

Exploiting Ring Oscillators to Leak Secret Information

FANNY LUNDBERG

MASTER'S THESIS

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY

FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY



Exploiting Ring Oscillators to Leak Secret Information

Fanny Lundberg
fa2023lu-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Markus Törmänen, Joel Åhlund

Examiner: Erik Larsson

May 30, 2025

Abstract

The manufacturing of hardware is a complex process where many different types of parties are involved. This creates risks of inserting hidden harmful parts, so called Hardware Trojans (HTs). HTs can have many different functions, such as reducing security, leaking information or degrading a system. This thesis examines how Ring Oscillators (ROs) can be exploited by a malicious party and modified to instead of protecting a system, leak secret information. This was examined by implementing a HT with the purpose of leaking the encryption key of an AES encryption module through power side-channels.

It was also examined how HTs can be categorized, and if the categorization could be improved. It was suggested that apart from existing categories: insertion phase, abstraction level, activation mechanism, effect, location and physical characteristic one should also add the categories: type of components/hardware, malicious actor, attack type and attack difficulty.

The effects of the designed HT were measured using an oscilloscope. The results of the measurements showed that even with a small RO of only five Look Up Tables, one could easily read out the encryption key from the power consumption. It was also discussed that there exist many different valid reasons for inserting ROs in a system and discovering one or two small ROs of the HT kind would be very difficult. Therefore, this is a problem that should be further examined.

Popular Science Summary

Manufacturing Hardware is a complex and expensive process. Therefore, most companies does not do the entire process by themselves, rather it is divided between many different companies, often from many different countries. The hardware produced is also very complex and often contains so many components that it becomes almost impossible to test every part of it effectively. The problem then becomes how to ensure that secure hardware actually is secure, and that no modifications have been made during the production process. A Hardware Trojan (HT) is a hidden modification of hardware with the purpose of doing harm to a system. These HTs, if successful, reduces the trust of secure applications.

This thesis demonstrates the problem above by proposing a HT design that can be used to attack cryptographic applications. The encryption algorithm most used today is the Advanced Encryption Standard (AES). The HT targets the encryption key of the AES encryption and tries to leak it through modifying the circuits power consumption.

The HT was built on a Ring Oscillator (RO), which is a loop of components, typically inverters. By inverting a signal and sending it around the loop before inverting it again, and so on, an oscillating signal is created. When a signal oscillates, it consumes extra power.

Many ROs also contain an enable signal that can turn the oscillation on or off. When the RO is enabled it consumes more power than when it is disabled. By enabling and disabling the RO in a controlled matter based on what value the encryption key is, it is possible to read out the key from measurements of the circuits power consumption.

The results showed that it was possible to easily read out the encryption key from measuring the power consumption for a circuit with as few as five components in the RO loop, and a total of 173 components in the whole HT.

Using a RO in a circuit can have many valid reasons, such as creating a clock or checking the timing. Therefore having an RO in a circuit is not strange. Normal, non-modified, circuits can have many thousands of ROs for valid reasons.

One RO that has been modified can therefore be hard to detect. Together with its small size, this makes the HT stealthy and difficult to discover. Leaking information this way can therefore be considered a threat to the security of applications and protection against it should be examined.

Acknowledgements

This thesis, for the degree of Master of Science in Engineering, has been carried out at Advenica.

Firstly, I would like to thank my supervisor at Advenica, Joel Åhlund, for all the help and guidance through out this thesis.

I would also like to thank my supervisor Markus Törmänen and my examiner Erik Larsson for many valuable discussions and advice.

Lastly I would like to thank my fellow thesis students at Advenica for providing encouragement and support.

Table of Contents

1	Introduction	1
1.1	Problems	1
1.2	Benchmarks	1
1.3	Ring Oscillators	2
1.4	Goals and Research Questions	2
1.5	Disposition	2
2	Background	5
2.1	Hardware Trojans	5
2.2	Insertion Phase	6
2.3	Attacks	7
2.4	HT Detection	11
2.5	Ring Oscillators	12
2.6	Side-Channels	12
3	Categorization	15
3.1	Existing Categories	15
3.2	Added Categories	15
4	Design	17
4.1	Implemented Hardware Trojan	17
4.2	Setup	19
5	Results	21
5.1	Experimental Results	21
5.2	Categorization of the Hardware Trojan	24
6	Discussion	27
6.1	Possible Discovery and Effectiveness	27
6.2	Possible Protection	28
7	Conclusion	29
7.1	Conclusion	29
7.2	Future Work	29

Appendix _____	32
A HT Code _____	33
B Activation Code _____	37

List of Figures

2.1	Standard RO circuits.	12
4.1	Schematic of the RO for $N=5$	18
4.2	Connection between the HT and the AES module.	19
4.3	Measuring setup.	20
5.1	Waveforms for different values of N	22
5.2	Waveform when the HT is not active.	23
5.3	Power consumption difference between a zero and a one key bit for different values of N	24

List of Tables

4.1	Truth tables for the LUTs.	18
5.1	Utilization of the FPGA resources for the HT with $N=5$	24

List of Abbreviations

<i>3PIP</i>	Third Party Intellectual Property
<i>AES</i>	Advanced Encryption Standard
<i>ASIC</i>	Application Specific Integrated Circuit
<i>BRAM</i>	Block Random-Access Memory
<i>COTS</i>	Commercial Off-the-Shelf
<i>EDA</i>	Electronic Design Automation
<i>FPGA</i>	Field-Programmable Gate Array
<i>HT</i>	Hardware Trojan
<i>IC</i>	Integrated Circuit
<i>IP</i>	Intellectual Property
<i>LED</i>	Light Emitting Diode
<i>LUT</i>	Look Up Table
<i>MSB</i>	Most Significant Bit
<i>MUX</i>	Multiplexer
<i>PRNG</i>	Pseudo Random Number Generator
<i>RO</i>	Ring Oscillator
<i>RSA</i>	Rivest-Shamir-Adleman
<i>RTL</i>	Register Transfer Level
<i>SoC</i>	System on Chip
<i>SRAM</i>	Static Random-Access Memory

1.1 Problems

The manufacturing of hardware is a complex process where many different types of parties are involved, often from many different countries. Because of this, there is a huge risk when it comes to parties adding or modifying the hardware in a harmful way. A Hardware Trojan (HT) is a hidden, deliberately harmful part that is built in to the hardware. The purpose of a HT is to attack the system from the inside and it can lay dormant in the system before the attack happens. The risk that a HT is built in affects the trust of these systems. The persons designing a system are often not the ones who tests the same system, and they can even be from different companies. Making sure that no one modifies or adds some parts of the system before it is released is therefore not an easy task. Even if a modification is noticed, there are often several valid reasons for making the modification in order to make the system securer, even though that might not be the case.

Although one might think that inserting a HT in circuit, knowing that it has been successfully inserted and being able to activate it might seem impossible, it is not such a difficult task. The hardest step is to insert a HT without being detected. However, if an employee, such as a designer or a tester can be bought to go against the company, this step also becomes much simpler, and even if this is not the case, there are ways to succeed.

1.2 Benchmarks

HT benchmarks are example circuits with built in HTs that are made to help evaluate the effectiveness of new methods created to detect these HTs. Today, the main collection of benchmarks used in research is TrustHUB [1]. The problem with these benchmarks is that many of them are not useful as they are. They contain many errors: Some do not have a specification, some cannot be implemented on hardware, and some work, but do not do any harm at all to the system according to Krieg [2]. When these benchmarks are to be used, they first need to be modified. This leads to many different versions of the benchmarks which makes it hard to compare the results of different methods, even if they state to have used the same benchmarks.

