Project Description Form (PDF): MSc Information Security

How to complete this form

This form should be completed during the second term. Every student must meet their supervisor at regular intervals during the term to discuss the scope of the project and the initial literature review. Drafts of the form and the literature review should be submitted to the project supervisor for review during the term, as plans for the project evolve. Apart from completing this front page, text should be provided under each of the four headings given after the declaration section below.

An initial project literature review should be prepared, again in consultation with your supervisor. This form should use the standard project front page. If at any stage the project deviates significantly from the description given, then the student must discuss this with their project supervisor and, if necessary, complete a revised form.

Royal Holloway encourages research of highest quality by ensuring that research ethics and good practices are followed. If the proposed work raises any ethical issues, ethical approval must be sought in advance.

Student checklist

- Two copies of this form must be completed, signed and submitted to your project supervisor for signature.
- A copy of the project literature review should be attached to both copies of this form.

Student and project details

Name	Luke Anthony Atherton
Student number	100905113
MSc track (if applicable)	MSc Information Security
Email address	Luke.atherton.2018@live.rhul.ac.uk
Supervisor name	Prof Konstantinos Markantonakis
Provisional project title	Securing Firmware for Embedded Systems: Binding Firmware and Hardware

Ethics (to be completed jointly by the student and the supervisor)

Question	Delete/tick as appropriate	
1. Will the study be covert in any way?	Yes ☐	No ☒
2. Will resulting data be used for purposes outside this study?	Yes ☐	No ☒
3. Are you working with a vulnerable population?	Yes ☐	No ☒
4. Is it possible that your study will cause distress or harm to participants?	Yes ☐	No ☒
If the answer to any of the above questions is 'YES', the supervisor should contact the MSc Project Director and arrange for Departmental/College ethical approval.		

Declarations and signatures

Student declaration: I declare that I have read and understood the MSc Project Handbook, in particular the sections on referencing and plagiarism. I declare that the contents of this Project Description Form are all my own work, and that I have acknowledged all quotations from published or unpublished work of other people. I also declare that the proposed work will not raise any ethical issues.
Signature: **Date**:

Supervisor declaration: I approve the attached project plan and literature review. I agree that the proposed project topic meets the requirements of the MSc track (if specified). I also agree that the project does not raise any ethical issues.
Signature: **Date**:

Statement of Objectives

What I plan to achieve

- Produce a clear definition of "firmware"
- Review real-world reasons justifying the need for further research into secure firmware, such as:
  - Intellectual property counterfeiting and its financial effects
  - Malicious code injection
  - Physical attacks
- Review existing solutions for binding hardware and software
- Understand the phaesible alterations/additions which can be made to widely used embedded system chips such as FPGAs and ARM Cortex-M Series. Understand the primitives which will be used for creating suggestions and improvements such as software control-flow graphs and PUFs.
- Provide suggestions for increasing firmware security though hardware to software binding including:
  - Hardware-software binding through encryption wrappers:
    - Whole program encryption and runtime decryption
    - Program block encryption and decryption through the use of control-flow graphs
  - Binding through the use of cryptographically protected control-flow graphs
  - "Virtual binding" through always on remote attestation
  - Secure boot sequence used to decrypt firmware
- Present remarks on secure firmware transfer using the suggested methods

Why I have chosen the project

As a software developer, neat and elegant solutions to problems appeal to me greatly. The computing constraints of embedded systems present the need for neat, elegant and efficient solutions. This cross-over is why the project interests me. To further my interest, embedded systems in the form of Internet of Things (IoT) and Operational Technology (OT) are becoming increasingly used and therefore their security is vital for consumers, businesses and the national critical infrastructure.

The project itself is of interest due to the lightweight use of encryption, examination and development of protocols and gaining a deeper understanding of embedded systems.

