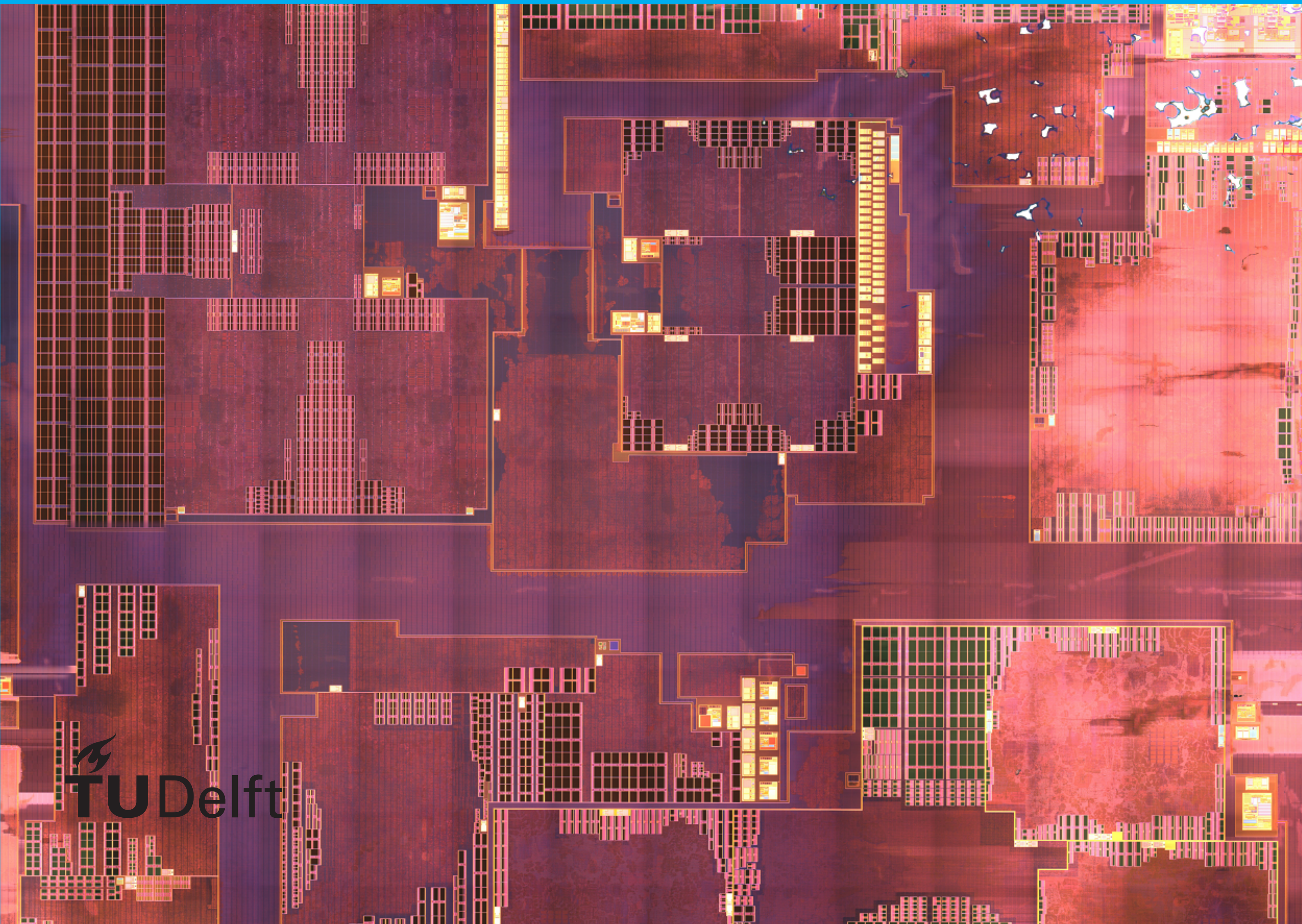
Circumventing Secure JTAG

A detailed plan of attack

M.L.J. van Beusekom



Circumventing Secure JTAG

A detailed plan of attack

by

M.L.J. van Beusekom

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday June 13, 2019 at 9:00 AM.

Student number:	4029127	
Project duration:	April 14, 2018 – January 15, 2019	
Thesis committee:	Prof. dr. ir. S. . Hamdioui,	TU Delft, supervisor
	M. Taouil,	TU Delft
	S. Picek,	TU Delft
	C. Klaver,	NFI

This thesis is confidential and cannot be made public until November 13, 2021.

TU Delft

Abstract

Forensic science is the cornerstone of the modern justice system, as it allows us to analyze evidence in order to discover the truth behind a crime scene. As mobile phones became more important to our daily lives they've taken up a bigger part in forensic research as well. These devices contain digital traces telling us the story of our lives. Digital forensic research traditionally focused on recovering data, in particular data from broken or undocumented systems. While security was not the primary concern of most manufacturers, this has changed in recent times. Cryptography has become a major roadblock in forensic science, which has shifted the focus from recovery towards exploitation.

Traditionally Digital forensic research focused on physical acquisition. While often tools supported logical extraction, this would leave many deleted traces inside of the memory chip intact. Instead methods such as Flasher tools, JTAG-based acquisition, In-System Programming and Chip-off were often used to make images of the physical contents of the memory chip. This would then be analyzed by data-analysts in order to extract evidence from the images. On many modern devices cryptography has ensured that these methods are no longer effective. Software exploitation techniques such as DirtyCow have proven to become much more important in digital forensic science, but more methods should be developed. JTAG is of particular interest in this regard. It is a testing system that was previously left unsecured and providing attackers with an easy way in. More recently manufacturers have started using novel ways of protecting JTAG from malicious use, but very little research exists testing these new security measures. This is a recipe for vulnerabilities.

This thesis focuses on attacking Secure JTAG; Samsung's authentication module for ARM CoreSight. Several potential attacks have been highlighted that are potentially applicable to Secure JTAG. A novel model has been developed that allows an attacker to evaluate the complexity of their attacks. While models such as CVSS and DREAD exist, these focus on threat analysis rather than offensive research. Using this model two attacks were chosen: One focusing on reversing the internal JTAG boundary scan chains and the other attempting to force authentication through fault injection.

In order to reverse the JTAG boundary scan chains a new set of tools had to be developed. Existing tools were often incapable of communicating with the internal scan chains or did not provide enough freedom to enable research. After their development these new tools were able to confirm that the internal scan chains were secured. The internal boundary scan chains will not be a viable attack vector until the authentication mechanism is circumvented.

The second attack focused on a potential fault injection vulnerability in the Secure JTAG authentication scheme. Because Secure JTAG is a purely hardware component it provided a unique challenge. Where normally firmware is analyzed to prove the existence of a vulnerability, Secure JTAG has no such firmware available. This makes discerning between failure because the device is not vulnerable and failure because of chance nearly impossible. To increase confidence in the attack and decrease attack space this project focused on Electromagnetic Side-channel Analysis. By locating the signals produced by Secure JTAG authentication it is possible to greatly reduce the attack space, thereby increasing confidence in the attack. In the end correlating signals were located near the high-power processor of the chip, albeit not the hashing engine itself. This lowered the attack space enough for an eventual fault injection attack.

Preface

This thesis marks the end of the 12 year long adventure that was my formal education. It has been a long road with many an obstacle, but not one i've ever regretted taking.

I'd like to thank everyone involved in this project. First, of course, my supervisors for the opportunity to work on this subject. This is apparently an uncommon thing to say, but i have legitimately loved writing this thesis. I also would like to thank them for their significant support, without which this thesis would no doubt have taken significantly longer to produce.

Secondly i'd like to thank everyone at NFI for their hospitality. For sharing their enthusiasm for their work with me, and teaching me all exciting techniques in the field. Thanks to you, nothing about this thesis felt like work and every day was exciting.

Lastly of course, i want to thank my family and friends for supporting me throughout my entire study. Without you i would not have gotten where i am today.

It's an odd feeling to end my study after so long. But luckily there's many more exciting ventures ahead!

M.L.J. van Beusekom
Delft, June 2019

Contents

Abbreviations

ADI	Arm Debug Interface. 59
ADIv5	Arm Debug Interface Version 5. 4, 25, 50
AHB	Advanced High-performance Bus. 26, 40
ALU	Arithmetic Logic Unit. 62, 65, 67
AP	Access Port. 25–27
APB	Advanced Peripheral Bus. 24, 26
AXI	Advanced eXtensible Interface. 26, 40
BSC	Boundary Scan Chain. 4, 5, 13–15, 17, 20, 22–24, 26, 28, 39, 40, 48–50, 53, 56–58, 69
DAP	Debug Access Port. 23, 25–28, 38, 50, 51, 54
DP	Debug Port. 25, 26
DR	Data Register. 49, 50
EM	Electro Magnetic. 8–11, 39, 51, 52, 54, 58, 60–62, 67, 70
EPROM	Erasable Programmable Read-Only Memory. 10
FIB	Focused Ion Beam. 11, 40, 41, 49, 69, 70
FMP	Flash Memory Protector. 38
IC	Integrated Circuit. 2, 4, 7–11, 15, 42, 54
IP	Intellectual Property. 7, 39, 40, 69
IR	Instruction Register. 56–58
ISP	In-System Programming. 3, 8, 9
JTAG-AP	JTAG Access Port. 28, 29
LIVA	Laser Induced Voltage Alteration. 11, 69
OBIC	Optical Beam Induced Current. 11
PLL	Phase-Locked Loop. 10
SECT	Standard for Embedded Core Test. 15
SJTAG	Secure JTAG. 1, 4, 5, 7, 16, 17, 19, 20, 22, 23, 25, 26, 33–40, 50–54, 59–62, 64–67, 69, 70, 78, 79
SoC	System on Chip. 3, 7–10, 15, 23, 40, 50, 52–56, 59, 61
SSS	Security SubSystem. 26, 28, 38, 39, 56
TAP	Test Access Port. 13, 14

1 Introduction

This chapter introduces the topic of this thesis, it's importance, context, related work and contributions. Section 1.1 presents the motivation and describes the forensic context of the thesis. Section 1.2 discusses historical techniques used in Digital Forensics and presents state of the art attacks on JTAG security mechanisms. Section 1.3 Describes the main contributions of this thesis. Lastly, 1.4 lists the organization of this thesis.

1.1. Motivation

Forensic science is a cornerstone of the modern justice system, as it allows us to analyze evidence in order to discover the truth behind a crime scene. As mobile phones became more important to our everyday lives they've taken up a bigger part in forensic research as well. These devices contain digital traces of our daily activities such as messaging and location history.

Digital forensic research has largely focused on recovering data, in particular recovery from broken or undocumented systems. While security was not generally the primary concern of most manufacturers, this has changed in recent times. Scandals such as the Snowden revelations [54] have underlined the importance of digital privacy for many. Manufacturers have answered this call, which has resulted in solutions such as the standard application of cryptography [21]. Cryptography has become a major roadblock to digital forensics, and has shifted the focus from recovery to exploitation.

There are three main attack vectors in order to defeat device encryption: Attacking the implementation of the cryptography, attacking the secret key of the cryptography or obtaining the key through alternative means. These vectors often have to be combined in order to stage a successful attack.

Cryptography is a difficult field and small mistakes can greatly reduce the computational complexity. By reducing the computational complexity, attacks that once took decades may be reduced to days or hours [30].

Attacking the secret key is similar in nature. While fully random keys are highly entropic and difficult to guess, human-friendly passwords are considerably simpler [13].

A lot of systems, including Android [47], combine the user key with a strong high-entropy key in order to strengthen the encryption against these kinds of attacks. This highly entropic key, commonly referred to as the hardware key, is typically stored in a secure area of the device. Obtaining the hardware key greatly simplifies cryptographic attacks by solving part of the encryption key.

There are numerous attack vectors to exploit a system. Historically, the testing mechanism known as JTAG was one such vector. JTAG is intended to be used during manufacturing in order to verify the correctness of circuits so as to minimize the shipping of defective units. Testing mechanisms like JTAG require full access to the system in order to completely test it. Full unrestricted access leads to obvious security problems [45]. Yet this mechanism was often left open after manufacturing. This enabled attackers to misuse JTAG in order to take complete control of the system. As a result, most JTAG ports are now disabled in an effort to increase device security.

Permanently disabling JTAG has a number of downsides for manufacturers, as this hinders repair and fault analysis. As such a lot of research has gone into developing novel methods to control access to the JTAG testing mechanism while remaining convenient for manufacturers to use. However, comparatively little public research exists which tests the security of these methods. This is a recipe for vulnerabilities and makes JTAG an interesting attack vector. This thesis focuses on attacking Secure JTAG (SJTAG) in order to resurrect

JTAG as an attack vector on modern devices. Re-enabling JTAG would result in the return of a powerful tool in the forensic toolkit.

1.1.1. The Forensic Process

Originating from a need for data recovery, digital forensics has become fundamental to criminal investigations. Virtually all electronic devices, such as personal computers and cellphones, contain a treasure trove of information [37]. Smartphones in particular document a person's entire life. Creating methods to extract information such as incriminating photographs or messages is of great benefit to law enforcement.

Forensic science resolves around the use of scientifically derived and proven methods throughout the following stages [6]:

1. **Recognition** and **Preservation** of *sources*, such as data carriers, so that potentially relevant traces may not be lost.

2. **Acquisition** of the raw *material* from the source, so that they may be analyzed. These include, for instance, a disk image of a hard drive.

3. **Extraction** of *features* from the raw material, e.g., photos and emails from a disk image.

4. **Relate** of the raw traces by correlating it to the known features of the relevant case. For instance, creating a graph of bitcoin transactions makes it possible to spot possible *relations* between suspects.

5. **Evaluation** of the results of the previous steps. Not all results are necessarily relevant to the case, and those that do should be probabilistically tested against several hypotheses.

Each of these stages has their own set of methods, tools and procedures. This is referred to as forensic knowledge, and is used to transition from each stage in the forensic process to the next. Methods comprise the knowledge and expertise required to transition from one stage and onto the next. These are based on scientific research, experiments and reverse engineering. Tools are intended to automate methods in order to make them applicable for case work. While this can include dedicated hardware and scripts for full automation, simpler tools such as a manual or a checklist also fall into this category. Lastly, procedures define how methods and tools are employed during investigations. These are typically standardized in official guidelines or ISO standards such as ISO 177025 [15].

Figure 1.1 shows forensic knowledge used to progress through each stage of the forensic process. By consistently applying and documenting all steps it's possible to maintain a chain of evidence. The ability to verify each step for correctness is critical for traces to maintain their evidentiary weight. If any part of the chain of evidence is breached, it cannot be ensured that evidence has not been tampered with thus making it inadmissible. Using forensically sound methodology as outlined above will keep evidence verifiable and admissible in court.

1.2. State of the Art

Historically, forensic research on mobile devices focused purely on data recovery. Early tools mainly relied on logical extraction, leaving many deleted traces inside of the memory chip untouched [10]. This section starts by discussing traditional methods of forensic data recovery and then discusses modern pitfalls and shortcomings.

- **Flasher Tools**

  Flashing used to be the easiest way to perform data extraction. Flasher tools can be connected to a device interface such as the USB port in order to extract all data. Ordinarily these are used by manufacturers or service centers in order to perform maintenance on their devices, but some are created by hacking groups in order to change the functionality of the device [10].

- **JTAG**

  JTAG is a testing mechanism for Integrated Circuits (IC), which allows manufacturers to control the values of a chip's external pins. This way interconnections can be tested quickly and through a simple interface. For an attacker this level of control can be used to gain complete control of a chip and its surroundings. Flash memory chips are not typically JTAG enabled, instead they are usually connected

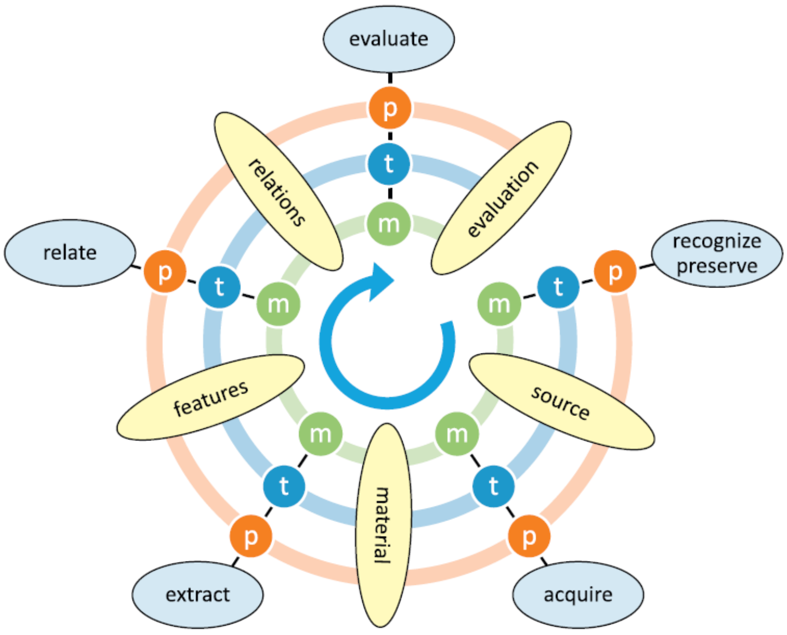


Figure 1.1: A circular graph of the forensic process [6]

to a JTAG enabled processor. An attacker can abuse the testing mechanism to gain access to the flash chip through control of the processor's external pins. JTAG itself is discussed thoroughly in Chapter 3. On modern devices the JTAG port is typically disabled, which is discussed in Chapter 4.

- **In-System Programming**

  The components of a device are typically assembled and then soldered onto the motherboard. Soldering is not done by hand but rather through the use of reflow stations. These reflow stations heat the entire board which causes the solder paste to flow and subsequently attach the components to the board. Data stored on flash memory chips cannot withstand the high temperatures involved during the reflow process. To ensure correctness any data is typically written to the flash memory after manufacturing of the board. A common method is In-System Programming (ISP). This method uses test points on the PCB that allow the manufacturer to connect to the flash memory. If left enabled an attacker can search for and subsequently connect to these points in order to gain direct access to the chip. Parasitic connections to System on Chip (SoC)-memory bus can also be used for ISP based reading of memory.

- **Chip-off**

  The most direct way to dump a memory chip is to physically remove it from the PCB so that it may be read using external tools. While conceptually simple, care needs to be taken not to damage either the memory chip, pcb or the data in the process [10].

On many modern devices, but smartphones in particular, using the methods above to obtain a dump of the flash memory is no longer enough. Android for instance supports full-disk encryption since version 5.0, which is enabled by default on devices that ship with Android 6.0 [21]. As such, digital forensic research has shifted more into the realm of software and hardware exploitation in order to circumvent this new layer of security. Privilege escalation attacks [18] such as DirtyCow [50] are potentially very useful tools in the digital forensic space.

Research on security of the debug system has been one-sided, focusing primarily on novel ways to secure JTAG while retaining its functionality. These have been further described in Chapter 3. Until recently, little research has been done to verify the security of these mechanisms.

Outsourcing testing to third parties is a growing trend among fabrication facilities. With this outsourcing comes the risk of IP piracy and reverse engineering. To counter this it is becoming more common to employ obfuscated test chains. These allow third party labs to verify correctness of the circuit without the ability to meaningfully control JTAG. In 2019 [1] proposed a novel method to counteract scan chain obfuscation by translating it into a satisfiability (SAT) problem. SAT problems are NP-complete problems for which reasonably efficient solvers exist. This allows the encryption key to be derived, thus breaking the obfuscation.

Also in 2019, [41] demonstrated the power of the ARM debug system as an attack vector on unsecured systems. The NAILGUN attack allows an attacker to run arbitrary code and obtain secured information through unsecured debug systems, effectively circumventing the protection supplied by ARM TrustZone.

1.3. Contributions

The primary objective of this thesis is the circumvention of the security provided by Secure JTAG (SJTAG). Thorough analysis has been performed on various hardware attacks. Several security measures for JTAG Boundary Scan Chain (BSC) have also been analyzed including Samsung's implementation. This resulted in various tools and methods that can be used to attack SJTAG.

The main contributions of this thesis are:

- **A classification of hardware attacks in the digital forensic attack space.**

  Attacks have been classified on their intended targets. Specifying whether an attack focuses on data, functionality or intellectual property results in a good overview of the digital forensic toolset for the hardware space.

- **A classification of security measures for JTAG**

  JTAG security has been classified into the three major classes based on their overarching method. Each method is complemented by a discussion of a particular implementation.

- **A new attacker focused model to judge attack complexity.**

  Most vulnerability models are intended for defense rather than offense. While they are excellent in evaluating a need to react, it is not suitable to choose an optimal attack vector. The model outlined in this thesis uses the existing CVSS v3 model [22] as a basis for an attacker oriented version. Usage and accuracy is demonstrated by evaluating potential SJTAG attacks with it.

- **A framework to enable easier attacks and standardization over the debug system.**

  The CoreSight Management script was created in order to address shortcomings in existing tooling. Primarily it provides a powerful scripting engine and a library focused on the target device. Most importantly, it implements JTAG-AP which enables research of the JTAG BSC on Arm Debug Interface Version 5 (ADIv5) supported systems.

- **Confirmation that the internal JTAG BSC is secured on the target device.**

  No security measures were described for the internal JTAG BSC. By converting existing BSC attack methodology to be applicable to the JTAG-AP interface and the creation of the appropriate tools, this thesis showed that the internal BSC is secured.

- **A novel method to prepare the target device for fault injection.**

  Preparing a modern Integrated Circuit (IC) for fault injection and side-channel analysis is a challenging task. This thesis provides several methods to prepare advanced targets for these kinds of attacks.

- **The successful application of side-channel analysis to determine key factors for fault injection.**

  The device under test is a complete black box, which complicates fault injection. In particular, it makes differentiating whether failure is because the device is secure or because of poor luck impossible. With side-channel analysis it was possible to determine the approximate location and timing of a fault, which greatly helps the confidence in an attack.

1.4. Organization

This thesis is divided into 7 chapters. The first three chapters aim to provide the appropriate background and context for an attack on Secure JTAG (SJTAG). The fourth chapter discusses the target, SJTAG, and relevant components of the device under test. The fifth chapter evaluates several potential attacks and describes the testing methodology. Chapter 6 implements the testing methodology and discusses the results. The last chapter concludes this thesis.

In detail, the remaining chapters contain the following:

Chapter 2: Hardware Attacks

- A classification of hardware attacks
- Mapping of existing attacks in the digital forensic space onto the classification
- Discussion of each attack to provide context for attacks.

Chapter 3: Overview of Hardware Testing Mechanisms

- Discussion of important boundary scan standards such as JTAG
- A classification of JTAG security measures.
- Explanation of several JTAG security measures and their mapping onto the classification

Chapter 4: Samsung Secure JTAG

- A detailed description of the CoreSight debug system
- Thorough explanation and discussion of SJTAG and it's components.
- Several attack vectors that are currently blocked by SJTAG.

Chapter 5: Attacks on Secure JTAG

- A discussion of attacks that are applicable on SJTAG
- Specification of a novel model to grade the complexity of attack vectors.
- Application and evaluation of the model on the proposed attacks.
- Attack methodology for BSC reversing attacks.
- Attack methodology for fault injection on SJTAG
- Methodology to decrease the attack space through side-channel analysis.

Chapter 6: Validation and Result Analysis

- A description of the attack platforms.
- Implementation and execution of the Boundary Scan Chain (BSC) reversal attack.
- A description of the trigger setup, a crucial component for side-channel analysis and fault injection.
- An overview of the various testing methodologies used to obtain results.
- Discussion and evaluation of the test methodologies

Chapter 7: Conclusion

- Provides a summary of the entire thesis
- Discusses future work

2
Hardware Attacks

Hardware, as opposed to software, is uniquely vulnerable to attacks on their physical properties. Even if the logic is sound the security of a chip can often still be broken through external manipulation. For instance, the CPU can deny access to the NAND flash memory, but it won't stop a determined attacker from physically removing and reading the memory externally. This chapter covers a variety of hardware-based attacks that may prove useful in circumventing Secure JTAG (SJTAG)'s security. Section 2.1 presents a classification of hardware attacks. Sections 5.1.1, 5.1.2 and 2.4 focuses on attacks that can be performed in the field. These are grouped by attacks on data, functionality and intellectual properties, respectively. Section 5.1.4 provides a brief discussion on hardware attacks in general.

2.1. Classification of Hardware Attacks

Hardware encompasses an vast amount of objects, from the smallest transistors to the largest space stations. It follows then that attacks on hardware can be performed in a myriad of ways. The attack space remains vast even when limiting attacks to System on Chips (SoC). In order to properly discuss and understand attacks in the hardware space it is necessary to classify attacks further. A useful starting point is asking four simple questions:

- **What** is being attacked?

  The first question refers to the **target** of the attack. This classification observes 3 types of targets to be attacked: Intellectual Property (IP), data and functionality. Attacks may focus on the IP of an Integrated Circuit (IC) to learn more about its design. This may be used to reproduce it for financial gain. Other attacks may focus on data inside of the IC such as cryptographic keys. Attacks on the functionality of the IC focus on modifying behavior such as bypassing a password check.

- **How** is it being attacked?

  This question refers to the type of **technique** used in the attack. These can be classified as invasive, non-invasive and semi-invasive [56]. Invasive attacks require the packaging of the IC to be opened so that the silicon die can be attacked directly, while non-invasive attacks do not require the packaging of the system to be removed. Semi-invasive attacks do remove the packaging but do not tamper with the silicon die otherwise.

- **When** is it being attacked?

  An IC or device goes through multiple **phases** throughout its lifetime. First they are designed by the manufacturer in order to meet user demands. Next, they are manufactured at a factory according to specifications. Finally, the completed product is delivered to the end-users in the field.

  Each of these phases provide an attacker with various capabilities. During the design phase, for instance, an attacker could modify the design to include a hardware trojan [7]. During the manufacturing phase certain components or secrets may be stolen to facilitate cloning and reproduction of the device [23]. Devices are typically easiest to acquire in the field, but as a result many of the attacks that are feasible during design or manufacturing phases are no longer viable. However, the lack of manufacturer oversight does provide an attacker with a certain degree of freedom. While the design and manufacturing processes can be tightly controlled, it is not feasible to have this level of control when the devices are in the hands of end-users.

- **Where** is it being attacked?

  A device consists of several components which allows attacks to be performed on several **locations**. Generally these can be classified in 3 types: Access signals, Nodes, and Inter-Node communication.

  Access signals refers to the intended interfaces for the end-user, such as an USB connection. These attacks typically exploit intended operations in unintended ways. JTAG based attacks described in Section 2.2.3 are an example of this.

  Nodes refers to the various components of a device, such as the SoC, memory and the internal dies. Electro Magnetic (EM) fault Injection as described in Section 2.3.2 is an example of a node-based attack.

  Lastly, inter-node communication consists of the communication lines between the various nodes. For instance, the connections between the SoC and the flash memory chip. These may be used in attacks such as In-System Programming (ISP) described in Section 2.2.2.

Within the field of cyber security it's often good to study who the attacker is and why an attack is taking place. These questions focus on threat analysis rather than offensive security research. As such these questions are considered to be out of the scope of this Thesis. This classification has been summarized into Figure 2.1.

The remainder of this chapter discusses various attacks that are possible in the hardware space. Given the forensic context of this Thesis only attacks that can be executed in the field are also considered to be out of scope. The attacks will be classified by target in order to present a digital forensic tool-set rather than a full overview of hardware attacks.

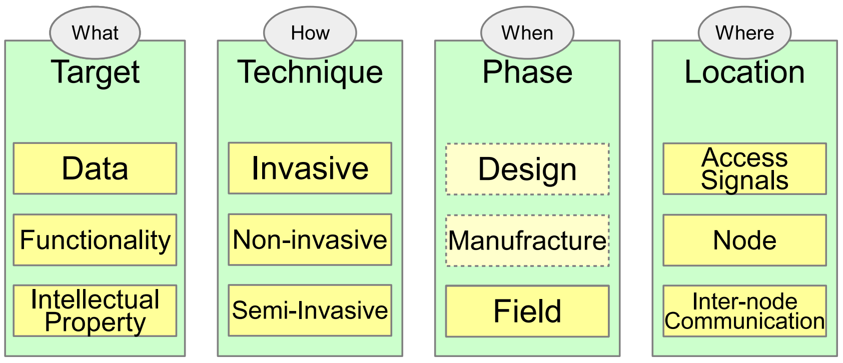


Figure 2.1: Classification of attacks in the hardware space

2.2. Attacks on Data

Attacks on data focus on retrieving secrets from inside of the Integrated Circuit (IC) or modifying data in order to alter functionality. It could be argued that attacks on data that alter functionality should be classified as an attack on functionality. However, as the IC is still functioning as designed these attacks are treated as a side-effect of data manipulation.

2.2.1. Chip-off

Chip-off attacks are conceptually very simple. Systems often require some kind of non-volatile memory chip in order to store persistent data. A chip-off attack removes this chip and accesses it through external means [9]. In practice there are a lot of variables to consider such as the protocol used in the chip, its connections or its physical properties. For example, a chip may not retain its data at the temperatures required to desolder it. While the manufacturing process of modern systems has made it more difficult to perform chip-off attacks, the primary obstacle is the application of cryptography. Encrypted data cannot be interpreted or modified in a meaningful way without breaking the cryptography.