These benchmarks could possibly be categorized regarding malicious actor (manufacturer, sub-module provider, rouge employee, etc.), attack type (Denial-of-Service, steal information, backdoor), attack impact (device malfunction, hi-jacking) and attack difficulty (does it require a lot of inside information? Specialist knowledge?). To unify these benchmarks would ease the comparisons, as well as the testing of new methods.

1.3 Ring Oscillators

Ring Oscillators (RO) are used for many purposes in circuits, such as clock generators, managing phase offsets or temperature sensors. The sensitive nature of ROs also give them many uses within creating visibility of internal signals and delays, which makes them a popular subject within research for detecting HTs. A small change in a delay, by the insertion of extra logic or degradation of a component, can show up in the frequency of the RO. Therefore, inserting ROs in a circuit are becoming more and more common.

To use ROs to get good visibility requires many ROs. For example, Cheng [3] used ROs for measuring delays in an circuit to keep track of the aging of components in a Field-Programmable Gate Array (FPGA). The results showed that to get accurate results one would have to fill the whole fabric with ROs, but that one could get away with a maximum error of 1.3 % if only filling the spaces that the original circuit did not occupy.

This example shows that using ROs to measure things inside of a circuit could require thousands of ROs. Apart from the benefits in increased visibility there is also a risk that the ROs could be modified and used by a malicious party instead. Using a circuit with thousands of ROs makes detecting one or two that has been maliciously modified very difficult.

1.4 Goals and Research Questions

The goal of this thesis is to examine how ROs can be exploited by a malicious actor with a HT. Questions to be answered are: How can HTs be categorized and connected to different types of threats and malicious parties? Can a HT be designed using only digital components to target FPGAs and how big does the HT have to be? What vulnerabilities are created by using ROs in a circuit and how can they be exploited by a malicious actor?

To examine these questions the thesis begins with a background study and then a HT is designed, implemented and tested. This HT is based on the vulnerabilities added by using ROs in a circuit. The focus of this thesis is specifically HTs in FPGAs.

1.5 Disposition

In this thesis we go through what a HT is and what different types exists, possible attacks and detection techniques, RO designs and side-channels in chapter

2. Chapter 3 contains a proposed categorization of HTs. Chapter 4 contains a description of the designed HT and the setup used for the experiments. Chapter 5 describes the experimental results and the categorization of the implemented HT. The results are discussed in chapter 6 and chapter 7 finishes the article with some conclusions.

2.1 Hardware Trojans

A HT is a hidden modification of a circuit with the purpose of doing harm to a system. The system targeted can be any type of hardware, for example Application Specific Integrated Circuits (ASICs) or FPGAs, and the HTs can have many different designs and ways of doing harm. To be classified as a HT it is important that the modification should not be easily detectable in any of the commonly used manufacturing test methods for hardware. Those tests can be for example functional tests, which tests whether each function or feature works as the specified requirements by inputting known values and checking if the outputs are correct.

Shakya *et al.* classify HTs according to two main types [4]: functional and parametric. A functional HT include specific malicious functionality that does harm by for example give unauthorized access, change functionality or leak information in some way. In order to do this, some gates are either removed or added from the original design. A parametric HT can do harm in the same way as a functional HT, but it does it through modifying existing physical parameters, for example wire thickness. The parametric HTs are, in difference to functional HTs, also often used to target the physical health of the hardware. By modifying the physical parameters one can put stress on different electronic parts, for example cables, in order to make parts of the device function differently or eventually reach denial of service.

A HT normally consists of two parts, a trigger and a payload. The trigger is the condition for when the HT should be activated. It could be both an internal trigger or external. The external triggers are triggers that react to an event or condition in the world outside of the circuit. The external triggers could be temperature, altitude or voltage levels, but they could also be a signal received through an radio antenna or a press of a button. The internal triggers are triggers based on events inside the circuits. It could be either input or state based, for example waiting for a specific input or sequence of inputs, or purely timing based, for example a time bomb. The payload is the signals, both inputs and outputs, from the HT-free implementation that the HT taps in on, or the nodes that are affected by the HT. It is in other words inputs that are used to trigger the HT or inputs that are used to change some output, and the outputs that are affected.

When the HT is inactive the circuit behaves as the HT-free version, which

makes most test results the same as if the HT was not present. When the HT is active, the test results might not be the same as the HT-free version. To avoid detection, it is important that the trigger is a condition that only happen very rarely, so it does not get accidentally activated during testing. A too rare trigger condition can on the other hand make the HT hard or even impossible to activate. A non-rare condition for the trigger would make the HT easier to activate but it would also increase the risk of the HT being activated during testing, and thereby discovered. Therefore, the trigger needs to be chosen wisely.

2.2 Insertion Phase

Where a HT can be inserted depends on the type of hardware it targets and the hardware’s supply chain. According to Shakya *et al.* [4] an ASIC production’s supply chain is typically divided into three parts. They are: Third Party Intellectual Property (3PIP) vendor, System on Chip (SoC) integrator and foundry. 3PIP vendor means all Intellectual Property (IP) designs that are made by another company or a person that is not an employee. The SoC integrator is the person integrating all different parts and different IPs to work together and can be both an employee or an outside person. The foundry is where the final design is sent and the ASIC is produced. A HT can be inserted in all parts of this supply chain, from specification to assembly and even by the Electronic Design Automation (EDA) tools. The part that is easiest to control is the SoC integrator, since this is often a part of the company that wants the final design. The most concerning parts are therefore 3PIP vendor and fabrication. Designing every part of a system from scratch takes a lot of time and costs a lot of money. Therefore it is common to use some IP blocks from outside, often to help with common functions. If a malicious part is inserted in the 3PIP vendor step in the design files it is referred to as a design-level HT. Owning a foundry is a huge cost and therefore most companies outsource the fabrication of the chips. This cost efficiency comes with the price of less control over the finished product. If the physical hardware or the fabrication process is targeted, the HT is called a physical-level HT.

The FPGA supply chain has many similarities with the ASIC supply chain, and is also divided into the same three parts, 3PIP vendor, SoC Integrator and foundry, but the order of these three are different. In the FPGA supply chain the foundry comes first, followed by the 3PIP vendor and lastly the SoC integrator. The order of these is important when it comes to possible HTs, since the order of the FPGA supply chain means that the IP-design is not included when the FPGA is produced in the foundry. An effective physical-level HT is therefore much harder to do on an FPGA than an ASIC. For the SoC integrator the same can be said as for the ASIC supply chain, the SoC integrator is often an employee and considered a trusted party. The 3PIP vendor is however a major concern regarding insertion phase and using untrusted IPs must be done with great care.

2.3 Attacks

2.3.1 Known Hardware Trojans

The main problem with developing detection techniques and examining HTs is that there are very little information from actual attacks. Krieg [2] even states that there are no known real incidents of an attack using a HT. The problem is that a successful HT is not discovered and we therefore get no knowledge of how the HT was designed. Also, a person or a company attacked by a HT that was discovered, does not want this to be known. By stating that a HT was discovered it is also stated that there exists some vulnerabilities regarding the possibility of getting a HT into this place. Even worse is stating what kind of attack was made since this shows that such an attack is possible. Both of these scenarios may lead to more attacks on said company or person and therefore used HTs remain unknown.

King *et al.* [5] states that although there are no known malicious circuits there are several examples of known attacks using some type of modified hardware devices. An attack that could be considered as a failed HT is the one described by Robertson and Riley [6]. Here a HT was successfully implemented and distributed to many important parties in the U.S. The purpose of these implants was to manipulate the core operating instructions of servers, to be able to insert code, change functionality and leak information. The implant was later discovered with no known consumer data stolen. Many believe that it would not have been discovered as fast and maybe not at all if it wasn't because of a received tip about the HT. This example shows that HTs are already being used and that it is important to learn how to prevent such attacks.

