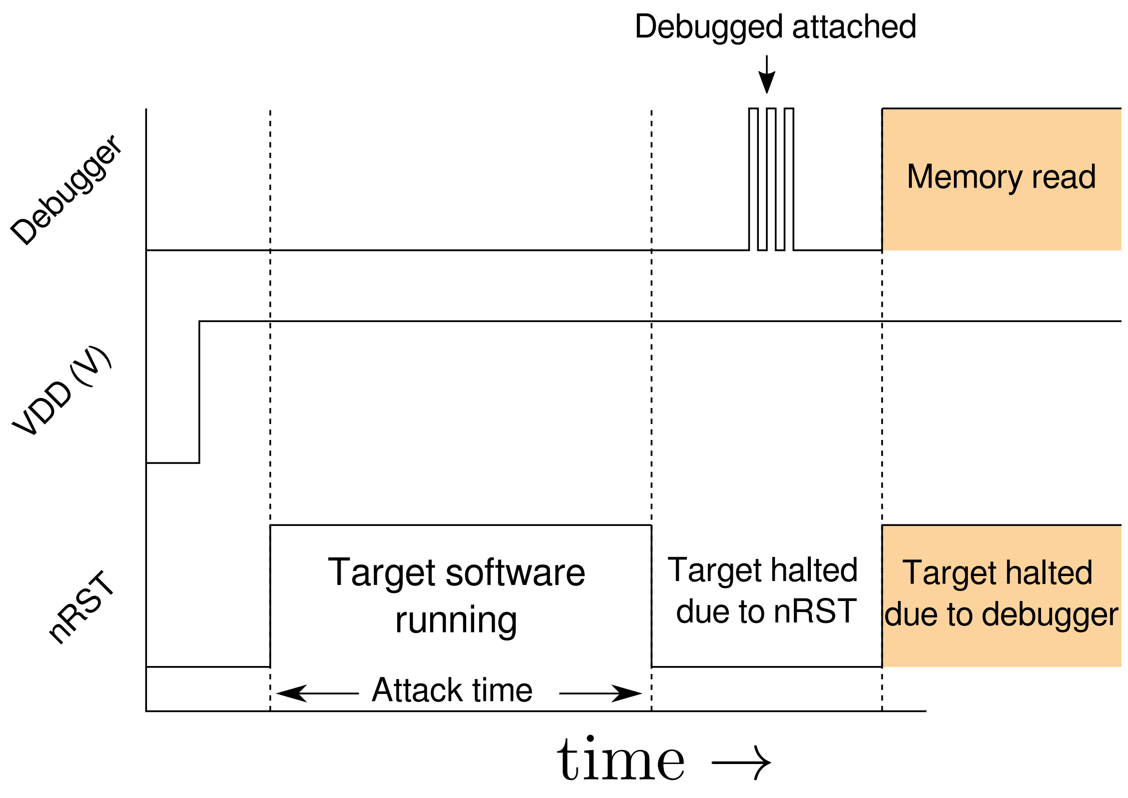


Figure 2.7: Timeline of the logic signals in Cold-Boot Stepping

2.3. Countermeasures

Several countermeasures have been developed to thwart possible hackers. Note that all countermeasures can only decrease the chance of a successful attack, not prevent them. However, by increasing the amount

2.3.1. Redundant Checks

Fault injection attacks can represent a major vulnerability to secure systems, as software features can be bypassed [32]. To prevent successful glitching attacks, additional checks can be added for detection [33], [34]. For instance, consider the following code:

```
if ( sensitive_check() == 0 ){
  if ( sensitive_check() != 0 ){
    // Should not be reachable under normal circumstances
    panic();
  }
}
```

If an attacker manages to bypass the first check, it would be detected by a second check. Note that the second check can be placed at arbitrary locations in the source code, e.g. before exiting the function or in the middle of a sensitive operation. While it can still be bypassed by glitching the second check, it slows down attackers by increasing the search space.

2.3.2. Debug Subsystem Protection

Debugging ports on microcontrollers are essential to their software development. However, since the debugging subsystem allows essentially unlimited access to every component, it can also pose a security risk for deployed devices. Certain Intellectual Property (IP) or secrets can be hidden in the firmware. With a debug port, this can be read using any compatible debugger. For this reason many MCU manufacturers provide options to limit or disable the debug port.

2.3.3. Brown-Out Reset

The Brown-Out Reset (BOR) is a piece of hardware that monitors the supply voltage of a MCU. When the voltage drops below a certain threshold, the whole system is reset. This can prevent certain types of glitching attacks.

2.3.4. Epoxy & Coatings

In order to hinder access to components of the Printed Circuit Board (PCB), sometimes an epoxy (or similar) coating is applied to the PCB. This non-conductive coating does not influence the behavior, but hides the traces and components markings. It also prevents measuring voltages with probes. There are ways to remove the coating with chemical or mechanical methods.

2.4. NIST FIPS 140-2

National Institute of Standards and Technology (NIST) is an organization founded in 1901 to advance the US industry competitiveness. At the time, the US industry lacked measurement technology, which resulted in lower-quality products compared to economic rivals such as the United Kingdom and Germany [35]. NIST served as a laboratory and standard organization for the US. Today, they still develop standards for both industry and government use. These standards are used by US government for various tasks, such as software quality assurance.

NIST publishes a set of software related standards called Federal Information Processing Standard (FIPS) [36]. It covers multiple areas, including cybersecurity, interopability and 3D graphics. All non-military government computer systems are expected to comply with these standards. FIPS-140-2 [15] is a set of requirements for cryptographic modules. A cryptographic module is defined as a hardware and/or software system that performs cryptographic functions, i.e. key generation, authentication or encryption. While the standards cover both physical and software security, we will only focus on the latter in the rest of this document.

One aspect of compliance with FIPS-140-2 is the use of exclusively FIPS-approved cryptographic algorithms, which are listed in [37]. Since these algorithms are extensive tested and verified by NIST, this gives reasonable assurance for certified devices.

Another important aspect is a specific way of documenting all cryptographic operations performed by the system. This includes a list of sensitive operations, a block diagram of the system components and the user types.

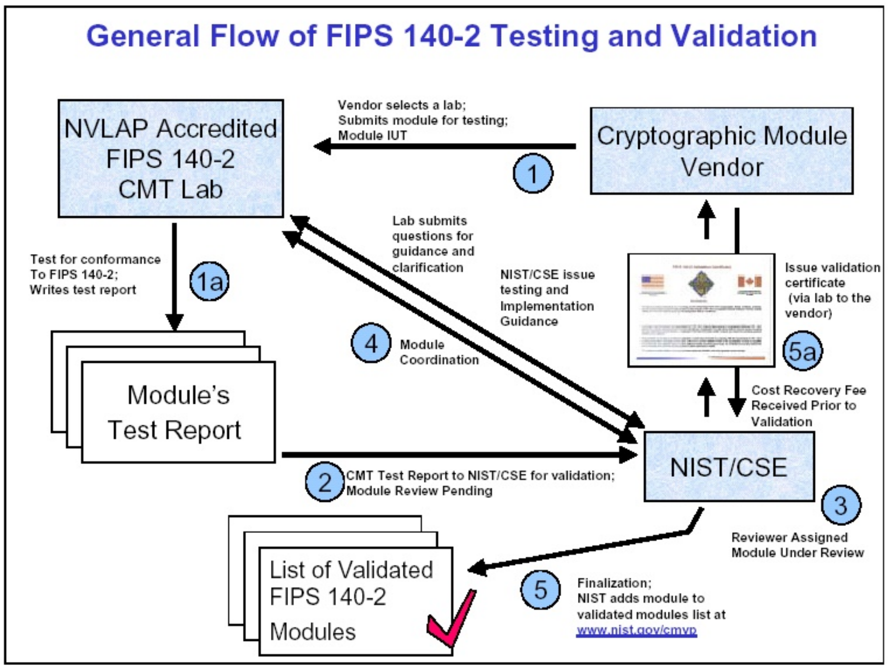
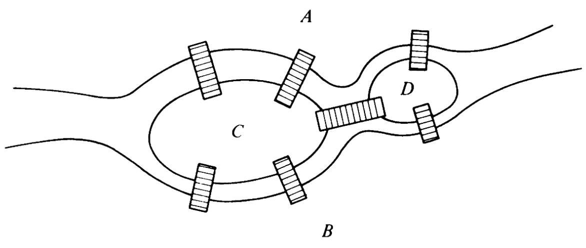

General Flow of FIPS 140-2 Testing and Validation
NVLAP Accredited FIPS 140-2 CMT Lab
Vendor selects a lab; Submits module for testing; Module IUT
1
Cryptographic Module Vendor
Test for conformance To FIPS 140-2; Writes test report
1a
Lab submits questions for guidance and clarification
NIST/CSE issue testing and Implementation Guidance
Issue validation certificate (via lab to the vendor)
5a
4
Module Coordination
Module's Test Report
Cost Recovery Fee Received Prior to Validation
NIST/CSE
2
CMT Test Report to NIST/CSE for validation; Module Review Pending
3
Reviewer Assigned Module Under Review
List of Validated FIPS 140-2 Modules
5
Finalization; NIST adds module to validated modules list at www.nist.gov/cmvp

Figure 2.8: FIPS 140-2 validation procedure. Image by [38]

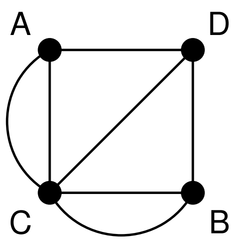
Validation of products is done by NIST, under the Cryptographic Module Validation Program (CVMP). All validations and manufacturer documentations are publicly accessible on the NIST website. The full process starts by submitting the module to certified lab, which write a report. That report is reviewed by NIST. Any issues are discussed between the lab and NIST. When the module is approved, the report is published on the website.

2.5. Graph Theory

Euler invented graph theory when walking along the scenery of Köningsberg. He saw seven bridges connecting four islands, identical to Figure 2.9a. He wondered whether it was possible to visit each island and return to the starting position by crossing each bridge exactly once. Euler visualized the problem in a graph, which is drawn in Figure 2.9b. Using this model, he could prove that the problem was unsolvable [39, Ch. 1].



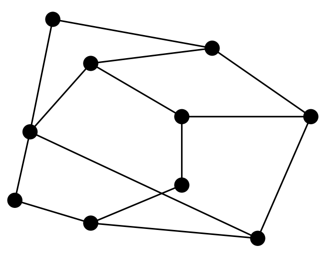
(a) The bridges in Köningsberg [39, Fig. 1.4]

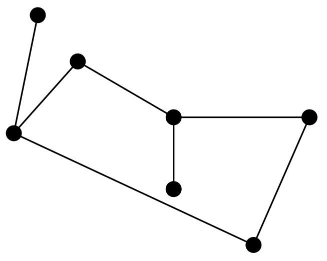


(b) The same situation represented as a graph

Graph theory is used in many different fields. It can be used to transform transportation networks[40], page links or electrical circuits[41] into graphs. These graphs can be analyzed mathematically, which in turn leads to more insight about the problem.

Vertices and edges are the main components of graph theory. A graph $G = (V, E)$ is the combination of a set of vertices V and a set of edges E. A vertex is a point on the graph with zero or more links to other vertices. Such a link is called an edge, notated as (p, q), meaning that it connects vertices p with q, where both p and q are from V. G contains zero or more subgraphs, which are graphs with a selection of the edges E and vertices V.

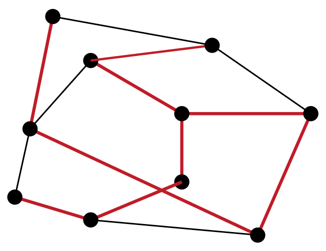

(a) Example of a graph

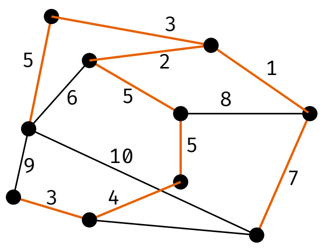

(b) A possible subgraph of the graph in (a)

A weighted graph is graph where each edge e_i has a corresponding weight $w(e_i)$. This can be used to model distances, costs or delays of real systems. As example, consider a graph of the multiple cities where the weights indicate the distance between two cities. One could use this graph to calculate the total distance traveled when visiting multiple cities.

A tree is a special type of graph where all vertices are connected without loops, meaning that there is only one path between two vertices. A familiar example is a family tree. Each person is a vertex, connected to their parents and children via an edge.

A spanning tree of a graph G is a subgraph that visits all vertices of G once each without loops, resulting in a tree. Spanning trees are only possible to create in graphs where all vertices are reachable. An example is shown in Figure 2.11a. A graph can contain multiple spanning trees. The set of all spanning trees possible in a graph is called the spanning forest.


(a) Example spanning tree of the graph from 2.10a



(b) Minimum Spanning tree of the graph from Figure 2.10a with weights

For weighted graphs, one can find the Minimum Spanning Tree (MST), which is the spanning tree with the smallest total weight. The total weight is calculated by summing the weights of all edges. There can be multiple MSTs for a given graph.