2.2.2. In-System Programming

During manufacturing of the IC flash chips will be put into a reflow oven in order to solder the components onto the PCB. Data consistency cannot be guaranteed when exposed to the conditions of the reflow process, so any programming must be done after assembly. In-System Programming (ISP) is a method used by manufacturers to provide connection points to be able to program and test flash memory. From an attacker's context these same points can be used to read and modify the flash memory. The effects of this attack are similar to chip-off attacks, but have the advantage that the chip does not need to be removed.

2.2.3. JTAG

Testing whether all components are assembled onto the board correctly is crucial for the manufacturer in order to minimize defective units shipped. It is generally cheaper to intercept defective units during production, compared to the costs of shipping and replacing a defective item. Testing efficiently can increase profits and customer satisfaction [5].

In order to be able to detect faults many ICs are designed for testing and include specialized testing hardware. The most popular hardware testing standard is JTAG, which is discussed in more detail in Chapter 3. In order to properly test the ICs the testing hardware needs to have full access and control over the chip. An attacker can exploit the testing hardware to gain full control and thus extract or modify any data contained within the IC [8].

In the past access to the testing hardware used to be completely open, which made the testing hardware an excellent point of attack. These days manufacturers have caught on to these attacks and have started securing the testing hardware. As such it is generally no longer a viable attack vector. The security measures are further discussed in 3.

It should also be noted that JTAG can also be used to attack functionality. In forensic context it is commonly used as a data extraction tool, which is why it has been categorized as an attack on data.

2.2.4. Side-channel Analysis

When an IC performs an operation it will usually generate some side-effect. Analyzing these side-effects and using them to learn more about the operation is the essence of side-channel analysis.

The simplest form of side-channel attacks are one dimensional and only measure time. These are also referred to as timing attacks. Timing attacks hinge on the concept that one branch of a program does not have the same execution time as another branch. For instance, an incorrect character during a password check may terminate execution immediately. Conversely, a correct character would progress further into the operation, thus resulting in a different execution time. This information can be used to distinguish correct characters in a password, thus greatly reducing time-complexity when brute-forcing. A more powerful application of this attack would be to derive the private key of an ECC cryptosystem [11]. To prevent these kinds of attacks the designer needs to take care to make processing time of all branches indistinguishable.

Similar analysis can be done by measuring power consumption over time, thus making the attack two-dimensional. These are referred to as power analysis attacks. A device's power consumption is dependent on the number of state changes of its transistors [56]. By analyzing the power spikes it becomes possible to distinguish persistent values used in computations such as the carry bit during shift operations. There have been various attacks on cryptosystems using both Simple Power-Analysis (SPA) [44] and Differential Power Analysis (DPA) [28]. In SPA the power consumption is directly interpreted by an attacker, while in DPA statistical analysis is used to detect small differences in power consumption.

A third side channel attack is through electromagnetic analysis. Electronic devices such as microchips leak Electro Magnetic (EM) radiation[19]. Like with power analysis, the amount of radiation changes when a CMOS gate changes states. By positioning an antenna over the IC it's possible to analyze the changes in radiation. This is especially interesting for System on Chips (SoC) as the location of the antenna is correlated to the components located inside of the SoC. Take note that this adds a spatial dimension, making it a three-dimensional side-channel attack.

2.3. Attacks on Functionality

These types of attacks focus on temporarily altering the behavior of the Integrated Circuit (IC) through external means. As noted in Section 5.1.1 it is possible to alter the behavior of the device by altering the data. However, as this is still correct behavior of the IC these attacks fall outside of the realm of attacks on functionality. Instead, attacks on functionality focus on generating incorrect behavior of the IC in order to reach a desired outcome. For instance, forcing a password check to report an incorrect result may lead an attacker to be authenticated using an invalid password. Incorrect behavior may lead to access or modification of data. This is considered to be a side-effect of attacks on functionality and is not classified as attacks on data.

2.3.1. Microprobing

Microprobing used to be an incredibly important attack [56]. It an attack on the intercommunication of the die. To gain access to these signals the die has to be exposed by decapsulating the IC. By also removing the passivation layer the internal connection lines become accessible. Using a microscope and small probing needles it's possible to connect to these lines. This allows an attacker to either eavesdrop on the chip or to inject signals. Due to the increased complexity and reduced size of modern ICs these attacks are becoming extremely hard.

2.3.2. Fault Injection

ICs are designed to operate within given specifications such as a certain voltage level or clock rate. Correct operation cannot be guaranteed when operating an IC outside of its specification. This is the essence of all fault injection attacks [56]. By briefly running the system outside of the specifications it is possible to generate an error in a way beneficial to an attacker. Password checks, for instance, could be skipped over completely.

There are several methods of fault injection, each with their own properties. These are discussed in the rest of this section

2.3.2.1. UV Attacks

UV attacks is one of the older variants of fault injection and a bit different to the other fault injection attacks covered in this chapter. It is an attack specific to Erasable Programmable Read-Only Memory (EPROM) chips. EPROM chips are erased when exposed to UV light after which they can be reprogrammed. Rewritable EPROM chips have a small window in the IC for this purpose. This window can be used to expose the memory to UV light. Some manufacturers use EPROM as one time programmable memory by removing the window. An attacker can expose the die once more through decapsulation, allowing it to be erased with UV light once more [56]. By exposing only part of the die to UV light it is even possible to target specific bits for erasure. This can for instance be used to remove important security bits without erasing critical data.

2.3.2.2. Clock Glitching

A processor is designed to run at a given clock speed so that it can be ensured operations are completed before the next clock tick. By briefly raising the clock speed past its specification it is possible to introduce errors during execution. This happens, because an instruction has not yet been loaded into the cache from memory in time for processing. There exist many more potential failures, but these failures can often be modeled as skipped instructions. These attacks are referred to as glitching attacks [56]. Modern System on Chips (SoC) are typically no longer vulnerable to clock glitching due to the application of Phase-Locked Loops (PLL). PLLs are designed to absorb jitter in order to produce a stable frequency. This causes them to absorb a glitch which makes clock glitching ineffective.

2.3.2.3. Voltage glitching

Voltage glitching [56] is very similar to clock glitching. An IC is designed to operate within a certain voltage range. When briefly underpowering an IC it can, for instance, no longer be guaranteed that every CMOS gate transistions states correctly. This causes errors that can generally be modeled to skipped instructions. Alternatively, by supplying too much power to the IC it is possible to create floating signals which can similarly lead to errors. Caution needs to be taken when supplying too much voltage, as it is easy to cause permanent damage.

2.3.2.4. Electro Magnetic Fault Injection

Strong Electro Magnetic (EM) fields are known to affect operation of electronic circuits. By using a coil to apply a high energy EM pulse it is possible to generate a fault on a specific location in the IC [16]. Similar to other techniques employed this causes transistors to enter an unintended state, thus leading to skipped instructions. EM fault attacks are interesting because they can be applied directly to a critical area of an IC or SoC without opening the chip. This makes it a more targeted attack than Clock and Voltage glitching.

2.3.2.5. Optical Fault injection.

Optical fault injection is similar to EM fault injection. Silicon is sensitive to light and transistors will start conducting when exposed to a powerful light source [57]. By decapsulating the IC it is possible to fire a laser onto the die. This makes an attack considerably more accurate than EM fault injection. It is even possible to flip specific values of a register using these techniques [56]. However, this accuracy is also to its detriment. Modern ICs are very dense, which makes finding the appropriate attack position considerably harder than when using EM attacks.

2.3.3. FIB Modifications

Connecting to the internal connection lines is harder as the fabrication technology gets smaller, and will need more sophisticated tools. A Focused Ion Beam (FIB) workstation is one of the best tools for technologies smaller than 0,5 µm [56]. Similar to an electron microscope, it uses accelerated gallium ions to create an image at a resolution of as low as 2.5 nm. However, unlike an electron microscope a FIB can be used to add and remove chip material. This allows an attacker to modify a chip's behavior by modifying the circuit. For instance, a line dedicated to monitoring security could be connected to either the ground or the supply line, thus forcing it into a specific state.

2.4. Attacks on IP

Attacks on intellectual properties are normally executed in order to make a profit. For instance by learning about the design of an Integrated Circuit (IC) it's possible to replicate it and sell it. Another example is faking the design of a cheap product to look like its expensive counterpart. These attacks on IP do not typically apply to a forensic context, but some methods can be used in order to learn more about the target. This section focuses on detailing some of the methods that can be used in a forensic context in order to enable further attacks.

2.4.1. Backside Imaging

Visual observation is a good first step to gain understanding an understanding of the IC. By putting a silicon die under the microscope it used to be possible to see the structures and gain some understanding on the chip. However, as the technology gets smaller and denser, visual inspection becomes more difficult. This is in part because the features are smaller, but mostly because of multiple metal layers blocking the view. An approach is to use infrared light and a sensitive camera to observe the chip from the backside. However, modern chips use highly doped silicon, which is less transparent to sensitive light so a stronger light source needs to be used.

2.4.2. Reverse Engineering

Reverse engineering is useful but a difficult method for understanding the chip. In order to reverse engineer a chip all layers of the die are removed and photographed. This allows a knowledgeable attacker to document the entire internal structure of the chip.[56] After all transistors and interconnections are documented a knowledgeable attacker may be able to understand the structures. Depending on the device, operation does not just depend on the internal structure but also on program code. In these cases the program code needs to be extracted and analyzed as well.

2.4.3. Laser scanning

Optical Beam Induced Current (OBIC) and Laser Induced Voltage Alteration (LIVA) are the two major laser scanning techniques. LIVA is used on a powered system in order to find the currently active areas. It works by monitoring the voltage changes of the power supply while a laser is scanned across the IC surface[56]. OBIC scans are supplied to an unpowered chip with its power supply pin connected to a current amplifier. A power current is generated when a laser is scanned across the surface of the IC. This current can be used to detect high-doped areas in the IC.

2.5. Discussion

A lot of the described attacks are no longer as effective as they initially were. Some, like JTAG attacks, are the result of deliberate countermeasures by manufacturers. Others, like backside imaging, are a result of the increased complexity of Integrated Circuits (IC).

JTAG attacks are an interesting case because manufacturers do not typically want to remove their own JTAG access. Restoring access to JTAG would once again grant access to one of the stronger attacks on data.

3 Overview of Hardware Testing mechanisms

Testing is a crucial part of manufacturing, and special testing mechanisms such as JTAG Boundary Scan Chain (BSC) have been developed to aid in this endeavor. This chapter discusses an overview of BSC standards and their security. However, providing high-privilege level access to the hardware also opens up a very powerful attack vector for malicious entities. As such, various mechanisms have been developed in order to keep BSC testing mechanisms secure while maintaining the advantages. Section 3.1.1 discusses the JTAG BSC standard and several extensions. Section 3.2 provides a novel classification of various ways to secure JTAG. Sections 3.3, 3.4 and 3.5 discusses the classifications and provides examples for each. Finally, 3.6 discusses the various security solutions in general.

3.1. Boundary Scan Standards

3.1.1. IEEE 1149.1 - JTAG

Testing circuit boards has become more difficult over time. As the number of components on a circuit board increased, so too has the number of pins that would be needed to access and test these components. Miniaturization has further compounded this problem as pins became harder or even impossible to reach with external tools. In response to this, the IEEE 1149.1 Standard [2] was developed by JTAG Technologies. It specifies a Boundary Scan Chain (BSC) implementation more commonly referred to as JTAG. It allows access to the pins of all JTAG compliant devices on the PCB through a serial connection, thus greatly improving testability. A basic layout is provided in figure 3.1.

The JTAG standard consists of three important components:

- The BSC register cells
- The Test Access Port (TAP)
- The TAP controller

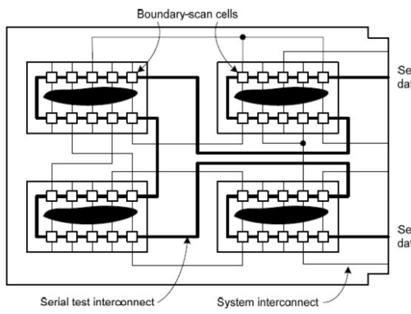


Figure 3.1: Example layout of a BSC equipped system [2].

The BSC register cells are placed on the pins of a compliant component. They consist of two flip-flops and the logic to support their operation. One provides the ability to capture and control the values of the pin, while the other is used to buffer control values before applying them on the pin [2].

The specification for the Test Access Port (TAP), provides four additional pins that are used for the control of the BSC mechanism. It requires four pins to be included into the design: one for the clock (TCK), another to enable various test modes (TMS), and two for in- and output of boundary cell data (TDI and TDO, respectively). These four lines can be daisy chained along various JTAG compatible components in order to provide full access to the system without requiring additional TAPs [2]. TMS, TDI and TDO are all serial connections that are sampled on the rising edge of TCK. The TAP controller controls JTAG's internal logic such as the instruction registers and associated circuitry.

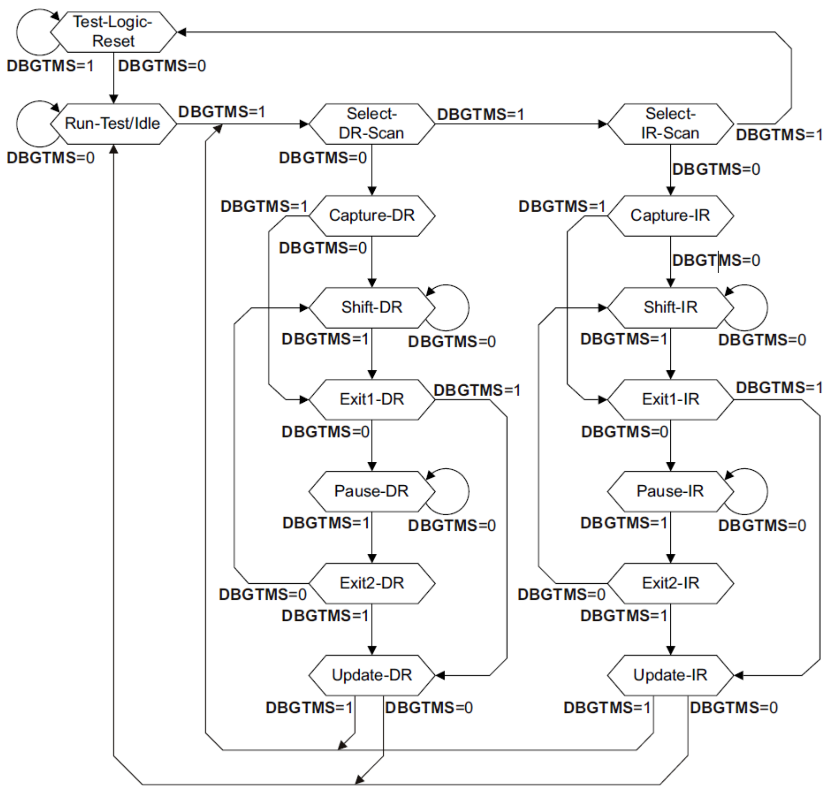


Figure 3.2: The JTAG State Machine [31]

3.1.1.1. The JTAG State Machine

The TAP controller implements the state machine shown in Figure 3.2 in order to control the BSC. The TAP controller state is controlled by the serial data on Test Mode Select. It has sixteen states, eight for the data registers and eight for the instruction register with similar functionality. The most important states are:

- Capture-DR: loads the input cells on the data registers as specified by the currently loaded instruction.
- Capture-IR: loads a fixed pattern into the instruction shift-register that causes the controller to exit it's current state.
- Shift-DR/Shift-IR: Allows to shift data from TDI onto the data register / input register and out into TDO.
- Pause: Pause a shifting operation
- Update: makes the currently shifted in value the live value.
- Test-Logic-Reset: resets JTAG and sets IDCODE as the default instruction

All other states are temporary states that exist only to assist with transitions.

3.1.1.2. Standard JTAG Instructions

Instructions can be shifted into the instruction register for more specific behavior. The standard specifies a number of required and several optional instructions. For this project, the most useful standardized instructions are:

- IDCODE: Sets the data register to access the IDCODE register, which can be used to identify components on the BSC
- BYPASS: Allows to temporarily disable the BSC registers of this component to speed up testing.
- SAMPLE: Allows to take a snapshot of normal operation of the component to be taken and examined.

- PRELOAD: Allows data to me shifted into the BSC register while the component is still in normal operation mode.

- EXTEST: Captures the current pin state and drives the value loaded in the BSC register onto the pins.

- INTEST (optional): Captures the pin state of the internal logic, and drives the values loaded into the BSC register onto the inner chip logic.

Additional Non-standard instructions, such as authentication registers, may be defined by the manufacturer and will need to be investigated separately.

3.1.2. IEEE 1500 - SECT

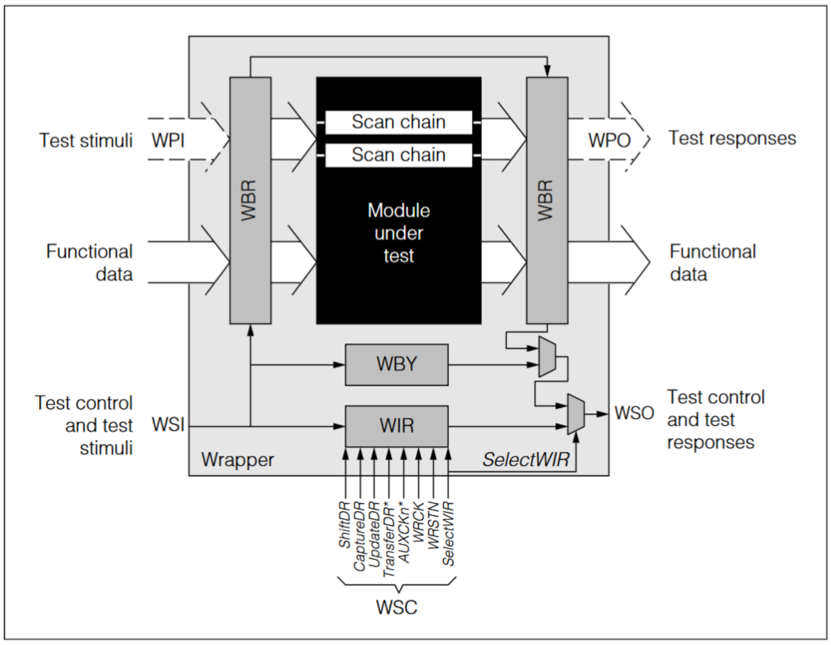


Figure 3.3: Architectural overview of the IEEE 1500 Sect wrapper. [36]

In order to increase functionality and performance the previously relatively simple Integrated Circuits (IC) have evolved into complete packages called a System on Chip (SoC). SoCs include numerous components and modules previously covered by several ICs, many not originally developed by the manufracturer. JTAG test chains need to be calculated for each chip's specific design, which becomes exceedingly complex with the highly modular design of SoCs. As such, treating the entire SoC as one entity is not ideal.

The IEEE 1500 Standard for Embedded Core Test (SECT)[36] is an extension of the original IEEE 1149.1 JTAG standard. It aims to expand the JTAG standard with plug-and-play interoperability, thus allowing for component manufacturers to design and exchange tests for the components in the SoCs. It achieves this by implementing wrapper around components of the soc, shown in Figure 3.3, and a formalized language: Core Test Language (CTL) [58]. CTL is used to formalize tests for components and is what allows tests to be shared. The wrapper interprets CTL and executes the intended test. When used together each component has it's own isolated scan chain with it's own associated tests. This allows component tests to be shared and reused over different SoC designs. The wrapper is otherwise conceptually similar to JTAG and further discussion is out of the scope of this thesis.

3.1.3. IEEE 1687 - IJTAG

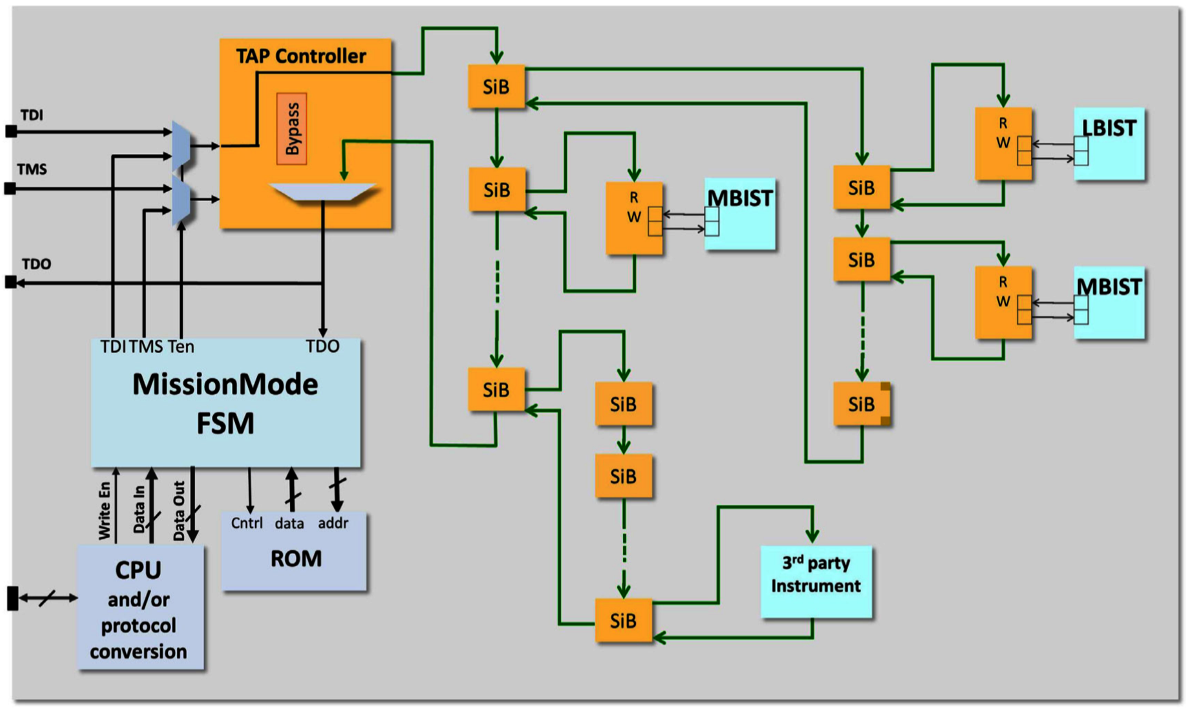


Figure 3.4: An example Layout for an IJTAG enabled system [38].

IJTAG, or instrument JTAG, is an important extension of the IEEE 1149.1 JTAG standard. It aims to extend JTAG with standards to manage configuration, operation and collection of data from embedded circuitry [26].

One problem with the JTAG standard surfaces when a JTAG compatible circuit contains tens or hundreds of components. These cases can result in extremely long scan paths with long test times as a result. Individual components can be bypassed in order to reduce the length. However, bypassing the various components requires a lot of instructions which is also considered to be inefficient.

IJTAG resolves this with the introduction of Segment-Insertion Bits (SIB) that allow the scan chain to be of variable length. These are single bit registers that, when closed, allow a shortcut on the segment of the scan chain. Thus potentially bypassing an entire network of instruments. The SIB can also be used in combination with a multiplexer, in which case it can be used to choose between alternative scan paths. These are referred to as Scanmux Control Bits (SCB).

3.2. Securing JTAG

One major problem in investigating secured variants of JTAG is the lack of standardization. As such, implementations may vary significantly between manufacturers. In order to aid in the conceptual understanding of Secure JTAG (SJTAG), figure 3.5 classifies a number of security measures for JTAG that can be found in the public domain. Examples will be discussed more thoroughly in chapter 3.2 and compared in chapter 3.6.

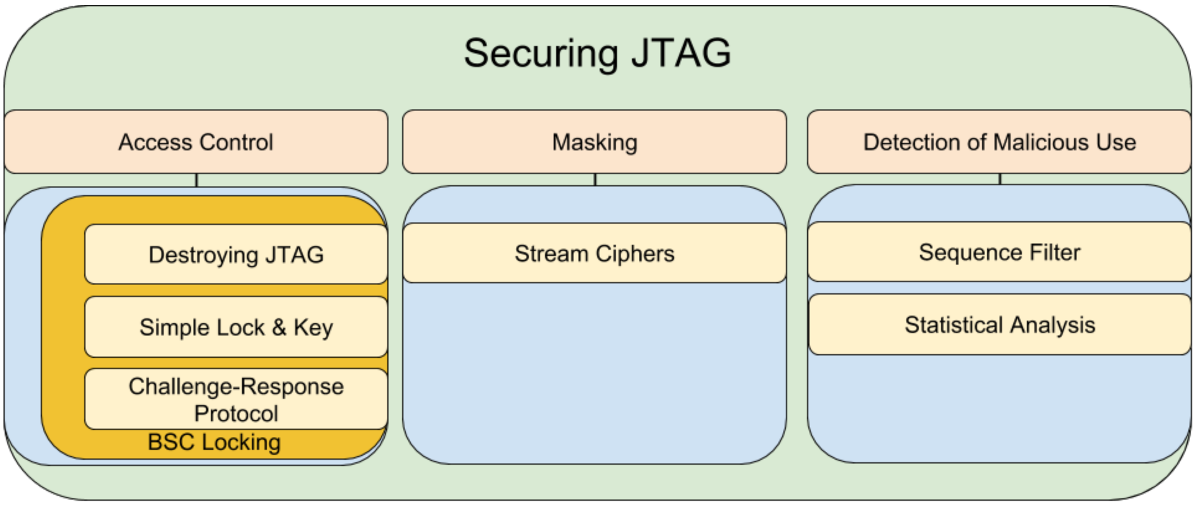


Figure 3.5: Classifications of security measures for JTAG.

There are three major schools of thought within JTAG security. The most important one of which is access control. These systems focus on denying access to the TAP controller, usually until the correct authorization can be determined. Masking is a method of protection where obfuscation of the data streams is central. While they do not block access to the TAP controller, they typically use encryption to make communication impossible. Detection of malicious usage is the class with the most recent advancements. Systems within these classes typically provide free access and rely on monitoring to detect a possible attacker in order to generate a response. They will often be used in combination with the other two classes

The Exynos 7420's implementation of SJTAG is equipped with a form of challenge-response authentication. These protocols are similar in concept but can still differ significantly in implementation, as such three sub-classifications for it have also been provided.

This remainder of this chapter discusses specific implementations of the classes mentioned in chapter 3.2. While there exist some variants of these implementations, on the conceptual level they are too similar to make a meaningful distinction for the scope of this paper.

3.3. Access Control

Access control mechanisms focus on restricting access to the JTAG Boundary Scan Chain (BSC) by (temporarily) disabling it. There exist various public implementations of this, and this section aims to provide an overview of various methods to implement access control on JTAG.

3.3.1. Disabling JTAG

The simplest and the most effective solution of securing JTAG is destroying the JTAG port. This can be done in many ways, ranging from physical destruction of the connections to burning a security fuse. While very secure, this also makes JTAG unavailable for fault analysis by the manufracturer [14]. Additionally, depending on the method used, it may be possible for an attacker to repair JTAG access. Securing JTAG without permanently disabling access is typically preferred on most chips because of these reasons.

3.3.2. Simple Lock & Key

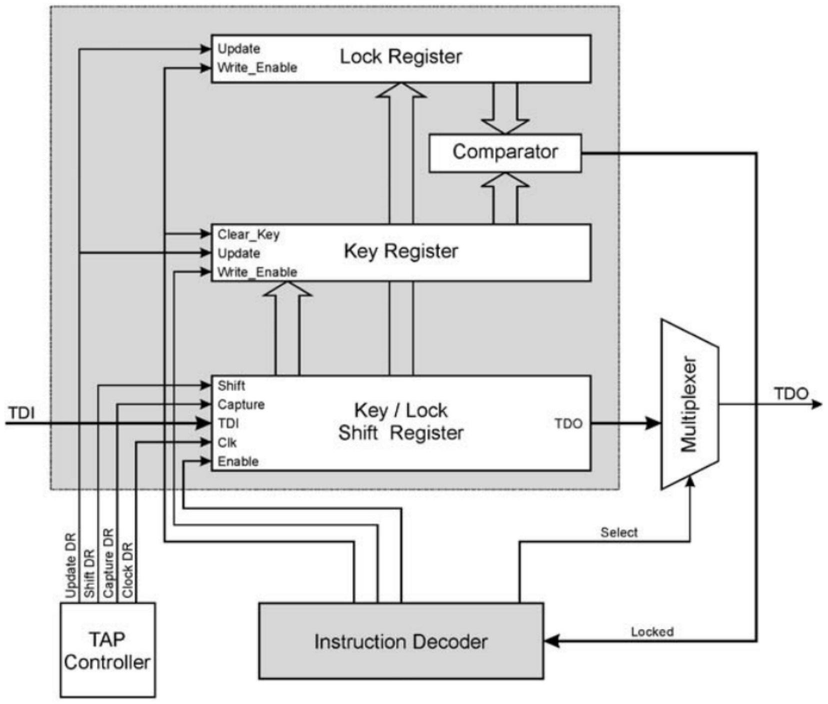


Figure 3.6: Structure of the locking mechanism [42].

The second class of security is a simple locking mechanism. This particular variant introduces three additional registers: the Lock register, the Key register and the Lock/Key register. Locking is done by shifting the LOCK instruction into the instruction register. This allows the user to shift in the key and enter it into the Lock register using Update DR. Unlocking is done in a similar fashion with the Key register and the UNLOCK instruction. All instructions, except UNLOCK, are mapped to the harmless BYPASS instruction while the chip is in a locked state [42]. Other implementations exist and may improve on security by, for instance, using a hash and a hashing engine to avoid storing the key in the system directly.