Methodology

How will I achieve the objectives of the above list

- Researching the following topics through journal articles, conference proceedings and books:
  - Binding of hardware and software
  - Secure software execution
  - Common attacks on embedded systems
  - Commercial cost of IP theft (through counterfeiting and other methods)
- Research, understand and critique primitives such as control-flow graphs and PUFs in line with my overall goal.
- Gain a deep enough understanding of FPGA and ARM Cortex-M to assess what is possible when it comes to hardware adjustments for the sake of security.
- Experiment with implementing basic hardware to software binding, this will include:
  - Gaining access to either an FGPA or and ARM Cortex-M based development board
  - Understanding the processes behind altering hardware, writing software, provisioning software and testing software
  - Assess existing solutions for hardware - software binding for applicability for testing (is it possible, how much work would it be, what advantage would I get out of implementing it)
- Consider the design of a novel solution
  - If solution requires remote interaction (through a server) a basic understanding and implementation of a server is required, this will be implemented either through a laptop acting as a server or running up a cloud instance.

- Complete design and (if possible) implement a novel solution/s
- Assess novel solution/s and investigate the firmware transfer process required

What is my strategy for getting started

- Using the objectives set and predefined project chapters review literature which has already been reviewed for the preliminary literature review, review literature which has already been gathered but not included in the literature review and review books for content which is applicable for this project.
- Gain an understanding of technologies used (FGPA and/or ARM), perhaps gain access to a corresponding developer board and write simple application.
- Develop knowledge of control-flow graphs and the method of creating them.
- Gain a deeper understanding of C, assembly language and C compilers.

Work plan

December 2018 - January 2019

Meet with project supervisor to begin discussions. Discuss potential project directions with supervisor. Gather and assess reading material followed by studying highly applicable material. Refine ideas for project title.

February 2019

Meet with project supervisor to discuss project topic choice, once topic is decided commence literature review of literature already possessed and any further work which comes to light during the literature review. Fill out Project Description Form (PDF) and formalise Preliminary Literature Review (PLR) before submitting to supervisor for review. Once changes suggested from review are made make final submission to supervisor on 1st March 2019.

March 2019

Begin work on understanding FPGA and ARM Cortex-M. Complete review-based chapters and submit to supervisor for review. Discussion with supervisor on learnings and thoughts so far.

April 2019

As this period will be used mostly for exam preparation project work will be not proceed at a high pace. Practical understanding of FPGA and/or ARM Cortex-M will be gained, including writing and implementing software for said systems. Notes will be made during this phase.

May 2019

As this period will also be used for exam preparation/partaking project work will not proceed at a high pace. Control-flow graphs and PUFs will be studied.

Notes will be made during this phase.

June 2019

Notes on understanding of hardware and primitives will be formalised into chapters. Existing solutions will be studied and compared, creating a chapter. This will be submitted to supervisor mid June 2019 (aiming for 14th June 2019). Begin work on producing novel method for binding hardware and firmware.

July 2019

Update chapters already submitted to supervisor with feedback provided. Meet with supervisor to discuss novel solution. Begin basic implementation of novel solution. Create chapter on firmware transfer. Submit first draft of project to supervisor at end of month (aim for 26th July 2019).

August 2019

Finalise implementation of solution. Discuss findings with supervisor and write up respective chapter for review (aim for 9th August 2019). Submit final draft to supervisor 16th August. Submit MSc Project 22nd August 2019.

Draft Table of Contents

1. Introduction
    (a) Usage of embedded systems
    (b) Definition of firmware
    (c) Types of attacks
    (d) Security and financial repercussions
2. Existing research
    (a) Secure software execution
    (b) Binding of hardware and software
    (c) Remote attestation
3. Processors used for embedded systems
    (a) Overview of the processors
    (b) FPGA special features and limitations relevant to binding of hardware and software
    (c) ARM Cortex-M special features and limitations relevant to binding of hardware and software
4. Primitives
    (a) Control-flow graphs
    (b) PUFs
    (c) Secured boot (through ROM or Trusted Zone)
5. Critique of existing solutions
    (a) Categorising solutions
    (b) Assess constraints of solutions
    (c) Determine strengths and weaknesses of solutions in comparisons to attacks and platforms
6. The novel solution

   (a) Founding principles
   (b) Hardware requirements
   (c) Programmer/compiler requirements and constraints
   (d) Description of solution

7. Implementation
   (a) Implementation steps
   (b) Implementation findings
   (c) Assessment of solution against attacks

8. Updating firmware using the novel solution
   (a) Key parties
   (b) Update mechanism

9. Remarks and Conclusion
   (a) Comparison with existing solutions
   (b) Conclusion

Contents

Chapter 1

Literature Review

1.1 Introduction

This section analyses and summarises contributing and related academic work applicable to this project.