There are multiple algorithms to find the MST. The two main algorithms are Kruskal and Prim. Kruskal works by listing all edges sorted by weight, and continuously selecting the smallest edges, without creating a loop. Prim's algorithm starts by initializing a new tree with a random vertex from the graph. In the second step the smallest edge with a vertex not in the tree is added. This step is repeated until all vertices are in the tree.

3 LUPIn

3.1. Motivation

Currently, there are two real options for attacking hardware are demo boards or a setup with a number of devices connected together using loose cables. Demo boards are used as a tool for educational and/or demonstration purposes, usually produced by research institutes or tool manufacturers to demonstrate their tools. Demo are easy to use due to their integrated nature: every needed component is present and properly connected on the board. Since these boards are commonly Printed Circuit Boards (PCBs), they are very robust. However, this approach is only really useful for a preset target, with known vulnerabilities. It is unfeasible for arbitrary targets. On the other hand, if the target is an unknown device with unknown weaknesses, one has to resort to connecting tools together using soldered cables. This is the only feasible way to archive the flexibility needed. Some examples of this kind of setup used in literature are [3], [6][14, Slide 86]. The main issue with this method is that is unreliable and cluttered. Multiple devices that interact with each other over unreliable connections is difficult to manage, which leads to faults that are difficult to pinpoint. A single broken wire can completely stop the setup from working, maybe even undetected. Another downside is that it can be difficult to replicate the setup, which is often the needed for research. This hinders the transfer of knowledge.

To improve on the situation, there are a number of improvements we can make. The first is that we use a PCB as basis for the attack platform, which gets us the most of the benefits that also apply to demo boards. The PCB integrates components that are commonly used as loose components, such as switches and power supplies. Different targets can be attacked, but is focused on encrypted USB sticks. With robust connections the system gets more reliable. Since the PCB provides a common building block, it also eases reproducibility of the entire attack and target.

3.2. Requirements

Three high-level requirements have been defined. First of all, LUPIn needs to be flexible, meaning that it should be able to perform multiple types of attacks on all common targets. Secondly, it should be robust. This entails that the setup should keep working over extended period of time without maintenance. Last import aspect is reproducibility. The same attack might need to be performed on different targets, or different attacks might need to be performed on the same target. Both these cases involve rebuilding the same setup with new components.

In the following sections, each high-level requirement is broken down into practical low-level requirements.

3.2.1. Target USB Drives

Based on Section 2.1, requirements are determined for LUPIn, to ensure the widest target range possible.

Physical
Devices are generally within $20 \times 70 \times 10$mm. LUPIn should be able to attack all smaller devices. Additionally, since there are sometimes multiple PCBs present, we require that the target can mounted on different orientations.

User Input
LUPIn should be able to simulate keypresses using electronic switches. The switches should be configured in a row/column layout to match with the standard keyscanning method.

User Output
LEDs are controlled using Pulse-Width Modulation (PWM), LUPIn should provide a way to read those values. Furthermore, it should be possible to read and write from common serial busses, including I2C and SPI.

Power
LUPIn should be able to provide power to the target besides the 5V USB power.

3.2.2. Hardware Attacks

A number of hardware attacks are named in 2.2. Different attacks require different types of connections and tools. Three main attacks are used as main focus: Cold-Boot stepping, side-channel analysis, glitching. Additionally, we consider the communication with the target over serial busses as a requirement for many hardware attacks.

Hardware tool	CBS	Glitching	Side-channel	Bus comm.
Debugger	x			x
Power/Reset toggle	x	x	x	
Serial Bus interaction		x		x
Generic IO				
USB Communication				x
Coax connections		x	x	

Table 3.1: Tools required for each of the listed attacks

Cold-Boot Stepping
With Cold-Boot Stepping (CBS), three components are needed: An SWD programmer, a power switch and a controller that can toggle the nRST line. This controller needs to be precise in timing, in the same order of magnitude as the Central Processing Unit (CPU) core clock to be useful. In the original paper, a regular microcontroller programmed with assembly was used.

Glitching
For glitching, the standard cycle is performing a glitch, reading the result, and resetting. For the glitching itself it is difficult to integrate resources on the board, as even short wires can influence the success rate. For the other aspects of this attack, there a number of this required. First we need to be able to read the result. This can be a message send over UART or similar serial bus, or a pin voltage change. LUPIn needs to be able to read this. For the reset, the same requirements as in subsection 3.2.2 apply.

Side-Channel Attacks
In order to perform a side-channel attack, the target has to perform the requested operation, and the power trace has to be collected. This includes a number of automation steps similar to other attacks. For LUPIn, a coax or High frequency (HF) port is required in order to collect the high-fidelity data.

Bus Communication
Microcontrollers commonly use low-speed serial busses to communicate with external components. Examples of these interfaces are I2C, SPI or CANbus. LUPIn should be able to interface with those busses directly, or be able to easily hook up external devices.

Final Requirements
In table Table 3.1 all of the requirements are visible for each attack.

3.2.3. Robustness

In order to provide a reliable platform, all connections with target have to be robust. In this sense, robust is defined as that it keeps working, even after a period of time without maintenance. A number of pieces are necessary to provide this robustness. First, a rigid, semi-permanent cable connection system is needed for have reliable electrical connections. Additionally, all cable lengths should be kept as short as possible in order prevent any unwanted ringing effects. There should be some way to physically attach the target to the PCB, in order to prevent shock from damaging any cables.

3.2.4. Flexibility

The field of hardware security is constantly evolving, and new techniques are developed continuously. For LUPIn, this means that the design should be expandable with arbitrary new components. In practice, connectors should be used to allow for easy expansion. It should be possible for the user to override the builtin components where possible, as new devices can have better specifications needed for certain novel attacks

3.2.5. Interaction with Target

In order to have meaningful attack possibilities, it is required to have some form of interaction with the target. In this section, the required methods are listed.

Keypad Switches
Hardware protected encrypted USB drives use some form of on-device user input in order to unlock the device. It should be possible to emulate keypresses by shorting connections using switches. Devices commonly use a matrix to scan for user input.

Analog Input
Certain pins on the target may not output a digital signal, but an analog one. LUPIn should provide ADCs with isolation for this use-case.

General Purpose Input/Output
General Purpose Input/Output (GPIO) is needed to interface with certain control signals, such as the nRST lines of a microcontroller.

Serial Bus Interface
Modern microcontrollers often make use serial bus interfaces such as I2C or SPI to control external peripherals. Being able to intercept and/or inject data on these busses is required for probing the target.

Power Switches
Many attacks in subsection 3.2.2 require that the target can be reset completely. While it is often possible to use a dedicated reset pin, removing the target power will always work.

3.3. Design

3.3.1. High-level Components

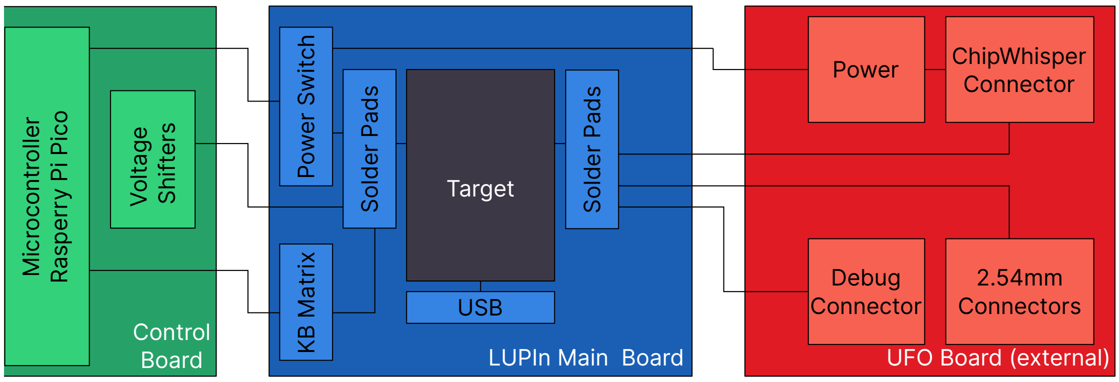


Figure 3.1: High-level overview of LUPIn

UFO Board

The ChipWhisperer CW308 UFO board is a modular PCB that is used as starting point for side-channel attacks. It was chosen because it is the only commercially available board that provides the connectors and form-factor for our particular use-case. It is accompanied by several smaller boards that function as targets. It integrates well with ChipWhisperer Lite, made by the same company. The intended way of using it is by installing one of the targets and use the ChipWisperer or an oscilloscope to record power traces. It has multiple connectors: The target connects via a 2.54mm-spaced 60-pin connector. Each target pin is exposed at least once on a separate connector, to allow for easy probing. Additionally, there are several special connectors: The 20-pin JTAG/SWD debug connector, a socket for placing a crystal oscillator and a coax connector. The 20-pin ChipWhisperer Lite connector is also present.

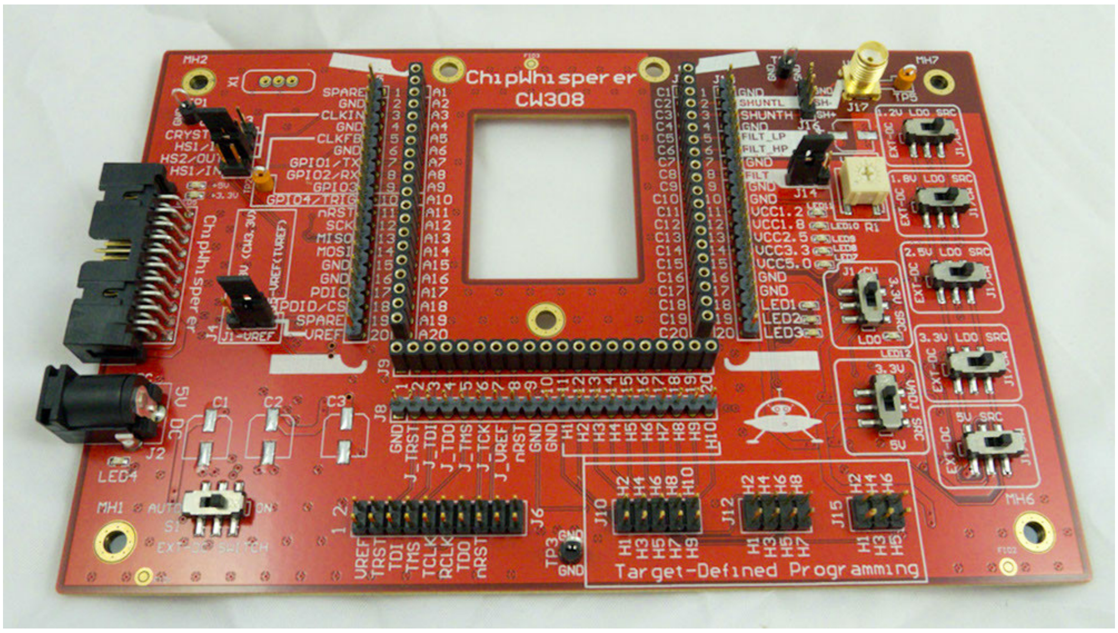


Figure 3.2: CW308 UFO board, picture by NewAE Technology, inc.

The board provides 5 power nets, of which 4 provide fixed voltages from 1.8V to 3.3V. One net has configurable voltage, using a potentiometer. It can be powered from a 5V supply, either from the ChipWhisperer connector or through a barrel jack connector. For user feedback, three LEDs are present that are connected to the target board. The UFO board is main component of the system, as it simplifies the design in several ways. The idea of interchangeable targets is very applicable here, as a low-cost PCB can be adapted to and fused with the target.

Main Board

For the main components of the system, the ChipWhisperer UFO board is leveraged. This board is designed to be used together with smaller target boards, for the purpose of demonstration. These smaller boards connect to the bigger board via a 60-pin connector, which consists of 3 20-pin 2.54mm headers. The main reasons for using the UFO board is that it is readily available, provides power with multiple voltages and has a plethora of connections. These connections are flexible: there are 10 pins dedicated to user specific applications. Other connections include: a oscilloscope probe connector, a standard 20-pin JTAG connector and an crystal oscillator. Furthermore, there is 20-pin connector to connect it directly with a ChipWhisperer Lite, a glitching device.

LUPIn

The LUPIn board itself connects into the CW UFO board. It is designed to hold one encrypted USB drive. The PCB has a “U”-shape. The board has an female USB-A port, for connecting the target. Two small “flaps” can be used to glue down the target, or can be removed completely if necessary. Furthermore, there a number of GPIO connectors, connections to the ChipWhisperer and two power pads. The latter is configurable using a two small headers, and the Raspberry Pi Pico microcontroller that is part of the system.

Raspberry Pi Pico Microcontroller

In order to control all peripherals on LUPIn and to be able to implement arbitrary functions, a microcontroller is present. The LUPIn PCB uses a 26-pin connector that is used to connect to another PCB containing the microcontroller. This small PCB contains some LEDs, level shifters and the microcontroller itself, the Raspberry Pi Pico. This specific MCU was chosen mainly for its Programmable Input/Output (PIO) feature. This subsystem consists of 8 tiny CPU cores, which run at the same speed as the main CPU. The cores have a very simple instruction set that can be used to perform IO, interact with first in first out (FIFO) buffers and trigger interrupts. With these cores, IO protocols can be implemented at high speed, with predictable timing, without consuming CPU time.

