Internet of Things

Rajat Subhra Chakraborty
Jimson Mathew
Athanasios V. Vasilakos *Editors*

Security and Fault Tolerance in Internet of Things

Springer

Internet of Things

Technology, Communications and Computing

More information about this series at http://www.springer.com/series/11636

Rajat Subhra Chakraborty
Jimson Mathew · Athanasios V. Vasilakos
Editors

Security and Fault Tolerance in Internet of Things

Springer

Editors
Rajat Subhra Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur, West Bengal, India

Jimson Mathew
Department of Computer Science and Engineering
Indian Institute of Technology Patna
Patna, Bihar, India

Athanasios V. Vasilakos
Department of Computer Science, Electrical and Space Engineering
Luleå University of Technology
Skellefteå, Sweden

ISSN 2199-1073 ISSN 2199-1081 (electronic)
Internet of Things
ISBN 978-3-030-02806-0 ISBN 978-3-030-02807-7 (eBook)
https://doi.org/10.1007/978-3-030-02807-7

Library of Congress Control Number: 2018958936

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The aim of this book is to cover various aspects of security, privacy and reliability in Internet of things (IoT) and cyber-physical system design, analysis and testing. Security is a prime design objective in current computing system design, including embedded systems, cyber-physical systems and the IoT devices. Over the years, various techniques have been proposed highlighting security and fault-tolerant system design, analysis and testing. Since cyber-physical systems are typically resource-constrained, these techniques are usually developed keeping the low energy and/or low-power availability into perspective. However, due to continued device miniaturization and technology scaling, reliability in the presence of ever-increasing number of faults is emerging as a design challenge, and more importantly, new security threats have emerged. This is because incorporating security, fault tolerance for reliability with low-energy or low-power consumption are conflicting design objectives. From IoT security to design robustness and big data processing, this monograph offers a potpourri of different methodologies on this rapidly expanding frontier of the new technology, with chapters written by leading international industry practitioners and academic researchers. It is intended that the readers receive both these perspectives, which is important for such cutting-edge technical topics of discourse.

Chapter "Security and Trust Verification of IoT SoCs" looks at the trustworthiness of System-on-Chips (SoCs). In order to ensure the security of IoT devices, it is crucial to guarantee the trustworthiness of SoCs, because advances in modern VLSI miniaturization technologies imply more and more functionality are being packed into complex SoCs than ever before. Verifying the trust in SoCs is a major challenge due to their long and globally distributed supply chain. Malicious components can be inserted in different stages of the design cycle. These malicious functionalities work as a backdoor to severely affect the security of the design by giving control of the system to adversaries. The threat creates a critical need for designing new validation approaches that are capable of identifying hidden Hardware Trojan Horses (HTHs). Existing validation techniques cannot efficiently activate and detect HTHs since Trojans are designed to be passive most of the time and usually triggered using very rare events. For example, if an adversary wants to

hide a HTH in register-transfer level (RTL) designs, rare branches in the control flow would be an ideal choice to host Trojans. To tackle this issue, the authors introduce a Trojan activation technique that utilizes an effective combination of symbolic simulation as well as concrete execution to identify Trojans that are hidden under rare branches and assignments. The technique is scalable as it considers one path at a time instead of considering the whole design. It uses *Satisfiability Modulo Theories* (SMT) solvers to satisfactorily solve the path constraints in order to generate a valid test to explore a new path in the design. The exploration continues until all of the rare branches in the design are activated in the search for hidden Trojans.

Chapter "Low Cost Dual-Phase Watermark for Protecting CE Devices in IoT Framework" presents a novel low-cost dual-phase watermarking methodology during high-level synthesis (HLS) for hardware intellectual property (IP) protection. Robust vendor signature is embedded in two subsequent phases of high-level synthesis to form an integrated watermark. A dual-phase watermarking methodology that embeds a multivariable double-phase watermarking during high-level synthesis for application-specific IPs, while incurring zero delay and register overhead as well as minimal hardware overhead, is presented. The dual-phase watermarking approach yields average reduction of embedding cost of 6% (which includes average area reduction of 7% and average latency reduction of 4%) when compared to two recent HLS-based watermarking approaches for application-specific IPs. Additionally, the approach also achieves stronger proof of authorship compared to two recent HLS-based watermarking approaches.