2.3.2 HTs developed by researchers

Aside from this example there are also many possible HTs developed by researchers. The largest and most famous of such examples is, as mentioned previously, the TrustHUB benchmark suite [1]. Although this collection of benchmarks might seem to be covering many aspects of HTs, the problems are many. Krieg [2] studied these benchmarks more closely and came to the conclusion that only three out of the 83 studied benchmarks were well made and he even went as far as stating that the rest could not even be considered HTs. In this benchmark suite there are however some HT benchmarks that are similar to the one implemented later in this thesis. They are designed to leak information through a side-channel. This is done for a couple of them by connecting the secret information to a capacitor, to let the capacitor affect the power consumption. One such example is the AES-T1800 that leak the encryption key from an Advanced Encryption Standard (AES) module through a capacitor. The problems with these benchmarks is however firstly that the capacitor is just put as a name of a wire in Register Transfer Level (RTL) code, there is no actual capacitor element in the code. Secondly since the capacitor wire, which is the only output of the HT, is not connected to anything, the whole HT is considered redundant and is removed when circuit optimization is made. These benchmarks are therefore not possible to use directly as is.

Another similar type of HT is described by Lin, Burleson and Paar [7] and is called MOLES. In this HT, they also leak information through a side-channel by

connecting the encryption key of an AES module with a capacitor. The differences compared to the TrustHUB benchmark is mainly two things. Firstly, MOLES connect the encryption key to a model of a capacitor in order to do simulation tests. Secondly, they also add a type of mask to the encryption key before it is leaked. The mask comes from a Pseudo Random Number Generator (PRNG). They then describe a method to extract the key from this seemingly random output. The experiments show that the technique is successful at leaking the key at least in theory and in simulation.

A third example of a slightly different type of attack is the HT described by Zhao and Suh [8]. In this example the targeted platform is a cloud based FPGA shared between users. In this platform all users share the same physical FPGA, but have separate online FPGA models where they can upload their design and get results from running on the actual FPGA. The different designs of the users are separated by unused logic on the physical FPGA. The goal of the HT is to steal the encryption key of an Rivest-Shamir-Adleman (RSA) encryption module used by one of the other users, even though it is blocked by unused logic in between. Stealing the encryption key is made by implementing a RO. Since the two designs are run on the same FPGA they are affected by the each others current and voltage usage, even if this effect is very small. The RO is so sensitive to small changes in these currents and voltages that the small differences in calculations inside the RSA module that depend on the changes in key bits can be measured. Thereby the RSA encryption key can also successfully be extracted.

Comparison

One problem with both the TrustHUB benchmarks [1] and the MOLES [7] is that using a capacitor limits the HT to only work on circuits with analog components. This type of HTs would not for example work on an FPGA since FPGAs normally does not contain a capacitor or any other analog elements. The HT designed in this thesis is based on the same idea of leaking information through side-channels as the TrustHUB and MOLES techniques, but is only using digital components, in order to target FPGAs. The problem with Zhao’s and Suh’s [8] approach is that the attack of measuring side-channels is widely known. Therefore, the majority of applications using secret information is nowadays designed to not show any differences in the side-channels for different secret information. The HT designed in this thesis is based on ROs but are made from the point of view that it would actively have to modify the side-channels in order to get any information from them, rather than just measure.

2.3.3 Possible attacks on FPGAs

Considering the previously mentioned insertion phases there are many ways an attack could be constructed. For attacks on designs implemented on FPGAs there are two main areas: HT in the IP and HT in the device. A HT in the IP means that the HT is inserted as a part of the IP before it is loaded onto the FPGA. This is done in the 3PIP vendor or SoC integrator step in the supply chain. A HT in the device means the HT is inserted in the foundry when the FPGA is

being produced. The HT in the device can both be HTs that target the IP that will be loaded onto the FPGA later or HTs that target the device itself, meaning parts that are common for all FPGA devices of that typical fabrication and do not depend on the IP loaded onto the FPGA. According to Mal-Sarkar *et al.* [9] the trust issues concerning HTs in FPGAs are mainly with HTs that attack the FPGA itself and that HTs that attack the final IP design are unlikely to happen due to a number of reasons. The main reason being that the IP design is not available during production of the FPGA. Only the base array of the FPGA is manufactured using a standard Integrated Circuit (IC) manufacturing flow. One would therefore have to either guess the later IP implementation or randomly place HTs and hope that they will be in a position where they can do the intended harm. The attack then becomes a probabilistic one, where the HTs need to be placed where they have a high probability of hitting an important function of the circuit. For maximum chance of the activated HT actually affecting the FPGA design, the triggers could be distributed on many parts throughout the FPGA, which then increases the discovery risk. The problem would first be to actually hit the circuit, and then also to hit a part where it actually matters.

HTs that change the netlist after it is loaded onto the FPGA are not a very big treat as long as the EDA tool can be trusted according to Trimberger [10]. This is because there exists many comparison tools that compare Layout-versus-schematic or directly compare two netlists to find added or removed logic.

Stealing the IP attacks

One of the most costly parts in an FPGA implementation is the IP design. Therefore, the IP design should have high security against theft and reverse engineering. According to Trimberger [10] the IP-design in an FPGA is stored in a Static Random-Access Memory (SRAM) and can be considered as an electronic message. An electronic message can be protected using information security techniques, which are already tested and understood, so there is no need to implement new protection techniques for that. Many FPGAs also have some measures installed to prevent stealing the IP, such as encrypting the bitstream, or clear the configuration data when an attempt is made to steal it. This makes both unauthorized copy and reverse-engineering hard to do. Trimberger [10] also states that there are no known reports of successful reverse-engineering of an FPGA bitstream. However, none of those techniques mentioned prevent the IP from being leaked by a HT, since the possible HTs pass the conventional information protection. Using a HT the IP can be leaked either by leaking the decryption key of the encrypted bitstream, by for example tapping in on the wires connecting the storage of the key and the decryption device, or leak some part of the design itself.

One method described by Trimberger [10] to prevent the bitstream from being stolen by tapping in on the wires is to load the FPGA bitstream only once, in a secure factory and then keep the FPGA powered constantly to ensure that the bitstream running inside is both secure and unmodified. This is however not a very good solution since constantly powering a device is both unpractical and wasteful. Therefore, many FPGA vendors have implemented more secure techniques to protect the bitstream. This does prevent stealing or modifying the

bitstream with a simple attack, but the keys and designs can still be stolen with a HT.

Device attacks

Layout-based HTs are not applicable for FPGAs, but there could still be pre-existing HTs in the FPGA fabric that were inserted during fabrication of the FPGA. Attacking the FPGA device itself requires no knowledge of the IP design that is going to be loaded onto the FPGA later. The HT can be inserted in the foundry when the FPGA is produced. An example of an attack on the FPGA device itself described by Trimberger [10] could be to try to weaken the security of the FPGA, in hopes of later being able to exploit the weakened state when the FPGA is inserted in the full system. The technique to prevent this attack is pretty simple. Since the security features are only a very small part of the device, it could be verified without significant time and cost. The same base-array can then be used to compare with a large number of designs, making sure they are trusted, without making the verification more times.

Another example for a IP-independent HT, described by Shakya *et al.* [4], is to attack the digital clock managers. By making small changes to the hardware the clock can be made inaccurate. Causing a fault in the clock frequency would change many sequential FPGA implementations behaviors. In the best case, a small clock fault cause only minor calculation errors in some rarely used cases, but in the worst case the whole functionality of the circuit could change. If the circuit in question is for example an encryption module then a fault could mean that the encryption is not made strong enough and can be easily broken or that the message becomes impossible to decrypt even with the correct encryption key.