3.3.3. Challenge-Response Protocol

Challenge-response authentication is a commonly used form of authentication that relies on public key cryptography. According to the protocol the verifier generates a random secret, and the prover uses this secret to prove he has knowledge of the access key. Commonly, this is done using public key cryptography where the prover can encrypt the secret using his private key. The verifier can then decrypt the data with the public key to recover the random secret to confirm the identity. There are some variants to this protocol within the JTAG ecosystem, including the one found on the Exynos 7420 [35]. As such, the public variants are discussed in the remainder of this section.

3.3.3.1. Challenge-response with SHA-256

The implementation discussed in [14] uses a SHA-256 hashing algorithm to provide the necessary authentication to JTAG [14]. It aims to provide a basic protection scheme that doesn't block access entirely, but is more secure than a simple password.

The implementation requires a secret key to be stored on the chip as well as a good random number generator and a SHA-256 hashing engine. In order to verify authentication, the device generates a random number and communicates it to the user. The user appends their secret key and hashes both with SHA-256. When this hash is communicated back to the device, it can generate it's own version of the hash using the stored secret key and random number to match it against the user's. If the hashes are equal, access is granted.

3.3.3.2. Challenge-response with PKC

Protected JTAG [12] is an implementation of the Challenge-Response authentication with public key cryptography. It's designed to only restrict the more sensitive internal processor functions, not the standard circuitry debugging ones.

Protected JTAG introduces three protection levels and four access modes, labeled PL0 through PL2 and AM0 through AM3 respectively. The protection levels define the security of the device, while the access mode is a configurable attribute that defines a default protection level with the ability to lower it. [12]. The protection levels can be summarized as:

- PL0 is the fully protected level and limits user access to external functions such as interconnections and the built in self test.
- PL1 extends the functionality found in PL0 by allowing access to registers, CPU and programmable flash memory.
- PL2 is the lowest level of protection, providing full access and is identical to JTAG.

And the access modes can be described as:

- AM3 is a non-protected access mode, which can be utilized during development to gain full unrestricted access. As such, the default protection mode is PL2.
- AM2, low protection mode, can be used during development once the devide's secure features have been tested and verified, and can be locked off. While the default access level is PL1, AM2 includes a feature to temporarily allow an authorized user to lower the protection level to PL2.
- AM1 is the high protection mode, and is typically used when a device is delivered to the customer. With the default level of protection set to PL0, it protects the device from being attacked through the JTAG port. Still, as JTAG is sometimes instrumental in recovering data, the protection level can be lowered to either PL1 or PL0 by an authorized user.
- AM0 is the maximum protection mode. The default protection mode is PL0 with no ability to downgrade it. It's intended for very sensitive devices that contain data that's not meant to be accessed by anyone.

Because Protected JTAG is implemented on an embedded device, it has limited database functionality. As such, authorization is handled through a trusted authorization server. In order to ensure authorization is done securely, a challenge-response based identification algorithm is used. The Protected JTAG module is outfitted with the ability to create challenges and verify responses to enable it. The authentication protocol is based on elliptic curve encryption, where the device holds the public key and the remote server holds the private key.

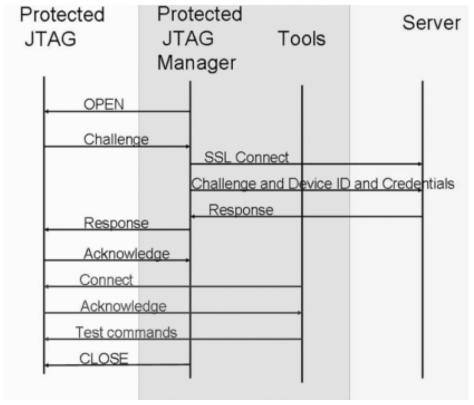


Figure 3.7: The authorization sequence found in Protected JTAG [12].

The authorization sequence, as displayed in figure 3.7, works as follows: First, the device generates a challenge that includes the requested access level and a random component, and signs it with the public key. This challenge, along with the device ID and the user's credentials, can then be communicated to the authorization server. Using the credentials, the server can then verify if the user's authorization matches the requested access. The device ID is used to find the correct private key, and the server generates an appropriate response with it. Once communicated to the Protected JTAG module, it can verify the response's authenticity using it's public key and grant the requested access.

3.3.3.3. Challenge-response with a certificate authority

Secure JTAG Implementation using Schnorr [17] is a Challenge-Response implementation that, unlike the name implies, is unrelated to Samsung's implementation of Secure JTAG (SJTAG). Previous approaches suppose a one-way authentication, where either the server or the circuit is considered trusted. This solution is suitable in cases where neither the circuit or server is trusted.

This solution presents another use of public key cryptography: Using Schnorr protocol it is possible to ensure that both the circuit and server can be trusted. In the Schnorr protocol, the prover (such as the user) has a signed certificate containing his public key. This certificate has been signed with the authentication server's private key. The validator (the circuit) can then validate a user's identity using the authentication server's private key, and extract the prover's public key. The validator then signs his own public key using the prover's, and communicates it to the prover. This securely exchanges public keys, which can then be used for safe communication.

3.3.4. Boundary Scan Cell Locking

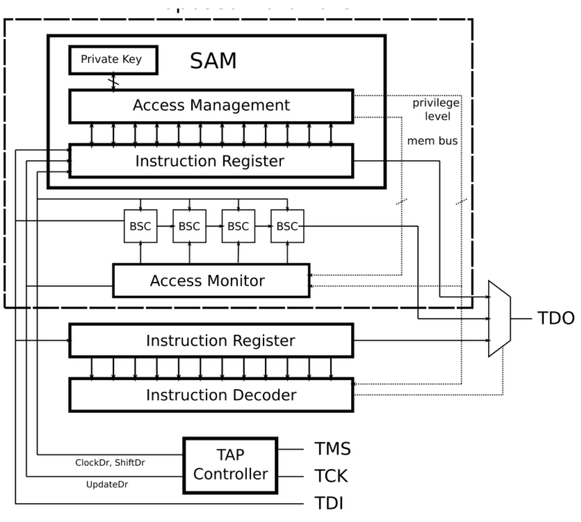


Figure 3.8: Overall architecture of BSC locking [46].

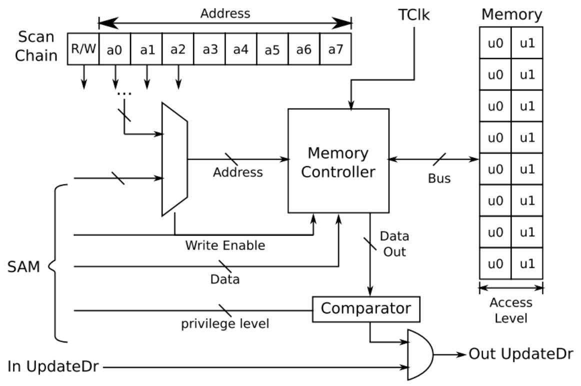


Figure 3.9: The access monitor of BSC locking [46].

Multi-level Secure JTAG is an extension of previous authentication systems but primarily based on Protected JTAG, described in section 3.3.3.2. While the aforementioned have detailed protocols for authentication, but do not control user access afterwards. They also generalize the permission scheme introduced by Protected JTAG by allowing the designer to set any amount of permission levels. A user can only execute functions that have an access level equal or lower than their permission level.

Multi-level SJTAG introduces a Secure Authentication Module and an Access Monitor. The Secure Authentication Module is designed to work with any challenge-response protocol, and works similarly as Protected JTAG's. After authentication, the Secure Authentication MODULE will pass the privilege level to the Access Monitors, and acts as their control interface.

The Access Monitor is the primary extension, and monitors the BSC cells. Specifically, it monitors the connection between the flip-flop used for Shift-dr and the flip-flop used for Update. This allows it to determine if an update to that cell is allowed, and to subsequently allow or deny it.

3.4. Masking

This security class focuses on masking the data streams sent through JTAG rather than controlling access. If implemented correctly this makes it impossible to control JTAG with any accuracy and impossible to interpret the output in any meaningful way.

Stream Ciphers are the only good way to implement masking on native JTAG. However, there are two variants that are conceptually in line with this class, but are unfit for inclusion:

- The first of these variants is obfuscating of the Boundary Scan Chain (BSC), where, for instance, the length of the scan chain is variable from device to device. This makes reversing the scan chain harder, but not significantly so. As such, this is not a proper security measure and has not been included in the classification.
- The second of these variants exists only when JTAG is used as the physical layer for a packet-based protocol. When using packets it's possible to use different cryptographic mechanisms such as RSA.

Conceptually this is similar to Stream ciphers, but no longer applied to the JTAG protocol. If the encryption is applied to a custom protocol it's security for that given protocol, and as such has not been included in the classification.

Stream Ciphers

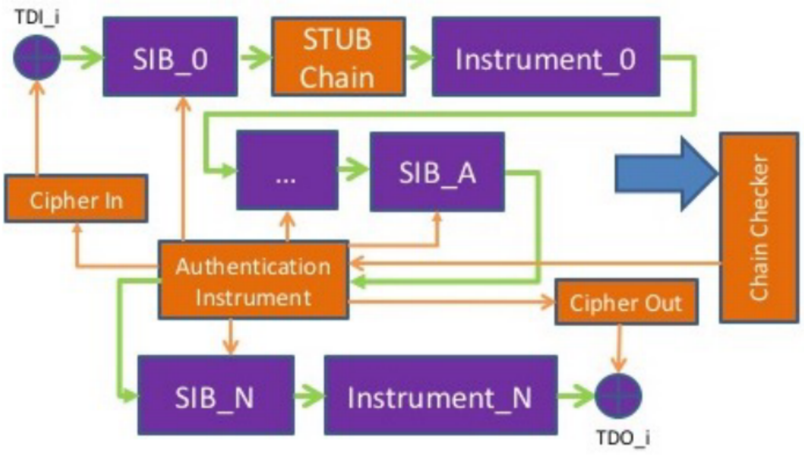


Figure 3.10: Stream cipher architecture [29].

A Stream cipher is a type of encryption where data is encrypted per bit. As JTAG is a bit-level protocol the sensible encryption is a bit-level encryption. Stream ciphers are employed in two places: Firstly, in order to maximize the data protection TDO is XOR'ed with the output of the cipher key stream [29]. Encrypting the output only protects the system data against reading, not control. A second, different, cipher is added on TDI pin in order to decode an encrypted input stream. This requires a user to encrypt the input Stream, thus securing against control of the system.

3.5. Detection of Malicious Use

Rather than trying to control who uses the JTAG interface, this class focuses on identifying malicious use.

3.5.1. Sequence filter

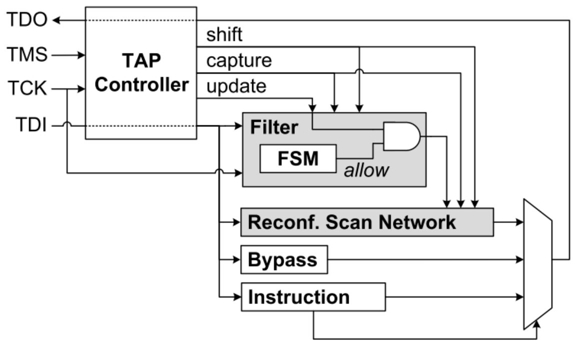


Figure 3.11: Structure of the sequence filter [4].

This method to secure JTAG is based on IJTAG's ability to reconfigure the scan network. The core concept behind this setup is to restrict access to protected SIBs so that sections containing sensitive components cannot be accessed through the TAP. This is achieved by using a sequence filter in the TAP controller [4]. The filter monitors shifted-in data and will block the update if the sequence would affect SIBs that would grant access to protected parts of the scan chain.

3.5.2. Statistical Analysis

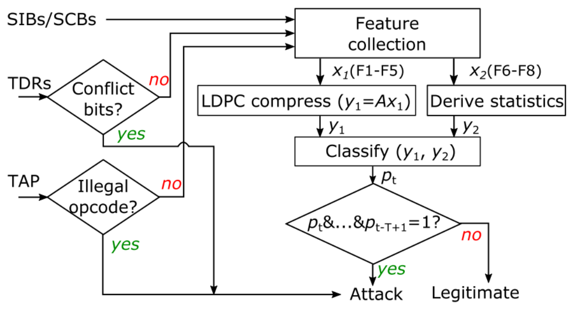


Figure 3.12: Flow Diagram of the detection of an attack in IJTAG [48].

Statistical analysis is a state of the art method to protect JTAG. It makes use of machine learning in order to detect attacks [48] [49]. It's based around the idea that illegitimate users will behave differently to legitimate users as they don't know what JTAG functions are implemented or how they should be operated. It employs an on-chip neural network classifier in order to recognize illegitimate access. For instance, an attacker may not know the appropriate length of the DR register and may attempt to shift an incorrect amount of times in order to determine the length. Another classifier that can be used are common transitions, as an attacker would be trying to discover and learn about them, rather than using them in the expected fashion. If enough consecutive predictions indicate an attacker, an alarm is raised and the attacker is locked out. The overall flow of a detection is shown in figure 3.12. While this has been proven to be an effective strategy for JTAG, with a 99.2% detection rate [49], this approach is ineffective in IJTAG. This is primarily because, due to it's hierarchical nature, many more combinations of operations are possible [48]. This results in a highly dimensional feature set which is challenging to store on a chip. While high dimensional, the feature set also very sparse; This makes it possible to use a low-density parity-check (LDPC) matrix to compress the dataset.

3.6. Comparison

Almost all solutions focus on restricting access to the TAP controller, rather than the scan chain. As all communication is done through the TAP controller this is a logical solution, but it does provide a single point of failure. This includes the stream cipher, as the data is unencrypted as it passes through the scan chain. These single point of failures might make it viable to circumvent the authentication mechanisms instead of attacking the authentication protocols directly. The major exception is Multi-level Secure JTAG (SJTAG) as it attempts to lock down the Boundary Scan Chain (BSC) cells rather than access to the scan chain.

Another common theme in the public key encryption is the use of Elliptic Curve Cryptography(ECC) rather than RSA. The reason for this is mainly economical: The implementation for ECC is much smaller on the chip than RSA is [17]. This makes it cheaper to fabricate, significantly so if a company produces thousands of chips. Despite being cheaper to implement, the security offered by ECC is similar to RSA.

Detection mechanisms, while effective, assume attackers are unfamiliar with the chip's architecture. With enough documentation or resources, it may still be possible for an attacker to pose as a legitimate user.

4 Samsung Secure JTAG

Chapter 3 discussed various methods in order secure JTAG Boundary Scan Chain (BSC). This chapter focuses on Samsung's implementation: Secure JTAG (SJTAG). A brief conceptual overview of SJTAG is provided in Section 4.1. To aid in the understanding of SJTAG and related attacks, Section 4.2 discusses the CoreSight debug system. SJTAG itself is detailed in Section 4.3.2. Lastly, Section 4.4 provides a brief explanation of a number of potential attack vectors if SJTAG were to be disabled.

4.1. Concept of Work

Section 2.2.3 discussed the power of JTAG as an attack vector. As JTAG is an important testing standard, several methods were discussed in Chapter 3 that attempt to secure it. Secure JTAG (SJTAG) [35] is Samsung's method to secure the debug system. While the name implies that it is a secure form of JTAG, it is not applied to the JTAG standard. Instead SJTAG is a module intended to extend the ARM CoreSight debug system with an access control mechanism. SJTAG is implemented in Samsung Exynos System on Chips (SoC), which are commonly found on various Samsung devices such as smartphones. The debug system and it's security is further described in Section 4.2, while SJTAG is detailed in Section 4.3

4.2. Debug system: CoreSight

This section discusses the debug system found in ARMv7 and ARMv8 architectures and it's important components. Sections 4.3.1 and 4.2.2 provide an overview of the debug system and it's security measures. Section 4.2.3 covers the Debug Access Port (DAP) which is an important component in accessing the debug system. JTAG-DP and JTAG-AP are critical components of the DAP, and are discussed separately in Sections 4.2.4 and 4.2.5.

4.2.1. Overview

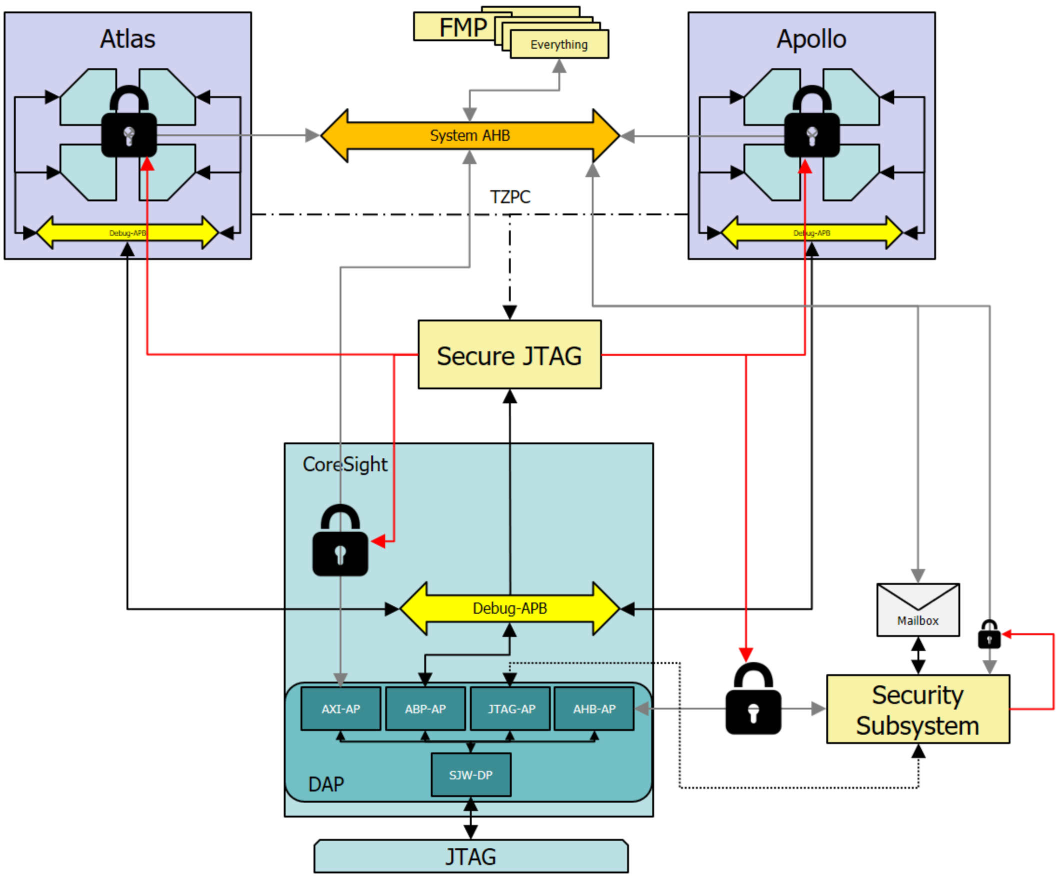


Figure 4.1: Simplified drawing of debug system

All ARMv7 and ARMv8 architectures contain an on-chip debug and trace architecture referred to as CoreSight. It replaces JTAG Boundary Scan Chain (BSC) where possible. Additionaly, the debug system provides support for JTAG BSC, which is used for components that do not support CoreSight. CoreSight-enabled devices have a separate debug bus called the Debug Advanced Peripheral Bus (APB) and all supported components have debug related registers memory mapped on it. This includes for instance registers governing breakpoints and trace information. The debug registers of the CPUs are grouped as follows:

- **DBG**: Debug, providing various debug info and controls.
- **CTI**: Cross Trigger Interface provides control mechanisms for simultaneous communication to multiple cores.
- **PMU**: Performance Monitoring Unit reports information about the core such as cache misses.
- **ETM**: Embedded Trace Macrocell is used in generation of execution traces for debugging purposes.

Further details can be found in Chapter H of the respective architecture's manuals [32] [34] and the CoreSight documentation [33] in general.

4.2.2. Security

As debug mechanisms have been proven to be a huge security concern, the ARMv7 and ARMv8 architectures specify up to four external security signals. These are:

- **DBGEN**: DeBuG ENable; enables invasive debug.

- **SPIDEN**: Secure Privileged Invasive Debug ENable; enables debug while the relevant processor core is in a secure state.
- **NIDEN**: Non-Invasive Debug ENable; enables trace functionality.
- **SPNIDEN**: Secure Privileged Non-Invasive Debug ENable; enables tracing while the relevant processor core is in a secure state.

Enabling these signals significantly impacts the device's security, as they govern, for instance, the ability to halt the CPU and step through instructions. Research has shown that the debug capabilities can be used by an attacker for a privilege escalation attack [41]. Notable about the attack described is that it can be executed purely from software, using CoreSights On-System Debugging features. The security signals are controlled by an external authentication mechanism. This mechanism is vendor specific and not specified by the ARM standard. Samsung's implementation as used on the Exynos 7420 is called Secure JTAG (SJTAG) and is discussed in Section 4.3.

4.2.3. Debug Access Port

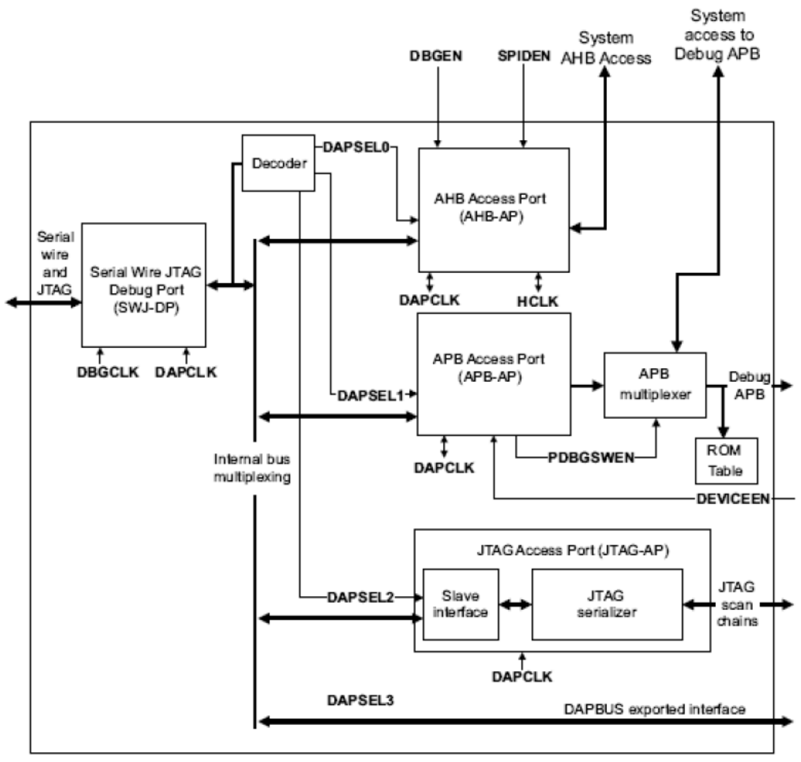


Figure 4.2: The Debug Access Port[35]

The DAP is the debug component in charge of external communication into the debug system. It is an implementation of the Arm Debug Interface Version 5 (ADIv5) [35]. As can be seen in Figure 4.2 the DAP comprises of two Debug Ports (DP), three Access Ports, a decoder and the DAPBUS. The decoder is used to select which AP is used, and the DAPBUS exposes the debug system to the CPU for on-system debugging.

4.2.3.1. Debug Ports

The DPs are divided into three classes, only two of which are actual DPs:

- **JTAG-DP**:

  The default operation mode. uses JTAG as an interface to communicate with the DAP. JTAG-DP is not directly used to gain access to the BSC of the device. It is described in more detail in Section 4.2.4.

- **SW-DP**:

  Allows communication over the Serial Wire protocol as an alternative to JTAG. It only uses two connections and may be preferred on systems where adidtional connections are expensive or difficult to place.

- **SWJ-DP**:

  Implementation of the external port that allows switching between JTAG-DP and SW-DP with a specific sequence. Does not have a communication protocol otherwise.

4.2.3.2. Access Port Classifications

Furthermore, there are two classes of AP. Specifically:

- **MEM-AP**:

  MEM-AP access ports allow read and write operations to specified memory addresses or registers. This type of AP is conceptually simple and well supported by various debug tools. As such further description of this class is unnecessary.

- **JTAG-AP**:

  JTAG-AP is a specific variant of MEM-AP that uses 8 registers to communicate with the internal JTAG BSC. Communication through this port is complex and poorly supported by common debug tools. A thorough understanding of this port is vital to this thesis and has been detailed in Section 4.2.5.

4.2.3.3. Access Ports

Three APs are implemented on the target device. Each AP accesses it's own area, and has it's own security measures. These are:

- **Advanced eXtensible Interface (AXI)-AP**:

  The first port is sometimes referred to as System Advanced High-performance Bus (AHB) by the documentation, including in Figure 4.2. The DAP internally refers to it as AXI-AP, so this terminology will be used. It is a MEM-AP that provides access to the main system bus, giving top-level access to the entire memory-mapped system. Because of the obvious security concerns this port is protected by the DBGEN and SPIDEN signals as described in Section 4.2.2.

- **APB-AP**:

  This unsecured MEM-AP provides access to the CoreSight debug bus or APB. This bus attaches to debug-related components such as the debug registers of the CPUs. SJTAG, The debug authentication mechanism, is also connected to this bus. This makes it possible to authenticate over the APB in order to gain access to the AXI bus.

- **JTAG-AP**:

  A JTAG-AP port that provides access to the BSC of the system. In-depth knowledge about this port is critical to understanding parts of this Thesis, and is described in Section 4.2.5

- **AHB-AP**:

  The AHB-AP is a MEM-AP connected to the Security SubSystem (SSS) of the target device. The SSS is an secure component that is not meant to be directly accessed while the system is in operation. Like AXI-AP, it is secured by DBGEN and SPIDEN. While DBGEN and SPIDEN are enabled the SSS can be accessed through the AHB-AP even while it operates in isolated mode.

4.2.4. JTAG-DP

This section aims to provide a general overview of JTAG-DP. While it is important to gain a rough understanding of how JTAG-DP operates for this Thesis, exact detailed operation is considered to be out of scope. Detailed information can be found in Chapter 3 of the Arm Debug Interface Architecture specification [31]

JTAG-DP is, first and foremost, compliant with the JTAG standard as outlined in Section 3.1.1. Most importantly this means the same concepts apply to both the JTAG state machine, shown in figure 3.2, and the instructions listed in Section 3.1.1.2.

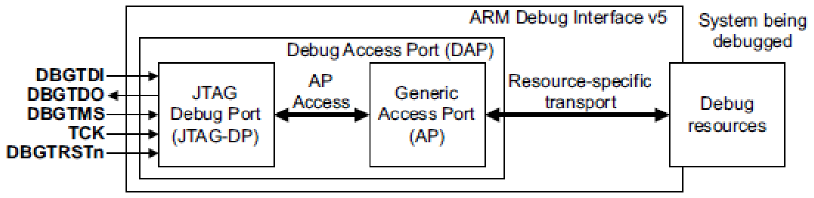


Figure 4.3: Relation of JTAG-DP to the DAP and debug resources[31]

Where JTAG-DP differs significantly from JTAG is in the way it is used. Rather than providing access to the test chains, Figure 4.3 shows how JTAG-DP is used as an interface to the DAP. Using the JTAG standard as a physical interface is a fairly common use case. For instance, as a programming interface for FPGA's and similar hardware [60].

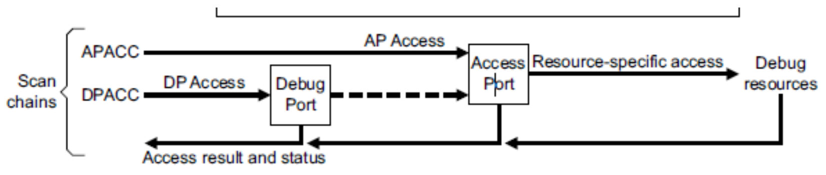


Figure 4.4: Relation of APACC and DPACC to debugging[31]

Two new instructions are introduced in JTAG-DP: DPACC and APACC. As can be seen in Figure 4.4 DPACC can be used to request information about the various APs in the DAP and select them. APACC is used to communicate with the selected AP in order to select, read and write memory addresses. Operation of these instructions are identical to eachother, but very different to the implementation of standard JTAG instructions. Specifically, the instructions incorporate the state machine as seen in 4.5. While exact operation is out of scope for this Thesis, a conceptual understanding of how this state machine works is important. Several JTAG transactions are used in succession to access the appropriate memory location at a given AP. Reading and writing to memory can happen simultaneously when accessing successive locations. More Information about JTAG-DP can be found in Chapter 3 of the the official ARM Debug Interface architecture specification [31].