It will be broken up into several sections: the first section "Subject-matter Surveys" will discuss works which describe the problems to be addressed and existing solutions to said problems, this section also included a subsection on FPGA security which makes useful reading as many embedded systems are FPGA-based. The second section "Attacks" contains a brief look at other physical attacks not addressed in the first section. The third section "Solutions" gives a deeper analysis of a handful of existing solutions, some of which directly address binding of software and hardware and some of which focus on the related subject of secure software execution. The fourth section "Primitives" provides a brief introduction into some of the founding principles used in many of the solutions already described and which will be heavily used in this project. Finally the conclusion will sum up the literature seen so far and describe its place in relation to this project.

1.2 Subject-matter Surveys

Many surveys on solutions which increase security for firmware/software in embedded systems have been completed. One such survey [1] focusses on existing mature solutions, while others [2], [3] and [4] provide a look at broader principles. All of these surveys also paint a picture of the attacks and threats which embedded systems face.

Other surveys exist on the technologies described in this project, one such survey is [5] which focuses on FPGA security.

1.2.1 Existing Solutions

[1] looks in to technologies designed to ascertain trust for embedded systems. They compare various technologies, some of which are mature and some in their infancy. Studied solutions include: Trusted Platform Module (TPM), Secure Elements(SE), hypervisors and virtualisation (e.g. Java Card and Intel's Trusted eXecution technology), Trusted Execution Environments (TEEs), Host Card Emulation (HCE) and Encryption Execution Environments (E3 - which has also been directly discussed in [6]). The paper sets out a series of criteria which solutions are tested against, including such criteria as "Centralised Control', where the trust technology is under the control of the issuer or the maintainer, and "Remote Attestation" where the trust technology provides assurance to remote verifiers that the system is running as expected. The paper goes on to describe each technology in a small amount of detail and populates a matrix of technologies vs. criteria.

In a survey of anti-tamper technologies [2], a series of *cracking* threats and software and hardware protection mechanisms are described, many of which apply to embedded systems. Such threats include:

- Reverse engineering, achieved through a variety of methods including gaining an understanding of software or simply *code lifting* where sections of code are re-used without understanding of their functionality;
- Violating code integrity, where code in injected into a running program to make it carry out illegal actions outside of the desired control-flow of the program.

Hardware solutions described include: using a trusted processor used to secure the boot of the system, using hardware to decrypt encrypted software from the hard-drive and RAM, using a hardware *token* which is required to be present for the software to run.

The general positives of using hardware solutions include: using a complex CPU which is difficult to defeat while not redirecting resource from the processor used for standard operation, it is more costly to repeat attacks on hardware than it is for software (physical access is required each time) and secure hardware can also control which peripherals can be connected to the system and which software (signatures) can be allowed to run. The negatives of using hardware solutions include: secure data traversing the secure to non-secure boundary needs to be encrypted (which creates an additional overhead for the main processor), hardware solutions tend to be inflexible, less secure than commonly assumed and additional components can add to the cost of manufacture.

Software solutions described include:

- Encryption wrappers, where all or just the critical portions of software are stored in a ciphertext form and dynamically decrypted. The value of this is that the attacker will not see all of the source program at the same time, however they can piece it together through snapshots or simply learn the

encryption key/s. This paper does not cite any references for the subject of encryption wrappers;

- Code obfuscation, where the look of the code is adjusted to make it not easily readable or understandable by the attacker but performs in the same manner;
- Software watermarking and fingerprinting, which can be used for proof of ownership or authorship and for finding the source of leak of the software;
- Guarding, which is the act of adding code purely to perform anti-tamper functionality. An example of guarding is comparing checksums of running code to expected value and performing certain actions if they do not match. It is recommended that guarding is implemented automatically rather than manually as providing sufficient coverage is a complex task. It is also noted that a guard should not react immediately so as to not reveal the point in the code which triggered it.

The paper also describes a series of steps to take when using anti-tamper technology as put forward by the "Defence Acquisition Guidebook" authored by the U.S. government.