The problem with HTs made in the foundry that attack the device itself is that most FPGAs are Commercially-off-the-shelf (COTS) FPGAs according to Trimberger [10]. This mean that the FPGAs are mass produced before they are sold, making it impossible to know what the device should be used for or even what company will buy them. Also, many different companies usually buy FPGAs from the same production batch. Therefore, an inserted HT in the foundry will affect all devices and every company that chooses to buy that specific batch of FPGAs. A party will most likely not load the FPGAs with the exact same IP design as some other party. This will create many opportunities to detect the HT since a lot of different implementations increases the chances of the HT being activated and detected. Also, since all the FPGAs are the same, a number of them could be destructively tested, to check if there exists any added or removed logic, without creating a huge cost.

3PIP HT

The main area for the HTs that remain is the 3PIP HTs. These are HTs that are inserted through IP blocks provided by an untrusted party and can target any part of the design. They are mostly IP-dependent and could for example be monitoring the logic values of Look Up Tables (LUTs), to corrupt them, load incorrect values into Block Random-Access Memories (BRAMs) or sabotage configuration cells.

With a bit of imagination and creativity a 3PIP can try to attack almost any device and design.

2.4 HT Detection

There are different types of countermeasures for HTs, either HT prevention or HT detection. The majority of research regarding HTs are made towards HT detection. HT detection can be divided into two subcategories: Destructive and Nondestructive. The destructive detection techniques render the IC or FPGA useless after the process is finished since it destroys the chip permanently. An example of a destructive detection technique is reverse engineering. Reverse engineering is when you take a chip and remove the metal layers layer by layer to scan how the inside of the chip looks like in order to get the final layout design. Destructive techniques are often not considered viable since they are high cost, takes a lot of time and render the IC useless after the process. The nondestructive techniques can also be divided into two main categories, post-silicon and pre-silicon. Post-silicon is used for testing the IC or FPGA after it is produced in the foundry. The test techniques for post-silicon are for example functional tests or side-channel signal analysis. The pre-silicon test focus on the code or IP-design before it is produced as an IC or loaded onto the FPGA. Examples of pre-silicon test techniques are functional validation, formal verification or code/circuit coverage.

Most detection techniques developed so far, especially the side-channel signal analysis techniques, require a golden design or golden IC to compare with in order to find the harmful modifications according to Xiao *et al.* [11]. A golden model is a HT-free version of the circuit that is trusted and is known to not include any harmful parts. The idea with these techniques is to extract different types of measurements, for example power consumption or execution time, from the golden model and then compare these with measurements from the untrusted design. If the measurements from the untrusted design differ too much from the golden model's then it is considered modified and can not be used. The problem with using a technique that require a golden model, although they are very precise, is that golden models are in most cases not available, which makes the techniques unpractical. Also, even if there exists a golden model, the HTs might be so small and well constructed that they can't be distinguished from noise sources or process variations.

Apart from the problems with many of the detection techniques mentioned above there are also many techniques that do work on some specific HTs, although Chakraborty, Narasimhan and Bhunia [12] states that there is not yet any one detection technique that can be used to detect all types of HTs. Many of the techniques designed to detect HTs without using a golden model use some type of RO. An example of a detection techniques using ROs is made by Deepthi [13]. Deepthi describes how ROs can be inserted in the circuit paths to measure changes in frequency to detect HTs.

2.5 Ring Oscillators

A RO consists of a number of inverters that are connected in a loop. The output of the last inverter is fed back as an input to the first inverter, creating a signal that oscillates. The standard model of a RO can be seen in fig 2.1a. It is also common to change the first inverter in the loop to a NAND-gate to be able to control the RO with an enable signal. The enable signal turns the ROs oscillations on or off. A RO with an enable signal can be seen in fig 2.1b. How fast the signal is oscillating depends firstly on how many inverters there are in the loop. It is important that the numbers of inverters are odd, otherwise the signal would not oscillate at all, since the bit sent back would be the same as was already an input to the first inverter. The oscillating frequency also depends on the specific components used, since there are small differences between inverters produced at different times. The frequency is also affected by surrounding electronics and some environment conditions, such as temperature, humidity and altitude. Although these might seem like unimportant factors, a RO is very sensitive and even small changes in these factors creates a change in the frequency.

Apart from the standard form of ROs there also exists some newer versions. Some of these newer designs are described by Sugawara *et al.* [14] where they use both latches and flip-flops to create the oscillating signals. These specific ROs are designed to avoid checkers that look for loops in circuits. Another version is to use LUTs instead of inverters.

ROs are also getting increasingly popular to use for detecting HTs. Because of the sensitive nature of ROs, they are very good for detecting small changes inside of a circuit. Inserting extra gates or making other changes of a path inside of the circuit will show up in a slightly changed frequency of the RO, indicating that a HT might be present.

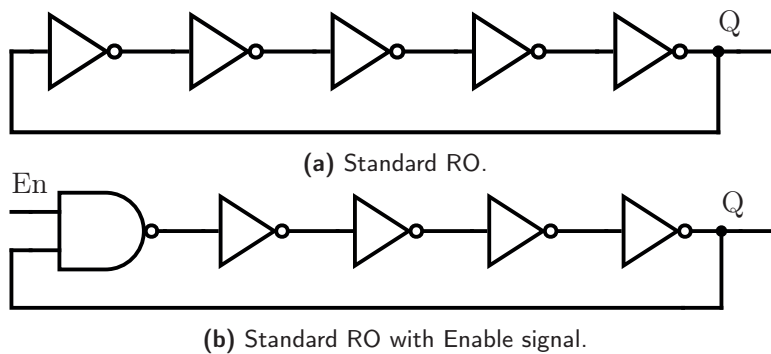


Figure 2.1: Standard RO circuits.

2.6 Side-Channels

A side-channel is a channel that is not a primary input or output of a circuit, but that still carry information about said inputs or outputs or an internal state of the

circuit. This information is unwanted and can be exploited as a security weakness. The reason why using different types of data can give different outcomes in the side-channels could be because of different branches being taken depending on some value, or calculations being heavier from larger numbers. By measuring these side-channels one can discover patterns that may lead to discovery of sensitive data, such as encryption keys. Some examples of side-channels are power consumption, timing or delay, electromagnetic emissions or heat radiation.

Unintentionally leaking information through side-channels is nowadays widely known as a problem when it comes to securing information. Therefore, implementing a new algorithm or problem that contains secret information must be done with care. For example, many of the encryption implementations are nowadays made to have constant side-channels regardless of what input or key is used, to make it hard or even impossible to get any information from the side-channels. Therefore, leaking information through side-channels might seem like solved problem, but using HTs one can still exploit these weaknesses in a seemingly secure system.

3.1 Existing Categories

For categorizing HTs there are a number of proposed ways. Shakya *et al.* [4] describes a possible categorization of HTs that are also used to categorize the HTs in the TrustHUB [1] Benchmark suite. They use six different categories: insertion phase, abstraction level, activation mechanism, effect, location and physical characteristic. The categories can be further explained as:

- Insertion Phase: In what stage the HT is inserted. Could be for example design phase or integration phase.
- Abstraction Level: Which abstraction level the HT has. Could be for example system level, where a HT must conform with the decided interfaces and interactions, or at physical level, where all locations are determined and only the white/dead spaces are available for insertion.
- Activation Mechanism: How the HT is activated. Either it is activated by some trigger, externally or internally, or it is always on.
- Effect: What effects the HT has on the system. The effects of the HT may be to change the functionality of the circuit, to produce faults, or to reduce the performance.
- Location: What part of the system the HT is affecting. Some examples are inputs/outputs, LUTs or Memories.
- Physical Characteristic: If the HT is a functional or a parametric HT and if it is loosely or tightly distributed.

3.2 Added Categories

Apart from the categories listed above, some more should be considered if a full picture of the HT should be acquired. The proposed added categories are:

- Type of Components/Hardware: What type of components are the HT using: analogue or digital or a combination of both? Based on the type of components, what type of hardware is the HT designed to be run at? The

type of hardware could be for example FPGA if the component are only digital and ASIC if the components are analogue or a mix of both digital and analogue.