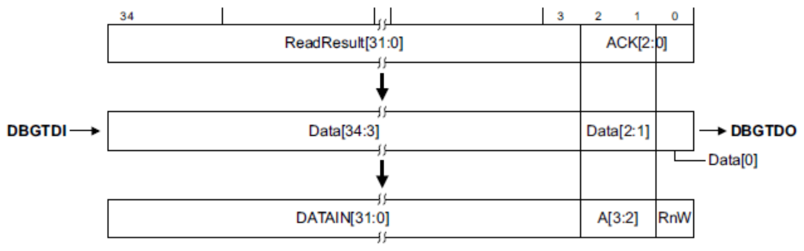


Figure 4.5: DPACC and APACC state machine[31]

4.2.5. JTAG-AP

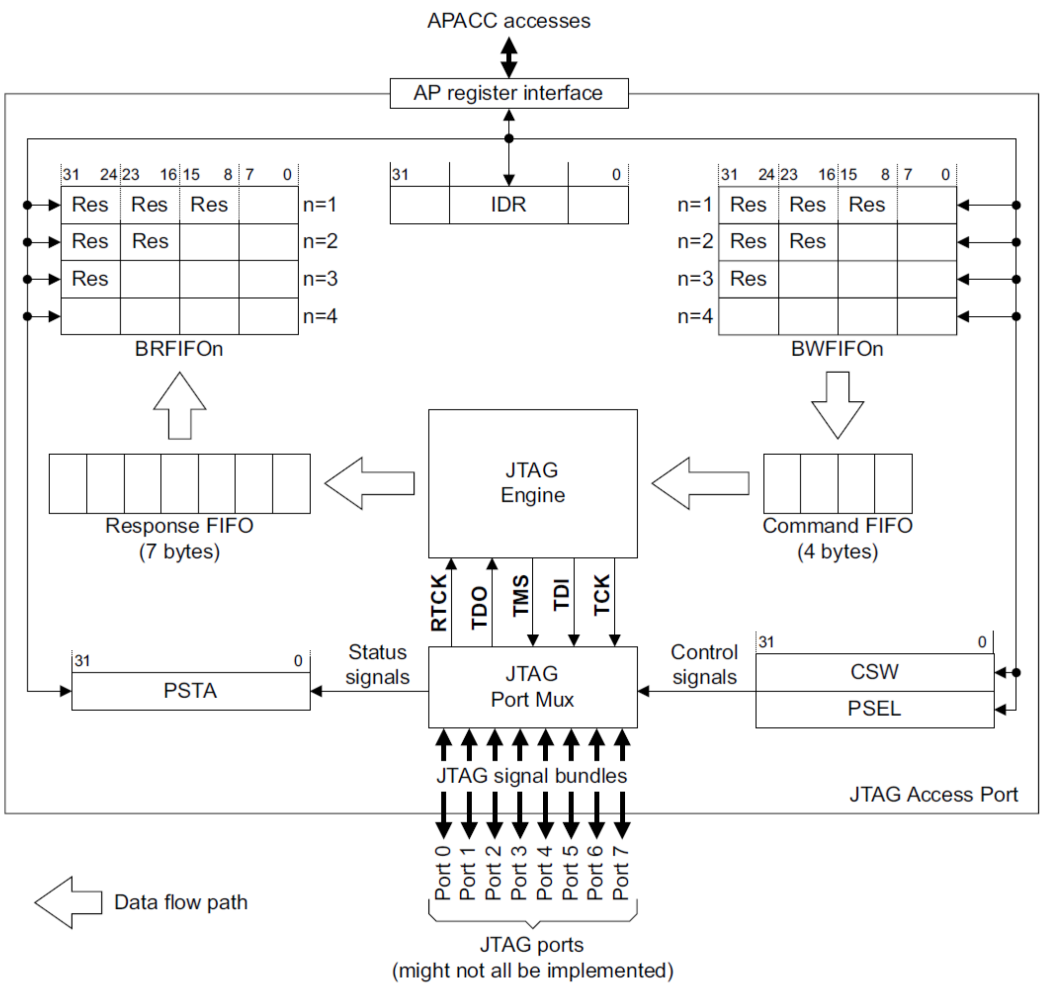


Figure 4.6: Overview of the JTAG-AP [31]

The JTAG Access Port (JTAG-AP) is one of the access port types supported by the DAP. It's designed to provide support for up to 8 legacy BSC connected devices. This chapter describes the JTAG-AP port and it's communication protocol. In-depth knowledge of this port is required for the attack described in Section 5.3. The JTAG-AP connects to internal BSCs which are employed to test such components as the SSS controller described in Section 4.4 and the Multi-Format Codec described in Chapter 47 of the Exynos 7420 documentation [35]. Older ARM architectures, such as the Cortex-M0 based controller of the SSS do not have CoreSight support. As such, these can only be tested through their more traditional JTAG interfaces.

This section discusses the registers of JTAG-AP and their contents followed by the communication protocol. Full descriptions can be found in Chapter 8 of the debug interface documentation [31].

4.2.5.1. Overview

JTAG-AP provides a parallel to serial interface to the BSC. As shown in Figure 4.6 the JTAG-AP consists of a JTAG Engine, a JTAG Port Multiplexer, Command and Response FIFOs and several configuration registers described below.

The JTAG Engine is the main processing component of the JTAG-AP. It is responsible for interpreting the command bytes in the Command FIFO, generating the correct JTAG signals for the Port multiplexer and interpreting TDO to generate a response to be stored in the Response FIFO. According to Section 54.3 of the Exynos 7420 documentation[35] it is be controlled by the CSTCK clock at a 25 MHz frequency.

The Port multiplexer is used to connect the JTAG ports to the (single) JTAG Engine. The multiplexer can be configured to connect to either individual or to multiple JTAG ports at the same time. Selecting multiple ports is useful to allow the JTAG engine to drive signals into several ports in parallel. However, this mechanism can only be used as an input to TDI; Reading from multiple JTAG ports is impossible because TDO output is undefined while multiple ports are selected.

The Command and Response FIFOs are used by the JTAG engine as input and output, respectively. These are serial interfaces that can be accessed using the parallel BxFIFO registers.

4.2.5.2. Overview of JTAG-AP Registers

The JTAG-AP contains eight registers, of which four are memory mapped. They can be accessed in CSAT using their offset or their numbers. CSW and IDR can also be accessed using their names. The registers, and their functions, are:

0. **CSW** is the Control/Status Word register. It shows information regarding the JTAG ports and allows them to be configured. Most importantly it contains how many bytes are to be processed by the JTAG ports and how many are waiting in the buffer to be read. This register is detailed in register 4.7. CSAT can access this register by using it's name.

1. **PSEL** contains an 8 bit bitmask ([0:7]) to determine and configure what JTAG ports the FIFO registers are currently connected to. Multiple ports can be selected for parallel input. When multiple ports are connected, the output of TDO is unknown.

2. **PSTA** is an 8 bit bitmask ([0:7]) describing which ports are connected AND disabled.

3. Reserved

4. **BxFIFO1** writes/reads one byte from the JTAG engine's FIFO registers.

5. **BxFIFO2** writes/reads two bytes from the JTAG engine's FIFO registers.

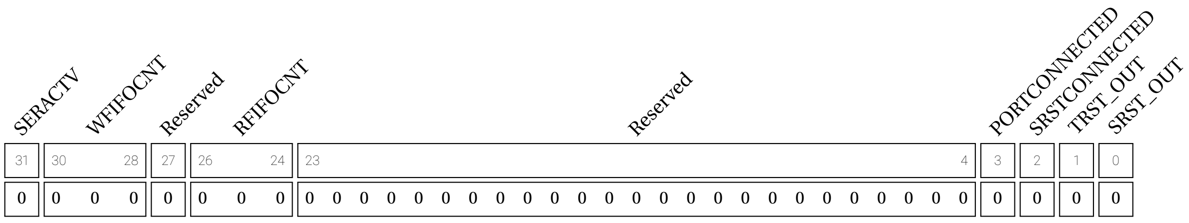
6. **BxFIFO3** writes/reads three bytes from the JTAG engine's FIFO registers.

7. **BxFIFO4** writes/reads four bytes from the JTAG engine's FIFO registers.

-. **IDR** is an identification register primarily to describe the AP as JTAG-AP and it's revision. This cannot be accessed by the CSAT interface, but is located at 0xfc.

4.2.5.3. CSW Register

Figure 4.7: CSW Register (0x00)



The CSW register, It's layout shown in Figure 4.7, is important in understanding how communication through the FIFO works. It contains the following registers:

SERACTV Indicates if the JTAG engine is processing pending commands

WFIFOCNT Pending number of command bytes. Writes to the FIFO registers increments this number and the JTAG engine decrements it as each byte is processed.

RFIFOCNT Pending number of response bytes. Gets incremented by the JTAG engine and decremented as the user reads bytes from the FIFO registers.

PORTCONNECTED This bit is high if all selected ports are connected to the JTAG Engine.

SRSTCONNECTED Is high if all selected ports have the SRST signal connected to the JTAG engine.

TRST_OUT Specifies if TRST is driven onto the TRST signal of the JTAG port. When this bit is set to 1 TRST output is LOW. This bit is read-only and does not self reset. It needs to be cleared by writing a 0 to this register.

Keeping track of the WFIFOCNT and RFIFOCNT registers is crucial during communication through the FIFO to avoid stalling the JTAG Engine. In order to explain their functionality it is important to first describe the BxFifo registers.

4.2.5.4. Communication Registers

The communication registers consist of five kinds of FIFO registers. These are:

Command FIFO Shifts communication bytes into the JTAG Engine to facilitate communication with TDI and TMS. This register is not directly exposed to the debugger.

Response FIFO Output from TDO is pushed serially into this register. Like the Command FIFO, it is not directly exposed to the debugger.

BWFIFO1-4 Four registers that will shift one to four bytes onto the Command FIFO. Writing to this register is exposed through the BxFIFO registers.

BRFIFO1-4 Four registers that will shift one to four bytes from the Response FIFO. Reading from this register is exposed through the BxFIFO registers.

BxFIFO1-4 Writing to these four registers will write to it's relevant BWFIFO register. Likewise, reading from these registers reads from the relevant BRFIFO Register. These are directly exposed to the debugger.

The FIFO registers are numbered and correspond to the amount of bytes that are being written to or read from the internal FIFOs. All FIFOs are 32 bits in length and ignore the relevant most significant bytes. This is shown in figure 4.8.

Figure 4.8: FIFO input/output structure

31 24	16	8	0	
Ignored (0x00)	Ignored (0x00)	Ignored (0x00)	First Byte	BxFIFO1
Ignored (0x00)	Ignored (0x00)	Second Byte	First Byte	BxFIFO2
Ignored (0x00)	Third Byte	Second Byte	First Byte	BxFIFO3
Fourth Byte	Third Byte	Second Byte	First Byte	BxFIFO4

The amount of pending bytes is stored in CSW's WFIFOCNT and RFIFOCNT fields. These fields need to be referred to when communicating with the FIFO registers. This is critical, because the JTAG engine will stall under the following conditions:

- More bytes are written to the Command FIFO than it can store. This is typically 4 bytes long but may be up to 7 bytes long on some implementations. This includes the Exynos 7420's implementation.
- The Response FIFO is full and the JTAG Engine has pending commands. The response FIFO is 7 bytes long.
- More bytes are read than there are pending in the Response FIFO.

4.2.5.5. JTAG Engine Byte Command Protocol

The JTAG Engine Byte Command protocol is used to communicate with the JTAG Engine through the BxFIFO registers. The protocol is designed to provide an interface to control the JTAG state machine. Further, it provides commands to control communication over TDI and TDO. All commands in the protocol are 1 byte long.

7	6	5	4	3	2	1	0	
0	TDI	1	TMS[4]	TMS[3]	TMS[2]	TMS[1]	TMS[0]	5 bits of TMS data
0	TDI	0	1	TMS[3]	TMS[2]	TMS[1]	TMS[0]	4 bits of TMS data
0	TDI	0	0	1	TMS[2]	TMS[1]	TMS[0]	3 bits of TMS data
0	TDI	0	0	0	1	TMS[1]	TMS[0]	2 bits of TMS data
0	TDI	0	0	0	0	1	TMS[0]	2 bits of TMS data

Bit 7 TMS Opcode. Designates start of TMS packet.

Bit 6 Value of TDI while clocking in TMS

Bit 5-2: Padding Unspecified amount of padding zeroes to pad byte to 8 bits.

Bit 5-1: Data flag Single bit specifying the start of TMS data

Bit 4-0 TMS data. LSB clocked in first.

Figure 4.9: TMS Packet structure

The TMS packet is a single byte long and contains between one and five bits of TMS data. Additionally it controls whether TDI is held high or low while TMS is clocked into the JTAG port. It's structure can be seen in figure 4.9.

Communication with TDI is done using the TDI_TDO packet. According to chapter 8.3.3 of the debug interface documentation [31] it is not used in a TDO response. Instead, TDO output is packed directly into bytes and inserted into the Response FIFO. The TDO_TDO packet consists of 3 types of packets that need to be sent in order. These packets are specified in figure 4.10.

4.2.5.6. Communication Example

The following enumeration presents a theoretical example of what communication resembles in practice. This example assumes a 3 byte command packet that will generate a 7 byte response. The byte used in the example shifts 53 bits over TDI and captures the TDO output.

1. Write a three byte command into BxFIFO3: 0x00003487
2. In theory, an immediate read of CSW could show 3 bytes in the WFIFOCNT and 0 bytes in RFIFOCNT. Note that in practice communication with the DAP is not fast enough to be able to perform a read in time, but this behaviour can be simulated by disconnecting the JTAG ports.
3. Wait for enough clock cycles to pass for the JTAG engine to process the command bytes.
4. A second read of the CSW register shows that WFIFOCNT has reduced to 0, and RFIFOCNT has been raised to 7 bytes.
5. Reading BxFIFO4 returns four bytes of data: 0xFFFFFFFF
6. Reading the CSW's RFIFOCNT field shows that 3 more bytes are available on the response FIFO.
7. Reading BxFIFO3 returns the remaining 3 bytes of data: 0x001FFFFF
8. Reading CFW's RFIFOCNT field confirms there are no more remaining bytes. Reading the SERATV field confirms the JTAG engine is no longer active, and no further bytes can be expected. Communication is complete.

This example also shows an implementation detail: The packet sends 53 bits over TDI which is three bits less than 7 bytes. Effectively this means that it's vital to keep track of the input length when parsing TDO data.

7	6	5	4	3	2	1	0	
1	0	0	Reserved(0)	TMS	RTDO	TDI	UTDI	TDI_TDO Opcode Byte
Format	Lenght / Packed TDI bits							TDI_TDO Lenght Byte
Packed TDI bits								TDI_TDO Data Byte
⋮								
Packed TDI bits								Subsequent Data Bytes

Opcode bits 7-5 TDI_TDO Opcode. Designates start of TDI_TDO packet.

Opcode bit 4 Reserved, should be 0.

TMS Specifies value TMS should be held at during last cycle of this packet. Useful for transitioning from SHIFT-IR/DR to EXIT-IR/DR after input.

RTDO Specifies if TDO output should be captured and pushed to Response FIFO.

TDI Value of TDI if UTDI is set to 1. Useful for shifting large amounts of 1's or 0's into TDI.

UTDI If set to 0, TDI values are held in subsequent bytes. If set to 1, use TDI value to determine TDI signal.

Format Specifies the format of the length byte. If value is 0, bits [6:0] contain unsigned integer specifying the amount of bits contained in subsequent bytes. This integer is offset by one, so a value of 0 results in 1 bit, while 127 results in the maximum of 128 bits. If UTDI is set to 1, this is the last byte as no further data is required. If the value of the format bit is 1, bits [6:0] contain the packed TDI bits. These are packed in a similar format as the TMS bits [5:0] in figure 4.9

Packed TDI bits Packed TDI bits. These are shifted in LSB first, with no format otherwise. The data bytes are only used with UTDI = 0 and Format = 0, and are repeated as needed. Unused bits in the final byte are kept 0.

Figure 4.10: TMS Packet structure

4.3. Samsung Secure JTAG Implementation

This chapter discusses the implementation of various components within the Secure JTAG (SJTAG) IP. First section 4.3.1 discusses SJTAG as a whole. This is followed by sections 4.3.2 and 4.3.3, which discuss both operational modes of SJTAG. Lastly, sections 4.3.4 and 4.3.5 discuss the Access controller and the JTAG detector. These are components that exist in both operational modes and are worthwhile to discuss.

4.3.1. Overview

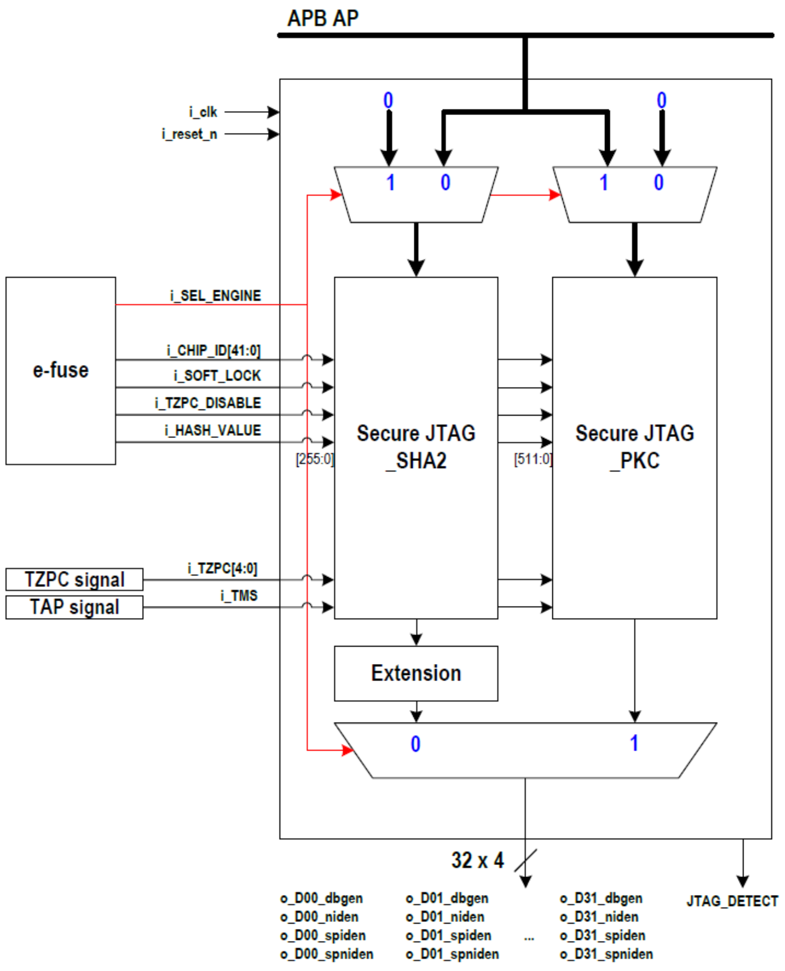


Figure 4.11: The overall layout of Secure JTAG [35].

SJTAG is Samsung's implementation of the authentication component for use in Coresight enabled devices. It's primary function is to authenticate legal users and generate the access signals specified in Section 4.2.2. While ARMv8 architectures only support four access signals, SJTAG outputs 32 sets in order to further compartmentalize access levels. This way full access can be given to a single core, while denying access to other secure areas. The device under target implements Secure_JTAG_combined version 1.0, as seen in figure 4.11, which combines SJTAG_SHA2 version 1.1 and SJTAG_PKC version 1.0. These components can only be operated exclusively of each other and are determined by the SJTAG_SEL_ENGINE eFuse[35]. This fuse is not exposed through a register, and identification needs to rely on the version register of SJTAG

eFuse registers are special registers whose values can only be changed once. While sometimes eFuses are implemented in changable memory such as NAND, this does not appear the be the case for Samsung. Samsung owns a patent for an eFuse [25] design that employs electro migration in order to physically and permanently change the value. The eFuse registers are used in as immutable configuration registers. Crucially, SOFT_LOCK can be fused to enable the security offered by SJTAG.

SJTAG_SHA2 uses a hashed password based scheme and is discussed in Section 4.3.2. The other operation mode, SJTAG_PKC, uses a challenge response authentication scheme and is discussed in Section 4.3.3. Both of these components implement an access monitor, described in Section 4.3.4 and a JTAG Detector, which is discussed in Section 4.3.5.

4.3.2. SJTAG-SHA2

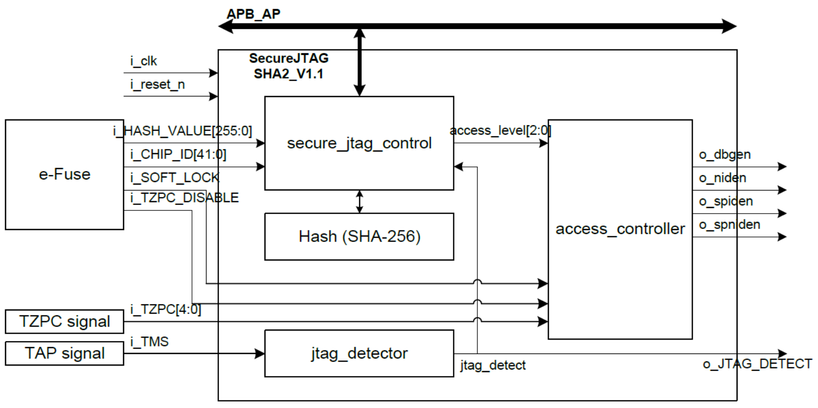


Figure 4.12: Block layout of Secure JTAG's SHA256 mode [35].

The SJTAG-SHA2 module [35], as seen in Figure 4.12, is one of the subcomponents and operation modes of SJTAG. It's operation is mutually exclusive from the SJTAG-PCK mode. It provides a password based multi-level authentication mechanism that is used to provide access to the debug system. SJTAG supports the ability to bypass authentication through the ARM TrustZone, but this feature may be disabled by fusing TZPC_DISABLE.

4.3.2.1. Components

- Controller

  The controller is completely undocumented and nothing concrete is known about it. Given the context of the component it is likely in charge of external communication, controlling the hashing engine and the access controller.

- Dedicated Hashing Engine

  A dedicated hashing engine that's used to hash the input password for comparison by the controller. It uses SHA-256 as its hashing algoritmn.

- eFuses

  Special purpose registers that can only be written to once, which makes them ideal to store secure and immutable data such as a a cryptographic key. In these are used to store the following values:

  – **Hash Values** The target hash to compare the results of the hashing engine to. As SHA-256 is used this is 256 bits long

  – **CHIP_ID**: A chip identification number. It is unclear what this maps to if anything. Allegedly it is not used in password generation, nor does it match the serial number of the device or the lot number of the CPU. It may be used by the vendor to uniquely identify the CPU and retrieve the appropriate password. This field is 42 bits long

  – **SOFT_LOCK**: The security bit. Enabling this bit engages security. While disabled all security signals will be sent by the access controller enabling full access to the debug system.

  – **TZPC_Disable** The debug security signals may be controlled by the trustzone. Setting this bit disables that functionality thus enforcing authentication.

- Access controller

  The access controller combines the inputs from the JTAG Controller, the eFuses and TrustZone input in order to output the debug signals. While in SHA-256 operation SJTAG only supports 4 access levels.

To generate 32 sets of the security signals the output of the access monitor is fed into the Extension component that can be seen in Figure 4.11. It is further discussed in Section 4.3.4.

- JTAG Detector

  Detects toggling of the JTAG TMS signal.

4.3.2.2. Operation

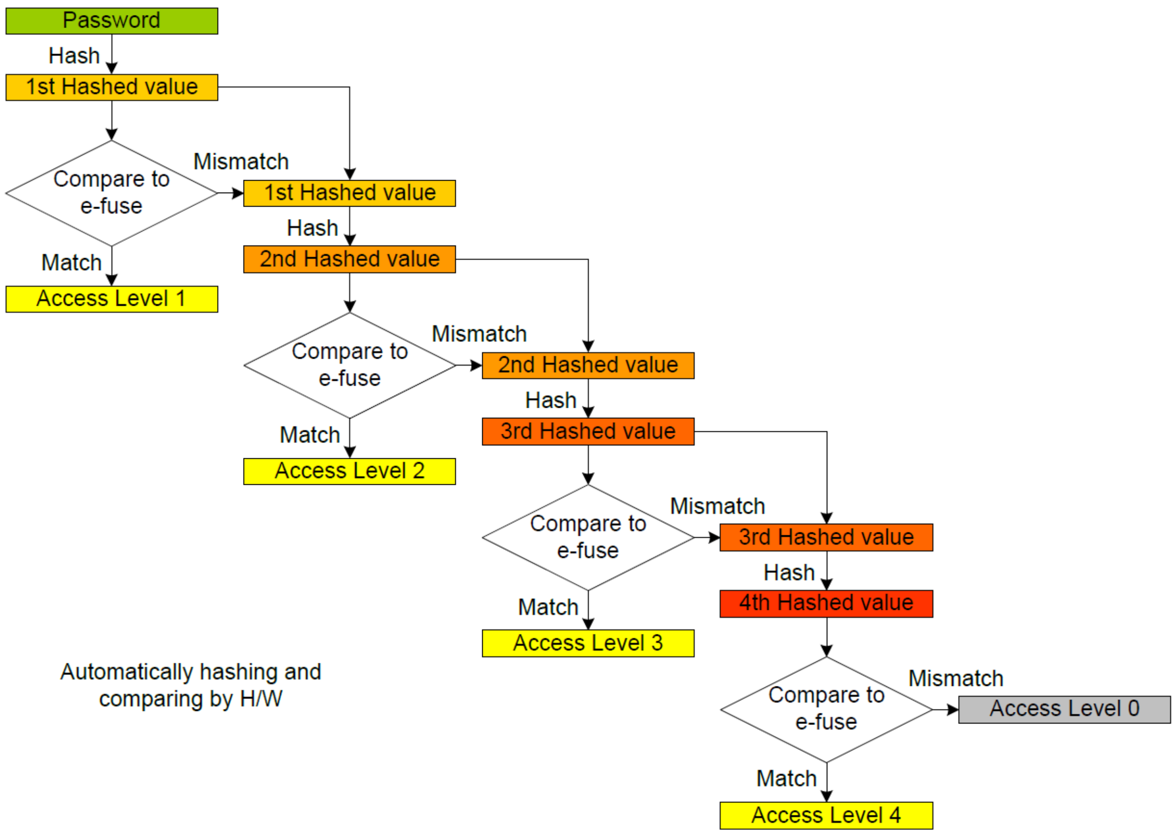


Figure 4.13: SHA2 password validation and access level determination [35]

SJTAG SHA256 authentication is based on sequential hashing of a password. During manufracturing a password p is chosen based on the lot number of the processor [53]. This password is hashed four using SHA256 function H:

$$H_0 = p$$

$$H_1 = H(p)$$

$$H_2 = H(H_2)$$

$$H_3 = H(H_3)$$

$$H_4 = H(H_4)$$

H_4 is stored in the device's eFuses. Intermediate hashes H_0 through H_3 can be stored by the manufacturer to serve as passwords. Because hashes are non-reversible they can be used in a multi-level authentication system. In this system, H_0 provides the highest level of access while H_4 provides the lowest.

As Figure 4.13 shows, password validation is performed by repeatedly hashing the input password and comparing the result to the value stored in the eFuses. The number of hashes required directly correlates to the access given. E.g., H_1 would need to be hashed 3 times in order to match eFuse value H_4. This results in access level 3. The input is hashed a maximum of four times, after which the password is considered to be incorrect. This results in an access level of 0.

Passwords are entered into by writing 8 32-bit WORDs into the password register. This causes SHA256 to perform it's hashing operation, completion of which is reflected in the status register. After completion SJTAG may be reset by writing a 1 to the SJTAG_DETATCH_CONF register or through a power cycle.

4.3.3. SJTAG-PCK

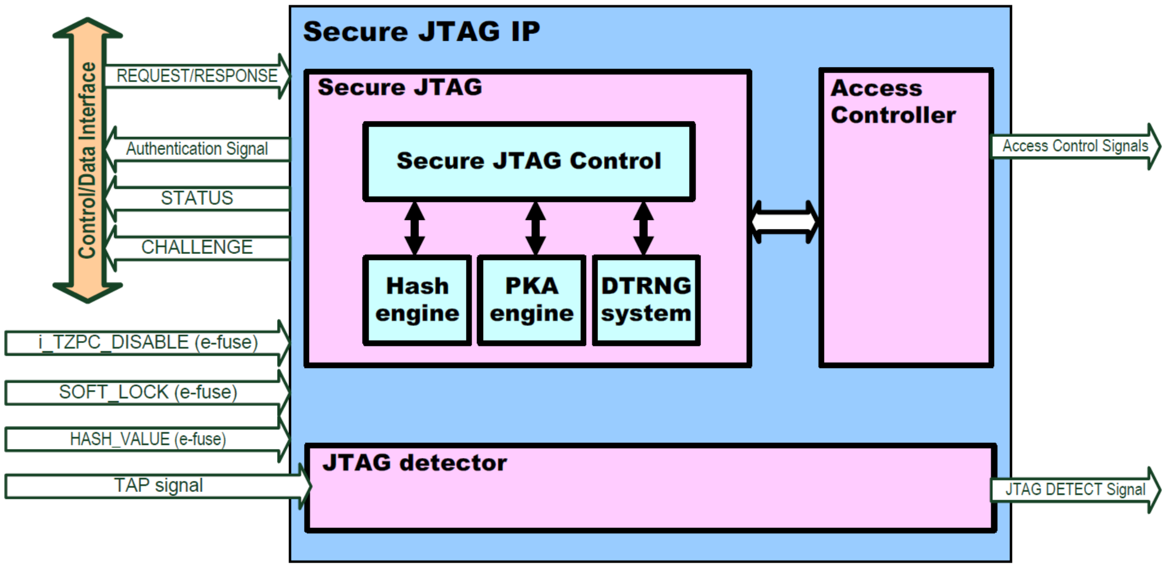


Figure 4.14: Block layout of Secure JTAG's PCK mode [35].