A similar survey [3] covers three types of attacks: reverse engineering, software piracy and tampering which it describes as "malicious host attacks". To defend against such attacks the paper states three corresponding defences: code obfuscation (as well as anti-disassembly and anti-debugging measures), watermarking and tamper-proofing. The authors note that they could not find a wealth of information on tamper-proofing at the time of writing (2002) but they do draw an interesting parallel with the anti-tamper mechanisms used in computer viruses.

1.2.2 Defence against fault injection

[4] ran high-coverage tests for security protections against fault injection attacks. It describes 17 different countermeasures, including: countermeasures protecting the data layer, combinations of data protection methods, countermeasures protecting control flow layer, combinations of control flow protection methods and combinations of data and control flow protection. To test these methods the authors produced a high number of simulated fault injections on a simulator of an ARM-Cortex-M3 processor running a benchmark application representing a bank card.

The experiments found that a combination of redundant condition checks (such as data duplication) and source and destination IDs reached the best coverage with moderate performance overhead. They also found that simple ID-based inter-block control checking were able to outperform more sophisticated (and complex) methods such as Control-Flow Checking by Software Signatures (CFCSS) as seen in [7] and Assertions for Control-Flow Checking (ACFC) seen in [8].

1.2.3 FPGA Security

[5] provides an excellent high-coverage survey on FPGA security, its contents include the background of FPGAs, attacks associated with FPGAs, defences for protecting FPGA implementations (existing at the time and ongoing research) and many more.

1.3 Attacks

The following are some examples of analysis of threats, all of which are aimed towards disrupting the flow of software, the likes of which are the focus secure software execution solutions.

[9] describes attacks which can be used to break instruction-level countermeasures. This paper discusses various attack countermeasures and how these are broken. The only countermeasures broken in this paper are algorithm-level and instruction-level (both of which are mostly redundancy-based). This paper suggests that a purely software-based countermeasure could be a futile defence.

[10] finds that physical faults can be injected in a non-random manner and in a low cost environment, this contradicts assumptions made in many of the examined solutions that physical attacks are too costly. It finds that instruction-skipping attacks create a vulnerability to skipped-instruction errors (which, in my opinion, drives the motivation behind control-flow monitoring right down to the intra-block level).

[11] Provides further details on side-channel attacks, as well as a brief description of the security concerns associated with FPGAs. This book could provide a good reference point.

1.4 Solutions

A myriad of creative technical solutions have been put forward which address the problems already discussed. They can be placed in to one of two categories - binding hardware ([6], [12], [13] and [14]) and software or secure software execution([7], [15], [16] and [17]). [18] and [19] are cross-overs of the two approaches.

1.4.1 Binding Hardware and Software

Hardware software binding is a technique where hardware and software a co-designed in such a way that software needs to be tailored to run on an individual instance of hardware. The same principle works the other way in that a individual piece of hardware will not execute software unless is it specifically tailored to it.

The first piece of work we consider is [6]. The problem the paper aims to address is device counterfeiting. An example of the requirement for binding of hardware and software is for Graphics Processing Units (GPUs), where GPUs are fabricated and then tested on their operating performance and subsequently

graded. Once graded, the GPUs are loaded with firmware which controls their voltage and clockspeed. The paper states that firmware aimed towards the superior graded GPUs could be installed on lesser graded GPUs which would then be sold on as superior GPUs.

The attacker identified is one which has several special attributes: they have physical access the the device, access to the device storage where they can read and copy the entire contents of memory, they are able to use hardware which has been built to the exact specifications as the original hardware and they can "read and copy any data which is loaded onto any of the buses which make up the embedded system". The attacker's aim is to either create a counterfeit platform which performs and functions in the same manner as the original or to install software retrieved from the legitimate product onto different (counterfeit) hardware.

The method described uses a function applied to either previous contents of memory or a randomly generated number to produce a mask which is applied to the program instructions residing in memory. The intention is that the CPU unmasks the contents as part of the execution process prior to actually carrying out the operation. The paper discusses the options for the mask-creating function, suggesting the use of hash-functions, block ciphers or PUFs before finally selecting PUFs due to their intrinsic nature. The act of masking the software has been undecided in this paper, which suggests that either the software is masked prior to loading or is masked during the loading process. The paper's author describes the provisioning process in a further paper [20].