3.3.2. LUPIn Components

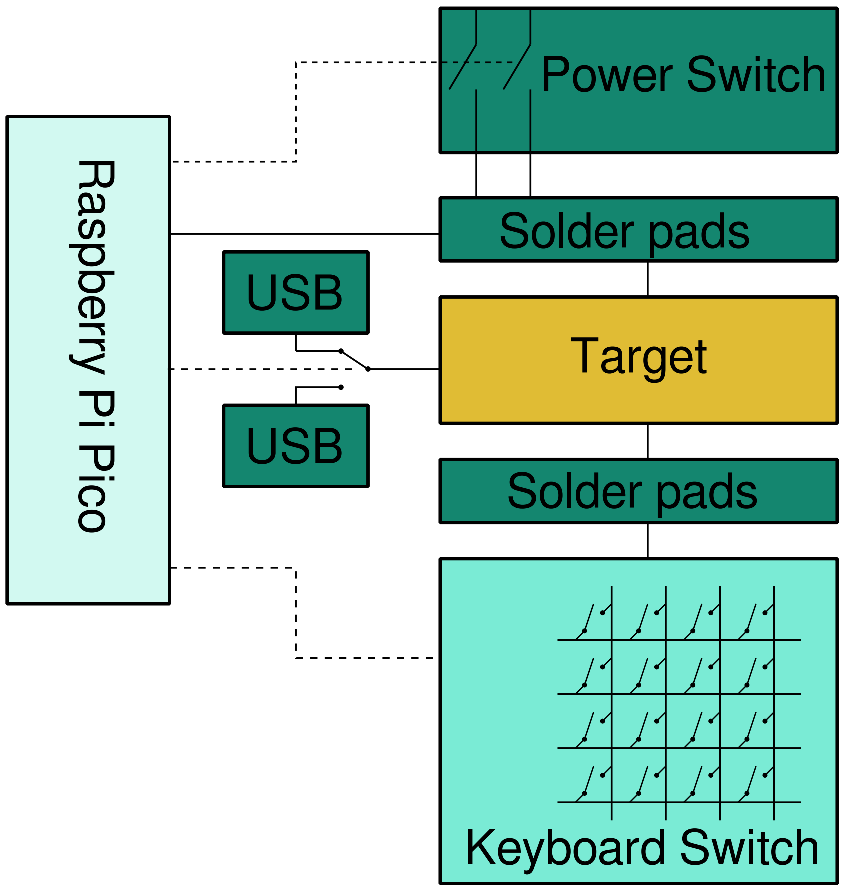


Figure 3.3: Detailed overview of LUPIn

LUPIn consists of a number of integrated components.

Power Switches

There are two pads on the PCB that can provide power. In order to reset the target system completely, the power can be switched using a switch. Since electromechanical relays have high switch times, low reliability, high cost and large footprints, a solid-state MOSFET power switch was chosen. The switch state is controlled by the Raspberry Pi Pico MCU, while the input voltage is selected using a small 1.27mm header. Using jumpers, the voltage for both the on and off state can be chosen. By removing the jumper, the pad is left floating.

Keypad Switch

Since encrypted USB drives commonly use buttons for the user to enter the code, a keyboard switch is present on the board. This makes it possible to enter a code electronically, which is required for certain attacks where the PIN needs to be entered automatically. It consists of 32 MOSFET switches, of which 25 are configured in a 5×5 matrix. After connecting the rows and columns to the target, these can be used to press keys electronically. The remaining 7 switches can be used for anything. Due to space restrictions, some of the switches are not independent, but share one pole.

USB Connector & Switch

An USB-A connector is placed on the board, both to connect to the device and to hold it in place mechanically. This connector can be soldered in two separate directions, in order to mount the target in two different ways. Additionally, the USB data lines are connected to a USB switch, which in turn connects to two USB ports. This way, the USB port can still be used during an attack. If there is an exploit that can be triggered over USB, one low-speed device can trigger the exploit, switch the port, and a high-speed device can pull the data from the target

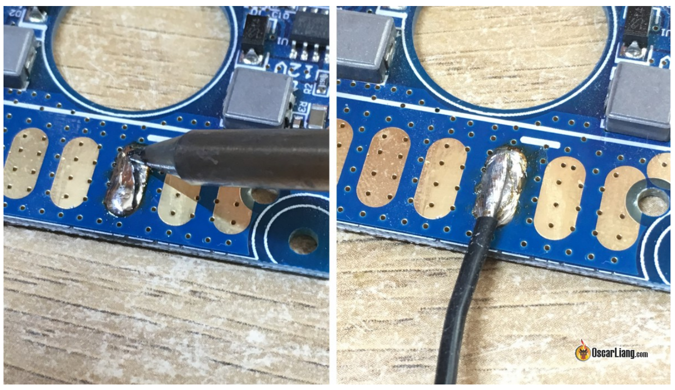


Figure 3.4: Example of an exposed pad used to connect a wire. Image by [42]

Input/Output Pads

Exposed copper pads, like in Figure 3.4 on the PCB are used for electrical connections. These were chosen because of their small footprint, reliable connection and compatibility with small wire thickness, as opposed to wire-to-board connectors. These pads are identical to regular SMD component pads. Wires can be soldered to a pad for a reliable connection. The pads that are present on the board are:

- 10 pads that lead to the user pins of the UFO board
- 6 pads that are connected to the Raspberry Pi Pico
- 6 pads that lead to the programming connector on the UFO board
- 8 pads that are linked to the 20-pin ChipWhisperer connector on the UFO Board

Raspberry Pi Pico Daughterboard

This detachable daugherboard is connected the main board using a 34-pin connector. The board can control most of the main LUPIn board, and interface directly with the target. This board has a number of components:

Voltage Level Shifters The Raspberry Pi Pico only supports input and output voltages of 3.3 V, while targets could be use different values. To remedy this, three level shifters are fitted. These voltage shifters have 2 channels and the direction of both channels is configurable. The Pico can change the direction per chip by toggling a pin. Input and output voltage can vary from 1.8 V to 5 V, which should cover most common logic standards [43].

LEDs To provide visual feedback, each of the 6 GPIO lines is connected to an LED. Because the LEDs could influence the switching characteristics, they are separate from the data lines with a single noninverting buffer IC.

Microcontroller The microcontroller on the board is a Raspberry Pi Pico. This MCU has two features that make it suitable for controlling (time-sensitive) hardware attacks. The first feature is the PIO subsystem. It consists of eight small cores that execute serial programs, with a simple instruction set of eight opcodes. It is optimized for implementing IO protocols in a straightforward way. Since the PIO runs at the same clock frequency as the main core (max 125 MHz) and is able to read or write one or more IO pins in a single clock cycle, it can provide very flexible and high-speed programmable interfaces. The second feature of the Pico MCU is that it is a dual-core processor.

3.3.3. Electrical Implementation

Schematic

The schematics were drawn in KiCAD, an open-source program for PCB design.

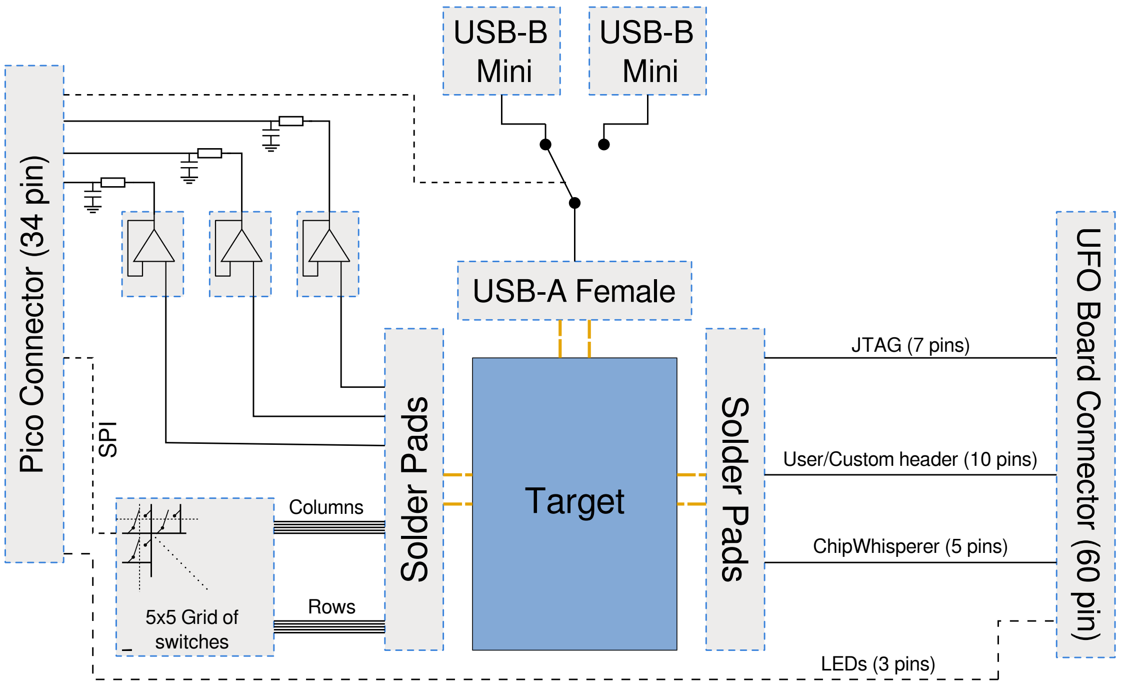


Figure 3.5: Signal connections between components on the LUPIn main board

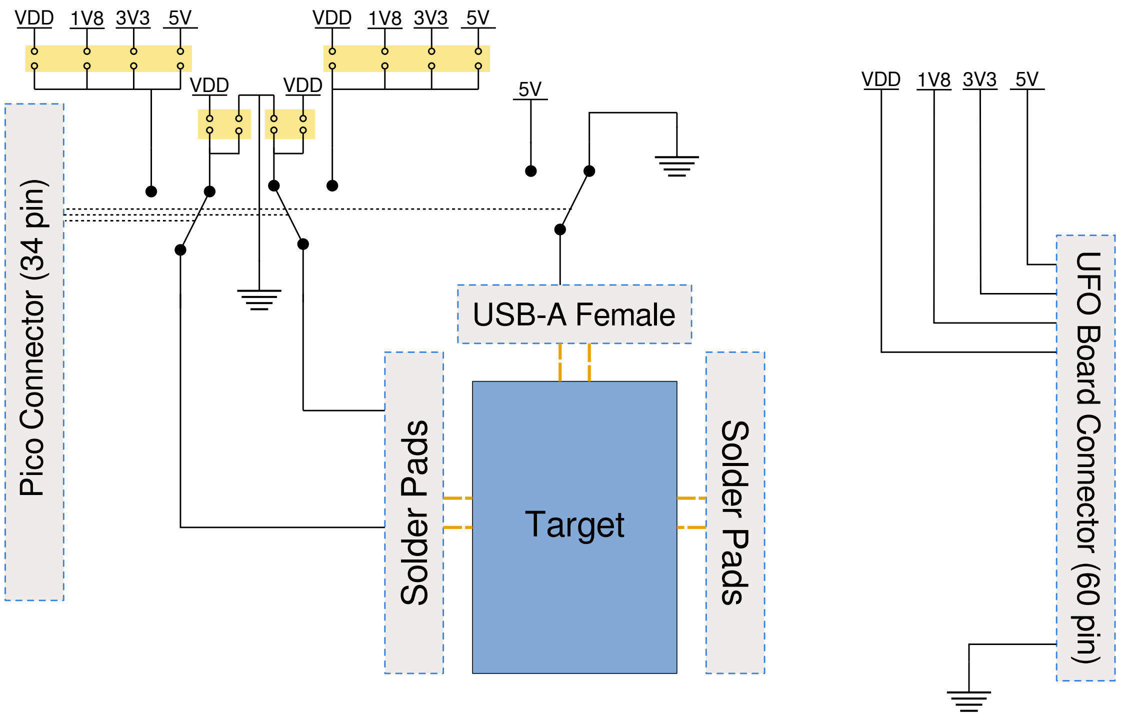


Figure 3.6: Power connections between components on the LUPIn main board

LUPIn Main Board A symbolic overview of the schematics is presented in Figures 3.5 and 3.6. The former details the signal connections between all subsystems, while the latter shows how the various power nets and switches are connected.

Four voltage nets from the UFO board are used: 1.8V, 3.3V, 5V and the user controlled voltage (marked as VDD in the schematic). There are two configurable outputs, where the voltage can be selected using the headers J15 and J16. Each is connected to a power switch that is toggled by the daugherboard.

The USB power is provided by the 5V rail from the UFO board. Three transistors are used to switch the power on and off. The first is n-MOSFET Q2B, which will pull down the gate of p-MOSFET Q3 when enabled, providing power to USB. When the input goes low, Q1 is will start conducting, to properly pull the voltage to 0V.

The board contains solder footprints for two different USB-A connectors. The first one is a regular female USB-A port, while the second is upside-down. On the PCB, both components share the same physical space: only one can be soldered on at the same time. This allows the user to mount the target upside-down.