SJTAG's second operating mode is performed by the SJTAG-PCK module [35], as seen in Figure 4.14. It's operation is mutually exclusive from the SHA-256 mode. It improves security over SHA-256 by providing a challenge-response authentication scheme using public key cryptography. While in PCK mode supports 32 debug domains, each with the four ARM debug signals. Like in SHA-256 operation the debug signals can be controlled by the TrustZone.

SJTAG stores a hash of the public key rather than storing the private key. According to the documentation this is so that, if SJTAG gets compromised, nothing of value can be obtained. Given that it's cryptographically impossible to determine the private key from the public key this is an odd claim. A more likely explanation could be that a eFuse memory may be more expensive than volatile storage and this may be worth the security reduction.

4.3.3.1. Components

The SJTAG-PCK module has the following components:

- Controller

  Like in SHA-256 operation the controller is undocumented and it's operation is unknown.

- eFuses

  Special purpose registers that can be written to only once, which makes them ideal to store secure and immutable data such as a cryptographic key. In SJTAG-PCK these are used to store the following values:

  – **Hash value**: a 512bit hash value representing the hash of the public key.
  – **CHIP_ID**: A chip identification number. It is unclear what this maps to if anything. Allegedly it is not used in password generation, nor does it match the serial number of the device or the lot number of the CPU. It may be used by the vendor to uniquely identify the CPU and retrieve the appropriate password. This field is 42 bits long
  – **SOFT_LOCK**: The security bit. Enabling this bit engages security. While disabled all security signals will be sent by the access controller enabling full access to the debug system.

  - **TZPC_Disable** The debug security signals may be controlled by the trustzone. Setting this bit disables that functionality thus enforcing authentication.

- Hashing engine
- A SHA-512 bit hashing engine that is used to confirm the correctness of the public key, which is to be shifted in prior to authentication.
- PKA Engine

  A public key accelerator that is used for ECDSA cryptography.
- a DTRNG engine

  a cryptographically secure random number generator, a required component in both ECDSA and generation of the challenge.
- 32 Access Controllers

  The access controllers combine inputs from the JTAG controller, eFuses and the TrustZone in order to output the debug signals. While SJTAG is in PCK operation, it supports separate signal configuration for each of the 32 debug domains. It is further discussed in Section 4.3.4.

4.3.3.2. Operation

SJTAG-PCK authentication is a three step process that generally requires a trusted server to work. it works as follows:

- The user requests the CHIP_ID from Secure JTAG and uses it to request the public key for the device. The trusted server is responsible for authorizing the user, the method of which is not part of the SJTAG authentication protocol.
- The public key is communicated to SJTAG which uses the hash stored in the eFuses to validate it's correctness. If correct, SJTAG uses the DTRG to generate a random challenge.
- The random challenge is passed to the trusted server. Using the associated private key, the trusted server encrypts a message containing the random challenge, the chip id, and the appropriate access levels.
- The package is then passed to SJTAG, which uses the public key to verify the package and generates the access signals specified in the package.

4.3.4. Access Controller

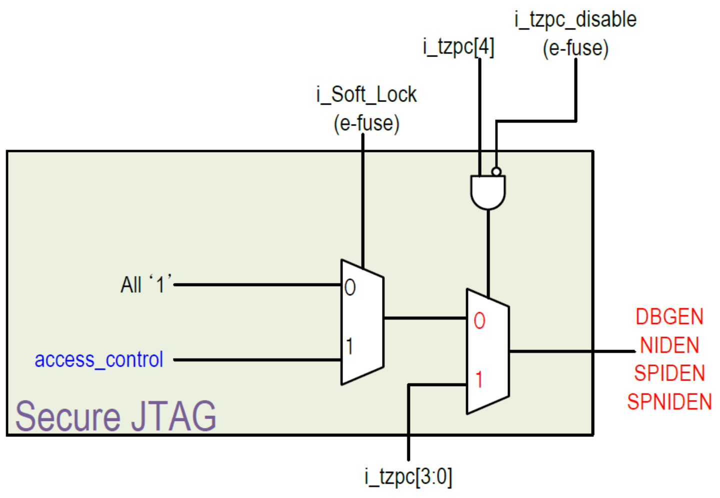


Figure 4.15: Access control signals found in secure JTAG [35].

The access controller shown in Figure 4.15 is a component in both SJTAG modules. It controls the debug signals and ultimately debug access. As such, is it worthwhile to describe the ingoing signals in details.

- **Soft Lock** is an eFuse controlled signal that enables security. All debug signals are sent while this bit is 0. Fusing this bit to 1 enables security and causes the state of the debug signals to be handled by the SJTAG controller.
- **access_control** is the input from the SJTAG controller. It determines the state of the debug signals as long as Soft Lock is set.
- **TZPC bits [3:0]** provide an alternative method of specifying the debug signals. This input is connected to the TrustZone.
- **TZPC bit 4** Overrides both Soft Lock and the SJTAG controller in favor of TrustZone control.
- **TZPC Disable** is an eFuse that, when set, permanently disables TrustZone control over the debug signals.

4.3.5. JTAG Detector

Very little is known about this component. It detects a JTAG connection by monitoring the TMS signal. It's purpose is otherwise unknown. It does resemble Patent US20120060067A1 [59] which describes an authentication mechanism for JTAG boundary scan. The patent references using SJTAG for password based authentication.

4.4. In Practice

This section aims to highlight some potential security vulnerabilities that rely on the ARM debug signals to be enabled. This section is by no means an exhaustive list and serves only to illustrate the dangers of a successful attack on Secure JTAG (SJTAG).

4.4.0.1. Privilege Escalation

ARM CPU's execute code at one of four execution levels. These are referred to as EL0 through EL3. EL0 has the lowest privilege and is meant for user-level code execution. EL3, conversely, is the highest level of privilege and used for TrustZone execution. As lower privilege levels cannot access areas dedicated to higher privilege levels this provides a compartmentalized means of security. The NAILGUN attack [41] circumvents this security measure by using the on-board debugging feature of ARM CPU. Using this feature it's possible to load and execute an arbitrary payload at EL3 privileges from EL0. This is possible because the CPU executes debug instructions at the highest level of privilege. This attack is prevented by the debug signals because the CPU cannot be put into debug mode without both DBGEN and SPIDEN signals active.

4.4.0.2. Security SubSystem

Some devices contains a component referred to as the Security SubSystem (SSS) [35]. It provides a dedicated isolated processor, a root of trust, secure storage and cryptographic hardware accelerators. To achieve a higher level of overall system security it is designed to be completely isolated during operation, even from the CPU. It provides a mailbox which is the sole means of communication between the SSS and the rest of the system. As a complex system like this needs to be tested and as such it is connected to the Debug Access Port (DAP) using the AHB-AP. Connecting to the SSS over AHB-AP gives full access over the entire subsystem, and makes for an interesting attack vector. Normally this approach is closed because AHB-AP access is locked by both DBGEN and SPIDEN.

4.4.0.3. Flash Memory Protectors

Modern Android devices encrypt their local storage for the protection of it's users. While this encryption can be performed on the CPU doing so is resource intensive. An alternative is using a hardware component such as an Flash Memory Protector (FMP) [35] that encrypts and decrypts data as it is passed to the storage medium. In it's operation it stores a high-entropy hardware key in non-volatile memory. This key is used in combination with a significantly simpler user key during encryption and decryption operations. Access to these registers can only be obtained by the TrustZone. As the debug system operates at the highest privilege level it can be used to obtain access to the registers and thus the hardware key. Provided, of course, that the debug access signals are active.

5 Attacks on Secure JTAG

This chapter focuses on potential attack methodologies for Secure JTAG (SJTAG). Section 5.1 discusses the applicability of the attacks discussed in Chapter 2 to SJTAG. Section 5.2 describes a novel model that measured the cost to perform an attack. This section also applies the model to the attacks, and determines the optimal attack vectors. Section 5.3 details the attack methodology of the first of these attacks: Boundary Scan Chain (BSC) reversing. Electro Magnetic (EM) fault injection is the second attack, and it's attack methodology is described in Section 6.3

5.1. Potential Attacks on Samsung Secure JTAG

Chapter 2 discussed a classification of hardware attacks. This Section discusses the feasibility of their application on the Secure JTAG (SJTAG) Intellectual Property (IP).

Three major classes of attacks were discussed in Chapter 2. Specifically, attacks were divided into attacks on data, functionality and intellectual property. This provides a good starting point for comparison. The device under target is configured to use STJAG_SHA2

5.1.1. Attacks on Data

Attacks on data focus on extracting or modifying a stored secret. Extracting a secret shows an immediate problem with this class: SJTAG does not store any secrets. Both modes rely on the non-reversible nature of the SHA2 hashing algorithm for their security. While these attacks could be used to extract the target hash, this is pointless for two major reasons: Firstly , obtaining this hash does not affect security whatsoever as the required input cannot feasibly be calculated from the hash. This also leads into the second reason: The target hash is exposed to the end-user through a register. This register can be read without any debugging privileges making any attack on it unnecessary.

Side-channel analysis deserves a special mention. While there is no secret to recover, thus making it useless as a primary attack vector, it may yet play a role. It may be possible to use it to derive the timing or location of certain mechanisms to support different attacks.

Attacking the values stored in the eFuses is a viable attack vector. An attacker could change the target hash into one with a known input. Alternatively, disabling the SOFT_LOCK fuse would disable security outright. This kind of attack, however, requires severely undermining the functionality of eFuses. As such, attacking eFuses is classified as an attack on functionality, and will be further discussed in Section 5.1.1

The target does appear to have JTAG boundary scan chains through JTAG-AP described in Section 4.2.5. Any possible security of JTAG-AP is undocumented and may be non-existent. If this is indeed the case, at least three interesting use cases exist:

1. The Security SubSystem (SSS) and holds most of the cryptographic components of the system. The SSS controller is an ARM Cortex-M0 and is connected to the internal JTAG Boundary Scan Chain (BSC). By taking control of the controller, it may be possible to exploit the rest of the SSS

2. SJTAG contains an otherwise unspecified controller. As SJTAG is not a computationally demanding component, this is most likely another low-powered controller such as the ARM Cortex-M0. These controllers do not typically have CoreSight support but still need to be tested. In light of this, it is not

unreasonable to expect the SJTAG controller on the BSC. Taking over the controller through JTAG could potentially let an attacker disable the debug systems outright.

3. Several Advanced eXtensible Interface (AXI) connected components, such as the Multi-Function Codec described in Chapter 47 of the Exynos 7420 Documentation [35], have JTAG interfaces. By exploiting them through JTAG it may be possible to use these components to connect to the AXI interface and undermine the device security of components connected to the Advanced High-performance Bus (AHB).

5.1.2. Attacks on Functionality

Attacks on functionality is the most interesting class for SJTAG. Disrupting normal operation in such a way that SJTAG starts sending the debug signals would result in full debug access on the System on Chip (SoC). Section 5.1.2 listed the following attacks:

- **Microprobing**

  Microprobing could work in theory. By probing the appropriate signal it may be possible to enable the debug signals directly by injecting a current. However, the target's SoC is fabricated at 14nm [35] making the technology too small for microprobing to be feasible.

- **Fault Injection**

  SJTAG has a number of components that, if they were to fail, could generate the access signals. A fault in the access controller shown in figure 4.15 could potentially generate the required access signals. Alternatively, a fault during authentication may cause SJTAG to accept an invalid password. This makes this attack vector applicable.

- **Focused Ion Beam (FIB) Modifications**

  Similar to microprobing, modifying the circuit to output the required access signals could work. FIB Modifications may also be employed to restore the eFuses to their original values, thus possibly disabling the security outright. FIB modifications are very difficult to perform, especially on a SoC this complex. However, it remains a applicable point of attack.

5.1.3. Attacks on IP

Attacks on IP are focused on obtaining more knowledge of a system, often with the intent to pirate the IP. As such, this class is not viable as an attack in order to gain access to the debug system by itself. This class may still be useful in gaining more knowledge about the system. For instance, reverse engineering the CPU may help locating critical components. It is even a required step in order to perform certain attacks such as FIB modifications.

5.1.4. Discussion

By comparing the attacks in discussed in Chapter 2, it is possible to conclude the following attacks to be applicable:

1. JTAG BSC reversing
2. Non-invasive Fault Injection
3. Invasive Fault Injection
4. FIB modifications

Non-invasive and Invasive fault injection have been separated into their own category because the requirements to perform the attack differ. One additional attack vector exists, but falls out of the hardware attack space:

5. Attacks on the SHA2 hashing algoritmn.

These are cryptographic attacks which would undermine the security of SJTAG. Although SHA2 is still considered to be secure, it should be included for the sake of completion. Attacking the password directly is not viable; not enough is known about it and attacks on the password would be equivalent to the entire SHA-2 keyspace.

5.2. Cost Evaluation of the attacks

In order to determine which of the attacks listed in Section 5.1.4 is worth investing in it's important to be able to evaluate the potential attacks. To evaluate vulnerabilities models as CVSS [22] and DREAD [39] have been developed. These models focus primarily on defense and threat analysis rather than offense which makes them unsuitable for this project.

To facilitate the needs of the project a new model aimed at cost evaluation of attacks was created. it's design has been documented in section 5.2.1 and was used to evaluate our attacks in Section 5.2.2.

5.2.1. Model Design

The model works similar to CVSS in that it has a number of categories with a selection of clearly specified options. These options each contribute to a final score between which correlates with the severity of the exploit. This makes it easy for a user of the model to quickly assess a vulnerability.

To do the same for an attack, one must first determine the core of the requirements. Cost to perform an attack and execution time are the most obvious two factors, but neglect the research complexity of an attack. The research complexity of an attack is best judged in both required knowledge and effort required to get to the point where an attack can be executed. These four variables determine the difficulty of a given attack but do not show anything about the usability of the attack. Performing a difficult attack may not be worth it if it rarely yields results. Likewise, one may be more worthwhile if a successful attack yields results that can be reused to speed up subsequent attacks. When considering an attack it is wise consider any ethical concerns of your attack. If a successful attack causes significant harm to people, the economy or the environment it may be wise to reconsider executing it. Lastly, especially when dealing with hardware, there is a risk of permanent damage to your target. Depending on the goal destruction may or may not matter in the end. This results in 13 classes:

- Research Complexity:

  1. **Knowhow**

     Knowhow represents the missing knowledge required to understand the theoretical background of an attack, and is measured in the amount of time that needs to be invested order to gain an understanding of the attack. This can vary between attackers and needs to be selected appropriately.

  2. **Effort**

     Effort represents the time needed in order to get to the point where an attack can be executed. This includes time required to resolve device specific knowledge or to build an attack setup. This can vary between attackers and needs to be selected appropriately.

- Execution Complexity:

  3. **Specialized Tools**

     Represents the cost of executing the attack. Attacks may require specialized tools such as a Focused Ion Beam (FIB) which would need to be purchased. Can vary per attacker as certain labs may already have the required equipment. This class does not include research costs, which is covered by the knowhow class.

  4. **Time Requirements**

     The pure execution time of an attack. Does not include any other phase, as that is covered by the effort class.

- Usefulness:

  5. **Reproducibility between devices**

     Successful attacks may expose additional knowledge about the system, which speeds up successive attacks on other, identical, devices. For example, an attack may result in access to some master key that can be used to speed up attacks. Depending on the goal of the attack it may even be possible that additional attacks are unneeded entirely.

6. **Reproducibility on exact same device**

   Successful attacks may expose additional knowledge about the device, which speeds up successive attacks on the same device. An attack that dumps firmware could, for example, expose the system through easier software attacks. Like the other reproducibility class, reproducing the attack may be unneeded entirely.

7. **Determinism**

   Determinism represents the chance that a successful attack yields results. Take note that this does not represent inherent randomness to certain attacks such as fault injection. To illustrate, if a fault injection attack crashes the device it is generally possible to try any number of additional times until success is achieved. Conversely, an attack may rely on a secret remaining in a temporary location. Once the device clears such a secret, the attack may never work again.

8. **Reusability**

   Reusability factors in the result of a successful attack. A difficult attack may be worthwhile if it results in a particulary valuable secret being exposed. Take note that this significantly extends past reproducibility amongst devices. An important enough breach may have far reaching consequences. Such as, for instance, the signing key of a root certificate authority.

9. **Device impact**

   Device impact judges the overall effect an attack has on the security of a device. Some attacks, while successful, don't affect the device's security at all. Others may give an attacker full access to the entire system.

- Risk:

10. **Personal Impact**

    Attacks may have very unfortunate side-effects depending on the target. Personal impact represents expected damage in areas such as personal privacy, security or freedom.

11. **Economic Impact**

    Attacks may also have very unfortunate economical side-effects. Economic impacts considers expected financial damages.

12. **Environmental Impact**

    Attacks may target critical infrastructure and can cause environmental damage as a result. Environmental Impact assesses likely damage to the environment.

13. **Destructiveness**

    Certain attacks can cause risk to the device. Decapsulation for instance removes a lot of an Integrated Circuit (IC)'s protection, making it very easy to permanently damage it. Destructiveness factors the risk of breaking a device, and likely ruining the attack in the process.

5.2.1.1. Model Weights

First step in determining the scores associated with the classes is considering how the model will be used. CVSS [22] gives clear descriptions and assigns a value rather than exposing the scoring to the end-user directly. This gives a consistent way of grading across multiple users. CVSS also uses a scoring system from 0 to 10, which is an intuitive grading scheme and a good starting point. However, attacks are very different to vulnerabilities in one key aspect: A small vulnerability may not be worth fixing immediately but is still a vulnerability. An attack on the other hand may take so long to execute that it isn't a viable attack vector. Scoring these attacks on a 1-10 scale is difficult: an easy and high value attack that takes decades to perform could at best be graded a 10 in difficulty. Comparing a 10 to a difficult but possible attack with a grade of 9 does not properly represent the difference in complexity. Instead, the choice was made to represent the most difficult but viable attack as a 10. Any attack that is impossible to execute due to either cost or time has it's score artificially raised by at least 10. This makes it possible to compare the impossible attacks between themselves while providing clear separation between possible and impossible attacks.

Last goal is to make choices clear to the end user. This is achievable by using a logarithmic scale for each choice. By having a clear scoring distance between choices the difference between options is distinct.

This leaves us with the following base constraints:

1. Scoring should be done on a basis of clear descriptors
2. Grading uses a 0 to 10 scale as a baseline, with 10 being the most difficult viable attack
3. Properties that make an attack infeasible or impossible should raise the grade artificially past 10.
4. Every step in the difficulty should be a significant step up in terms of complexity.

These can be deviated from on a per-class basis but gives a good reference point to start with. The constraints have been applied as follows:

1. **Knowhow** has no special requirements and ranges from no outstanding knowledge to extreme time required to be resolved.
2. **Effort** has no special requirements. Ranges from no building/research time to extreme time investment.
3. **Specialized Tools** is the first class where attacks can become infeasible or impossible. The normal range ranges from no special tools required to so cost prohibitive only high-end labs have the budget. However, an attack may require a budget that is outside of the realm of known entities. An budget of this magnitude would be extremely challenging to acquire, but not impossible. The final step requires an impossible amount of resources or may not be resolvable through money.
4. **Time Requirements** is the other class that can make an attack infeasible or impossible. The maximum acceptable score is a few years, but certain attacks can go significantly over. A brute-force attack of a low-bit key may take 50 years to execute. While possible, 50 years for an attack is not feasible. However, exhausting an 2048-bit keyspace could take billions of years and is considered to be impossible.
5. **Reproducibility between devices** is highly dependent on the time requirements of an attack. The score of this category is a fraction of the Time Requirement score. Ranging from a multiplication factor of zero when reproducibility is unneeded to a factor of one if reproducing takes the exact same amount of time.
6. **Reproducibility on the exact same device** , like the other reproducibility class, is highly dependent on the time requirement of an attack. Likewise, its score is determined as a fraction of the time requirement ranging from 0 to 1.
7. **Determinism** has no special requirements.
8. **Reusability** is a positive factor making a difficult attack more worthwhile. As such, it ranges from -10 to 0 in order to lower the overall score of an attack.
9. **Device Impact** is another positive factor, but less important than reusability. As such, it ranges from -5 to 0.
10. **Personal Impact**, like both other impact classes, does not have special requirements of itself. Together they may contribute too heavily to the final score and are averaged together to compensate.
11. **Economic Impact**, like both other impact classes, does not have special requirements of itself. Together they may contribute too heavily to the final score and are averaged together to compensate.
12. **Environmental Impact**, like both other impact classes, does not have special requirements of itself. Together they may contribute too heavily to the final score and are averaged together to compensate.
13. **Destructiveness** has no special requirements.

These constraints and exceptions have been compiled in scoring tables 5.1, 5.2 and 5.3. They show both descriptions and the appropriate score assigned to them. This way a user of the model can easily identify and select the appropriate score for an attack. Note that table 5.1 contains the values IF and IM to describe infeasible and impossible attacks. As these selections artificially raise the score their weights need to be calculated to reflect an addition of 10 and 50 score, respectively. This makes it so that the highest score a feasible attack can have is 10, an infeasible attack ranges from 10 to 50, and an impossible attack has a score higher than 50. This also allows a user to compare impossible attacks to possible attacks by their first digits.

Table 5.1: Scoring table for the Research and execution complexity of the attack

1. Knowhow		**2. Effort**		**3. Specialized Tools**		**4. Time Requirements**	
No outstanding knowledge or assumptions	0	Virtually no building / research time	0	Requires no special tools or specialism	0	Executing the attack takes a few minutes	0
Missing knowledge or outstanding assumptions require little time investment to resolve	1	Some research or setup time	1	Tools cheap to acquire and don't need a lot of specialistic knowledge	1	Executing an attack takes a few hours	1
Missing knowledge and outstanding assumptions require considerable time investment to resolve	3	Considerable research or setup time	3	Tools steep to acquire privately but doable in most labs; requires specialistic knowledge to operate	5	Executing an attack takes up to a few days	2
Missing knowledge or outstanding assumptions require sigificant time investment to resolve	6	Significant research or setup time	6	Tool costs so expensive only high-end specialistic labs may be able to afford it. Requires significant time-investment by experts to operate	10	Executing an attack takes up to a few months	5
Missing knowledge or outstanding assumptions require extreme time investment to resolve	10	Extreme research or setup time	10	No known entity has this budget. Raising the budget is not impossible but extremely challenging.	IF (105)	Attack takes a few years to execute	10
				No amount of money or resources will be enough. Without some significant breakthrough the required equipment is out of reach for anyone	IM (525)	Attack takes a significant number of years that makes an attack impractical but not impossible	IF (105)
						Heat death of the universe will occur before this attack is completed. Without significant advances this attack is not feasible	IM (525)

Table 5.2: Scoring table for the usefulness of an attack

5. Reproducibility between Devices		6. Reproducibility on exact Same Device		7. Determinism		8. Reusability		9. Device Impact	
Attack does not need to be reproduced to achieve results	x0,0	Attack does not need to be reproduced to achieve results	x0,0	Properly executed attack will always yield results	0	Results can be used beyond manufracturer	-10	Attack results in total loss of security of entire device	-5
Reproducing the attack requires minimal effort	x0,1	Reproducing the attack requires minimal effort	x0,1	Properly executed attack will generally yield results	2	Results can be used beyond device family	-7	Attack results in total loss of security for important subsystems but not entire device	-4
Reproducing the attack requires moderate effort or time	x0,4	Reproducing the attack requires moderate effort or time	x0,4	Properly executed attack will sometimes yield results	5	Results can be used amongst device family	-4	Attack results in partial loss of the device's security	-2
Reproducing the attack costs a significant amount of time	x0,8	Reproducing the attack costs a significant amount of time	x0,8	Properly executed attack will rarely yield results	10	Results can be used amongst production batch	-2	Attack results in partial loss of the device's subsystems	-1
Reproducing the attack on another device requires roughly equal amount of research effort as the initial attack	x1,0	Reproducing the attack on another device requires roughly equal amount of research effort as the initial attack	x1,0			Results are per device and do not carry over	0	Attack does not result in (additional) loss of security of the device or its subsystems	0

Table 5.3: Scoring table for the risk of an attack

10. Personal Impact	**x1**	**11. Economic Impact**	**x1**	**12. Environmental Impact**	**x1**	**13. Destructiveness**	**x1**
no impact expected in areas such as personal privacy, security or freedom	0	No economic impact expected	0	No environmental Impact Expected	0	Attack does not put the device at risk	0
Slight impact expected in areas such as personal privacy, security or freedom	1	Slight economic impact expected; Economic losses as a reaction but not a targeted attack	1	Slight environmental impact expected	1	Low risk to the device; Opening the casing, soldering on the board etc.	2
Considerable impact expected in areas such as personal privacy, security or freedom	3	Considerable economic impact expected; Targeted Attack designed to cause economic losses	3	Considerable environmental impact expected	3	Medium risk to the device. Desoldering, transplants	6
Significant impact expected in areas such as personal privacy, security or freedom	6	Significant economic impact expected. Targeted attack aimed to destroy financial assets	6	Significant environmental impact expected	6	HIGH risk to the device; Any step may destroy device (decapsulation die modifications)	10
Severe impact expected in areas such as personal privacy, security or freedom	10	Severe Economic impact expected. Targeted attack with far-reaching economic losses	10	Severe environmental impact expected	10		

5.2.1.2. Model formula specification

To formally specify the method of calculating the score of an attack some variable definitions are required:

$$S_n : \text{Value selected by user in column } n \text{ of the scoring tables}$$
$$M_n : \text{Vector of all values contained in column} n \text{of the scoring tables}$$
$$G : \text{Final grade}$$
$$L_n : \text{maximum value in column } M_n \text{ that is not } \mathit{IF} \text{ or } IM.$$

For clarification, the following variables will also be used:

$$\mathit{IF} : \text{ Value representing the artifical score of Infeasible rated selections}$$
$$\mathit{IM} : \text{ Value representing the artificial score of Impossible rated selections}$$

$$G_{maxcomplexity} : \mathit{IF} \text{ subtotal for the maximum of the complexity class}$$
$$G_{maxusefulness} : \mathit{IF} \text{ subtotal for the maximum of the usefulness class}$$
$$G_{maxrisk} : \mathit{IF} \text{ subtotal for the maximum of the risk class}$$
$$G_{max} : \text{Maximum score that can be attained in the model}$$

$$G_{mincomplexity} : \mathit{IF} \text{ subtotal for the minimum of the complexity class}$$
$$G_{minusefulness} : \mathit{IF} \text{ subtotal for the minimum of the usefulness class}$$
$$G_{minrisk} : \mathit{IF} \text{ subtotal for the minimum of the risk class}$$
$$G_{min} : \text{minimum score that can be attained in the model}$$

$$R_{diff} : \text{ Value for the subtotal resulting from column 5}$$
$$R_{same} : \text{ Value for the subtotal resulting from column 6}$$

Supporting variables are calculated as follows:

$$R_{diff} = S_4 S_5$$

$$R_{same} = S_4 S_6$$

$$G_{maxcomplexity} = \sum_{i=0}^{4} \min(\max(M_{i,0} \dots M_{i,n}), L_i)$$

$$G_{maxusefulness} = \sum_{i=5}^{6} \min(\max(M_{4,0} \dots M_{4,n}), L_i) M_i + \sum_{i=7}^{9} \min(\max(M_{i,0} \dots M_{i,n}), L_i)$$

$$G_{maxrisk} = \frac{\sum_{i=10}^{12} \min(\max(M_{i,0} \dots M_{i,n}), L_i)}{3} + \min(\max(M_{13,0} \dots M_{13,n}), L_i)$$

$$G_{max} = G_{maxcomplexity} + G_{maxusefulness} + G_{maxrisk}$$

$$G_{maxcomplexity} = \sum_{i=0}^{4} \min(\max(M_{i,0} \dots M_{i,n}), L_i)$$

$$G_{maxusefulness} = \sum_{i=5}^{6} \min(\max(M_{4,0} \dots M_{4,n}), L_i) M_i + \sum_{i=7}^{9} \min(\max(M_{i,0} \dots M_{i,n}), L_i)$$

$$G_{maxrisk} = \frac{\sum_{i=10}^{12} \min(\max(M_{i,0} \dots M_{i,n}), L_i)}{3} + \min(\max(M_{13,0} \dots M_{13,n}), L_i)$$

$$G_{max} = G_{maxcomplexity} + G_{maxusefulness} + G_{maxrisk}$$

$$G_{mincomplexity} = \sum_{i=0}^{4} \min(M_{i,0} \dots M_{i,n})$$

$$G_{minusefulness} = \sum_{i=5}^{6} \min(M_{4,0} \dots M_{4,n}) M_i + \sum_{i=7}^{9} \min(M_{i,0} \dots M_{i,n})$$

$$G_{minrisk} = \frac{\sum_{i=10}^{12} \min(M_{i,0} \dots M_{i,n})}{3} + \min(M_{13,0} \dots M_{13,n})$$

$$G_{min} = G_{maxcomplexity} + G_{maxusefulness} + G_{maxrisk}$$

$$IF = G_{max} - G_{min}$$

$$IM = 5IF$$

To calculate the final grade of an attack the following formula is used:

$$G = 10 \frac{\sum_{i=0}^{4} S_i + R_{diff} + R_{same} + \sum_{i=7}^{9} S_i + \frac{\sum_{i=10}^{12} S_i}{3} + S_{13} + |G_{min}|}{G_{max} - G_{min}}$$

To elaborate on these equations:

- *IF* is calculated as maximum value the grading can have, offset by the lowest value. This causes an *IF* score to cause the grading to be increased by 10 points.
- *IM* is calculated by being five times *IF*, because *IF* can show up in an equation up to four times plus the normal grading. This causes impossible scores to offset the final score by 50.
- The final grade is the average of all scores, scaled to fit in a 0-10 range for feasible attacks.