The second piece of work we consider is [12] where the goal is to "protect against intellectual property (IP) extraction or modification on embedded devices without dedicated security mechanisms". In this paper the attacker is aiming to extract IP (in the form of software or secrets) stored on the device. The attacker may use this information in any of the following ways: they may implement the extracted information on counterfeit devices, they may modify the software or data to remove licensing restrictions or unlock premium features, they may downgrade to earlier firmware versions in order exploit previous vulnerabilities, they may wish to alter firmware to capture valuable data such as password, change output data such as readings on smart meters or reveal secrets such as cryptographic keys.

In this paper the attacker is assumed to have physical access to the device, can read the contents of external memory and can inspect and modify on-chip memory values. The assumed limitations placed on the attacker by the paper are that the attacker cannot change the code of the boot-loader as it is stored in a masked read-only memory (ROM), they cannot replace the ROM chip with one with a boot-loader under the attacker's control as the ROM chip would be heavily integrated on a system-on-a-chip (SoC) so would require skill levels outside of those expected of the attacker and finally the attacker cannot read the start-up values of the on-chip SRAM during start-up which are protected by the boot-loader and are erased once read by the boot-loader.

The method described heavily utilises PUFs created using the SRAM start-up values to derive a key used to decrypt the firmware. The firmware is de-

crypted by the the boot-loader before being loaded and executed. The system was implemented on a SoC platform using a two-stage bootloader (u-boot).

This paper provides an extensive review of SRAM PUFs for ARM Cortex-M and Pandaboard's IMAP and includes description of its Fuzzy Extractor design.

The third piece of work [18] has not been published in a well-established journal however the authors have been invited to present their findings in [21]. Here the problem of injection of malicious code is also addressed, as well as prevention of code reverse-engineering. This paper uses secure execution to bind hardware to software.

The attacker identified has physical access to the processor and peripheral connections and that they can read out contents of memory or registers. They are also assumed to be able to place arbitrary data into the main memory of the processor (either locally or remotely). Attacks comprising denial-of-service (DoS)achieved by, for example, injection of random invalid instructions and hardware side-channel attacks have not been addressed.

The method described has been labelled "Secure Execution PUF-based Processor" or SEPP. The operating principle of SEPP is the encryption of basic blocks (which have exactly one entry point and one exit point) which make up programs. The blocks are encrypted using a symmetric cipher in CTR mode with the parameters set in relation to instruction location within a block and the block's location within memory. The key used for this encryption is set by the user. SEPP utilises a ring-operator (RO) PUF to create a new key used to encrypt the users key. The decryption module is included in the instruction fetch stage of the processor's pipeline and makes use of first-in, first-out (FIFO) buffer to store encryption pads before they are needed by the processor (therefore making use of spare time provided by instructions which take more than one processor cycle). This system also implements u-boot as a bootloader which has been modified to provide the functions of the security kernel. It appears that due to the nature of this method (the device tailoring the software to itself), it does not prevent malices uploading of a new program to device which the device then processes.

The fourth method identified in [19] had identified illegitimate reproduction as a problem that requires a solution. It also identifies modification of software to bypass the need for purchasing a license for particular features as another attack scenario. Here the attacker has the ability to read and modify the content of external memory such as flash memory or RAM, they can also do the same with internal memory including software with hard-coded secrets and cryptographic keys.

The method consists of four basic mechanisms: two check functions and two response functions. The first check function hashes the native program code and compares this to the current running code. The second check function uses a SRAM-derived PUF to measure the authenticity of the device. If these functions indicate that either the software is not in its intended state or is running on the incorrect device the first response function adjusts the flow of the program to move in a random manner and the second response corrupts the program's execution stack. Both response functions are designed to cause a malfunction

in the program. One has to question the safety of having a program jump to a random block.

The fifth method described in [13] is developed to protect against IP theft or reverse engineering. This paper does not describe the attacker but makes some assumptions that they will not be able to access the PUF-based key used for encryption as it is internal to the FPGA. This method uses an obfuscated secure ROM to start the boot process, checking and running the integrity kernel which decrypts and runs the software using the PUF-based key. The problem with this solution is the reliance on an obfuscated ROM, once there is an understanding of a ROM could it be possible that malicious software could be run if it is accepted by the ROM?