The full schematics as drawn in KiCAD can be found in Appendix A.1

Raspberry Pi Pico Daugherboard The daughterboard contains the microcontroller (Raspberry Pi Pico), voltage shifters and LEDS. The microcontroller (MCU) is powered by either the USB host port, or through the connector. It uses a setup with a p-MOSFET for efficiency. While the bulk diode of the transistor could be used, an additional schottky diode is placed, for increased robustness.

PCB

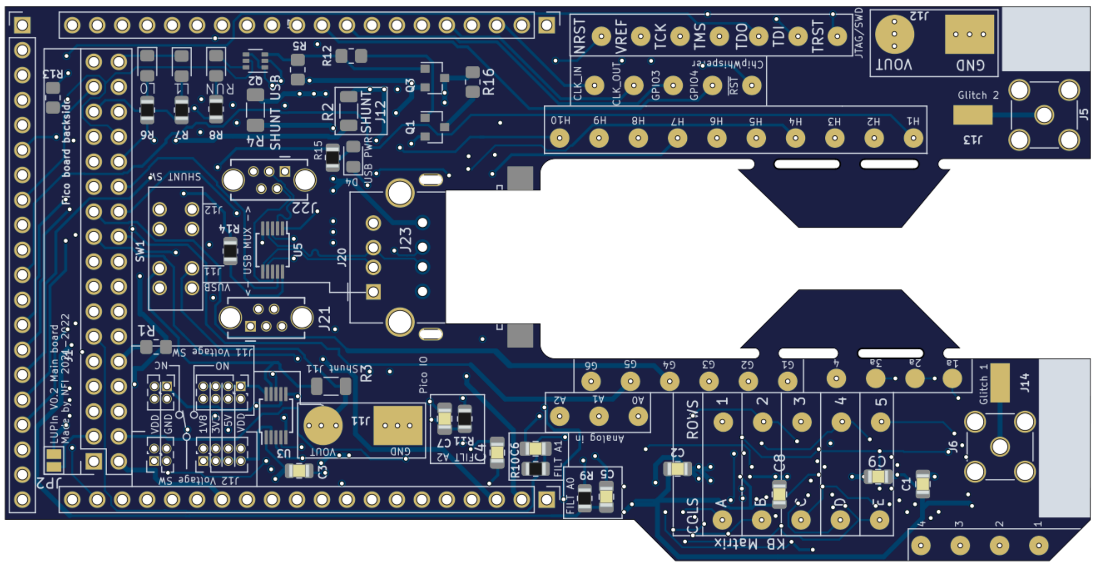


Figure 3.7: Main PCB top view

The PCB has U-like shape, where the target is mounted in the middle. Wires can be soldered to the pads to make connections to the target. The two “flaps” in the middle can be used to glue the target for robustness, or can be removed at the cut-away line.

3.3.4. Software Implementation

The software running on the Raspberry Pi Pico daughterboard is based on MicroPython [44]. MicroPython is a small implementation of the Python programming language, suitable for running om microcontrollers with a few 100 kB of RAM. Most of the Python syntax is supported, but the standard library is significantly slimmed down.

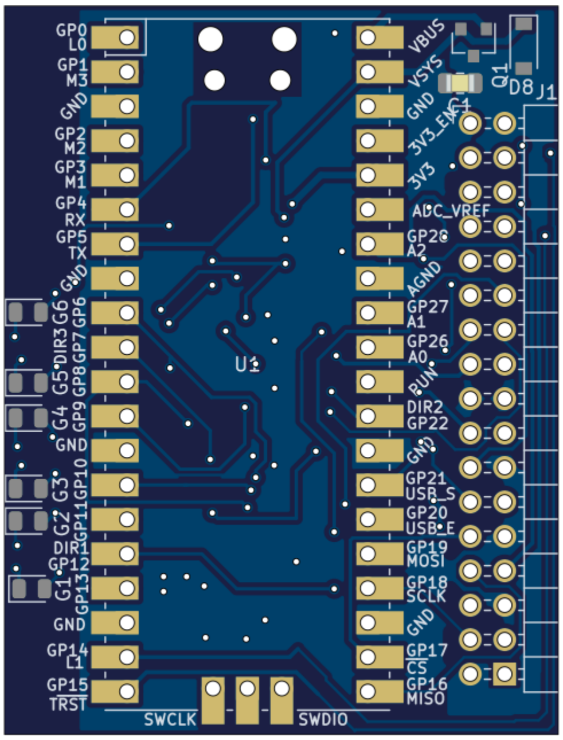


Figure 3.8: Secondary PCB top view

Usually, firmware for microcontrollers is written in a compiled language like C. Modifying the source code requires a recompilation step and uploading it to the internal flash. MicroPython, like the original Python, is interpreted. Over the interactive Read-Evaluate-Print-Loop (REPL) session code can be directly executed on the MCU. It additionally incorporates a simple filesystem, which can be used store scripts. These features allow for quick iterations and real-time feedback, in turn decreasing development time.

The interpreter gives no guarantees about the accuracy of the timing functions [45]. Instead, tasks that require precise timing can be delegated to the PIO subsystem. PIO cores run programs written in an assembly language consisting of only eight instructions. With MicroPython, these tiny programs can be written and run inline with other code.

3.4. Results

The design resulted in two separate PCBs. The main PCB is designed around the physical and electrical specifications of common encrypted USB drives. Such a drive can be mounted and attached to the PCB. Connections can be made by soldering wire to the exposed solder pads. The PCB is designed to be plugged into an UFO board, an inexpensive component that provides power and additional connections.

The smaller second PCB adapts an off-the-shelf microcontroller to the connector of the first PCB. It is used to control solid-state switches on the main board. Furthermore, it can be used for direct communication with the target over GPIO lines.

Combined, the three boards are a robust and flexible platform for attacking encrypted USB drives. Each PCB can be independently modified or replaced for maximum flexibility.

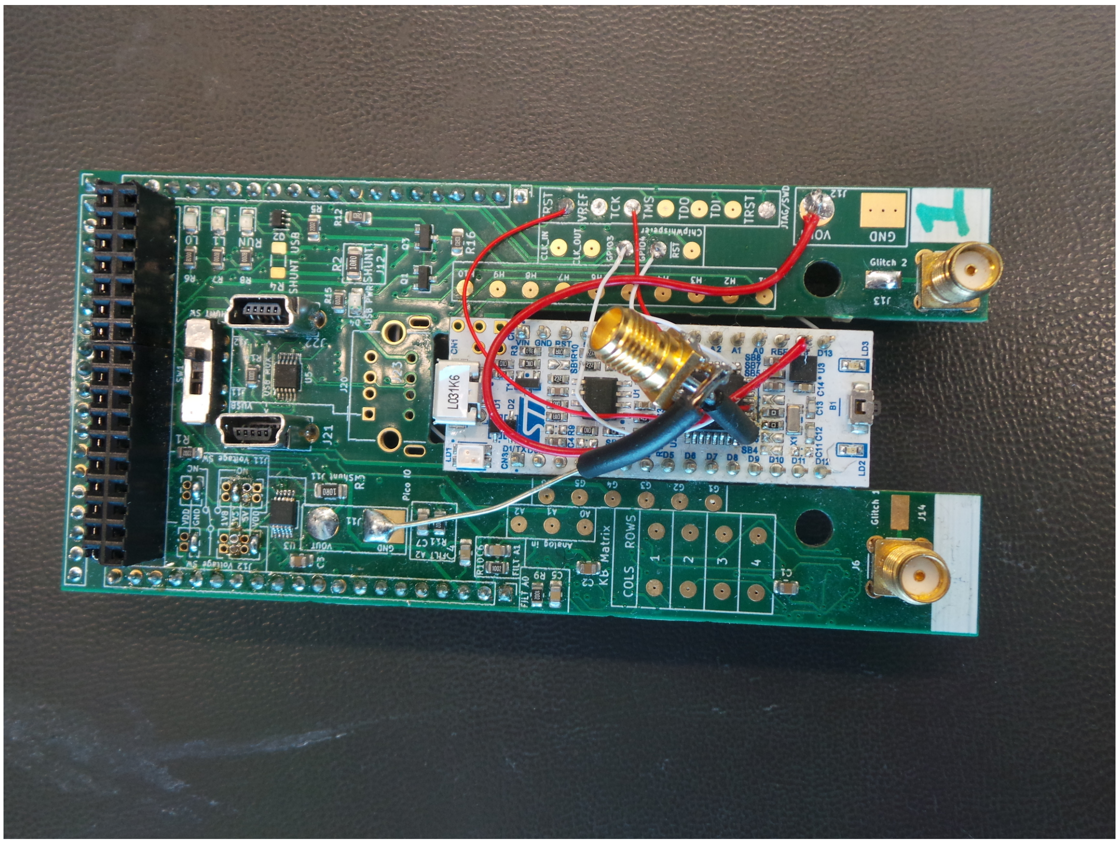


Figure 3.9: Photo of the LUPIn PCB with the an STM32 development board (white) mounted

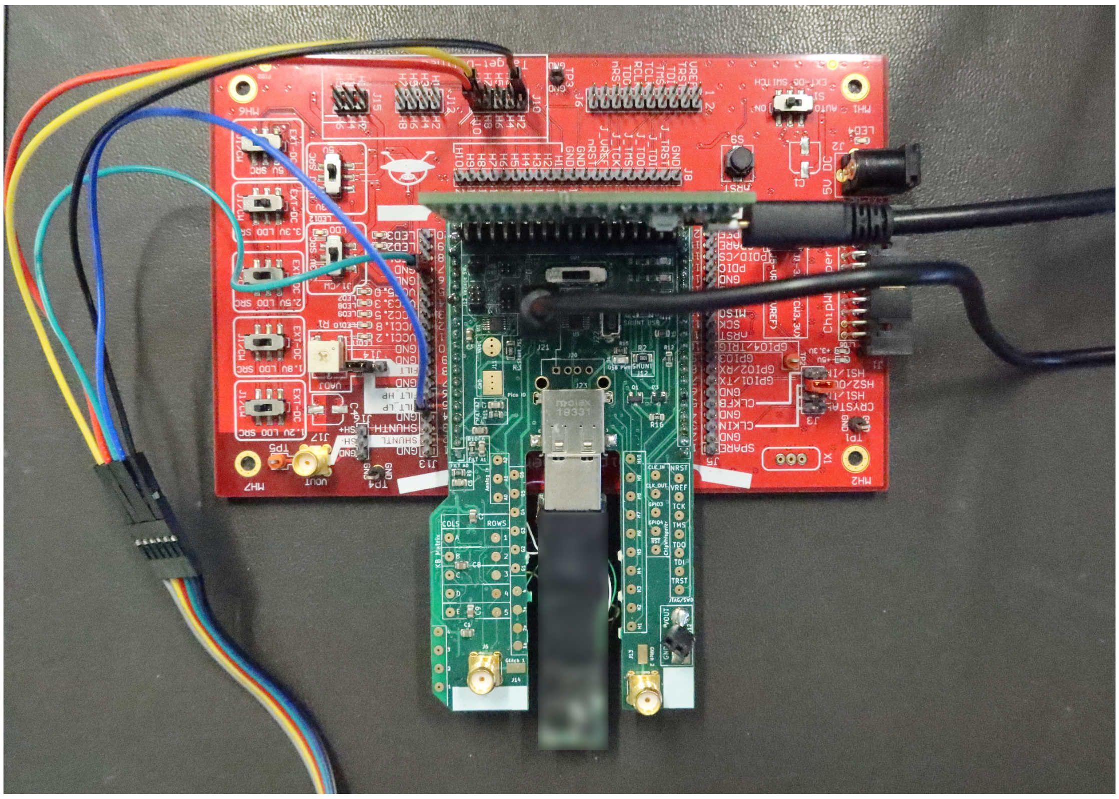


Figure 3.10: Photo of complete LUPIn setup, mounted on a UFO board, with a target

4 Using LUPIn

4.1. Motivation

In order to verify the performance of LUPIn, we use it to attack a consumer-grade encrypted USB drive. This target was chosen because it has no known vulnerabilities The USB drive contains a microcontroller that handles the cryptography. For the remaining part of this chapter, this microcontroller will be considered as the main target.

4.2. Target

The chosen target is a microcontroller with 8 kB of Random Access Memory (RAM) that performs cryptographic operations. It confirms to NIST standard FIPS 140-2 Level 3[15]. The input is a PIN of 7 to 15 digits. The digits are read using 10 membrane switches that are connected in a 3×4 keyscanning matrix. After reading the digits, it proceeds to the key derivation phase. Here the PIN and a salt value are used to derive a Key-Encrypting Key (KEK). The resulting KEK is used to decrypt a block of data using AES. This block contains the Data-Encrypting Key (DEK). Additionally some check data is included, since AES has no authentication/verification properties. If the check passes, the DEK is used to decrypt user data. If the check fails, the DEK contains invalid data that cannot be used to decrypt the drive. In this scheme, the DEK is never directly checked or stored, ruling out simple fault injection and side-channel attacks.

The drive has a locked down debug port: only RAM can be read. Internal flash storage is unreadable. Since we cannot read out the flash, the code remains unknown.