5.2.2. Attack Evaluation

The attacks specified in section 5.1.4 can be evaluated using the model defined in Section 5.2.1. Table 5.4 shows the summary of the evaluation, while the complete evaluation has in documented in Tables A.1, A.2, A.3,A.4 and A.5 of Appendix A. The rest of this section discusses the attack's various scores, sorted by their total cost.

Attack	Total Cost	Top Contributors
JTAG BSC	1.84	Research complexity
Non-invasive fault injection	2.66	Execution complexity
Invasive fault injection	3.78	Execution complexity and risk
FIB attacks	5.24	Research complexity and execution complexity
SHA2 attacks	201.46	Execution complexity and reproducibility

Table 5.4: Summary of attack evaluation

1. JTAG Boundary Scan reversing

   JTAG Boundary Scan Chain (BSC) reversing scores the lowest of all attacks, which makes it a very interesting attack. The success of this attack hinges on the JTAG boundary scan chains being entirely unsecured. This makes verifying whether an attack works very simple, and this is properly reflected in the model. Execution time of an attack is low, but non trivial, and would need to be executed again for each attempt. Given that the nature of a mobile phone it is likely that there will be considerable impact to the privacy of the device's owner, but not beyond that.

2. Non-invasive Fault injection

   Fault injection in general requires more research and effort in order to create a good attack method than JTAG reversing does, but still manageable. Budget is a larger concern, as some fault injection equipment can be expensive. Due to the random nature of fault injection attacks an attack can be hard to perform, but once the variables are known it should be easier to reproduce.

3. Invasive Fault Injection

   The invasive variant of fault injection is very similarly scored to non-invasive fault injection. The required decapsulation makes an attack significantly more dangerous to the device which is reflected by a higher final score.

4. FIB modifications

   FIB modifications require significant amounts of research in order to find the vulnerable components. Execution is comparatively easy, but still a problem. Budget may prove to be an issue as an FIB is very expensive. Attacks are persistent however and do not need to be executed a second time on the same device. Execution of the attack is very dangerous as any small mistake can destroy the device.

5. SHA2 Attack

   Attacking SHA2 is the only attack which had a final score of over 50. This makes the attack to be considered impossible to execute. This result is expected, as attacking the entire 256bit keyspace is computationally unreasonable. The score of this attack remains noteworthy as it makes a good case for the scoring mechanism. The score of 201,46 clearly places it out of the feasible attack space. This is primarily caused by the execution complexity of attacking SHA2. This score also shows the effects of a potential critical vulnerability in the SHA2 algorithm. If one were to be discovered that trivializes execution time then the final score would drop down to 1,46. Such a scenario would make this the optimal path of attack.

According to the model, JTAG BSC reversal is the easiest attack vector. While it does rely on the assumption that the JTAG BSC are not secured, this can be verified early in the process. If this assumption proves to be false, Non-invasive fault injection is a second best method.

5.3. BSC Reversing Attack

As discussed in Section 5.2.2, an attack over JTAG Boundary Scan Chain (BSC) would be the least costly attack. These attacks rely on the on-board JTAG BSC test system being entirely unsecured in order to gain total control of the connected devices.

JTAG attacks start with the following steps:

1. Locate the JTAG testing points on the board. This may require reparation of the JTAG connectors.
2. Connect to JTAG.
3. Put the JTAG state machine (Figure 3.2) in Shift-IR state.
4. Shift 1's into the instruction register until the device shifts out successive 1's. This allows an attacker to determine the length of the instruction register. The active instructions are now BYPASS for each component thereby reducing the direct register to the number of components.
5. Set the JTAG State machine to Shift-DR.
6. Shift in a pattern until TDO shifts the pattern back out. This allows an the determine the current length of the selected Data Register (DR). Because BYPASS is the active instruction, this is equal to the amount of connected components.
7. Determine the length of the data-registers for each possible instruction:
   (a) Start with binary number $n = 0$
   (b) Put the JTAG-state machine into SHIFT-IR.
   (c) Shift in n, pad with zeroes for the remainder for the length of the register.
   (d) Put the JTAG state machine in Shift-DR.
   (e) Shift in pattern until pattern is detected. This is the allows an attacker to determine the lenght of the DR for instruction n.
   (f) $n = n + 1$.

(g) Repeat from step 7b for the rest of the instruction length.

These steps allow an attacker to derive the amount of components on the scan chain as well as the amount of supported instructions. These provide the basic building blocks of a JTAG based attack. Optionally, the IDCODE instruction can be used to determine which components are connected on the scan chain. This is not a required instruction and may be encoded in differently by various vendors. However, IDCODE registers are standardized as 32 bits long. Previous steps determined the amount of components and the length of each possible chain. The IDCODE instruction's DR length should be 32 times the amount of components, and match with one of the discovered DR's lengths.

5.3.0.1. Tools

Automated tools such as the JTAGulator [27] exist and are a tremendous help in locating JTAG connections and executing initial steps outlined above. However, in the case of Arm Debug Interface Version 5 (ADIv5) equipped devices, JTAGulator only gets an attacker halfway. Specifically, it will only aid in locating JTAG-DP. JTAG-DP is the external JTAG interface that provides access to the Debug Access Port (DAP). JTAG-DP is further described in Section 4.2.4. JTAG-AP, detailed in Section 4.2.5, provides access to the actual scan chains of the System on Chip (SoC) and is the actual target of the attack. The JTAGulator provides no support for JTAG-AP , and is purely intended for external JTAG interfaces such as JTAG-DP.

Communication with JTAG-DP is well documented and fully supported by tools such as OpenOCD [43] and the official ARM DS-5 debug tools [3]. Both tools allow quick access to the CoreSight debug system by providing full support for MEM-AP. JTAG-AP is not implemented by OpenOCD at all. The DS-5 suite provides support for components on JTAG-AP provided the user can identify the exact component and it's location on the JTAG BSC. This does not give an attacker enough control to be able to explore the JTAG scan chain. The DS-5 debug software version 5.29.1 comes with CSAT: the CoreSight Access Tool. This is a commandline application that interfaces well with the official ARM DS-5 debug hardware and provides low-level support for CoreSight debugging. Like OpenOCD, it provides excellent support for MEM-AP but none for JTAG-AP.

Because JTAG-AP is a variant of MEM-AP support for JTAG-AP can be implemented using either OpenOCD or CSAT. This implementation resulted in the CoreSight Management Script, a wrapper for CSAT expanding the standard functionality and implementing communication with JTAG-AP, The CoreSight Management Script has been further described in Section 6.1.3.

5.4. EM Fault Injection Attack

This section explores Fault injection attacks on Secure JTAG (SJTAG), which poses it's own unique challenges.

Generally, fault injection attack methodologies rely on what is called the skipped instruction model [40]. When a fault is injected into the system any number of operations can fail. For instance, reading an instruction from memory could fail or be corrupted. Alternatively required computational blocks could be unresponsive or take too long to process. Modeling all possible faults is a complex task indeed, let alone verifying which fault occurred. Luckily, most of these faults all result in the same thing: Instructions not being executed as they should, effectively skipping them.

A common approach in setting up an attack methodology is inspecting the firmware to find locations where a skipped instruction would circumvent device security. This inspection needs to be performed on the binaries, rather than the source code as the compiler significantly affects executed code. Demonstrating said vulnerability requires very precise timing on the correct position. Innate inconsistencies can also introduce slight variations in the required timing of an attack. As such, demonstrating an existing vulnerability is complicated and partially left to chance. By using the knowledge of a potential skipped instruction vulnerability and testing whether or not the device responds to fault injection attacks it's possible to conclude that a theoretical vulnerability exists.

For SJTAG this leads to an obvious problem: There is no way to get the SJTAG firmware, if it even exists. It is a dedicated hardware component embedded into the System on Chip (SoC). It is designed to be active during manufacturing without any external data sources attached. This makes it impossible to apply the skipped instruction fault model to SJTAG. While this does not mean it cannot be glitched, building a good theoretical foundation is extremely challenging. Without such a foundation it is impossible to make a distinction between an attack that failed because the target is not vulnerable or an attack that failed despite the target being vulnerable. Because chance is such a large factor it is vital to have a high amount of confidence in the other factors of the attack.

5.4.1. Potential Fault Injection Vulnerabilities in Secure JTAG

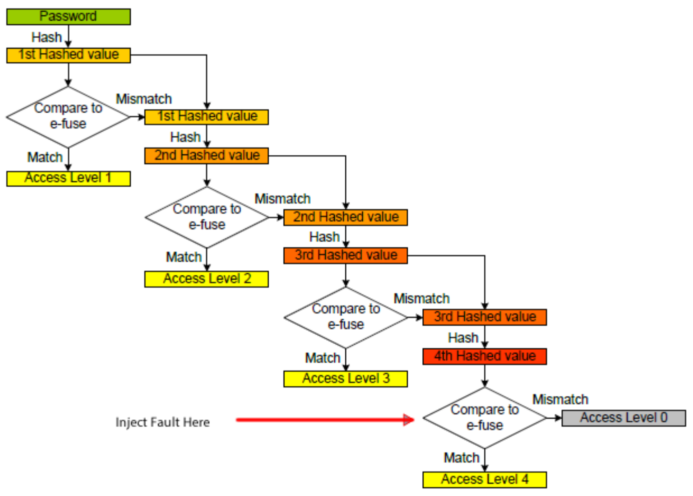


Figure 5.1: Potential fault injection vulnerability in the authentication scheme of Secure JTAG [35]

SJTAG authentication is handled by two components: The SJTAG controller and the Access Controller. The Access controller was discussed in Section 4.3.4. Figure 4.15 showed two multiplexers that, if glitched, could potentially lead to a transient fault in the debug security. While such a transient fault can potentially be exploited in the Debug Access Port (DAP), this would add yet another unknown factor to the attack.

The SJTAG SHA2 authentication mechanism is much more interesting. Figure 5.1 shows that a fault on the fourth comparison of the hash. Depending on the implementation of this check it's possible that a fault could accept an incorrect password. Such a fault would potentially accept the wrong password, thus escalating privilege to the highest level.

5.4.2. Fault injection on Secure JTAG

As described in Section 2.3.2, There exist 3 major attack vectors for non-invasive fault injection: the Clock, Voltage and Electro Magnetic (EM). According to the documentation [35], the target uses Phase-locked loops for it's clock inputs. These make the system impervious to clock glitches, as they are designed to absorb them. Voltage glitches have more of a chance to work, but the many capacitors in the system would likely absorb such an attack. Because of this the choice was made to use state of the art EM fault injection techniques.

Because it is impossible to know if the attack can be successful it's important to have confidence in the attack. EM Fault injections consist of 3 primary variables:

- **Time**

  Time considers the is the timing of Secure JTAG's authentication, respective of the measurement point. By locating the four hashing rounds in the EM spectrum it's possible to determine the moment that compares the hashing result to the stored password hash. This would be the optimal time to attack.

- **Location**

  After locating the Secure JTAG hashing rounds, the next step is determining the location of the hashing engine. This can be done by finding the location with the strongest signal radiation. While this is most likely not the correct location for glitching, it gives a good initial indication, thus decreasing the attack space.

- **Power**

  The amount of power required to inject a fault is variable for each device and other various external factors. This can not be determined beforehand and must be done during the attack.

By determining the time and location as much as possible beforehand, the attack space is greatly reduced. This allows for greater confidence in the attack, which in turn leads to greater confidence in the final conclusion.

5.4.3. Side-channel analysis to locate Secure JTAG

Section 5.4.2 described Side-Channel analysis as an interesting way to determine the timing and location of Secure JTAG. This consists of 3 major steps:

- Trigger setup

  Both Side-channel analysis and Fault Injection require a consistent point to start measuring from. When attacking software common methods for this are enabling a GPIO pin or performing communication over a serial connection just prior to the target code. This does need code execution, so this is not always an option. Because SJTAG is a hardware component accessed through the debug system this provides some challenge. Solutions are device specific and described in Section 6.3.1.

- Finding the signal of the hashing rounds

  Riscure Inspector EM Probe Station is a solution for EM analysis. It allows a researcher to capture EM radiation over time and store it as a trace. By filtering these traces patterns can be revealed hinting at the behaviour of the SoC. This step focuses on finding patterns that correlate with SJTAG's hashing mechanism. Exact methodology is device specific and will be detailed in Section 6.3.2.

  A signal's correlation with SJTAG's hashing engine has to be confirmed in two ways:

  1. Entering 8 password bytes should consistently trigger the hashing engine, and this should correlate over several traces.
  2. When entering 7 password bytes the hashing engine should not trigger, showing a clear difference between the traces made in step 1. There is a chance that the hashing engine runs on each entry. If this step does not show a clear difference, entering no password bytes or disabling the SJTAG clock should be considered.

- Locating Secure JTAG

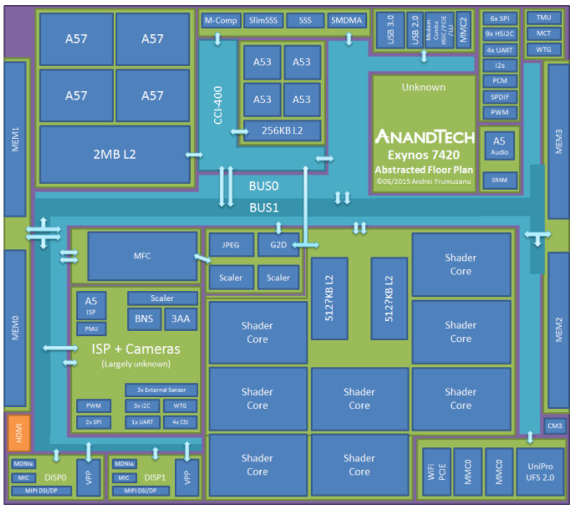
  With the knowledge of where the Secure JTAG signals show up in the EM spectrum it's possible to find the location where that signal is the most intense. This location should correlate with the approximate location of the Secure JTAG hashing engine.

6 Validation and Result Analysis

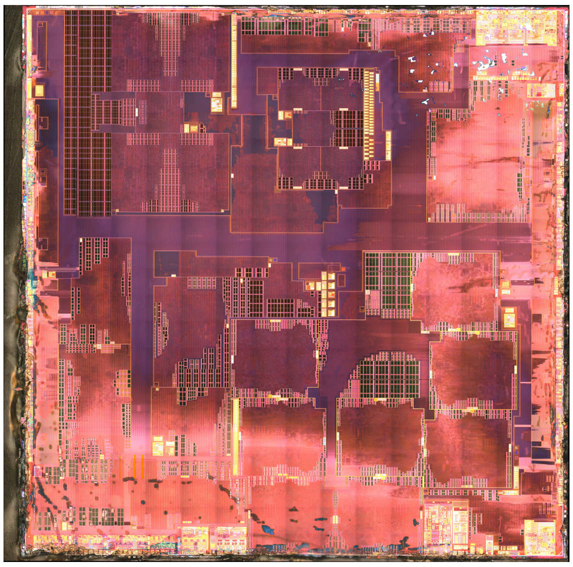
This chapter discusses the implementation of the attacks discussed in Chapter 5. Section 6.1 describes the attack platforms. The implementation and results of the first attack, Boundary Scan Chain (BSC) reversing, are discussed in Section 6.2. The second attack, it's implementation and results, are discussed in Section 6.3.

6.1. Platform

Secure JTAG (SJTAG) can be found on the Exynos 7420 System on Chip (SoC). This SoC has been developed for the Samsung Galaxy S6 family of devices, but can also be found on the Samsung Galaxy Note 5, Samsung Galaxy A5 (2016) and the Meizu PRO 5 [51]. It was created using Samsung 14 nm finFET process, and is a package that has the RAM stacked on top of the SoC. Figure 6.1a shows a reverse engineered floorplan of the die made by Anandtech [20]. Figure 6.1b shows the layout of the die as it was reversed during the course of the project in order to validate Anandtech's floorplan. While some discrepancies exist, such as the location of the L2 cache, it does appear to be the same SoC. It is possible that the floorplan was based on a slightly different revision of the SoC.



(a) Reverse engineered floorplan of the Exynos 7420 by Anandtech [20]



(b) Reversed Engineered silicon die to verify Anandtech's floorplan

Figure 6.1: Overview of the Exynos 7420 die

The SoC contains two primary CPUs in what's referred to as a big.LITTLE Architecture:

- **Atlas**: the high-performance ARM Cortex-A57 based on an ARMv8 Architecture at 2.1GHz. It is labeled A57 in Figure 6.1a.

- **Apollo**: the power-efficient ARM Cortex-A53 based on an ARMv8 architecture at 1.5GHz. It is labeled A53 in figure 6.1a.

Apollo is used to save battery life while Atlas is only turned on when system requires additional computational power. Testing, available in the project documentation, verified this behavior and Atlas seamlessly enabled when the SoC was placed under stress. While making Apollo the standard processor is not ideal for performance, it is ideal for various external attacks. Indeed, a slower target platform gives potential attacks a larger window of time, thus making various time-reliant attacks easier to perform.

Section 4.4.0.2 referenced the Security Subsystem. This can be found directly north of Apollo, labeled SSS in Figure 6.1a.

The Galaxy S6 is described in Section 6.1.1, and will be the primary target platform. To support the attack the HowChip HC7420-UFSeMM is used as a development platform, it is described in Section 6.1.1. Lastly, Section 6.1.3.

6.1.1. Samsung Galaxy S6 (SM-G920F)

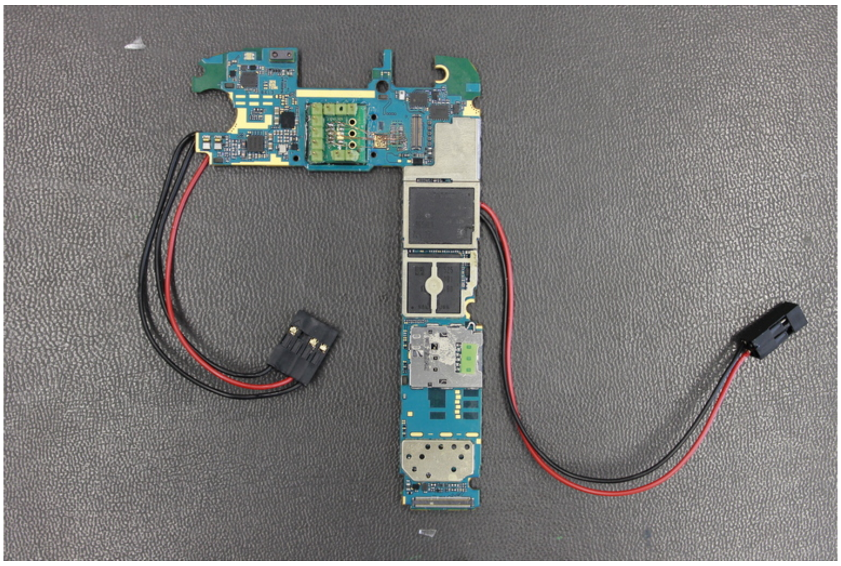


Figure 6.2: Prepared SM-G920F platform.

The Samsung Galaxy S6 is formally known by as the SM-G920F and is the default model of the Samsung Galaxy 6 family of devices. It was Samsung's Flagship phone on it's release in 2015 and has remained popular [52]. It's specifications are, outside of the Exynos 7420, irrelevant for this thesis.

Figure 6.2 shows the SM-G920F motherboard, which has connections soldered to the power and volume buttons for automated toggling. It also shows the attached header pins used to connect the DS-5 hardware debugger to JTAG-DP. JTAG-DP is hidden on the motherboard and the pads need to be exposed prior to soldering connections onto them. Connecting the ARM debugger to the JTAG-DP pads provides a connection to the Debug Access Port (DAP) and allows communication with the CoreSight debug system. The ARM debugger is further described in Section 6.1.3. SOFT_LOCK is fused on SM-G920F platforms, so full debug access is unavailable until successful SJTAG authentication.

For both fault injection and side-channel analysis there is a good chance the stacked ram dissipates the Electro Magnetic (EM) waves. This could make EM fault injecting Secure JTAG impossible. For this reason a second version of the SM-G920F Integrated Circuit (IC) has been prepared. To circumvent issues with the RAM this version has it's RAM milled away. This causes the device to loop during it's initial boot phases. This is not ideal for side-channel analysis as it provides a lot of random noise, but it is acceptable for fault injection.

With the RAM removed the system draws approximately 1 ampere from the power supply, indicating it is both on and operating. Note that this is significantly more than a device at rest, which draws around 0.1 ampere. While in this state the device still offers 1.8 volt on JTAG-DP's VDD-Ref, indicating the debug system is enabled. Connecting to the debug system allows access to SJTAG, which shows that the system is can be operated sufficiently for fault injection attacks.

3-pin connector Pin	RS-232 Serial Pin	Purpose
1	5	Ground
2	2	RX
3	3	TX

Table 6.1: RS-232 adapter Pinout

6.1.2. Development Platform: Howchip HC7420-UFSeMM

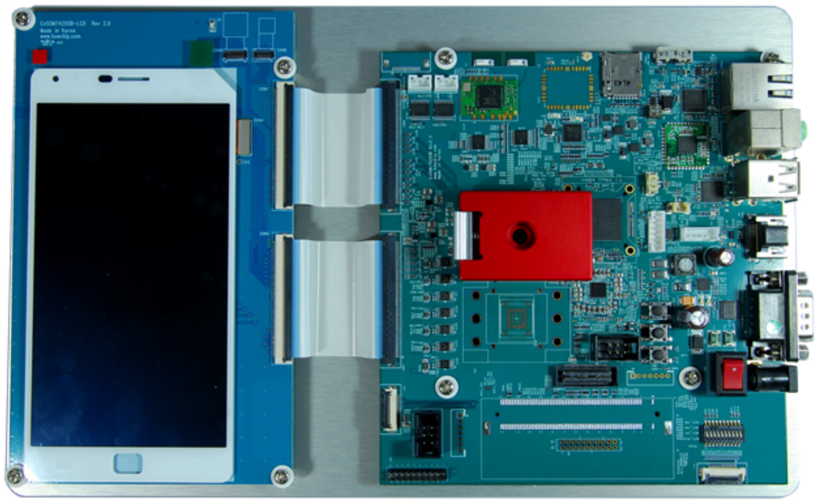


Figure 6.3: HC7420-UFSeMM developement platform

The HC7420-UFSeMM shown in Figure 6.3 is test and evaluation platform for UFS and eMMC chips[24]. While designed for rapid evaluation of flash chips, it also functions as a development platform. The platform allows unsigned code execution and has available sources to work from. This allows bare-metal code execution, which greatly benefits side-channel analysis. During boot it outputs logging information through UART based on RS-232 levels. This makes a lot of standard USB-to serial chips unusable, and an appropriate RS-232 adapter should be created. The pinout for the RS-232 adapter is shown in Table 6.1: SOFT_LOCK is not fused on this platform, enabling full debug support. This made the platform ideal for development of the CoreSight Management Script described in section 6.1.3.

6.1.3. CoreSight Management Script

ARM provides the ARM DS-5 Development Studio[3] to aid in the debugging of ARM based applications. It provides a high-level user interface for debugging applications on development hardware. While powerful, it's abstractions do not make it entirely fit to use as an attack platform on an unknown device. More importantly, at the start of the project no license for the DS-5 Development suite was available.

CSAT is an application included with the DS-5 Development studio. It provides an interactive shell that connects to the DS-5 debug hardware and does not require a license to operate. The shell has full support for ARMv8 and ARMv7 debug architectures, including MEM-AP, trace support, and more. It also provides convenient named access to standard debug registers, avoiding manual entry of the memory addresses.

However, CSAT has some shortcomings that make it a poor fit for research purposes. Specifically, these are:

1. No scripting support. It has batch support to execute a series of commands, but any sort of conditional branching is out of reach.
2. No SoC specific support whatsoever. While it is unfair to expect knowledge about arbitrary devices, knowledge about the system cannot be added by the user outside of the most basic debug components.
3. No parsing functionality. Memory reads return a 4 byte hexadecimal value requiring the user to make sense of the bits. Again an unfair expectation, but results in manual labor that is best avoided.

4. No command history (on Windows). This makes CSAT inconvenient to use, especially during research phases where a lot of iterations need to be executed.

This results in a system that requires the user, but especially an attacker, to constantly refer to the system's documentation for the appropriate memory addresses. This is very prone to user-error and is not a sustainable way to do research.

Lastly, CSAT has no support for JTAG-AP. As described in Section 5.3, this is the critical missing ingredient in reversing the scan chain. While theoretically JTAG-AP can be accessed through CSAT using the built-in MEM-AP support, it is a tremendously complex component.

The Coresight Management script has been developed to remedy CSAT's shortcomings and expand it's functionality. It is at it's core a Python3 wrapper around CSAT. This allows the script to reuse all the CSAT functionality while leveraging the extra power offered by Python3.

Python 3 rectifies CSAT's shortcomings in the following ways:

1. Scripting support. By wrapping Python3 around CSAT the full power of Python3 scripting is available.

2. Partial library for the Exynos 7420. Everything that has been researched has been implemented into a Python3 library for use with CoreSight Management Script. This allows easy reuse for various scripts.

3. Parsing functionality embedded in the library. Important status registers can be requested with descriptive names and parsed values.

4. Command history no longer necessary. Scripts operate vastly different to a commandline interface. Research iterations can be made quickly by modifying and executing the script.

Using CoreSight Management Script an attacker or researcher can implement SoC functionality once and make it permanently available in the Exynos 7420 library. This cuts down on a huge amount of referencing the documentation and makes the entire system less prone to user-error.

CoreSight Management Script also provides the following extensions to CSAT:

5. Full JTAG-AP support. Using Python3 allows CoreSight Management Script to leverage CSAT's MEM-AP support; It implements an intuitive set of functions to communicate with JTAG-AP.

6. Full Secure JTAG SHA-2 support. Allowing for easy authentication and reading the various status registers. Because none of the devices under test implement it, SJTAG_PCK mode is not supported.

7. Debug functionality such as halting the SoC and powering down cores. Requires DBGEN and SPIDEN flags.

8. Arbitrary Payload Execution through the debug system. Requires DBGEN and SPIDEN flags.

9. Static ELF loading through the debug system. Requires DBGEN and SPIDEN flags.

10. Firmware dumping of the Security SubSystem (SSS). Requires DBGEN and SPIDEN flags.

11. Disable SJTAG by disabling it's clock. This does not downgrade security.

12. GPIO toggling. Requires DBGEN and SPIDEN flags.

6.2. BSC Reversing Attack

Section 5.3 discussed a step by step method of the first phase of Boundary Scan Chain (BSC) reversing. The location of the external JTAG-DP ports is discussed in Section 6.1 and JTAG communication is handled by the CoreSight Management Script discussed in Section 6.1.3. The next step is figuring out the lenght of the Instruction Register (IR). If JTAG-AP is not secured then retrieving the lenght of the IR is possible.

6.2.1. Determining the Length of the Instruction Register

The principles behind determining the IR's length are straightforward. As discussed in Section 2.2.3 TDI and TDO are connected; Any input shifted into TDI will be shifted out of TDO after a period with enough clock cycles. This period corresponds to the length of the scan chain or, in this case, the IR. To determine the period an attacker shifts a recognizable pattern into TDI and shifts until it returns on TDO.

During development of JTAG-AP functionality in the Coresight Management Script it appeared that the IR was not responding in a way that was compliant with the JTAG standard [31]. Specifically, the JTAG-AP ports would not return any non-zero data over TDO. This is the first sign that JTAG-AP may be secured. Each JTAG-AP supports up to 8 JTAG ports, and it is important to validate that this occurs for all ports.

This test will be performed on the developer board. With SOFT_LOCK disabled in Secure JTAG it is assumed to have the lowest debug security.

6.2.2. Validating TDO output data

Since the JTAG ports are only returning zeroes, the most clearly recognizable pattern is it's direct opposite: All ones. By shifting more ones in than the length of the scan chain, we can easily determine whether the JTAG port is not returning TDI data. The problem with this approach is that the length of the IR scan chain is unknown, and shifting one bit too few would result in a false-negative. The solution is to shift a very large amount of bits in order to be reasonably sure it's longer than the IR. A normal IR is typically between 3 to 5 bits long per TAP, but as TAPs can be chained the total could be significantly larger.