- Malicious actor: What type of malicious actor is required for inserting the HT. This could be for example a manufacturer, a sub-module provider or a rouge employee.
- Attack type: What type of attack the HT performs. Could be for example Denial-of-Service, steal information or create a backdoor.
- Attack difficulty: How hard is the attack to perform. Does it require a lot of inside information or specialist knowledge?

The category "Type of Components/Hardware" should be an added category since a HT designed to attack an ASIC are not necessary functioning on an FPGA and vice versa. When using a HT benchmark it is therefore important to now what a certain HT is compatible with without having to search through the design files.

The category "Malicious actor" should be added to help distinguishing the problematic party. If for example a company are suspicious against their factory and wants to develop detection techniques targeting the types of HTs inserted in a factory, it will be easier to filter benchmarks on that and focus the efforts on that specific category.

The category "Attack Type" should be added to give a faster overview over what the HT does.

The category "Attack difficulty" is important if one knows the skills of the parties most likely to try to insert a HT. One can then focus the efforts on the type of benchmarks of appropriate difficulty.

4.1 Implemented Hardware Trojan

The HT implemented has the purpose to leak an encryption key from an AES module through side-channels. The leakage is done through increasing and decreasing the current that the circuit requires during run time, so that the key can be read from the current consumption. This is realized through a type of RO that is turned on to increase the current and turned off to decrease it. The RO is modeled as N LUTs connected in a ring, and can be seen in fig 4.1. The number of LUTs in the ring can be varied. The LUTs used was Vivado’s LUT2 primitives. Each LUT has two inputs, R1 and R0. R1 is the signal connecting the LUTs to each other. R0 is both the activation signal and the key bit input.

The first $N-1$ LUTs are modeled as AND gates and the truth table for these can be seen in table 4.1a. The last LUT is also an AND gate but with one input inverted. The inverted input is the one connected to the previous LUT. This makes the output of the RO inverted, and since it is this signal that is fed back to the start of the ring, it is this inversion that creates the oscillation. The truth table for the last LUT can be seen in table 4.1b. An example of the full RO with this design, with $N=5$, can be seen in fig 4.1. As can be seen in both of the truth tables, as long as the R0 signal is zero all LUTs outputs only zeros and there is no switching of the signals. This can be seen as the RO being inactive. When the R0 signal is 1 then the first $N-1$ LUTs outputs the value of the R1, and the last LUT inverts it. When the trigger condition is not met, the R0 is set to be zero, meaning the RO is inactive. When the trigger condition happens the R0 signal is connected to the individual bits of the encryption key, starting from the Most Significant Bit (MSB) and shifting down to the next bit after a certain amount of clock cycles. If the key bit is a 1 then the RO will start oscillating and the circuit will consume more power. If the key bit is a 0 then the RO will not have any switching signals and will act as if it was inactive. When the whole key has been traversed the R0 signal will again be connected to a zero signal and will stay that way until the next time the trigger condition is met.

The HT has five main signals: four inputs and one output. The inputs are the signals: clock, reset, activate and information. The activate signal is the signal activating the HT when the trigger condition is happening. The information signal is the secret information that should be leaked by the HT. The output signal is

R1	R0	O
0	0	0
0	1	0
1	0	0
1	1	1

R1	R0	O
0	0	0
0	1	1
1	0	0
1	1	0

(a) Truth table for the first N-1 LUTs (LUT2a). (b) Truth table for the last LUT (LUT2b).

Table 4.1: Truth tables for the LUTs.

the output of the last RO. The most important signals inside the HT interface is two counters. One counter keeps track of which bit position in the encryption key that is currently being leaked. This counter also acts as the checker for when the whole key is leaked and the HT should go back to the state of being inactive and waiting for the trigger condition to occur again. The other counter keeps track of for how many clock cycles each key bit should be leaked until moving on to the next one.

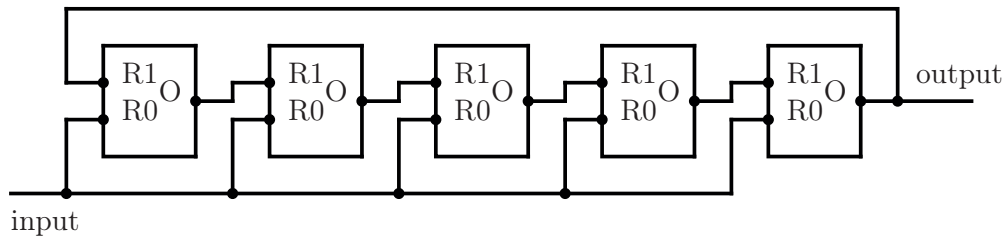


Figure 4.1: Schematic of the RO for N=5.

The circuit used to test the HT on was an implementation of the AES algorithm and the goal of the HT is to leak the encryption key. The information input to the HT is therefore the encryption key. The connection between the HT and the AES module can be seen in fig 4.2. The AES module receives keys and message blocks through an input port called `usb_rx`. When the encryption key is loaded into the AES, it is stored in key register. The key can then be read from the key registers by the HT. The input of the HT was therefore connected to the output of these registers. The message blocks are the messages that will be encrypted and they are also stored in registers. The block registers are connected to the trigger and the trigger condition was set to be the 128 bit message block: `00112233_44556677_8899aabb_ccddeeff`.

If the output of the RO is left unconnected, the design programs will think that the RO does not contribute to any output of the system. It would then remove the whole HT when optimizing the RTL code for placement on the FPGA. To prevent the HT from being seen as redundant logic by the optimization tool, the output therefore needs to be connected to something. At the same time, we want a low probability of the HT being discovered so it cannot be connected to something obvious. The output of the HT was therefore chosen to be connected to a segment selector of the FPGA. This segment selector chooses which group of

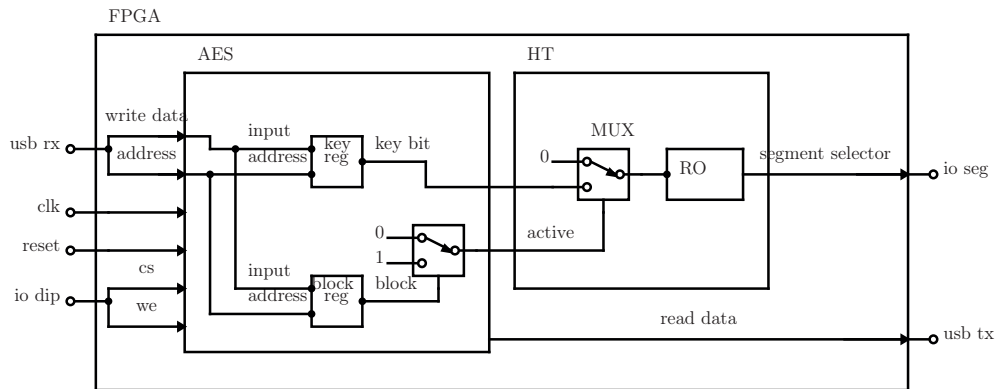


Figure 4.2: Connection between the HT and the AES module.

Light Emitting Diodes (LEDs) should be controlled on the FPGA. None of the LEDs are used for anything else and are therefore set to be turned off. Oscillating the segment selector signal will therefore not give any visual indication of the HT on the FPGA.

4.2 Setup

The HT was implemented on an FPGA board of model Alchitry AU with an extra Alchitry IO board connected. The board’s FPGA part name was xc7a35tftg256-1. The RTL code and necessary simulations was made in Xilinx Vivado. The power to the FPGA was provided by a Power Generator set at 5V and connected to the FPGAs power pins. Connected in series with the board was also a 2.5Ω resistor created by connecting four 10Ω resistors in parallel. To measure the current that the FPGA consumed at different times an oscilloscope was then connected in parallel over the resistor. The oscilloscope measures the voltage and using this and the value of the resistor the current can be calculated using

$$P = V^2/R \quad (4.1)$$

where P is the power consumption, V is the voltage and R is the resistance. The setup is shown in fig 4.3.