It also implements a brute-force detection: after 10 consecutive incorrect unlocking attempts, the internal key data will be wiped, and the stick will reset to factory settings. The original user data is rendered unreadable, since the decryption key cannot be retrieved.

The aim of this attack is to extract the salt value, which is stored in flash. The only way we can observe or derive the salt is via RAM. The certifications detail that the the PBKDF2 algorithm is used to derive the KEK. As input, it takes the PIN and the salt. PBKDF2 [46] is a generic password-derivation algorithm, which can be used to derive keys of any length. It also needs an underlying pseudorandom function (PRF), which in our case is HMAC-SHA1 [47]. PBKDF2 will repeatedly call the PRF on its own output, in order to generate high-entropy, secure output bytes.

A complete overview of the algorithm can be seen in Figure 4.1.

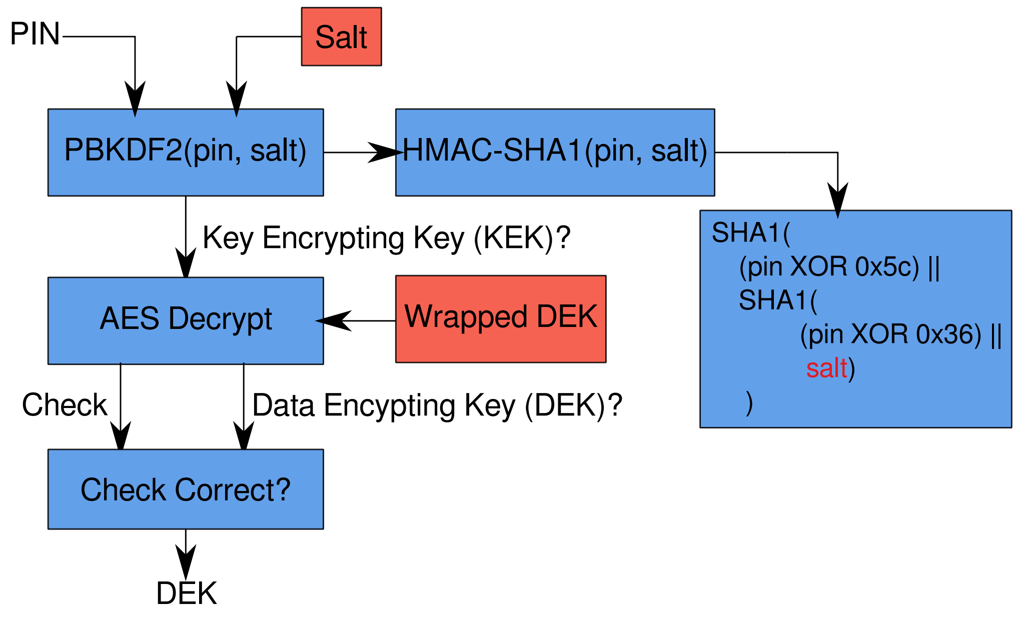


Figure 4.1: Target cryptography overview. Red boxes indicate that the data is stored in flash. Blue blocks are operations performed by the microcontroller.

4.3. Attack

As described in Section 2.2, multiple different attacks are available for these types of microcontrollers. Ideally, we would like to get the secrets off the chip with 100% success rate, without damaging the chip. Using these goals, we can elimate some of the hardware attacks discussed earlier. A glitching attack could be used to bypass the debug lock, and access the flash region. However, these types of attacks have been extensively researched in the past, resulting in manufacturers building countermeasures into their devices. Preliminary tests revealed that simple glitching tools would be insufficient to bypass the debug lock.
Getting to the flash would also be possible by completely “decapping” the chip itself, and reading out the storage under an electronic microscope[48]. However, this method risks destroying the chip with its contents. Therefore it would not be a very practical attack for forensic investigation.

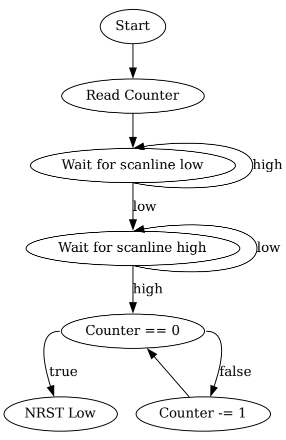


Figure 4.2: State machine used to implement the CBS attack on the PIO

Likewise, side-channel attacks would be difficult to perform, given that we need to find a precise and repeatable moment where the secret data is mixed with data we control. Since the firmware is unknown, this would be a difficult task.

For the attack we will use the Cold-Boot Stepping method as described in Section 2.2.3. This attack was chosen for a number of reasons. First, this attack only requires an open debug port, which matches our target. This allows us to study the capabilities and limitations of CBS. Additionally, we can test whether LUPIn is compatible with this type of attack. During the attack our goal will be to extract the internal salt, stored in the internal flash memory. LUPIn helps us with the setup and execution of the attack in numerous ways, which will be discussed in the following sections.

4.3.1. Setup

Our aim is to extract the salt value from flash. From there we can perform an offline bruteforce attack, to find the original PIN. As the attack method CBS is used, as it designed to bypass flash readout restrictions.

CBS is modified to fit this specific target. First, the cryptographic operations only start after a Personal Identification Number (PIN) has been entered. In the original CBS paper[5], the timing was fixed from the microcontroller boot, which is a reliable method. This setup can be seen in Figure 4.4a. Since the goal is to snapshot encryption data from RAM, the attack needs to be started when the device starts checking the PIN. One option is to simply start the timer after the last input has been made, like in Figure 4.4b. The problem is that this introduces jitter, as the key is sampled asynchronously. Therefore we define the start point after the target has sampled the input, using the matrix switching signals as reference, see Figure 4.4d. LUPIn will aid in the attack by setting up the electrical connections and the execution of the main tasks. The overview of all connections can be found in Figure 4.3. The target connects to the LUPIn Printed Circuit Board (PCB), from which it connects to various other subsystems: The nRST pin, keypad, keypad scanning pin and power switch are connected to the daughterboard. Power and debugger lines are routed to the UFO board. The UFO Board provides a stable voltage and a debugging connector, to which a standard JTAG debugger is connected.

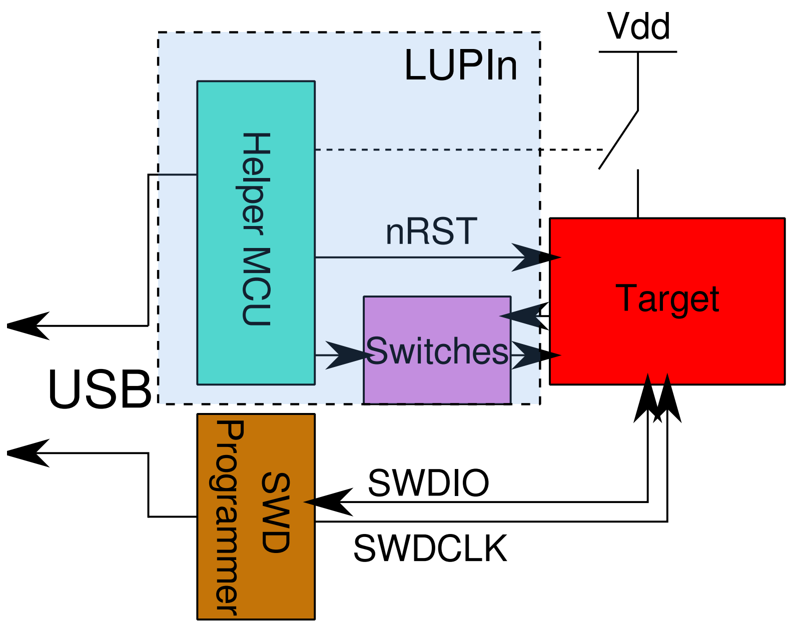


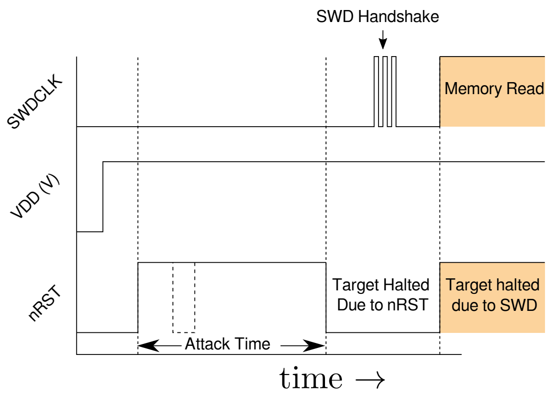
Figure 4.3: Cold-Boot Setting block diagram

The daughterboard runs the main code responsible for the attack. Recall that after the PIN has been entered, we need to wait for a specific line to change state, wait for a number of cycles and finally pull down the nRST line. These actions are time-sensitive, and therefore implemented on the PIO subsystem, as explained in Section 3.3.1. The code is derived from the state machine diagram in Figure 4.2.

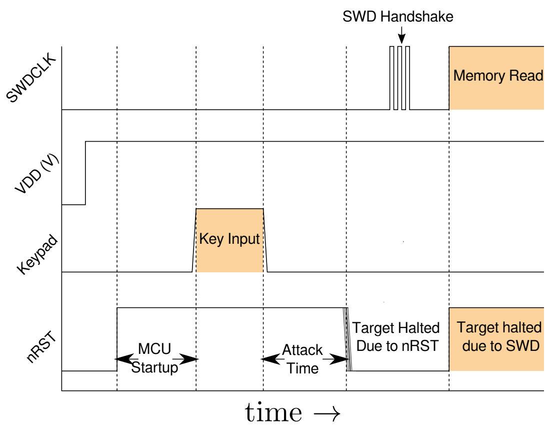
A PC is connected to both the daughterboard and the programmer. After the daughterboard finished executing, the debugger is used to extract the RAM data. The target needs to be in active in order to read the data, which means that the nRST has to be restored before the data transfer takes place. All RAM samples are saved in a database together with the attack parameters.

To conclude, the target is powered on and the nRST line is pulled high. After the initialization of the target, the PIN is entered using the electronic switches. At the last keypress, the PIO system will wait for an rising edge, and start counting down from the preset counter. If the counter reaches zero, the nRST line is pulled low, and the debugger is connected. Once the debugger has done the handshake, the nRST line is set to a high voltage one last time. At last, the RAM is read. To prepare for the next round, power is switched off.

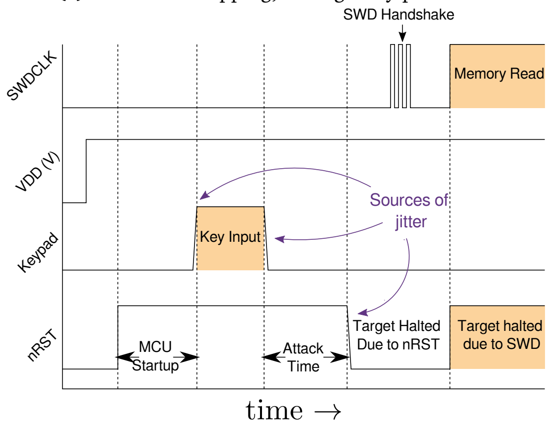
This setup allows us to read RAM from timestamp $t = 0$ to the end of execution, which is when the USB has checked the input code. While we have established earlier that the bruteforce detection is bypassed when resetting before the final check, the attack is performed on a prepared USB drive where the PIN is known in advance. This way, we can develop this attack without possibility of accidentally triggering the bruteforce detection mechanism.



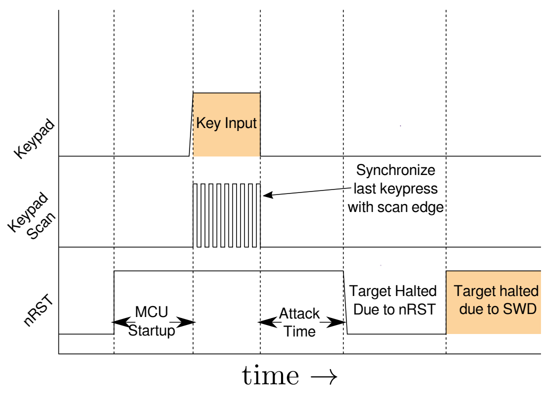
(a) Cold-Boot Stepping, as originally presented



(b) Cold-Boot Stepping, with PIN



(c) Cold-Boot Stepping, with jitter sources



(d) Cold-Boot Stepping as done in the attack

Figure 4.4: Cold-Boot Stepping setups

4.3.2. Results

The attack performed as intended, LUPIn proved to be a reliable addition to the setup. A large number of RAM dumps was collected, from interval 0 to 2s, with each individual sample being 8 kB of binary data. No additional metadata was available. However, there is some information that can be deduced from the data. The higher addresses are filled with `0xAA` and the lower addresses are filled from `0xBB`. The data at higher addresses changes infrequently, suggesting that it is used as heap, and grows upward. The lower part is the stack, growing down. Following this logic, we can conclude that the maximum RAM usage recorded is about 5 kB. This value is derived from the first non-fill values between the edges of the stack and heap. Certain 32-bit values appear to be pointers to various locations in RAM (`0x2000xxxx`) and flash (`0x0800xxxx`). This could mean there are pointers that point to allocated memory else. Other values seem to indicate cryptographic operations, e.g. constants used by SHA-1 or HMAC.