An honourable mention should be made for [14] which was published in 2006 and led the way in using PUFs to secure software and also clearly describes the enrolment process with defined message exchanges.

1.4.2 Secure execution

Secure software execution (or control-flow security/integrity) has the goal of preventing the desired results of attacks against program control flow, which aim to use physical attacks to make running programs execution jump to blocks which should not be running (e.g. an administration function).

The first solution [7] raises the point that most research on fault-attacks has been aimed towards cryptographic functions which result in gaining knowledge of the secret key. This paper takes this further by considering the modification of games consoles to make them skip the function which checks the validity of loaded software. The paper focuses on securing against fault-attacks.

The method mainly utilises control-flow integrity to solve the stated problem. The basic principle behind control-flow integrity is the understanding of the basic blocks of a making up a program. The blocks will flow into one another control-flow instructions. This flow can be described as the control-flow graph (CFG) and a correctly functioning program will abide by this graph. The a signature of the program flow is created and compared against the expected value according to the CFG. The solution provides assistance to C programmers in that it automatically inserts signature updates, although programmers can also insert them manually for critical sections of the program. This functionality has been provided via the editing of LLVM compiler. If using assembly code the programmer must manually insert signature updates whenever branches, loops and function calls are encountered. The control flow signatures are calculated through a recursive disassembling approach. Important principles introduced in this paper (along the same lines as CFG) are generalized path signature analysis (GPSA) and continuous-signature monitoring (CSM). This paper does require modifications to be made to Cortex-M3 architecture.

The second paper [15] (ConFirm) states that "given the critical role of firmware, implementation of effective security controls against firmware malicious actions is essential", having read the various prior examples seen we can safely agree with this.

The attacker is assumed to have the ability to inject and execute malicious code, or call existing functions not abiding by the control-flow graph. These attacks are assumed to be possible either on-line or off-line (which would require a device reboot after uploading of malicious firmware image) and depending on the design of the device the firmware alterations could be achieved locally or remotely.

The method employed revolves around making use of hardware performance counters (HPCs) which count various types of events. The HPCs are utilized in conjunction with a bootloader (which sets checkpoints, initialises an HPC handler and contains a database of valid HPC-based signatures). While the program is run, HPC values are checked once checkpoints are reached (checkpoints are placed at the beginning and end of each basic block, as well as one randomly inserted between). The checkpoints actually redirect the control flow to the ConFirm core module to compare the HPC value with those stored in the database containing valid values. If the check fails ConFirm will report a deviation, which could be used to run a fault sequence, such as "rebooting the system, generating an alarm and disabling the device".

The third paper [16] approaches secure execution from a slightly different direction: remote attestation. It aims to provide remote attestation of an application's control flow path during operation. The paper excludes physical attacks and instead focusses on execution path attacks, they assume that the subject device features data execution prevention (DEP) and a secure trust anchor that provides an isolated measurement engine and can generate the fresh authenticated attestation report.

The method builds a hash of the path taken from node to node which is then reported to the verifier. Loops are dealt with in a novel manner to work around the issue posed due to the infinite hash available when the number of iterations of a loop is set dynamically.

The fourth paper [17] lists various attacks, ranging from buffer overflow attacks to physical attacks. Their method measures inter-procedural control flow, intra-procedural control flow and instruction stream integrity. Control flow monitoring is provided by an additional hardware element which tracks instruction addresses and compares them to known acceptable values stored in lookup tables. Instruction steam integrity monitoring utilises the lookup tables in addition to corresponding hash values of the basic clock. If a violation is discovered it is reported to the processor which should then terminate execution of the current program. Details of the violation are included in the report to the processor to enable a finer-grained view of the violation.

1.5 Primitives

1.5.1 Control-flow Graphs

[8] considers the use of control flow checking for accidental program flow changes, but still presents the basic principle. It describes basic blocks, control-flow

between blocks, how these make up program graphs and finally how these graphs can be used to indicate control-flow error. The paper presents two existing error detection methods but finds faults in both so presents a novel solution addressing the previous solutions' shortcomings.