By leveraging JTAG-AP's communication protocol it is possible to shift a large amount of ones into all TDI ports at the same time. Reading TDO is limited to 4 bytes per read, and has to be done sequentially for each JTAG port. This is significantly slower than the input speed and is an unwelcome bottleneck on the test. The goal of this test is validating whether it's possible to derive the lenght of the IR chain which makes it's possible to answer the research question with just the last bit. Knowing this, we can formulate the testing methodology as follows:

1. Transition JTAG State machine to SHIFT-IR. The IR has a set length and never changes during operation.
2. Shift N amount of bits without capturing the data and with N significantly large.
3. Shift an M amount of bits while capturing data; with $M > 0$.
4. If the captured data contains at least a single 1, the port returns data

The maximum size of a TDI_TDO packet is 128 bits. Sending this packet 10000 times results in an 1280000 bit input. Assuming a long IR of 32 bits for each connected TAP results in a total of 40000 connected components. Each of these components would likely have an even longer BSC, leading to really long tests. Due to these factors it is extremely unlikely that the input is shorter than the IR when 1280000 is chosen as N.

While a single bit is enough for M, 8 bits were chosen for aesthetic reasons. 0xFF is more visible in the logs than 0x01 when compared to 0x00. As one byte is the smallest readable measure of data there are no significant performance downsides whatsoever to this choice.

6.2.2.1. Results

Snippet 6.1 is a summarized result of this test as produced at the end of the CoreSight Management Script. The summary displays the TDO output after one byte was shifted into the JTAG port. The full log has been made available in the project documentation.

Snippet 6.1: Summary of BSC responsiveness test

```
RESPONSES:
JTAG PORT0: bytearray(b'')
JTAG PORT1: bytearray(b'')
JTAG PORT2: bytearray(b'')
JTAG PORT3: bytearray(b'\x00\x00\x00\x00')
JTAG PORT4: bytearray(b'\x00')
JTAG PORT5: bytearray(b'\x00')
JTAG PORT6: bytearray(b'\x00')
NONZERO RESPONSES: 0
```

Because ports 0 to 3 exhibit behavior not in line with the test their initial CSW values have been extracted from the log and compiled into snippet 6.2.

Snippet 6.2: Summary of BCD responsiveness test

```
Port 0: JTAGState(EngineActive=True, OutstandingCommands=0, OutstandingResponse=0,
    PortConnected=True, SRST_Connected=True, TRST_OUT=False, SRST_OUT=False)
Port 1: JTAGState(EngineActive=True, OutstandingCommands=2, OutstandingResponse=0,
    PortConnected=True, SRST_Connected=True, TRST_OUT=False, SRST_OUT=False)
Port 2: JTAGState(EngineActive=True, OutstandingCommands=4, OutstandingResponse=0,
    PortConnected=True, SRST_Connected=True, TRST_OUT=False, SRST_OUT=False)
Port 3: JTAGState(EngineActive=False, OutstandingCommands=0, OutstandingResponse=4,
    PortConnected=False, SRST_Connected=False, TRST_OUT=False, SRST_OUT=False)
```

6.2.3. Discussion and Evaluation

Ports 3-6 respond with all zeroes and we can conclude that they do not return TDI Data. Ports 0 to 2 do not return any data whatsoever and that is remarkable. Judging by the incrementing OutstandingCommand value in snippet 6.2 it appears that the ports 0 to 2 are stalling. This is corroborated by the EngineActive value, which indicates that the JTAG Engine is running. Once port 3 is selected all commands are processed and return the combined bytes of ports 0 to 3. Further testing verified this behavior by writing data to the TDI of port 0 and then switching to port 5. The byte would not get processed until port 5 is selected, after which it returns all-zero TDO data immediately. The full version of this test is available in the project documentation.

This behavior may indicate that Ports 0 to 2 contain a JTAG TAP without a responsive clock. Ports 3 to 6 may not have an implemented JTAG TAP. Unless these issues are resolved, this makes further exploration of the BSC impossible.

Because the BSC cannot be explored further it is not possible to determine the length of JTAG-AP's IR. This leaves two possible conclusions:

1. CoreSight Management Script's implementation of JTAG-AP is incorrect
2. The internal JTAG BSCs are secured.

To prove correctness of the JTAG-AP implementation it would be ideal to test it on an unsecured device. Locating a device with an open JTAG-AP has not been successful, so this is not an option. During development of JTAG-AP a lot of testing has been done to validate the correctness of JTAG-AP. Some of these tests are available in the project documentation. JTAG-AP does consistently return the expected amount of bits, which is a significant indicator of correctness.

The JTAG scan chains pose a significant risk not just to the device's security but also to Samsung's IP. It is not unreasonable to assume they are locked even on development devices. Further research discovered that Samsung owns Patent US 2010/0153797 A1. This describes an authentication system for JTAG boundary scan. It is a system similar to the simple lock and key scheme introduced in Section 3.3.2 where JTAG is disconnected unless the appropriate password is entered. Crucially, it contains Figure 6.4 which shows both the Unknown JTAG detector and Secure JTAG.

Given these factors, it is appropriate to conclude that the JTAG Boundary scan chains are secured and are not a viable attack vector at this time. Most likely this is a feature of Secure JTAG that does not adhere to the status of the SOFT_LOCK fuse. Alternative attacks, on Secure JTAG or otherwise, could potentially disable JTAG BSC security in the future. Attacks over the JTAG BSC remain lucrative. As such, attackers should take note that an attack on Secure JTAG may unlock the JTAG BSC as a side-effect.

6.3. EM Fault Injection Attack

This section documents the implementation of the test methodology, as well as any achieved results. Performing both Electro Magnetic (EM) fault injection and side-channel analysis requires platform specific implementations, which are also discussed in this section.

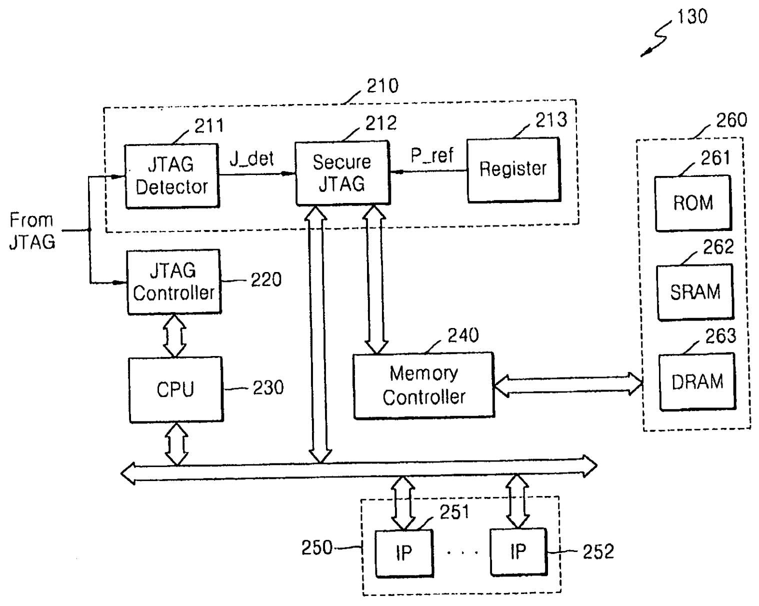


Figure 6.4: Excerpt from patent US20120060067A1 Showing both JTAG Detector and Secure JTAG [59]

6.3.1. Triggering on Secure JTAG authentication

For both side-channel analysis and fault injection it is crucial to have a consistent point in time to start measurement. That way it can be ensured an attack targets the same processes on each execution. This is typically not always exactly the same as various processes have some variance within the System on Chip (SoC).

Two separate trigger methods have been created, each with their own advantages and disadvantages.

6.3.1.1. JTAG-based triggering

The first triggering method is based on the external debug interface and targets the communication to JTAG-DP. When working on closed systems such as the SM-G920F platform it is hard to define a software based trigger. When the device no longer boots, such as when the RAM is removed, a software based trigger is impossible entirely. A hardware based trigger on one of the transmission lines of Secure JTAG (SJTAG) is also not an option because the entire debug system is embedded inside of the SoC.

That leaves the external JTAG-DP communication as accessible means. The JTAG-DP communication protocol is well defined in the ARM debug documentation. By analyzing the communication it is possible to know exactly when the final authentication password byte is sent into the DAP. This is the exact moment that SJTAG can validate the authentication.

To determine the appropriate trigger JTAG signals were tapped using an USbee DX logical analyzer at 25Msps. These signals are not human readable and had to be translated into a more convenient format.

The sigrok software-suite [55] is an open source library for signal analysis. It has two popular frontends: sigrok-cli and Pulseview. sigrok-cli is a commandline tool aimed at conversion and parsing. Pulseview is a GUI front-end which makes comparing the logic signals to the parsed results convenient for an end-user. In this project sigrok-cli proved to be very useful to create a good overview of JTAG communication, while Pulseview was mainly used to verify the parsing for correctness.

The sigrok tools come with a standard set of parsers. Importantly it has a parser for JTAG, which takes a comma separated file of recorded JTAG connection samples, and outputs the JTAG states and communication into a human-readable format. More importantly sigrok allows parsers to be stacked onto each other, so that, for instance, TDI and TDO data can be further analyzed. The STM32 parser is especially worth noting, as it parses JTAG debugging information for an older, but similar, debug interface as the Exynos 7420. Most likely this interface would be Arm Debug Interface (ADI)v4, while the Exynos 7420 uses ADIv5. Documentation on ADIv4 is hard to find, which makes understanding the parser not straightforward. By using the ADIv5 documentation and knowledge of the samples it was possible to derive the necessary changes and gain support for the interface. A second parser was written that outputs the total amount of JTAG Clock cycles for each UPDATE-IR. This makes determining the appropriate amount of clock cycles for a given input convenient.

The exact modifications are available in the project documentation.

Using this new parser the following method can be used to generate an appropriate trigger:

1. Write a testing script in the CoreSight Management Script that needs to be triggered on.
2. Attach the USbee DX logical analyzer to the JTAG-DP connections.
3. Trigger on the RST or TMS line while recording the trace. These lines shift prior to the actual JTAG-DP transaction and ensure a full capture. Output the result as a comma separated file.
4. Using sigrok-cli or pulseview with the new parsers allows the end-user to determine the appropriate point and read the corresponding amount of clock ticks.

By connecting the oscilloscope to the JTAG CLK line and setting it up to trigger only after the correct amount of clock cycles results in a highly accurate trigger. This trigger has been verified using the development board. Polling code that outputs on the UART after authentication completes showed a reaction within 15.2 μs of the trigger.

This method is universal and works for both platforms. The downside is that Secure JTAG authentication is relatively slow, the entire chain costing 500ms to completely execute. The trigger consistency is also compromised when authenticating several times. This can be resolved by restarting the device between each run. Because of these two reasons it is impossible to capture an EM trace containing SJTAG authentication in very short succession, complicating Side-channel analysis. Fault injection is not affected by these downsides.

6.3.1.2. UART-based triggering

The second trigger is a software based trigger. It was developed because side-channel analysis was limited by JTAG based triggering. This created a need to do SJTAG authentication on the device, rather than externally through the CoreSight Management Script.

Executing code on an Android based platform would cause a lot of extra noise due to various Android processes. Ideally, side-channel code should run with as little extra processes as possible. The bootloader is a single-threaded executable that is effectively a real-time OS. Taking over the bootloader would therefore be ideal to minimize other noise caused by other processes.

The development platform uses U-boot as the bootloader. U-boot sources were supplied with the Howchip HC7420-UFSeMM development board. Modification after board specific initialization allows a payload to be injected in order to, for instance, authenticate repeatedly with Secure JTAG. Printing `0xFF` causes the UART to output a brief single 5v power spike. This is the start bit used in serial communication. Because RS-232 uses inverted logic the printed `0xFF` is outputted as 0 volt. Printing prior to the critical moment results in a very clear and very consistent trigger.

It should be noted that the UART prints asynchronously to the processor. Care should as such be taken to create a payload with a long enough processing time to ensure the consistency of the trigger.

The primary advantage of this method is a double-edged sword: Code-execution allows a researcher to construct clear patterns in the EM trace, which can be used as markers to aid side-channel analysis. The accompanying downside is that code-execution causes the Atlas to emit significantly more EM radiation.

The biggest downside of the method is that it only works on the development board. This makes it useless for fault injection and it cannot be performed on the RAMless SM-G920F.

6.3.2. Locating Secure JTAG

Locating SJTAG is, first and foremost, the development of a method. Over the course of the project there have been a lot of iterations, so this section will only describe key methods. Full descriptions of all iterations are available in the project documentation.

6.3.2.1. Initial approach: JTAG-DP triggering

Initial approaches were done exclusively on with JTAG-DP based triggering as outlined in Section 6.3.1.1. Development of the method started on the RAMless SM-G920F platform described in Section 6.1.1. The SM-G920F was chosen in order to minimize the dampening effect the RAM. Traces showed a lot of random bursts of EM radiation along with high power usage of the platform, making them useless. As such, it was abandoned in favor of the Development Board. Without a boot medium the platform stays in BOOTROM with virtually no power usage. This indicated absolutely minimal activity which outweighed the downsides of the stacked RAM.

Early testing revolved around making scans of the full chip and analyzing the spectrum for differences. The idea behind this method was that the hashing engine should radiate at certain frequency band. By comparing traces where the hashing engine was active against traces where it was not there should have been a clear difference. This turned out not to be the case, not even when reducing the traces to a mere 20μs.

With Spectrum analysis not leading to results, the better solution is to manually go through each trace and analyze frequency bands. This is a labor intensive process, so performing it on the entirety of the SoC is not an option. Instead, by making educated guesses to the location of SJTAG it's possible to significantly reduce the attack space.

The first of these educated guesses was a component referred to as Slim_SSS. It can be found next to the Security SubSystem in Figure 6.1a, but is not described in the official documentation [35]. The theory is that, because SJTAG is a controller with some cryptographic hardware, it may have been mistaken for a smaller Security SubSystem. The Security SubSystem is after all a controller with some cryptographic hardware itself.

During the project both X-ray photographs and images of the die were created. By combining these with the official measurements of the markings found in the documentation [35] allowed for the creation of an accurate model. This model, as seen in figure 6.5 shows the required alignment of the EM in order to target the Slim_SSS.

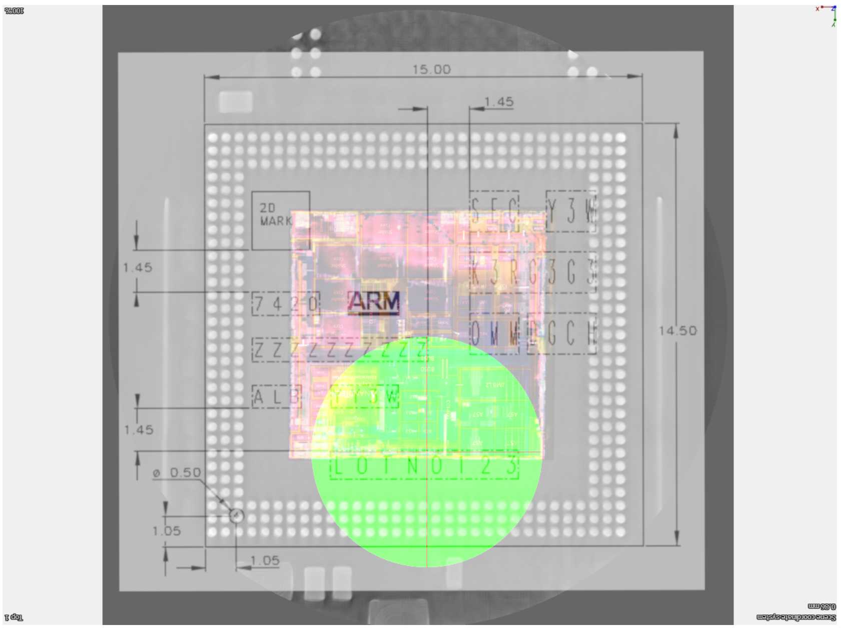


Figure 6.5: Model of the Exynos7420 aligning the die with the external markings. EM probe, marked in Green, aimed at the Slim_SSS

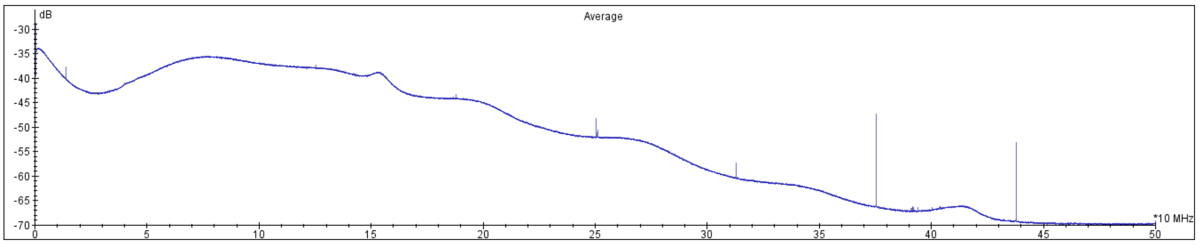
Prior to the creation of this model meant targeting the Slim_SSS was not very accurate. Figure 6.6 shows that this model led to an immediate improvement for aiming the EM probe. Closer analysis showed a repetitive pattern centered around 160 Mhz. However, this pattern stayed present even when ensuring the hashing engine could not be executing.

Further research determined that the SJTAG clock is derived from Atlas. Coresight, closely tied to the debug system in general, shares this same behavior but is also in the same power domain as Atlas. Given these facts it is likely that SJTAG is located near Atlas. Analyzing the image of the die does reveal some extra, unknown, components near Atlas. While most have a counterpart near Apollo, the two details depicted in Figure 6.7 stand out. As CoreSight and SJTAG are components that do not exist on Apollo these are very good candidates.

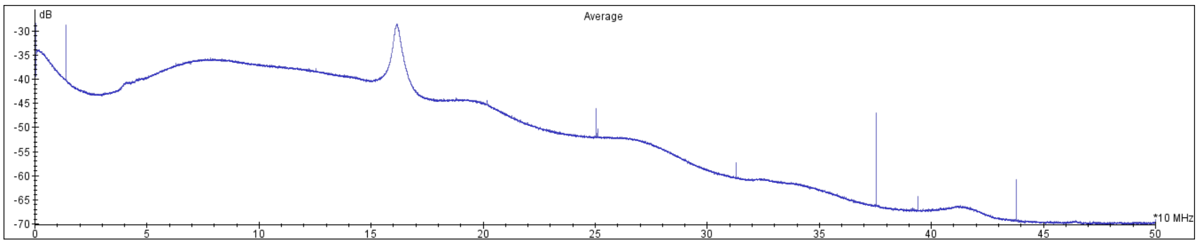
Measuring on this location did not provide a recurring signal however, nor did it show any correlation over several frequency bands. Another approach may be required.

6.3.2.2. Second approach: UART based triggering

As described in Section 6.3.1.2, the development platform allows for code execution. Code-execution can be used to rapidly trigger SJTAG authentication so that the signal may stand out more. Additionally, processor heavy operations can be repeated in order to create a recognizable block wave. This can then be used for



(a) Averaged spectrum of traces prior to creation of the model.



(b) Averaged spectrum of traces using the model to aim the EM probe.

Figure 6.6: Comparison of the spectrum of two separate trace sets. Higher means more signal at that particular location.

alignment, enabling the averaging of traces. Using averages of aligned traces should make recurring signals, such as SJTAG authentication, more visible.

Snippet 6.3 is the payload added to U-Boot. It loops through the following steps:

1. Multiplicate 500 times
2. Execute SJTAG authentication 10 times:
    (a) Enter password bytes 8 times, triggering the hashing engine
    (b) Poll the SJTAG status register to confirm completion of authentication
    (c) reset SJTAG
3. Multiplicate 100 times
4. Idle for 500ns

Multiplication is a computation intensive operation, and should be clearly visible on EM traces. Computation uses a dedicated component in each core, which should be distinct from SJTAG.

A scan over the entire surface of Atlas did not immediately show any distinct signals. Applying a low-pass filter to all traces revealed block-like behaviour on one of the traces. This trace originated from the position outlined in Figure 6.8. This is one of Atlas's cores, which contains a dedicated Arithmetic Logic Unit (ALU) for computation. Applying frequency band composition reveals a clear block wave at several bands. Figure 6.9 shows the 85.94 Mhz frequency band containing one such waveform. It has been annotated to show the clear difference between the multiplications and SJTAG.

Using filter guidance all correlating traces can be grouped, which results in two groups. One group similar to trace 6.9, and one it's inverse. By applying a strong low-pass filter this creates a very clean line that can be aligned on. These alignments can then be applied to the original traces and averaged. This resulted in a new correlation between two traces, but these did not appear to resemble SJTAG communication.

Measuring above one of the cores is useful to create a testing methodology, but less so for picking up a SJTAG signal. As mentioned in Section 6.3.2.1, the most probable location for SJTAG is near the components found in Figure 6.7. As such, the EM probe was positioned as shown in Figure 6.10 and new traces were made.

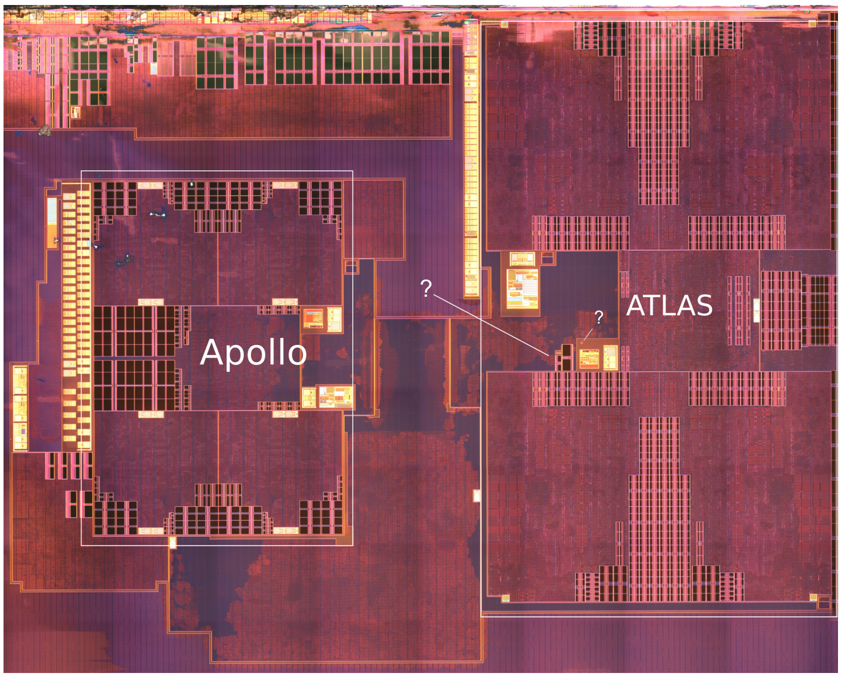


Figure 6.7: Interesting components near Atlas

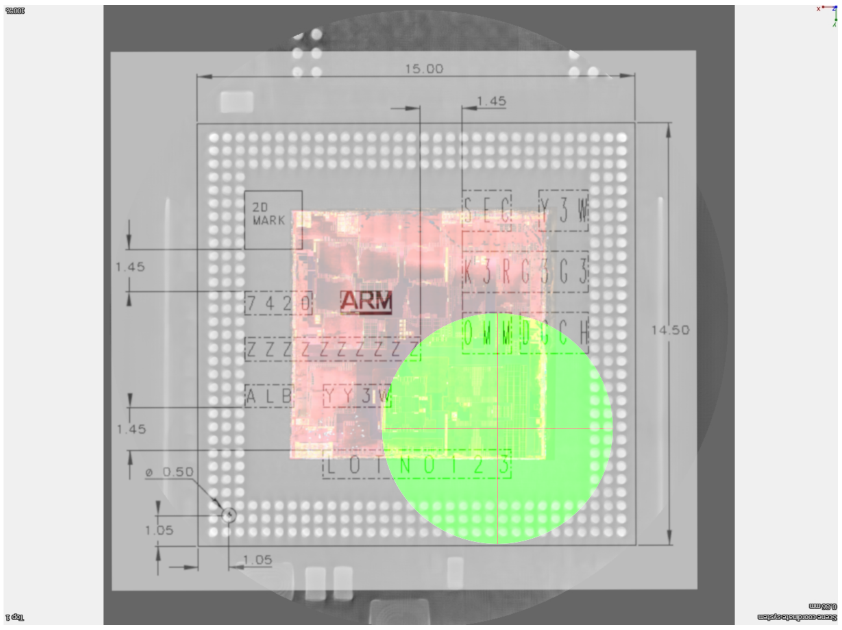


Figure 6.8: Origin in block signal for UART Trigger-based trace

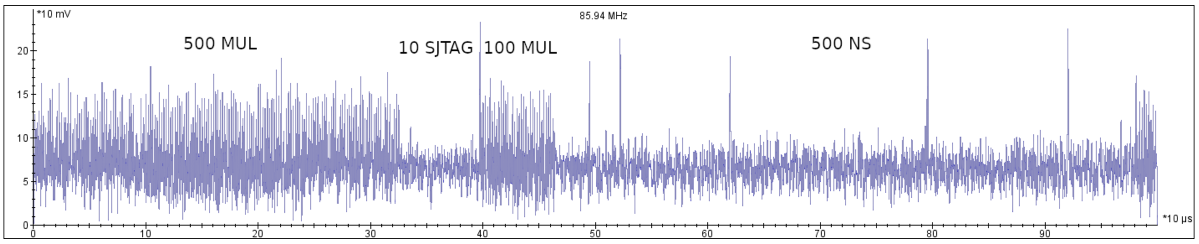


Figure 6.9: Annotated block wave that clearly shows the various stages of the code

Snippet 6.3: U-boot payload executing on development device

```
while(1){
   printf("\xFF");
   volatile int i;
   volatile int mult = 5;
      for(i=0; i<500; i++){
      mult = mult*i; //500 Multiplications to create a wide block marking the start of SJTAG
          authentication
      }
   int j;
   for(j=0;j<10;j++){ //10 times: Authenticate, poll, reset
      for(i=0; i<8;i++){
         writel(0xcafebabe,0x16008010); //Authenticate with SJTAG
      }
      int loop = 1;
      while(loop){
         int result = readl(0x16008004)&2; //Poll for SJTAG_DONE
         if(result){
            writel(0x00000001,0x1600800C); //Detatch / Reset SJTAG
            loop = 0;
            }
         }
      }
   for(i=0; i<100; i++){
   mult = mult*i; //100 multiplications to designate end marker of SJTAG
   }
   udelay(500); //Do nothing for 500ns as end-marker for the sample
}
```

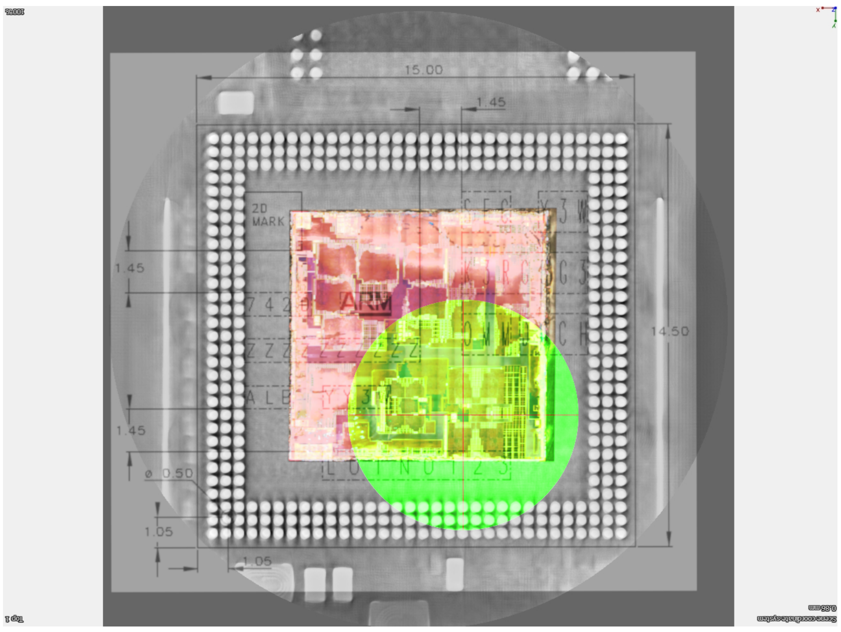


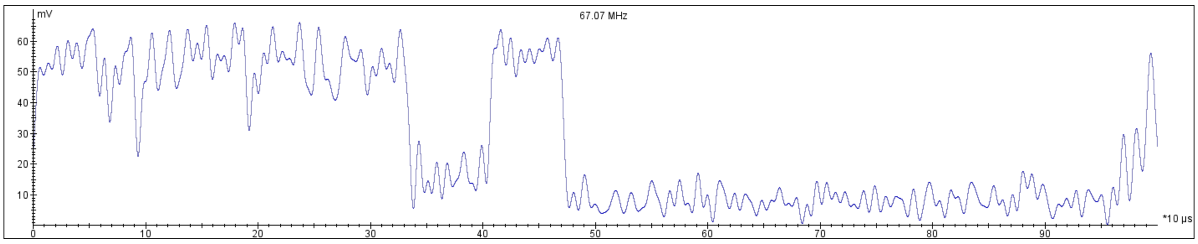
Figure 6.10: Optimal location for detecting SJTAG signals

6.3.2.3. Results

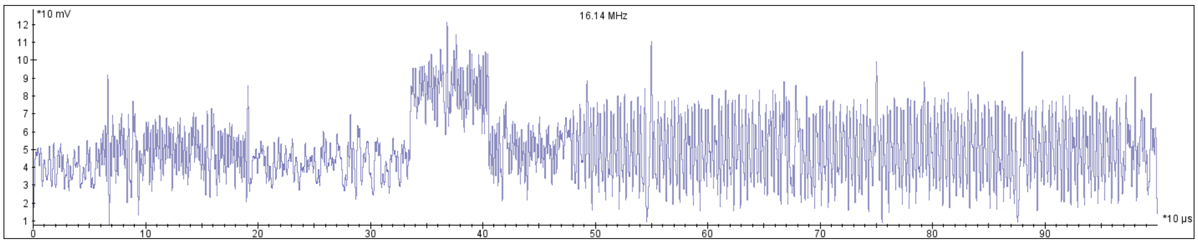
Filter guidance did not discover any meaningful new groups. However, brief manual analysis revealed the waveform in Figure 6.11b. Where previous waveforms showed high correlation between SJTAG and the delay, this one was remarkably distinct. Closer examination revealed similar slight differences between SJTAG and the delay period over several spectra.