4.4. Data Filtering

4.4.1. Motivation

From the data of the attack in Section 4.3.2 it is concluded that the data has inconsistencies. In this section we will analyze why these inconsistencies are present and how we try to minimize their impact. Each RAM snapshot has corresponding sample time t_s, which is the time between entering the PIN and taking the snapshot. One way of analyzing the collected data is by checking the bytes that are different between two consecutive samples. These bytes would indicate where the processor is currently writing data, which identify which data structures are being modified.

Looking a the number of bytes that differ we can quickly find large inconsistencies. Samples that are a small number of clock cycles apart differ with hundreds of bytes. Using the fact that the Central Processing Unit (CPU) is capable of writing up to 4 bytes per clockcycle, we can determine the samples that do not accurately represent the state relative to other samples.

Any part of the attack that take varying amounts of time between runs could be the cause for the behavior described above. For example, we want to take a snapshot after the processor has run for t_s seconds, but a random delay between 0 and t_r is present. The captured sample now is somewhere between $t_s - t_r$ to t_s.

By examining the setup, a number of processes are found that take random amount of time. Firstly, the attacking software setup for the input setup phase is not timed using deterministic way . Secondly, the targets reads button presses completely asynchronous from actual moment of the button press, leading to uncertainty. There are also some physical limitations that might influence the timing, such as switching times. The target uses an internal RC oscillator as clock, which can also take different times to stabilize, and can vary between runs. The target software might not be deterministic either: there is a hardware random number generator on the MCU, which is not controllable by our attack. There might be interrupts caused by timers, that can trigger at unpredictable intervals.

4.4.2. Method

As mentioned before, the CPU can write at maximum 4 bytes per clockcycle into RAM. This is done by using the store instruction, which can write one register[1].

Therefore, two RAM samples that differ only 4 bytes have a high probability of being correlated in terms of program state. The filtering algorithm takes this into account. The main working of the algorithm are presented in Figure 4.5. In the first step, we apply some prefiltering. This means trimming the samples to the used memory region, and removing bytes that are different for more than 90% of the samples These bytes are random state, sampled from the hardware random number generator. All remaining samples are put into a fully connected graph. Each connection has a weight that corresponds to the hamming distance between the two ends. We then filter nodes that contain edges with 0 weight, which means the samples are exactly the same. Next, we find the Minimal Spanning Tree (MST) of the graph, which will find the minimum weight (hamming distance)

[1]The PUSH instruction can write n registers, but takes $1 + n$ cycles

needed to traverse the entire graph. The resulting tree has only connections between samples that are very likely to be consecutive in terms of program state.

The psuedocode for the algorithm is as follows:

```
samples = [ram0, ram1, ram2, ...] # 2D array of n ram samples

for i in len(samples):
  for j in (i+1, len(samples)):
    hamming_distance[i,j] = hamming(samples[i], samples[j])

G = Graph()
for i in len(samples):
  G.add_node(i)
  for j in (i+1, len(samples)):
    G.add_edge((i,j), weight=hamming_distance[i,j])

G.drop_nodes_with_zero_weight_edges()

mst = G.minimum_spanning_tree()
```

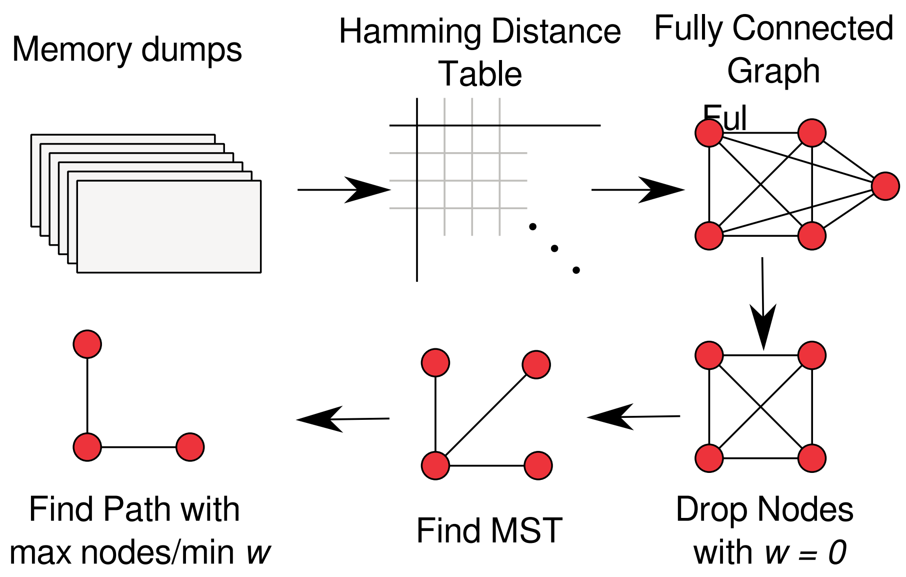


Figure 4.5: Data filtering algorithm

4.5. Simulation

4.5.1. Setup

In order to test the algorithm, a simulation was performed. The main goal is to measure the ability to restore samples into correct order. By simulating an entire CPU core, the system state can be determined exactly at clock cycle accuracy. Because the software is known is advance, we determine if the algorithm reconstruction actually matches the simulated reality.

For this simulation, an entire system with microcontroller CPU was emulated using QEMU [49]. This core runs a program that performs cryptography similar to the target, namely PBKDF2 with HMAC-SHA1, using an open-source library called mbedTLS [50].

```c
int main(void) {
  // setup code omitted

  // Input values
  const unsigned char key[] = "key";//{0x1, 0x2, 0x3, 0x4, 0x5};
  uint8_t hash[MBEDTLS_MD_MAX_SIZE] = {0};
  const char input[] = "The quick brown fox jumps over the lazy dog";
  const int outsize = 32; //Output size of

  // mbedtls context for HMAC-SHA1
  mbedtls_md_context_t ctx;
  mbedtls_md_init(&ctx);
  mbedtls_md_setup(&ctx, mbedtls_md_info_from_type(MBEDTLS_MD_SHA1), 1); // Setup
    SHA1

  mbedtls_pkcs5_pbkdf2_hmac(&ctx, (const unsigned char*)input, sizeof(input)-1,
    key, sizeof(key)-1, 1000, outsize, hash);

  hex_print(hash); //output to prevent compiler from optimizing everything away
}
```

Listing 4.1: Code running in the simulation

The short version of the code is listed in Listing 4.1. After the virtual CPU has started up, the machine is halted. From there, each instruction is executed by itself (single-stepping), and the RAM and registers are saved to disk at each instruction. The result is several thousand RAM samples, corresponding to each instruction cycle.

In the next steps, we will attempt to verify the algorithm by running it against the memory data we collected before. In order to simulate a the CBS attack, we randomly sample from the snapshots, and apply the algorithm to reconstruct the most probable order the snapshots occurred in. Lastly, we verify the algorithm results with our known real results. Sampling is done by dividing the total range into small intervals. At each interval, one or more samples is picked based on a normal distribution. The interval size, number of samples per interval and the σ of the normal distribution are used are parameters. Then, the algorithm is applied on those samples. Since we know the full context of the samples, we can infer whether the algorithm has made correct predictions.

4.5.2. Results

Since the result of the algorithm is a connected graph, there are two types of errors theoretically possible: The nodes in the graph are not ordered correctly or there are branches in the graph (nodes with >1 degree). The first failure mode was not reproducible with the simulation, and therefore not considered for further investigation. However, branching was clearly visible. Branches can be interpreted as two paths that are both very likely. However, there is another aspect of branching. In C, when a function is executed, registers are pushed to the stack. When the function terminated, the data is read back from the stack into the registers. These two situation have very similar RAM contents, resulting in branches in the final tree. In order to test the algorithm performance, a Monte-Carlo approach was used: over 400 trials, data was randomly sampled, reconstructed and performance was measured. The number of nodes on the longest tree (e.g. not on a small branch) is classified as "percentage matched". The result can be seen in Figure 4.6. The three parameters correspond to the random sampling, as described in the previous section.

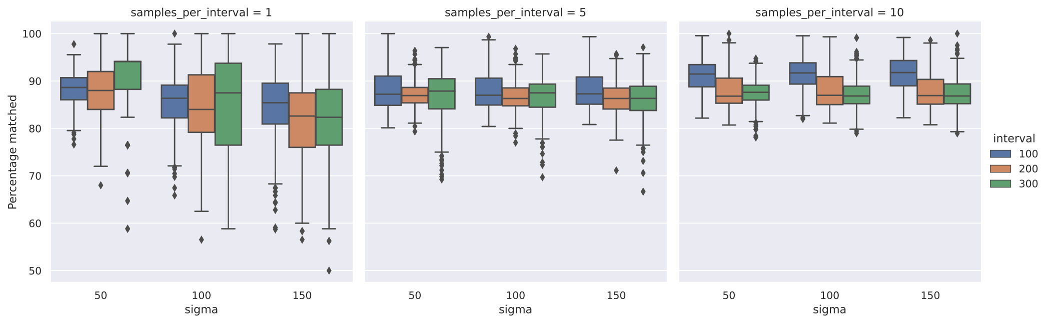


Figure 4.6: Simulation results over 400 runs with different parameters. The percentage matched refers to the number of nodes that are placed in the main branch

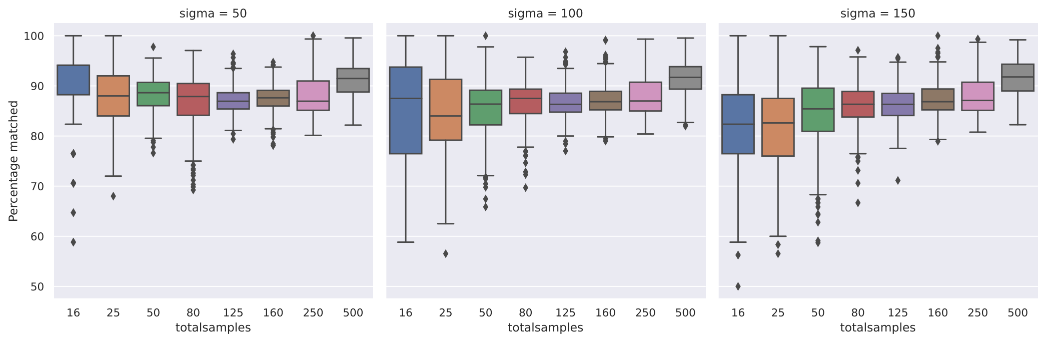


Figure 4.7: Simulation results, grouped by number of samples and σ (standard deviation)

Figures 4.6 and 4.7 show the results, grouped by the samples per interval and the σ of the random distribution, respectively. There are a number of things to conclude from the results. The sigma variable is significant when only a single sample is taken per interval. For an increased number of samples, the sigma is less significant, while additionally decreasing the error margins. In other words, when there is more samples to chose from, the percentage of mispredicted samples goes down, likely due to the average smaller hamming distance between the samples. To conclude, we most significant aspect is to collect more data, as more data results in better reconstruction.

4.6. Filtered Results

4.6.1. Filtered Data

Now that the algorithm was verified, the next step is to apply it to the real data. From Figure 4.7, we can conclude that we need to collect as much data as possible to decrease the uncertainty in the resulting graph. Contrary to the simulation, the random deviation during sampling is unknown.

The aforementioned algorithm is applied on the data gathered from the attack. No verification data is available, however certain metrics can be derived from the reconstructed tree itself. Most importantly, the number of branches in the resulting graph give an indication of the validity. Many branches means that at many points, a good decision could not be made. An example can be seen in Figure 4.8. Since the data itself is too large to plot in a single graph, 4000 random point were chosen. While at points there are paths visible with little branches, 1988 of the 4000 nodes in the graph have more then one edge, showing that the gathered data is has low correlation, as opposed to the simulations.