Chapter 9 of [22] contains a wealth of information on data-flow and will provide a good basis for understanding both the essence of data(control)-flow and how compilers use them for code optimisation. This chapter should be referenced in order to gain a meaningful understanding of control-flow.

1.5.2 PUFs

A huge amount of prior research has been undertaken into PUFs and their application towards security in embedded systems. One very good overview is the PhD thesis [23], which provides a thorough examination into most aspects of PUFs including the types, analysis of each type in terms of uniqueness and reproducibility and uses including entity-authentication and key generation.

Much of the literature ([6], [12], [18], [19], [13] and [14]) already described explain the use of PUFs and their reasoning behind their choice in PUFs.

1.6 Conclusion

In this literature review we have seen the reasons for security of firmware of embedded devices. We then saw reviews of existing solutions and introductions to primitives used. After a light look at physical attacks we then provided an in depth review of binding of hardware and software solutions and secure software executions. Finally we looked at a couple of primitives which will be useful to this project.

It can be seen through the wealth of literature available on many of the subjects that this project will address that the focus of the project is an important one, and one which still has not been fully addressed.

Bibliography

[1] C. Shepherd, G. Arfaoui, I. Gurulian, R. P. Lee, K. Markantonakis, R. N. Akram, D. Sauveron, and E. Conchon, "Secure and Trusted Execution: Past, Present, and Future - A Critical Review in the Context of the Internet of Things and Cyber-Physical Systems," in *2016 IEEE Trustcom/BigDataSE/ISPA*, IEEE, Aug. 2016, pp. 168–177, ISBN: 978-1-5090-3205-1. DOI: 10.1109/TrustCom.2016.0060. [Online]. Available: http://ieeexplore.ieee.org/document/7846943/.

[2] E. D. Bryant, M. J. Atallah, M. R. Stytz, M. J. Atallah, E. D. Bryant, and M. R. Stytz, "A Survey of Anti-Tamper Technologies," *The Journal of Defense Software Engineering*, no. November, pp. 12–16, 2004.

[3] C. S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation - Tools for software protection," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 735–746, 2002, ISSN: 00985589. DOI: 10.1109/TSE.2002.1027797.

[4] N. Theissing, D. Merli, M. Smola, F. Stumpf, and G. Sigl, "Comprehensive Analysis of Software Countermeasures against Fault Attacks," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pp. 404–409, 2013, ISSN: 15301591. DOI: 10.7873/DATE.2013.092. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6513538.

[5] S. Drimer, "Volatile FPGA design security – a survey," *University of Cambridge*, pp. 1–51, 2008. [Online]. Available: http://www.cl.cam.ac.uk/%7B~%7Dsd410/papers/fpga%7B%5C_%7Dsecurity.pdf.

[6] R. P. Lee, K. Markantonakis, and R. N. Akram, "Binding Hardware and Software to Prevent Firmware Modification and Device Counterfeiting," *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security - CPSS '16*, pp. 70–81, 2016. DOI: 10.1145/2899015.2899029. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2899015.2899029.

[7] M. Werner, E. Wenger, and S. Mangard, "Protecting the Control Flow of Embedded Processors against Fault Attacks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9514, 2016, pp. 161–176, ISBN:

978-3-642-37287-2. DOI: 10.1007/978-3-319-31271-2_10. arXiv: 9780201398298. [Online]. Available: http://link.springer.com/10.1007/978-3-319-31271-2%7B%5C_%7D10.

[8] O. Goloubeva, M. Rebaudengo, M. Sonza Reorda, and M. Violante, "Soft-error detection using control flow assertions," in *Proceedings. 16th IEEE Symposium on Computer Arithmetic*, vol. 16, IEEE Comput. Soc, Nov. 2003, pp. 581–588, ISBN: 0-7695-2042-1. DOI: 10.1109/DFTVS.2003.1250158. [Online]. Available: http://ieeexplore.ieee.org/document/1250158/.

[9] B. Yuce, N. F. Ghalaty, H. Santapuri, C. Deshpande, C. Patrick, and P. Schaumont, "Software Fault Resistance is Futile: Effective Single-Glitch Attacks," *Proceedings - 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016*, pp. 47–58, 2016. DOI: 10.1109/FDTC.2016.21.