For verify correlation with SJTAG, the code in Snippet 6.3 was modified to only write the password byte 7 times. This resulted in two key changes: Firstly, the SJTAG authentication block in Figure 6.11c appeared to be slightly shorter than it's 8 byte counterpart in Figure 6.11a. Indeed, performing 10 operations less naturally shortens this block. More importantly, the block wave in Figure 6.11b completely disappeared in 6.11d. Note that the signal generated by the delay function is present in both frequency bands.

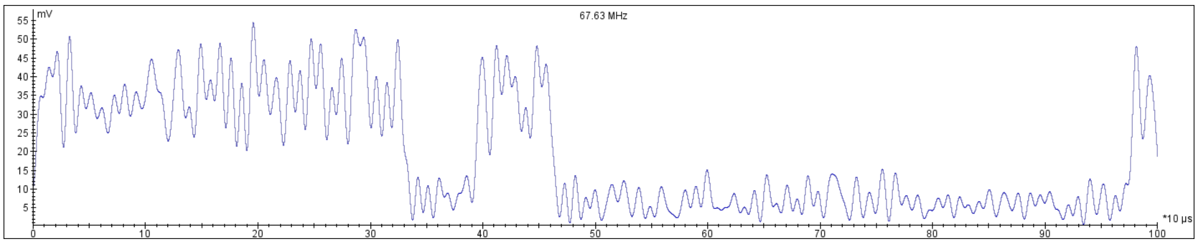
Note that Figure 6.11b is centered around 16.14 Mhz while 6.11d is centered around 16.48 Mhz. This happens because the process of sampling down to 2.2 Ghz is an approximation. This is a necessary step in the process in order to stop the software from processing the noise located above 1.1ghz. Because of the difference in frequency bands it's possible that the SJTAG authentication is not in this particular band. However, decreasing the size of the bands to 250mhz on the trace where 8 bytes were entered results in Figure 6.12. This shows that SJTAG authentication is tightly centered around 16.14Mhz. It does not show up in the neighboring bands whatsoever. Likewise, the signal does not show up in any of the nearby bands when 7 password bytes are entered. This proves that the discovered signal correlates with the SJTAG hashing engine.



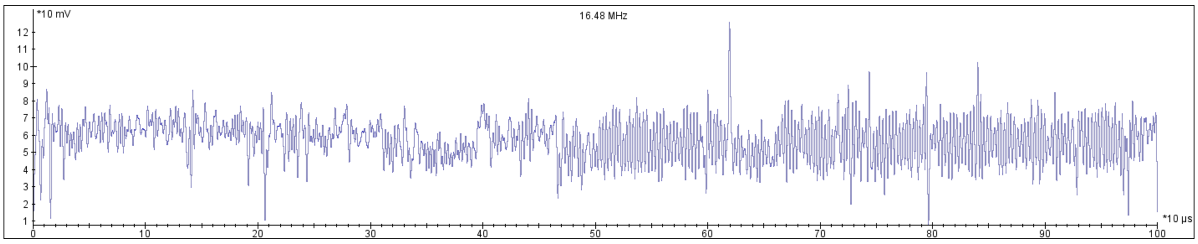
(a) Signal correlating with the ALU when 8 passwords are entered



(b) Trace of the 1Mhz frequency band centered around 16.14 Mhz band when supplying 8 password bytes



(c) Signal correlating with the ALU when 7 passwords are entered



(d) Trace of the 1Mhz frequency band centered around 16.48 Mhz band when supplying 7 password bytes

Figure 6.11: Comparison of the signals located discovered around components near Atlas

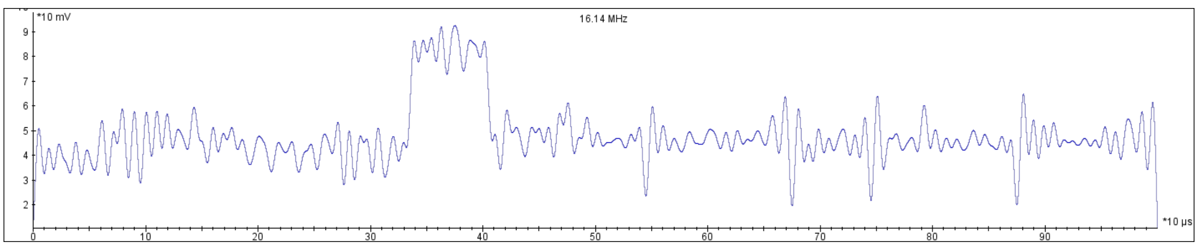


Figure 6.12: 250 Mhz band centered around 16.14 Mhz, showing only SJTAG authentication.

For verification, the code in 6.3 was modified to only write the password byte 7 times. This resulted in two key changes: Firstly, the SJTAG authentication block appeared to be slightly shorter than it's 8 byte counterpart. Indeed, performing 10 operations less naturally shortens this block. More importantly, the block wave at 16.14 MHz completely disappeared. This proves that the discovered signal correlates with the SJTAG hashing engine.

6.3.3. Discussion and Evaluation

Several methodologies were researched but not all of them were successful. This section aims to evaluate each methodology, and finally discusses the results.

- JTAG-DP based triggering
  - Full scan of Chip with spectrum analysis:

    The SJTAG hashing engine is most likely too fast and too quiet to show up on a spectrum. These spectra are effectively averages over the entire trace for a given frequency, which would drown out smaller signals.
  - Measuring EM radiation on Slim_SSS:

    The pattern found during this test did not correlate with SJTAG's hashing engine. Furthermore, despite the Slim_SSS not being described in the documentation [35], closer inspection revealed references to it. Critically, the Slim_SSS is not in Atlas's power domain, which SJTAG is known to be. As such, Slim_SSS is not the appropriate location to search.
  - Measuring EM radiation near Atlas:

    There are three major possibilities why this test did not result in correlating signals: Firstly, this may still be the wrong location. However, given the knowledge of the SJTAG clock domains it should be nearby enough. More likely is that the hashing engine moves in respect to the trigger. Traces can be aligned to negate this, but this requires a consistent waveform to align on. Lastly, it's still possible the EM signals do not penetrate the RAM.
- UART based triggering
  - Measuring above one of the cores:

    Measurement location is on top of one of the Atlas cores. While ideal for the block wave, there is a good chance this interferes with SJTAG signals. For best results this should be reproduced on the appropriate location near Atlas as outlined in Figure 6.7.
  - Measuring EM radiation on the components near Atlas:

    Scanning on theorized location revealed, for the very first time, a signal that correlates with the SJTAG hashing engine. As such, the location outlined in figure 6.10 is the approximate location for SJTAG. However, the testing methodology did not reveal this signal. This is potentially because the SJTAG hashing engine shows strong correlation with the ALU. Removing multiplication from the executed code may allow better discernment between SJTAG and the delay function.

Testing in Section 6.3.1.1 showed that SJTAG authentication finishes within 15 μs. The results of Side-channel analysis have revealed the location of the SJTAG hashing engine to be near the area shown in Figure 6.10. Additional investigation may be performed to increase the precision of the timing and location of the SJTAG hashing engine. However, EM fault injection is not an exact process, and appropriate parameters will need to be experimented with. As such, these results adequately lower the total attack space and provide an excellent starting point for fault injection.

7 Conclusion

This chapter provides a summary of the research and achievements in this thesis in Section 7.1. It also provides potential future work in Section 7.2

7.1. Summary

Chapter 1 of this thesis explained the transition of digital forensics from recovery to exploitation. This transition is primarily caused by introduction of standard cryptography. It details how access to the debug system could be a tremendous tool in defeating such cryptographic techniques.

Chapter 2 provided a classification of attacks in the hardware attack space and described existing attacks in the digital forensic space in its context. The classification was made from the perspective of exploitation, and differentiates between attacks on data, functionality and Intellectual Property (IP).

Chapter 3 discussed important Boundary Scan Chain (BSC) standards such as JTAG and its extensions. It also introduced a novel classification of JTAG security measures, with example implementations of each class.

Chapter 4 detailed Samsung's implementation of JTAG security: Secure JTAG (SJTAG). The chapter also discusses various related components such as the CoreSight debug system in order to provide the necessary context, and it showed potential attack vectors that the device is currently protected from by SJTAG.

Chapter 5 discussed the applicability of the hardware attacks described in Chapter 2 on SJTAG. It also introduced a novel model that allows an attacker to grade the difficulty of attacks. This model was used to evaluate the aforementioned attacks and decide the optimal course. The chapter also detailed testing methodology for the two chosen attack vectors.

Chapter 6 provides a description of the target platforms. This includes a description of one of the developed tools for the project: CoreSight Management Script. It details the implementation, evaluation and execution of both attack methodologies.

7.2. Future Work

This section gives several recommendations for future work:

- To further lower the attack space:
    1. The 15µs attack space was discovered through polling, which introduces an unknown amount of overhead. Since the appropriate location and frequency of the signal is known since discovering the Secure JTAG (SJTAG) signal through the use of the UART based trigger. This information can be used to locate the signal with a JTAG-DP based trigger. Locating the signal respective to the JTAG-DP based trigger will give greater accuracy for the attacks.
    2. By measuring the strength of the 16.14ghz signal over surrounding locations it is possible to determine the location with the clearest signal. This should be the exact location of the hashing engine. A Laser Induced Voltage Alteration (LIVA) scan such as described in Section 2.4.3 may show the exact location of SJTAG. The merit of this level of accuracy for Fault injection is highly debatable, but it may be good preparation for an eventual Focused Ion Beam (FIB) based attack.

- To circumvent SJTAG security:
    1. The results obtained in this thesis provide an excellent basis for a Electro Magnetic (EM) fault injection attack. As such, the primary recommendation is to proceed with fault injection.
    2. The approximate location of SJTAG should also provide a good basis for FIB research. However, this remains an incredibly complicated attack.
- In general:
    1. As mentioned in Sections 1.1 and 1.2, research into JTAG security is very one sided. While there is a big focus on securing JTAG, there is a pattern of keeping the benefits of JTAG available for manufacturers. At the same time, there is very little public research into ensuring that these methods are actually secure. This is a recipe for disaster and makes it a very interesting field of research. Given the likelihood of vulnerability and the power of attacks over the debug system, it will be very worthwhile to research other implementations.

A
Attack Evaluation Tables

This appendix contains the evaluations of the attacks discussed in Section 5.1 based on the model designed in Section 5.2.1.

Table A.1: Attack Evaluation of JTAG Boundary Scan Reversing

Classes	**Attack: JTAG BSC**	
1. Knowhow	Missing knowledge or outstanding assumptions require little time investment to resolve	1,00
2. Effort	Some research or setup time	1,00
3. Specialized Tools	Tools cheap to acquire and don't need a lot of specialistic knowledge	1,00
4. Time Requirements	Executing an attack takes a few hours	1,00
5. Reproducibility between Devices	Reproducing the attack on another device requires roughly equal amount of research effort as the initial attack	1,00
6. Reproducibility on exact Same Device	Reproducing the attack on another device requires roughly equal amount of research effort as the initial attack	1,00
7. Determinism	Properly executed attack will always yield results	0,00
8. Reusability	Results are per device and do not carry over	0,00
9. Device Impact	Attack results in total loss of security of entire device	-5,00
Social Impact		1,33
10. Personal Impact	Considerable impact expected in areas such as personal privacy, security or freedom	3,00
11. Economic Impact	Slight economic impact expected; Economic losses as a reaction but not a targeted attack	1,00
12. Environmental Impact	No environmental Impact Expected	0,00
13. Destructiveness	Low risk to the device; Opening the casing, soldering on the board etc.	2,00
Total Cost		1,84

Table A.2: Attack Evaluation of non-invasive fault injection

Classes	**Attack: Non-invasive fault Injection**	
1. Knowhow	Missing knowledge and outstanding assumptions require considerable time investment to resolve	3,00
2. Effort	Considerable research or setup time	3,00
3. Specialized Tools	Tools steep to acquire privately but doable in most labs; requires specialistic knowledge to operate	5,00
4. Time Requirements	Executing an attack takes up to a few days	2,00
5. Reproducibility between Devices	Reproducing the attack requires moderate effort or time	0,80
6. Reproducibility on exact Same Device	Reproducing the attack requires moderate effort or time	0,80
7. Determinism	Properly executed attack will always yield results	0,00
8. Reusability	Results are per device and do not carry over	0,00
9. Device Impact	Attack results in total loss of security of entire device	-5,00
Social Impact		1,33
10. Personal Impact	Considerable impact expected in areas such as personal privacy, security or freedom	3,00
11. Economic Impact	Slight economic impact expected; Economic losses as a reaction but not a targeted attack	1,00
12. Environmental Impact	No environmental Impact Expected	0,00
13. Destructiveness	Low risk to the device; Opening the casing, soldering on the board etc.	2,00
Total Cost		2,66

Table A.3: Attack Evaluation of non-invasive fault injection

Classes	Weights	**Attack: Invasive fault Injection**	
1. Knowhow		Missing knowledge or outstanding assumptions require sigificant time investment to resolve	6,00
2. Effort		Considerable research or setup time	3,00
3. Specialized Tools		Tools steep to acquire privately but doable in most labs; requires specialistic knowledge to operate	5,00
4. Time Requirements		Executing an attack takes up to a few days	2,00
5. Reproducibility between Devices		Reproducing the attack costs a significant amount of time	1,60
6. Reproducibility on exact Same Device		Reproducing the attack requires moderate effort or time	0,80
7. Determinism		Properly executed attack will always yield results	0,00
8. Reusability		Results are per device and do not carry over	0,00
9. Device Impact		Attack results in total loss of security of entire device	-5,00
Social Impact			1,33
10. Personal Impact		Considerable impact expected in areas such as personal privacy, security or freedom	3,00
11. Economic Impact		Slight economic impact expected; Economic losses as a reaction but not a targeted attack	1,00
12. Environmental Impact		No environmental Impact Expected	0,00
13. Destructiveness		HIGH risk to the device; Any step may destroy device (decapsulation die modifications)	10,00
Total Cost			3,78

Table A.4: Attack evaluation for FIB modifications

Classes	Weights	Attack: FIB Modifications	
1. Knowhow		Missing knowledge or outstanding assumptions require extreme time investment to resolve	10,00
2. Effort		Extreme research or setup time	10,00
3. Specialized Tools		Tool costs so expensive only high-end specialistic labs may be able to afford it. Requires significant time-investment by experts to operate	10,00
4. Time Requirements		Executing an attack takes up to a few days	2,00
5. Reproducibility between Devices		Reproducing the attack on another device requires roughly equal amount of research effort as the initial attack	2,00
6. Reproducibility on exact Same Device		Attack does not need to be reproduced to achieve results	0,00
7. Determinism		Properly executed attack will always yield results	0,00
8. Reusability		Results are per device and do not carry over	0,00
9. Device Impact		Attack results in total loss of security of entire device	-5,00
Social Impact			1,00
10. Personal Impact		Considerable impact expected in areas such as personal privacy, security or freedom	3,00
11. Economic Impact		No economic impact expected	0,00
12. Environmental Impact		No environmental Impact Expected	0,00
13. Destructiveness		HIGH risk to the device; Any step may destroy device (decapsulation die modifications)	10,00
Total Cost			5,24

Table A.5: Attack evaluation of attacks on SHA2

Classes	Weights	**Attack: SHA2 Attacks**	
1. Knowhow		Missing knowledge and outstanding assumptions require considerable time investment to resolve	3,00
2. Effort		Some research or setup time	1,00
3. Specialized Tools		No amount of money or resources will be enough. Without some significant breakthrough the required equipment is out of reach for anyone	525,00
4. Time Requirements		Heat death of the universe will occur before this attack is completed. Without significant advances this attack is not feasible	525,00
5. Reproducibility between Devices		Reproducing the attack on another device requires roughly equal amount of research effort as the initial attack	525,00
6. Reproducibility on exact Same Device		Reproducing the attack on another device requires roughly equal amount of research effort as the initial attack	525,00
7. Determinism		Properly executed attack will always yield results	0,00
8. Reusability		Results are per device and do not carry over	0,00
9. Device Impact		Attack results in total loss of security of entire device	-5,00
Social Impact			1,33
10. Personal Impact		no impact expected in areas such as personal privacy, security or freedom	0,00
11. Economic Impact		Considerable economic impact expected; Targeted Attack designed to cause economic losses	3,00
12. Environmental Impact		Slight environmental impact expected	1,00
13. Destructiveness		Attack does not put the device at risk	0,00
Total Cost			201,46

Bibliography

[1] Lilas Alrahis, Muhammad Yasin, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. Scansat: Unlocking obfuscated scan chains, 2019. URL `http://doi.acm.org/10.1145/3287624.3287693`.

[2] Boundary-scan Architecture. *IEEE Standard for Test Access Port and Boundary-Scan Architecture - Redline*, volume 2013. 2013. ISBN VO -.

[3] ARM. Arm ds-5. URL `https://developer.arm.com/tools-and-software/embedded/legacy-tools/ds-5-development-studio`.

[4] Rafal Baranowski, Michael A. Kochte, and Hans Joachim Wunderlich. Securing access to reconfigurable scan networks. *Proceedings of the Asian Test Symposium*, (November):295–300, 2013. ISSN 10817735. doi: 10.1109/ATS.2013.61.

[5] John T Bateson. *In-circuit testing*. Springer Science & Business Media, 2012.

[6] Harm Van Beek. Science & Justice A forensic visual aid : Traces versus knowledge. *Science & Justice*, 58(6):425–432, 2018. ISSN 1355-0306. doi: 10.1016/j.scijus.2018.08.006. URL `https://doi.org/10.1016/j.scijus.2018.08.006`.

[7] Swarup Bhunia, Michael S Hsiao, Mainak Banga, and Seetharam Narasimhan. Hardware Trojan Attacks : Threat Analysis and Countermeasures. *Proceedings of the IEEE*, 102:1229–1247, 2014. doi: 10.1109/JPROC.2014.2334493.

[8] M. F. Breeuwsma. Forensic imaging of embedded systems using JTAG (boundary-scan). *Digital Investigation*, 3(1):32–42, 2006. ISSN 17422876. doi: 10.1016/j.diin.2006.01.003.

[9] Marcel Breeuwsma and Martien De Jongh. Forensic data recovery from flash memory. *Small Scale Digital ...*, 1(1):1–17, 2007. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.5697{&}rep=rep1{&}type=pdf{%}5Cnhttp://www.ssddfj.org/papers/SSDDFJ{_}V1{_}1{_}Breeuwsma{_}et{_}al.pdf`.

[10] Marcel Breeuwsma and Martien De Jongh. Forensic data recovery from flash memory. *Small Scale Digital ...*, 1(1):1–17, 2007. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.5697{&}rep=rep1{&}type=pdf{%}5Cnhttp://www.ssddfj.org/papers/SSDDFJ{_}V1{_}1{_}Breeuwsma{_}et{_}al.pdf`.

[11] Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6879 LNCS:355–371, 2011. ISSN 03029743. doi: 10.1007/978-3-642-23822-2_20.

[12] Ronald F. Buskey and Barbara B. Frosik. Protected JTAG. *Proceedings of the International Conference on Parallel Processing Workshops*, pages 405–412, 2006. ISSN 15302016. doi: 10.1109/ICPPW.2006.65.

[13] Katha Chanda. Password security: an analysis of password strengths and vulnerabilities. *International Journal of Computer Network and Information Security*, 8(7):23, 2016.

[14] C. J. Clark. Anti-tamper JTAG TAP design enables DRM to JTAG registers and P1687 on-chip instruments. *Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2010*, pages 19–24, 2010. doi: 10.1109/HST.2010.5513119.

[15] International Electrotechnical Commission. General requirements for the competence of testing and calibration laboratories. *Order A Journal On The Theory Of Ordered Sets And Its Applications*, 2006:2005–2007, 2006.

[16] Ang Cui and Rick Housley. BADFET : Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection.

[17] Amitabh Das, Jean Da Rolt, Santosh Ghosh, Stefaan Seys, Sophie Dupuis, Giorgio Di Natale, Marie Lise Flottes, Bruno Rouzeyre, and Ingrid Verbauwhede. Secure JTAG (SJTAG) implementation using schnorr protocol. In *Journal of Electronic Testing: Theory and Applications (JETTA)*, volume 29, pages 193–209, 2013. ISBN 1083601353699. doi: 10.1007/s10836-013-5369-9.

[18] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Marcel Winandy. Privilege escalation attacks on android. In *international conference on Information security*, pages 346–360. Springer, 2010.

[19] E. De Mulder, S. B. Örs, B. Preneel, and I. Verbauwhede. Differential electromagnetic attack on an fpga implementation of elliptic curve cryptosystems. *2006 World Automation Congress, WAC'06*, 0(0):1–7, 2007. doi: 10.1109/WAC.2006.375739.

[20] Andrei Frumusanu. The samsung exynos 7420 deep dive - inside a modern 14nm soc. URL `https://www.anandtech.com/show/9330/exynos-7420-deep-dive/2`.

[21] Google Inc. Compatibility Definition Android 6.0. pages 1–62, 2015. URL `http://www.businessdictionary.com/definition/compatibility.html`.

[22] CVSS Special Interest Group. Common vulnerability scoring system v3.0: Specification document. URL `https://www.first.org/cvss/specification-document`.

[23] Ujjwal Guin and Daniel Dimase. Counterfeit Integrated Circuits : Detection , Avoidance , and the Challenges Ahead. pages 9–23, 2014. doi: 10.1007/s10836-013-5430-8.

[24] HOWCHIP.com. Howchip hc7420-ufsemm. URL `http://www.howchip.com/products/exsom7420/HC7420_UFSeMM.php`.

[25] CHOI Hyun-Min and Shigenobu Maeda. E-fuse structure of semiconductor device, jun 2016. URL `https://patents.google.com/patent/US9368445B2`.

[26] IEEE Computer Society. Test Technology Standards Committee., Institute of Electrical and Electronics Engineers., and IEEE-SA Standards Board. *IEEE standard for access and control of instrumentation embedded within a semiconductor device - 1687*. 2014. ISBN 9780738194165. doi: 10.1109/IEEESTD.2014.6974961. URL `http://ieeexplore.ieee.org/document/6974961/`.

[27] Parralax inc. JTAGulator Product Brief, 2013. URL `https://www.parallax.com/sites/default/files/downloads/32115-JTAGulator-Product-Brief-1.1.pdf`.

[28] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. pages 129–143, 2003. ISSN 03029743. doi: 10.1007/3-540-36400-5_11. URL `http://link.springer.com/10.1007/3-540-36400-5{_}11`.

[29] Senwen Kan, Jennifer Dworak, and James George Dunham. Echeloned IJTAG data protection. *Proceedings of the 2016 IEEE Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2016*, 2017. doi: 10.1109/AsianHOST.2016.7835558.

[30] Yogesh Kumar, Rajiv Munjal, and Harsh Sharma. Comparison of symmetric and asymmetric cryptography with existing vulnerabilities and countermeasures. *International Journal of Computer Science and Management Studies*, 11(03), 2011.

[31] ARM limited. ARM ® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2. 7013(c):1–48, 2013.

[32] Arm Ltd. ARM Architecture Reference Manual (ARMv7-A and ARMv7-R edition). pages 1–1138, 2007. ISSN 09537112. doi: 10.1016/j.cacc.2004.08.014.

[33] Arm Ltd. CoreSight Components - Technical Reference Manual. 2009.

[34] Arm Ltd. ARM Architecture Reference Manual (ARMv8, for ARMv8-A architecture profile). 180(2):566–571, 2017. ISSN 10902104. doi: 10.1016/S0006-291X(05)81102-7.

[35] Samsung Electronics Co. Ltd. *Exynos 7420 User's Manual.*

[36] E J Marinissen and Y Zorian. IEEE Std 1500 Enables Modular SoC Testing. *IEEE Design & Test of Computers*, 26(1):8–17, 2009. ISSN 0740-7475 VO - 26. doi: 10.1109/MDT.2009.12.

[37] Rodney McKemmish. When is digital evidence forensically sound? In *IFIP international conference on digital forensics*, pages 3–15. Springer, 2008.

[38] Mentor. Tessent missionmode: New, runtime dft technology paves way for self-correcting automotive electronics. `https://www.mentor.com/products/silicon-yield/resources/overview/-078ab35f-c862-475c-8115-da33e3e4bffb`. Online; accessed 19 June 2018.

[39] Microsoft. Improving web application security: Threats and countermeasures - chapter 3 - threat modeling. URL `https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648644(v=pandp.10)`.

[40] N. Moro, K. Heydemann, E. Encrenaz, and B. Robisson. Formal verification of a software countermeasure against instruction skip attacks. *Journal of Cryptographic Engineering*, 4(3):145–156, 2014. ISSN 21908516. doi: 10.1007/s13389-014-0077-7.

[41] Zhenyu Ning and Fengwei Zhang. Understanding the Security of ARM Debugging Features. *S&P*, 2019.

[42] Franc Novak and Anton Biasizzo. Security extension for IEEE Std 1149.1. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 22(3):301–303, 2006. ISSN 09238174. doi: 10.1007/s10836-006-7720-x.

[43] OpenOCD. Open on-chip debugger. URL `http://openocd.org/`.

[44] Elisabeth Oswald. Enhancing simple power-analysis attacks on elliptic curve cryptosystems. *Ches*, pages 82–97, 2003. ISSN 03029743. doi: 10.1007/3-540-36400-5_8. URL `http://link.springer.com/chapter/10.1007/3-540-36400-5{_}8`.

[45] Keunyoung Park, Sang Guun Yoo, Taejun Kim, and Juho Kim. JTAG security system based on credentials. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 26(5):549–557, 2010. ISSN 09238174. doi: 10.1007/s10836-010-5170-y.

[46] Luke Pierce and Spyros Tragoudas. Multi-level SJTAG architecture. In *Proceedings of the 2011 IEEE 17th International On-Line Testing Symposium, IOLTS 2011*, pages 208–209, 2011. ISBN 9781457710551. doi: 10.1109/IOLTS.2011.5993845.

[47] Android Open Source Project. Encryption. URL `https://source.android.com/security/encryption,`.

[48] Xuanle Ren and R D Shawn Blanton. Detection of IJTAG Attacks Using LDPC-based Feature Reduction and Machine Learning. (Norte 2020), 2018.

[49] Xuanle Ren, Grade Tavares, and R D Shawn Blanton. Detection of Illegitimate Access to JTAG via Statistical Learning in Chip. *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 109–114, 2015. ISSN 15301591.

[50] AP Saleel, Mohamed Nazeer, and Babak D Beheshti. Linux kernel os local root exploit. In *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–5. IEEE, 2017.

[51] Samsung. Smartphone showcase, . URL `https://www.samsung.com/semiconductor/minisite/exynos/showcase/smartphone/`.

[52] Samsung. Samsung galaxy s6, . URL `url`.

[53] Samsung. Artik documentation: Jtag port lock, . URL `https://developer.artik.io/documentation/advanced-concepts/secure-os/jtag-lock.html`.

[54] David E Sanger and Brian X Chen. Signaling post-snowden era, new iphone locks out nsa. *New York Times*, 26, 2014.

[55] sigrok. sigrok. URL `https://sigrok.org/`.

[56] Sergei P. Skorobogatov. Semi-invasive attacks-a new approach to hardware security analysis. *Technical report, University of Cambridge, Computer Laboratory*, (630):144, 2005. ISSN 1476-2986. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.228.2204{&}rep=rep1{&}type=pdf`.

[57] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. *International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002)*, 2523:2–12, 2003. ISSN 03029743. doi: 10.1007/3-540-36400-5_2.

[58] Test Technology and Standards Committee. *IEEE Standard for Memory Modeling in Core Test Language.* 2014. ISBN VO -. doi: 10.1109/IEEESTD.2014.6832422.

[59] Yun-Ho Youm, Mi-Jung Noh, Hong-Mook Choi, FENG Xingguang, Easyg Llc Evans, T.C., Gavrilovich, E., Mihai, R.C. and Isbasescu, I., Darryl Thelen, J A Martin, S M Allen, and Slane SA. Apparatus and method of authenticating joint test action group (JTAG). 002(15):354, nov 2011. ISSN 13871811. doi: 10.1037/t24245-000.

[60] Timothy Zadigian, Jonathan Stroud, Michael Moriarty, P Morin, T H Applebaum, R Bowman, Y Zhao, Compact Multilingual, Word Recog, P Morin, B A Hanson, T H Applebaum, Primary Examiner, and Vijay B Chawan. JTAG-based programming and debug. 1(12), oct 2014.