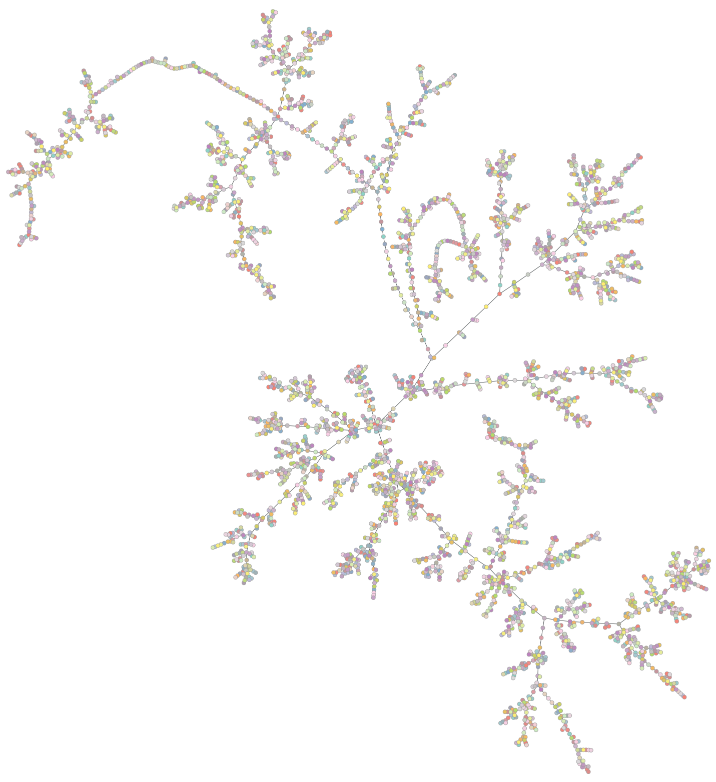


Figure 4.8: Result from the gathered data, 4000 random points

4.6.2. Extracting Secrets

Some cryptographic algorithms use constants that can be traced back to the RAM dumps. The target uses PBKDF2, which in turn uses HMAC-SHA1. HMAC-SHA1 takes a key K and a message m as arguments and performs the operations according to Equation 4.1. Note that the key is XORed with the $opad$ and $ipad$, which are both 64 byte buffers filled with `0x5C` and `0x36`, respectively[47]. PBKDF2 calls the HMAC function with "password" as key K, which should correspond to the user-entered PIN. Searching through the data samples, an address can be found that seems to contain both padding buffers at different times. Undoing the XOR at both buffers results in a 28 byte value followed by zero bytes, which can be seen in Figure 4.9. This value stays constant over time, and changes when another PIN is entered. The question remains how this value is derived, as it does not contain the PIN itself. Additionally, a 28 byte length is unusual, as it does not match common hashing algorithms.

$$\text{HMAC-SHA1}(K, m) = \text{SHA1}\Big((K' \oplus opad) \parallel \text{SHA1}\big((K' \oplus ipad) \parallel m\big)\Big) \tag{4.1}$$

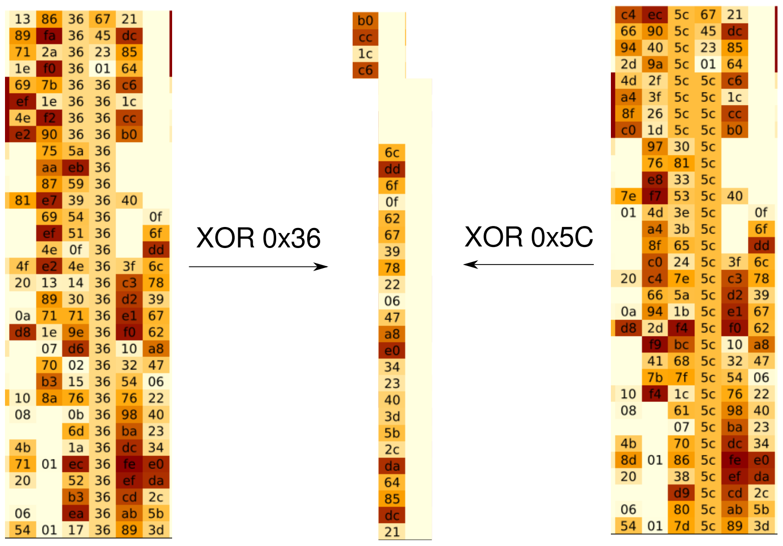


Figure 4.9: Memory view of the two buffers used in HMAC. Both are used to uncover a 28 byte value. Squares with no text contain the value `0x00`

4.6.3. Discussion

It was not possible for to extract the salt, which was our goal. There are a number of likely causes that will be expanded on in the following paragraphs.

Data Filtering Algorithm The created algorithm did very well in the simulation, where it was able to match over 80% of the samples given enough data. However, we cannot replicate those numbers with the real data. There is likely an unforeseen factor that influenced the results. The software is one aspect that is not really taken into account with the simulation, as a very simple program was being run. It could be that the real software uses some random data and

Compiler Optimization SHA-1 consists of 80 rounds of bitwise operations. Each round modifies all internal state registers. However, an optimizing compiler is free to reorganize the instructions for fastest path. Additionally, internal state of SHA-1 only requires six 32-bit variables, which can all be kept in registers. Therefore, looking at a RAM sample is not guaranteed to actually contain the internal SHA1 state.

Memory-Mapped IO The target MCU maps the entire flash region into memory. This makes accessing the flash data from code simple, as one can use the regular load/store instruction. However, this also means that secret data, such as the salt, might not even appear in memory. The data could be modified before it is written back to memory, if ever.

SHA-1 Internal State The resulting hash of SHA-1 is five internal variables concatenated together. Therefore, reading the state alone does not reveal information whether the operation has finished or is still ongoing. This can make it difficult to determine the input of the SHA-1 function, as the output cannot really be determined.

Lack of Analysis Software The memory samples contain no metadata, and are just binary dumps. While there exist tools that can analyze binary files, in our case the data has an additional dimension: time. We need to find data correlation in binaries that are sequential. The author is unaware of any software that can (be extended to) fulfill this requirement.

5 Conclusion & Future Work

5.1. Conclusion

In this research a hardware tooling PCB (LUPIn) was successfully designed, assembled and tested. It proved to be a reliable platform for performing different hardware attacks against encrypted USB drives. All beforehand set requirements were satisfied, being flexibility, robustness and reproducibility. Integrated components allowed for a wide variety of attacks, while the addition of semi-permanent connections kept the system robust and reproducible. This made development of attacks simpler and less time-consuming.

LUPIn was adapted to attack a real world target. During this attack, a LUPIn PCB was bonded together with the target. A Cold-Boot Stepping attack was successfully executed, where RAM was read thousands of times with precise timing. The aim was to extract a salt value using in unlocking an encrypted data partition.

However, the chosen target was not crackable using the Cold-Boot Stepping attack. Mainly due to difficulties of analyzing the retrieved RAM data, even with additional data filtering. However an attack on a simulated target yielded promising results. The different outcomes are likely attributed to lack of information about the target firmware.

5.2. Future Work

Future work can be dedicated to two different subjects: LUPIn and the attack on the target.

For LUPIn, future work can be performed by verifying the compatibility with other major hardware attack, namely glitching and side-channel analysis.

For future CBS attack, reading out the flash data with invasive SEM techniques should be considered. This flash data can be used to reverse-engineer the firmware, which will lead to better identification of sensitive material. Additionally, having completely deterministic signals during the input phase could lead to less time inaccuracy during readout. It can be achieved using FPGAs or extending the PIO subsystem of the Raspberry Pi Pico. Lastly, data processing can be improved by using several novel techniques such as SAT solvers or deep learning. SAT could be used to find logical operations corresponding to data modification between RAM samples. Using simulated data, it might be possible to train a neural network to identify sensitive data. Another opportunity in data processing could analysis of the pointers in RAM. Pointers might contain information about the internal layout of data structures. This information could be used to create more extensive models of the performed operations.

Bibliography

[1] LimitedResults. “Enter the EFM32 Gecko.” (Jun. 22, 2021), [Online]. Available: https://limitedresults.com/2021/06/enter-the-efm32-gecko/.

[2] N. Wisiol, P. Gersch, and J.-P. Seifert, “Cycle-accurate power side-channel analysis using the chipwhisperer: A case study on gaussian sampling,” 2022. [Online]. Available: https://eprint.iacr.org/2022/903.

[3] C. O’Flynn, “Brute-forcing lockdown harddrive PIN codes,” Aug. 2016. [Online]. Available: https://www.blackhat.com/docs/us-16/materials/us-16-OFlynn-Brute-Forcing-Lockdown-Harddrive-PIN-Codes.pdf.

[4] R. Rigo and J. Czarny, “Analysis of an encrypted hdd,” May 11, 2015. [Online]. Available: https://syscall.eu/pdf/SSTIC2015-Article-hardware_re_for_software_reversers-czarny_rigo.pdf.

[5] J. Obermaier and S. Tatschner, “Shedding too much light on a microcontroller’s firmware protection,” Aug. 2017.

[6] Z. Chen, G. Vasilakis, K. Murdock, E. Dean, D. Oswald, and F. D. Garcia, “Voltpillager: Hardware-based fault injection attacks against intel sgx enclaves using the svid voltage scaling interface.,” in *USENIX Security Symposium*, 2021, pp. 699–716.

[7] “About riscure,” Riscure. (), [Online]. Available: https://www.riscure.com/about-riscure/.

[8] *Riscure inspector*, Riscure, 2017. [Online]. Available: https://riscureprodstorage.blob.core.windows.net/production/2017/08/inspector_brochure.pdf.

[9] C. O’Flynn and Z. D. Chen, “Chipwhisperer: An open-source platform for hardware embedded security research,” in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, Springer, 2014, pp. 243–260.

[10] “Overview & comparison - newae hardware product documentation,” NewAE Technology Inc. (), [Online]. Available: https://rtfm.newae.com/Capture/.

[11] C. O’Flynn. “CW308-UFO CW PRO setup,” NewAE Technology Inc. (May 19, 2017), [Online]. Available: https://wiki.newae.com/File:Cwpro_setup.jpg.

[12] “CW308 UFO target board,” NewAE Technology Inc. (2018), [Online]. Available: https://media.newae.com/datasheets/NAE-CW308-datasheet.pdf.

[13] T. Geron, “Something ventured: Uncle sam is staking start-ups,” *VentureWire*, Mar. 12, 2008.

[14] J.-M. Picod, R. Audebert, and E. Bursztein, "Attacking encrypted usb keys the hard(ware) way," 2017. [Online]. Available: https://elie.net/talk/attacking-encrypted-usb-keys-the-hardware-way/.

[15] N. I. of Standards and Technology, "Security requirements for cryptographic modules," Mar. 25, 2001. DOI: 10.6028/NIST.FIPS.140-2.

[16] M. Schink and J. Obermaier. "Exception(al) failure - breaking the stm32f1 read-out protection." (Mar. 17, 2020), [Online]. Available: https://blog.zapb.de/stm32f1-exceptional-failure/.

[17] *Datashur user manual*, iStorage Ltd, 2013. [Online]. Available: https://istorage-uk.com/wp-content/uploads/2019/03/iStorage-datAshur-user-manual.pdf.

[18] S. Skorobogatov, "Teardown and feasibility study of ironkey – the most secure usb flash drive," Oct. 2021. DOI: 10.48550/arXiv.2110.14090.

[19] *Sandisk ultra fit usb 3.1 flash drive*, Western Digital Technologies, Inc., Jan. 2020. [Online]. Available: https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/sandisk/product/usb-flash/ultra-fit-usb-3-1/data-sheet-ultra-fit-usb-3-1.pdf.

[20] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, C. Whelan, and G. International, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, Aug. 2004. DOI: 10.1109/JPROC.2005.862424.

[21] C. Bozzato, R. Focardi, and F. Palmarini, "Shaping the glitch: Optimizing voltage fault injection attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 2, pp. 199–224, Feb. 2019. DOI: 10.13154/tches.v2019.i2.199-224. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7390.

[22] C. O'Flynn, "Voltage fault injection on a modern RPi SBC," *Circuit Cellar*, Mar. 2021. [Online]. Available: https://circuitcellar.com/research-design-hub/design-solutions/voltage-fault-injection-on-a-modern-rpi-sbc/.

[23] N. Kühnapfel, R. Buhren, H. N. Jacob, T. Krachenfels, C. Werling, and J.-P. Seifert, "Em-fault it yourself: Building a replicable emfi setup for desktop and server hardware," in *2022 IEEE Physical Assurance and Inspection of Electronics (PAINE)*, 2022, pp. 1–7. DOI: 10.1109/PAINE56030.2022.10014927.

[24] K. Tobich, P. Maurine, P.-Y. Liardet, M. Lisart, and T. Ordas, "Voltage spikes on the substrate to obtain timing faults," in *2013 Euromicro Conference on Digital System Design*, 2013, pp. 483–486. DOI: 10.1109/DSD.2013.146.

[25] P. C. Kocher, "Timing attacks on implementations of Die-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology| Crypto*, vol. 96, 1996, p. 104 113.

[26] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," 2016. [Online]. Available: https://eprint.iacr.org/2016/596.

[27] D. Genkin, A. Shamir, and E. Tromer, “Rsa key extraction via low-bandwidth acoustic cryptanalysis,” 2013. [Online]. Available: https://eprint.iacr.org/2013/857.

[28] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Advances in Cryptology — CRYPTO’ 99*, M. Wiener, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397, ISBN: 978-3-540-48405-9.

[29] “Tempest: A signal problem,” National Security Agency, 1972. [Online]. Available: https://www.nsa.gov/portals/75/documents/news-features/declassified-documents/cryptologic-spectrum/tempest.pdf.

[30] *STM32F0x1/STM32F0x2/STM32F0x8 advanced Arm-based 32-bit MCUs - reference manual*, rev. 10, STMicroelectronics, Apr. 18, 2022. [Online]. Available: https://www.st.com/resource/en/reference_manual/rm0091-stm32f0x1stm32f0x2stm32f0x8-advanced-armbased-32bit-mcus-stmicroelectronics.pdf.

[31] *Arm debug interface v5 architecture specification*, ARM Limited, Feb. 8, 2006. [Online]. Available: https://documentation-service.arm.com/static/5f900a61f86e16515cdc0610.

[32] “Kraken identifies critical flaw in trezor hardware wallets,” Kraken. (Jan. 31, 2020), [Online]. Available: https://blog.kraken.com/post/3662/kraken-identifies-critical-flaw-in-trezor-hardware-wallets/.