[10] M. S. Kelly, K. Mayes, and J. F. Walker, "Characterising a CPU fault attack model via run-time data analysis," in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, May 2017, pp. 79–84, ISBN: 978-1-5386-3929-0. DOI: 10.1109/HST.2017.7951802. [Online]. Available: http://ieeexplore.ieee.org/document/7951802/.

[11] C. H. Gebotys, *Security in embedded devices*, ser. Embedded systems. New York ; London: Springer, 2010, ISBN: 144191529x.

[12] A. Schaller, T. Arul, V. Van Der Leest, and S. Katzenbeisser, "Lightweight anti-counterfeiting solution for low-end commodity hardware using inherent PUFs," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8564 LNCS, pp. 83–100, 2014, ISSN: 16113349. DOI: 10.1007/978-3-319-08593-7_6.

[13] M. A. Gora, A. Maiti, and P. Schaumont, "A flexible design flow for software IP binding in FPGA," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 719–728, 2010, ISSN: 15513203. DOI: 10.1109/TII.2010.2068303.

[14] E. Simpson and P. Schaumont, "Offline HW / SW Authentication for Reconfigurable Platforms," *Cryptographic Hardware and Embedded Systems (CHES)*, pp. 1–13, 2006.

[15] X. Wang, C. Konstantinou, M. Maniatakos, and R. Karri, "ConFirm: Detecting firmware modifications in embedded systems using Hardware Performance Counters," *2015 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2015*, pp. 544–551, 2016, ISSN: 1933-7760. DOI: 10.1109/ICCAD.2015.7372617.

[16] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-FLAT: Control-Flow Attestation for Embedded Systems Software," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, New York, New York, USA: ACM Press, 2016, pp. 743–754, ISBN: 9781450341394. DOI: 10.1145/2976749.2978358. arXiv: 1605.07763. [Online]. Available: http://arxiv.org/abs/1605.07763%20http://dl.acm.org/citation.cfm?doid=2976749.2978358.

[17] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha, "Hardware-Assisted Run-Time Monitoring for Secure Program Execution on Embedded Processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 12, pp. 1295–1308, Dec. 2006, ISSN: 1063-8210. DOI: 10.1109/TVLSI.2006.887799. [Online]. Available: http://ieeexplore.ieee.org/document/4052340/.

[18] S. Kleber, F. Unterstein, M. Matousek, F. Kargl, F. Slomka, and M. Hiller, "Secure Execution Architecture based on PUF-driven Instruction Level Code Encryption," *Cryptology ePrint Archive, Report 2015/651*, 2015. DOI: cr.org/2015/651.

[19] F. Kohnhäuser, A. Schaller, and S. Katzenbeisser, "PUF-Based Software Protection for Low-End Embedded Devices," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9229, 2015, pp. 3–21, ISBN: 9783319228457. DOI: 10.1007/978-3-319-22846-4_1. arXiv: arXiv:1506.07739v2. [Online]. Available: http://link.springer.com/10.1007/978-3-319-22846-4%7B%5C_%7D1.

[20] R. P. Lee, K. Markantonakis, and R. N. Akram, "Provisioning Software with Hardware-Software Binding," in *Proceedings of the 12th International Conference on Availability, Reliability and Security - ARES '17*, New York, New York, USA: ACM Press, 2017, pp. 1–9, ISBN: 9781450352574. DOI: 10.1145/3098954.3103158. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3098954.3103158.

[21] S. Kleber, F. Unterstein, M. Matousek, F. Kargl, F. Slomka, and M. Hiller, "Design of the Secure Execution PUF-based Processor (SEPP)," *Workshop on Trustworthy Manufacturing and Utilization of Secure Devices, TRUDEVICE 2015*, no. 2, pp. 1–5, 2015. DOI: http://dx.doi.org/10.18725/OPARU-3255.

[22] A. V. Aho, *Compilers : principles, techniques, and tools.* Second edi, ser. Pearson custom library. 2014, ISBN: 9781292024349.

[23] R. Maes, *Physically Unclonable Functions: Constructions, Properties and Applications (Fysisch onkloonbare functies: constructies, eigenschappen en toepassingen)*, August. 2012, ISBN: 9789460185618. [Online]. Available: https://lirias.kuleuven.be/handle/123456789/353455.