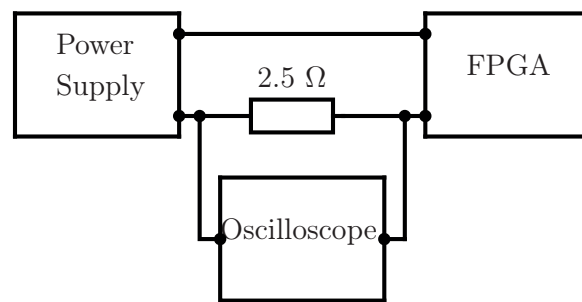


Figure 4.3: Measuring setup.

Chapter 5

Results

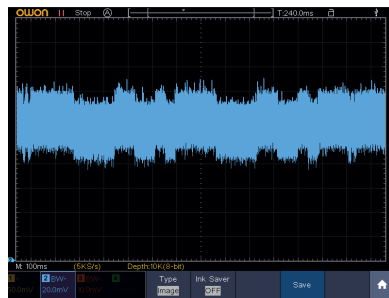
5.1 Experimental Results

The AES circuit implemented was tested through simulation only for functional testing, and not to see the effects of the HT. A RO is not possible to simulate fully since its behavior depends very much on the exact hardware used. In many of the simulation programs a RO even makes the simulation crash if no simulation delay is added in the RO's loop. This delay, even if made very small, would affect the power consumption and make the simulation results inaccurate. Therefore, the measurements of the HT could not be simulated, and was instead made directly using the hardware.

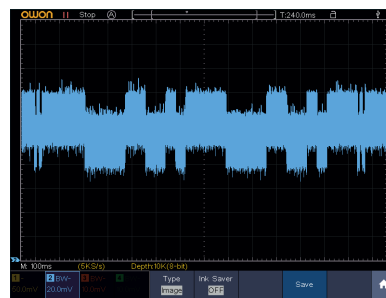
The circuit was tested using different sizes for the HT. It was tested for $N=3$ up to $N=99$. The voltage was measured through the oscilloscope and the waveforms for some of the different values of N can be seen in fig 5.1. The waveform for when the HT is inactive can be seen in fig 5.2. The key used in the measurements was 128 bits and the pattern was: `fff0000ff00f0fff0000ff00f0ff5`. This pattern was chosen to test many different lengths of consecutive 1's and 0's. The difference in power between a one and a zero bit in the key was calculated using equation 4.1. The resulting power difference between a zero and a one key bit can be seen for different values of N in fig 5.3.

As can be seen from the figures 5.1 and 5.3 there is a difference in power between the zeroes and ones and this difference is what changes when using different values of N . When using $N=3$ the difference in power between a zero and a one was only about 25uW. This was a bit too small to see the different bits clearly, especially when the key switched a lot between ones and zeros. However, already at $N=5$ the difference increased to about 115uW, which made the different bits clearly distinguishable, even when switching every bit. When continuing to increase N the power difference continues to increase, but with smaller and smaller increases, until around $N=51$. With values of N larger than 51 the power difference stayed at around 810uW.

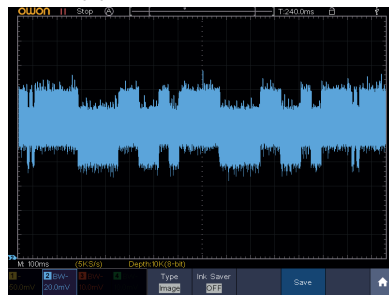
The register counting how long each key bit was leaked was for these measurements set to 20 bits. This makes each key bit being leaked for about 1 million clock cycles. With a clock cycle period of 10ns, 1 million clock cycles equals about 10,5ms, so each key bit controls the RO for 10,5ms. Leaking the whole key takes 1,34 seconds for a 128-bit key and 2,68 seconds for a 256-bit key.



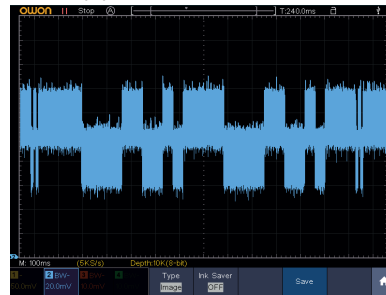
(a) Waveform for N=3.



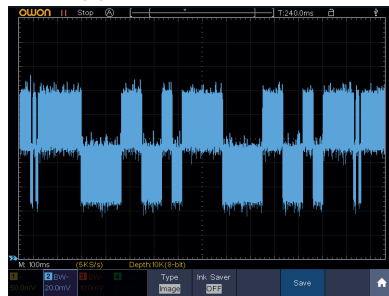
(b) Waveform for N=5.



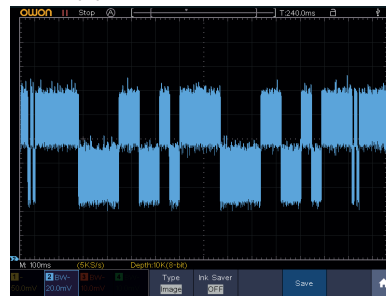
(c) Waveform for N=7.



(d) Waveform for N=15.



(e) Waveform for N=31.



(f) Waveform for N=99.

Figure 5.1: Waveforms for different values of N.

When the HT was tested without the counting register, making each key bit being leaked for only one clock cycle, it was not possible to see the difference in power consumption on the oscilloscope. This was probably due to the small power peaks and dips being removed by the voltage and current buffers that exist on the FPGAs inputs. The value 10,5ms per bit was determined experimentally. This value was chosen because at this value the peaks and dips in voltage were to big for the buffers to remove, and also the different bits were clearly distinguishable from each other.

The reason why some measurements seem a bit off, for example N=7 and N=9 having a smaller power difference than N=5 could be because of mainly two reasons. Firstly, the power difference is calculated from manually measuring voltage levels on the oscilloscope using cursors. This is not an exact measurement

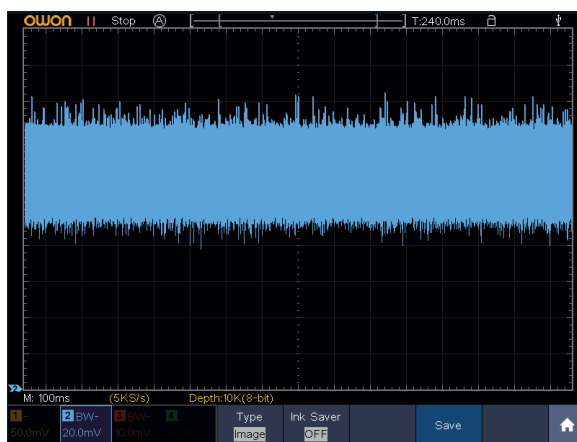


Figure 5.2: Waveform when the HT is not active.

and can give some errors in the result. The second reason is that the optimization tool can place the same design on the FPGA in many different ways. Therefore, changing the value of N can change the layout of the whole design and might result in a more or less optimized value.

It should also be noted that these measurements are taken when the AES module is not actively doing any encryption or decryption. To leak the encryption key using this HT does not require any timing of when the key is being loaded or used to be able to extract it. It is enough that the AES module has a key loaded in and that the FPGA has power to run.

5.1.1 Warnings

When the HT is inserted in the RTL code it generates a critical warning that states:

"Combinatorial Loop Alert: N LUT cells form a combinatorial loop."

It also generates a normal warning that states:

"Found switching activity that implies high-fanout reset nets being asserted for excessive periods of time which may result in inaccurate power analysis."