[33] T. Ban. “Hw fault injection mitigation,” ARM Limited. (2020), [Online]. Available: https://www.trustedfirmware.org/docs/TF-M_fault_injection_mitigation.pdf.

[34] M. Witteman. “Secure application programming in the presence of side channel attacks,” Riscure. (Nov. 2018), [Online]. Available: https://web.archive.org/web/20210118203607/https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf.

[35] “National institute of standards and technology,” U.S. Department of Commerce. (), [Online]. Available: https://www.commerce.gov/bureaus-and-offices/nist.

[36] “Fips general information,” National Institute of Standards and Technology. (May 21, 2018), [Online]. Available: https://www.nist.gov/itl/fips-general-information.

[37] “Cryptographic algorithm validation program,” National Institute of Standards. (Feb. 24, 2022), [Online]. Available: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program.

[38] “A diagram which maps the general flow of the cmvp fips 140-2 testing process.” (Aug. 2004), [Online]. Available: https://commons.wikimedia.org/wiki/File:FIPS140-2validationflowchart.jpg.

[39] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall, Inc., 1974.

[40] M. Barthélemy, “Spatial networks,” *Physics Reports*, vol. 499, no. 1-3, pp. 1–101, Feb. 2011. DOI: 10.1016/j.physrep.2010.11.002. [Online]. Available: https://doi.org/10.1016%2Fj.physrep.2010.11.002.

[41] J. A. Barnes and F. Harary, "Graph theory in network analysis," *Social networks*, vol. 5, no. 2, pp. 235–244, 1983.

[42] O. Liang. "How to solder guide for fpv beginners." (Dec. 2019), [Online]. Available: https://oscarliang.com/soldering-guide/.

[43] M. Pearson, "A brief recap of popular logic standards," *EE Times*, May 26, 2004. [Online]. Available: https://www.eetimes.com/a-brief-recap-of-popular-logic-standards/.

[44] D. George. "Micropython – Python for microcontrollers," George Robotics Limited. (), [Online]. Available: https://micropython.org/.

[45] D. George. "time – time related function – micropython documentation." (), [Online]. Available: https://docs.micropython.org/en/latest/library/time.html.

[46] B. Kaliski, *PKCS #5: Password-Based Cryptography Specification Version 2.0*, RFC 2898, Sep. 2000. DOI: 10.17487/RFC2898. [Online]. Available: https://www.rfc-editor.org/info/rfc2898.

[47] D. H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, RFC 2104, Feb. 1997. DOI: 10.17487/RFC2104. [Online]. Available: https://www.rfc-editor.org/info/rfc2104.

[48] F. Courbon, S. Skorobogatov, and C. Woods, "Reverse engineering flash eeprom memories using scanning electron microscopy," in *Smart Card Research and Advanced Applications*, K. Lemke-Rust and M. Tunstall, Eds., Cham: Springer International Publishing, 2017, pp. 57–72, ISBN: 978-3-319-54669-8.

[49] "Qemu: A generic and open source machine emulator and virtualizer." (), [Online]. Available: https://www.qemu.org/.

[50] "Mbed tls," Linaro Limited. (2023), [Online]. Available: https://www.trustedfirmware.org/projects/mbed-tls/.

A
Schematics

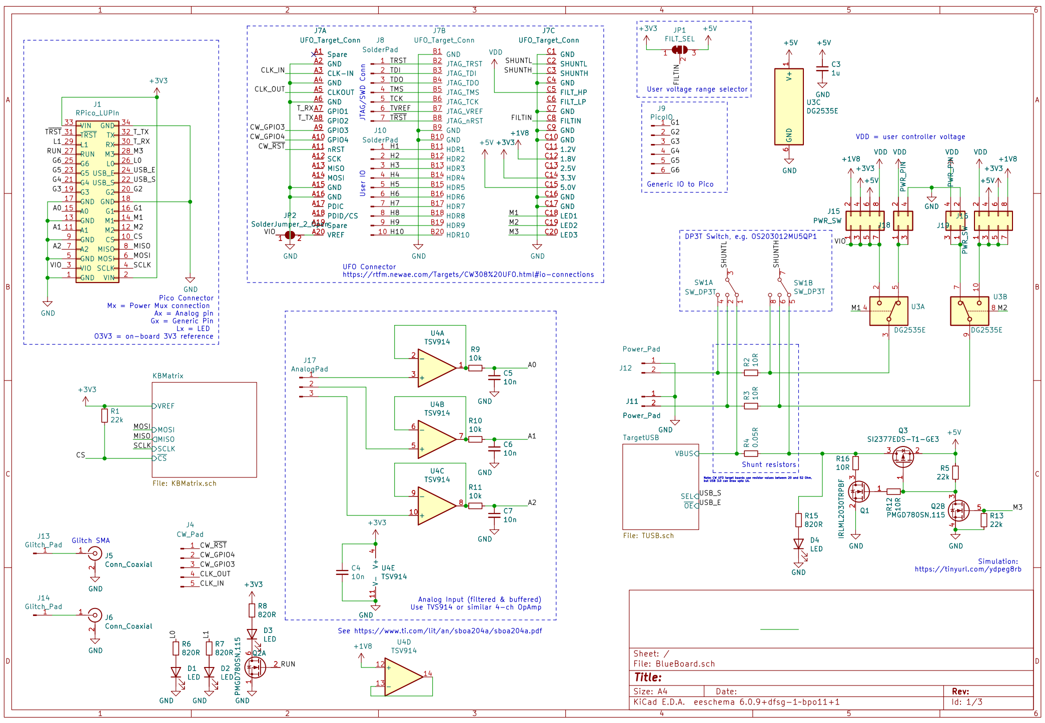


Figure A.1: Main LUPIn board schematic

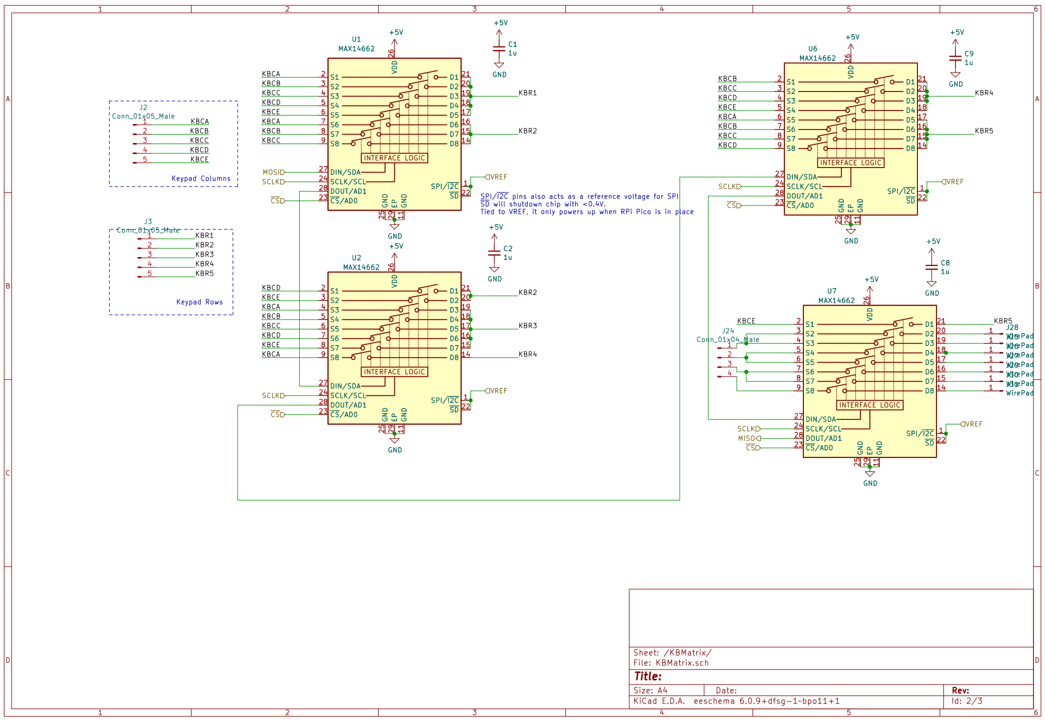


Figure A.2: Main LUPIn board schematic detail: the keyboard matrix switches

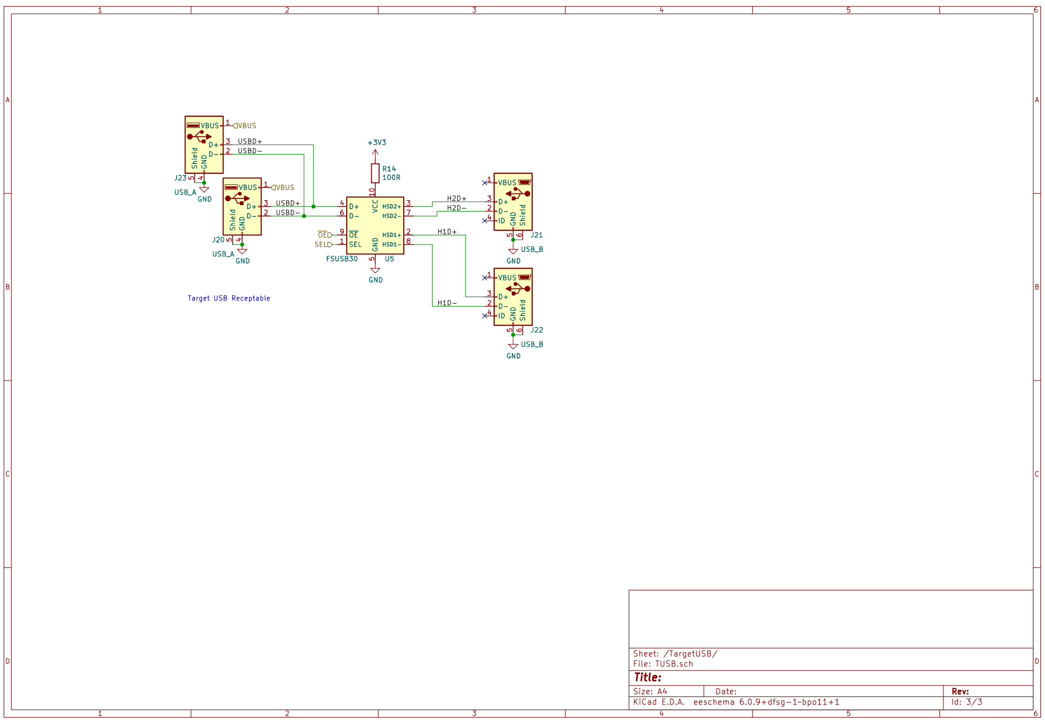


Figure A.3: Main LUPIn board schematic detail: USB subsystem

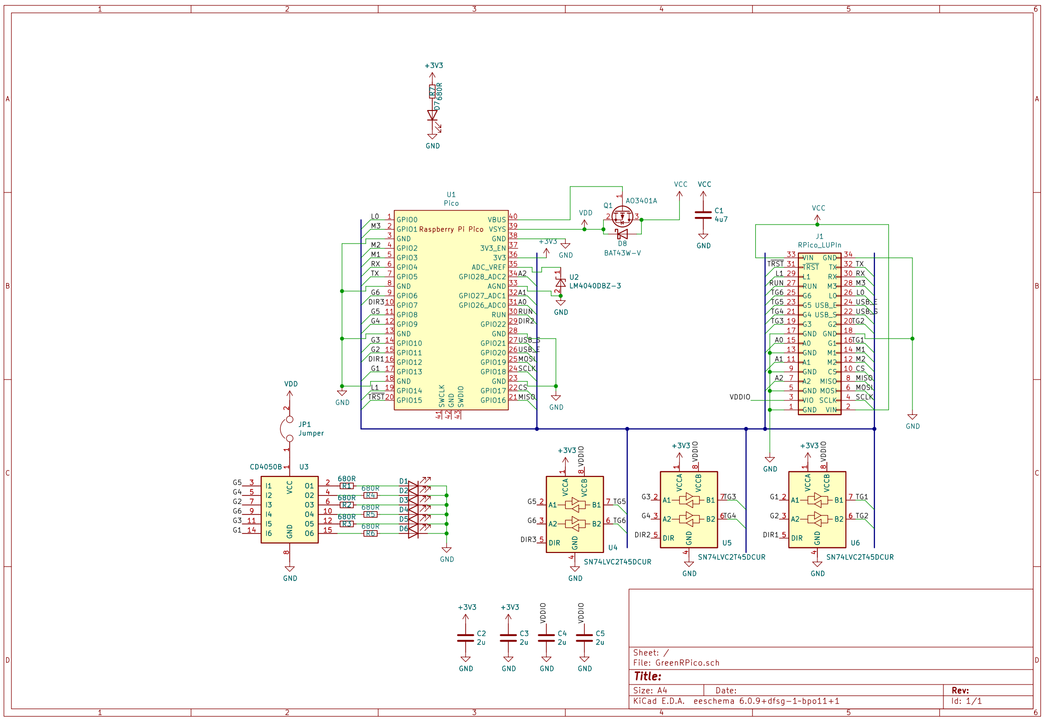


Figure A.4: Raspberry Pi Pico daughterboard schematic