The critical warning can be removed by adding the row:

```
"set_property allow_combinatorial_loops true [get_nets aes/core/ro/ro_sig_1]"
```

to the constraints file. However, the normal warning still remains.

5.1.2 Utilization

When taking into consideration all parts of the HT, which are RO, controlling the key bits to the RO and checking for the trigger condition, the total number

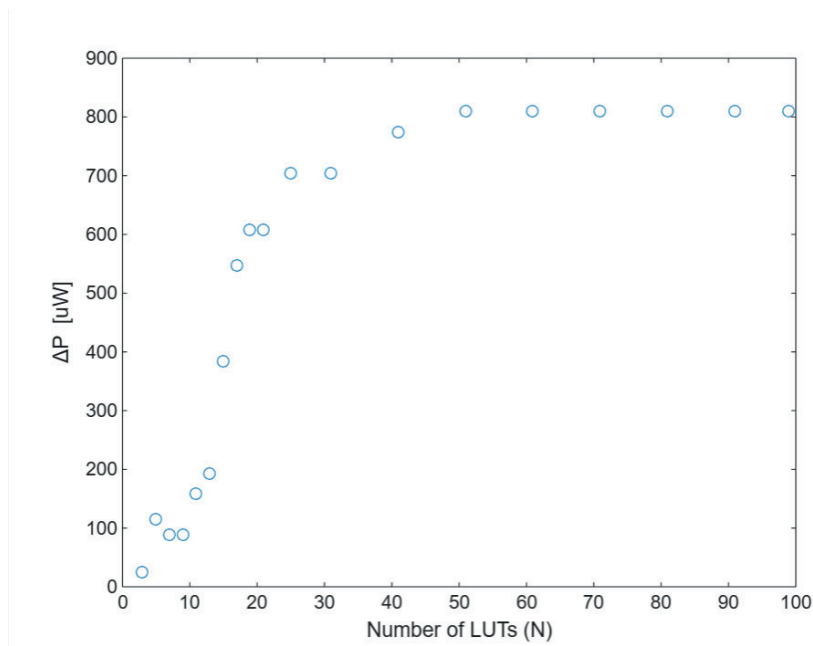


Figure 5.3: Power consumption difference between a zero and a one key bit for different values of N.

of components becomes as listed in table 5.1. The measurements were taken for N=5. Most of the components utilized were due to the big counters for keeping track of the timing of how long each bit should be leaked, and also for storing the key.

Components	HT Utilization	Total on FPGA	Utilization (%)
LUTs	110	20800	0,529
Registers	30	41600	0,072
MUX	33	16300	0,020

Table 5.1: Utilization of the FPGA resources for the HT with N=5.

5.2 Categorization of the Hardware Trojan

The HT that have been implemented can be categorized using the categories listed by Shakya *et al.* [4] and mentioned in chapter 3. They are: insertion phase, abstraction level, activation mechanism, effect, location and physical characteristic.

- Insertion phase: For insertion phase the HT could be categorized in more than one way. The most straight forward way to insert the HT is through a 3PIP. When the HT comes in through a 3PIP no golden model is available

and the HT can easily be added. Another way the HT could be inserted is through a malicious employee. That employee would then have knowledge of the IP design and could just add the extra RTL code. Adding the HT in the factory when the FPGA is being produced is possible but not very practical for this type of HT. Since you need to connect the HT to the signal you want to leak you need to have knowledge of where that signal is placed if it is going to have the right effect. Since the fabricated FPGA might be used for many different purposes, adding the HT without any knowledge of the actual usage of the circuit gives only a small probability of choosing the right place. Insertion in the factory is therefore not very likely to give a functioning HT.

- Abstraction level: The HT is inserted before the locations of the original circuit is decided so it categorizes as a system level HT and only needs to conform with the decided interfaces and interactions.
- Activation mechanism: For the activation mechanism, the HT is activated with an internal trigger that is based on the message blocks inputted to the cryptographic device.
- Effect: The effect of the HT is that it leaks secret information.
- Location: The location of the HT depends on where the secret that should be leaked is stored. The generalized HT could be inserted anywhere as long as it is connected to the signal that should be leaked. This could be for example secrets stored in LUTs or in the memory. For the AES circuit used to test the HT the location was the registers that stored the targeted encryption key.
- Physical characteristics: Since the HT adds gates to the circuit and does not alter any of the circuits original parameters, the physical characteristics is a functional HT.

Regarding the additional proposed categories, the HT implemented in this thesis could be categorized as follows.

- Type of Components/Hardware: The HT implemented is designed to be inserted on an FPGA and is mainly focused on the FPGA pipeline. Therefore the type of components used are only digital. However, since all digital circuit elements used are also possible to implement on an ASIC it could be added as a possible implementation.
- Malicious actor: The HT can be inserted by different types of malicious actors. The most likely would be a sub-module provider, a rogue employee or a test-creator.
- Attack type: The attack type of this HT is to steal information.
- Attack difficulty: Since this HT is based on simple components and can be inserted directly in the RTL code it does not require specialist knowledge. It does however require inside information about what signal should be leaked and where the HT should be placed to have the desired effect.

6.1 Possible Discovery and Effectiveness

The designed HT was successful at leaking the encryption key of the AES module. Leaking the encryption key is a malicious behavior since it reduces the security of the encryption module. The HT can therefore be seen as both correct and malicious.

Discussing whether or not the designed HT is stealthy when compared to a golden model is irrelevant. Since there is no golden model in the scenarios described for this HT then the problems that rises with comparison to a golden model are not applicable. A few examples of these problems are that there should not be any added design files, there should be no added input or outputs in the interface and the side-channels should give the same values, with just a small difference due to noise and process variations. These problems are not considered.

The RO was able to leak the encryption key even for the small size of five LUT in the RO. Even if all components of the HT are taken into consideration the size is still very small. With utilization of 0,529% for LUTs, 0,072% for registers and 0,020% for Multiplexers (MUXes) the probability of someone noticing this very small increase in a design of thousands or even millions of components is very small. Also, since the HT worked for values of $N=5$ and larger, it would be easy to change the length of the HT's RO to match that of ROs already present in a circuit that the HT is targeting. This makes the HT stealthy and hard to discover when it comes to size.

6.1.1 Warning Discovery

Even though the critical warning was easy to remove, the normal warning still remains. This warning will not hinder any simulation or implementation of the circuit, however it might give an indication to a tester that a RO exists in the circuit. This might not however reveal the HT since there are many reasonable arguments for inserting a ROs in a circuit, and there might even already be many other ROs inserted. Many HT detection techniques builds on extra ROs being inserted so the presence of ROs are not strange. If a malicious party do not want the possibility of being discovered by this warning the LUT-based RO can be exchanged with another design, for example one of the designs mentioned by

Sugawara [14]. Those ROs are designed to not give this type of warnings.

6.1.2 Simulation Discovery

The code could be tested through running simulations and measuring the code coverage. The trigger, consisting of an if statement checking the input for the correct activation input, would then be highlighted as not covered and would most likely be added as a test case. If no simulation delay is added to the RO the whole simulation will stop as soon as the RO is activated. The RO's output is supposed to be connected to an unused part of the circuit and in the circuit used to test the HT the output is chosen to be an unused segment selector. Since this signal has no usage for the circuit there is no reason for a tester to add this particular signal to the simulation. Checking all signals in a circuit is typically not done since there would be too many. Therefore, there is little chance that the RO signal would be noticed as an oscillating signal even if the HT is accidentally activated. If the unused signals for some reason were added to the simulation window, then the HT output would display only a 0 for as long as the HT is not activated. Also, if it was activated, it only oscillates during less than three seconds, until it is deactivated again. If the small chance of the HT being active and added to the simulation window occurs, the presence could still be explained by the multiple ways a RO can be used for non malicious purposes in a circuit, for example being part of a detection technique.

6.2 Possible Protection

As shown by the measuring setup described earlier, it is enough to only measure the power consumption of the connected power supply to the board in order to get the changes in power consumption of the specific RO. Once the HT have been inserted, the malicious party would therefore not have to keep track of which RO was modified in order to know where on the circuit one should measure to read the key.

Although protection from this kind of attack might seem as simple as keeping the device secure enough that no one gets physical access to it, there are possible ways to get around it. Without physical access, it would not be possible to measure the power side-channel with an oscilloscope. However, many systems also have built in power, voltage or current sensors, and if those are present the problem of extracting power consumption becomes hacking into those sensors. If the internal sensors could be reached, then this kind of attack would not even need physical access, making it much harder to protect from.

7.1 Conclusion

In conclusion, this thesis has presented a way to design a HT to exploit ROs in order to leak secret information. The different measurements made shows that it was possible to leak an encryption key through power side-channels, using a RO of as few as five LUTs in a loop. This was concerning results, since an extra RO of 5 LUTs, or one modified RO of adaptable length could be very hard to discover in a big system with thousands of ROs. This makes the HT very effective.

It was also stated that the measurements required to gather the information leaked by the HT was very simple. It was only required to connect an oscilloscope to the power pins of the FPGA, and it was not even required to know where in the circuit the HT was inserted. The other option mentioned was to hack the internal power sensors of the FPGA board to get the information without having physical access.

The HT designed was shown to be easy to use, having many ways into a system and could also be very hard to discover. The results are concerning for the security of systems storing secret information and how to prevent this kind of attack should be examined.

7.2 Future Work

One thing that should be researched further is how to protect circuits from this type of attack. It could either be techniques for preventing this HT from being inserted in the first place, or ways of detecting it once it is there.

Since ROs are so sensitive to surrounding electronics it might be possible to leak the encryption key using cross talk. It would therefore be interesting to examine if it is enough to place the RO in the vicinity of the sensitive signal instead of in direct connection with it. Especially for implementations, like the AES module used, where the design is made to have constant side-channels it could be examined how much leakage one would get with cross-talk. If that is possible then the HT would be even easier to insert and the threat even bigger.

Another example of future work could be to create a number of HTs and example circuits that can be combined with each other to create a system of HT Benchmarks. This system of Benchmarks would help evaluate different techniques

developed for detecting HTs, and would make comparison between the effectiveness of different detection techniques much easier.

Bibliography

- [1] *Chip-level Trojan Benchmarks*. URL: <https://trust-hub.org/#/benchmarks/chip-level-trojan> (visited on 05/11/2025).
- [2] Christian Krieg. “Reflections on trusting TrustHUB”. In: *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE. 2023, pp. 1–9.
- [3] Pengxiang Cheng. “Study of monitoring circuitry for ageing in FPGAs”. In: (2021).
- [4] Bicky Shakya *et al.* “Benchmarking of hardware trojans and maliciously affected circuits”. In: *Journal of Hardware and Systems Security* 1 (2017), pp. 85–102.
- [5] Samuel T. King *et al.* “Designing and Implementing Malicious Hardware.” In: *Leet* 8 (2008), pp. 1–8.
- [6] Jordan Robertson and Michael Riley. “The big hack: How china used a tiny chip to infiltrate us companies”. In: *Bloomberg Businessweek* 4.2018 (2018).
- [7] Lang Lin, Wayne Burleson, and Christof Paar. “MOLES: Malicious off-chip leakage enabled by side-channels”. In: *Proceedings of the 2009 international conference on computer-aided design*. 2009, pp. 117–122.
- [8] Mark Zhao and G Edward Suh. “FPGA-based remote power side-channel attacks”. In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, pp. 229–244.
- [9] Sanchita Mal-Sarkar *et al.* “Hardware trojan attacks in fpga devices: threat analysis and effective counter measures”. In: *Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI*. 2014, pp. 287–292.
- [10] Steve Trimberger. “Trusted design in FPGAs”. In: *Proceedings of the 44th annual Design Automation Conference*. 2007, pp. 5–8.

- [11] Kan Xiao *et al.* “Hardware trojans: Lessons learned after one decade of research”. In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 22.1 (2016), pp. 1–23.
- [12] Rajat Subhra Chakraborty, Seetharam Narasimhan, and Swarup Bhunia. “Hardware Trojan: Threats and emerging solutions”. In: *2009 IEEE International high level design validation and test workshop*. IEEE. 2009, pp. 166–171.
- [13] S Deepthi *et al.* “Hardware trojan detection using ring oscillator”. In: *2021 6th International conference on communication and electronics systems (ICCES)*. IEEE. 2021, pp. 362–368.
- [14] Takeshi Sugawara *et al.* “Oscillator without a combinatorial loop and its threat to FPGA in data centre”. In: *Electronics Letters* 55.11 (2019), pp. 640–642.

```

module ring_oscillator(
    input wire clk,
    input wire reset,
    input wire activate,
    input wire [255:0] key,
    output wire ring_out
);

parameter N=5; //How many LUT there will be in the RO
wire [N:0] ro_sig;
reg key_bit_reg, key_bit_new;
reg state_reg, state_new;
reg [19:0] count_clock_reg, count_clock_new;
reg [7:0] count_key_bits_reg, count_key_bits_new;

//Generate N-1 LUT connected together
generate genvar i;
    for (i=0; i<N-1; i=i+1) begin: generate_ro
        LUT2 #(
            .INIT(4'b1000)
        ) inst (
            .O(ro_sig[i+1]),
            .I0(key_bit_reg),
            .I1(ro_sig[i])
        );
    end
endgenerate

//Last LUT to invert the output
LUT2 #(
    .INIT(4'b0010)
) inst_5 (
    .O(ro_sig[N]),
    .I0(key_bit_reg),

```

```

        .I1(ro_sig[N-1])
    );

    //Connecting the LUTs together in a loop
    assign ro_sig[0] = ro_sig[N];
    assign ring_out = ro_sig[N];

    always @ (posedge clk or negedge reset)
    begin: reg_update
        if (!reset)
            begin
                key_bit_reg <= 0;
                state_reg <= 2'h0;
                count_clock_reg <= 20'h0;
                count_key_bits_reg <= 8'h0;
            end
        else
            begin
                state_reg <= state_new;
                count_clock_reg <= count_clock_new;
                count_key_bits_reg <= count_key_bits_new;
                key_bit_reg <= key_bit_new;
            end
        end
    end // reg_update

    always @*
    begin: output_key
        state_new <= state_reg;
        count_clock_new <= count_clock_reg;
        count_key_bits_new <= count_key_bits_reg;
        key_bit_new <= key_bit_reg;
        case(state_reg)
            1'b0:
                begin
                    if(activate == 1'b1) begin
                        state_new <= 2'b1;
                        key_bit_new <= key[255-count_key_bits_reg];
                        count_key_bits_new <= count_key_bits_reg +1;
                    end
                end
            1'b1:
                begin
                    count_clock_new <= count_clock_reg +1;
                    if(count_clock_reg == 20'hffff) begin
                        if(count_key_bits_reg == 8'b11111111) begin

```

```
        count_key_bits_new <= 8'b0;
        state_new <= 2'b0;
        key_bit_new <= 0;
    end else begin
        count_key_bits_new<=count_key_bits_reg+1;
        key_bit_new<=key[255-count_key_bits_reg];
    end
end
end
endcase
end //Output_key
endmodule
```

Appendix **B**

Activation Code

```
always @* //ring_oscillator
  begin: activate_oscillator
    if(block==128'h00112233_44556677_8899aabb_ccddeeff) begin
      activate_ro = 1;
    end else begin
      activate_ro = 0;
    end
  end
end
```



LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2025-1058
<http://www.eit.lth.se>