Chapter "Secure Multicast Communication Techniques for IoT" discusses multicast security approaches mainly based on extending the DTLS protocol and its drawbacks. Also included are the major requirements of secure group communication and different secure multicast communication techniques emphasizing an interesting and effective approach called S-CPABE, based on attribute-based encryption (ABE).

Chapter "An Adaptable System-on-Chip Security Architecture for Internet of Things Applications" addresses emerging threats and security design constraints of architectures catering to IoT and automotive applications and their subsequent limitations. Then, it presents a novel, flexible and adaptable SoC security architecture that efficiently implements diverse security policies. The architecture and associated CAD flow enable "hardware patching", i.e., hardware security policy engines that can be seamlessly and securely upgraded infield to address unanticipated attacks or new security requirements. The chapter describes the implementation of (1) a centralized Reconfigurable Security Policy Engine (RSPE); (2) smart security wrappers and (3) Design-for-Debug (DfD) infrastructure interface as the building blocks of the architecture. The proposed framework provides a systematic approach to represent and synthesize diverse security policies. Through extensive analysis using representative SoC model, it is demonstrated that the proposed framework provides a high level of adaptability with minimal energy and performance overhead. Consequently, the architecture is highly suitable for devices operating under a tight boundary of energy and performance.

Homomorphic encryption constitutes a powerful cryptographic method that enables data aggregation in distributed applications over large datasets, such as storage of data on cloud, electronic voting, electronic wallets, secure auctions, lotteries and secret sharing. At the same time, as attack trends move towards the lower levels of the computation stack and new threats continue to emerge, the lack of trust in contemporary computing paradigms keeps increasing. Since homomorphic encryption helps preserve the confidentiality of sensitive information, it offers a powerful countermeasure against contemporary and future privacy threats, while allowing meaningful processing even though the data remains unreadable. Nevertheless, when homomorphic primitives are mapped to hardware circuits to improve performance, they become vulnerable to random faults and soft errors, since homomorphic operations are malleable by construction and do not provide any explicit assurance towards data integrity. Chapter "Lightweight Fault Tolerance for Secure Aggregation of Homomorphic Data" presents a fault tolerance methodology that protects homomorphic aggregation circuits through concurrent detection of random errors in homomorphic ALUs and encrypted values stored in the memory. The proposed approach establishes the theoretical foundations to extend residue numbering to additive homomorphic operations, which enables lightweight fault detection with detection rates of more than 99.98% for ALU operations and 100% for single bit-flips and clustered faults in memory values. Using an efficient modular reduction algorithm, the method incurs a runtime overhead between 3.6 and 8% and a small area cost.

In Chapter "An Approach to Integrating Security and Fault Tolerance Mechanisms into the Military IoT" the authors look at security and dependability of IoT implementation in military applications. This is an approach for integrating security techniques on the access layer and the fault-tolerant techniques at sensor nodes. Presented solutions for securing the military IoT network ensure strong node authentication within network clusters and securing data transmissions between sensor nodes (SN) and gateways with the use of COTS IoT platforms equipped with TPM modules. Fault diagnosis is based on the comparison method within network clusters. An experimental network called SFTN is also explored to demonstrate the approach.

Physically unclonable function (PUF) circuits are very promising lightweight hardware security primitives, which depend on manufacturing process variations, act as fingerprint generators of electronic devices such as integrated circuits (ICs). PUF circuits can be used as alternatives to computationally expensive cryptographic algorithms and protocols and are extremely suitable for application domains such as IoTs. However, they are vulnerable to injected faults that can help an adversary to launch sophisticated computational attacks on them, revealing secret information that can impact system security. Chapter "Fault-Tolerant Implementations of Physically Unclonable Functions on FPGA" describes fault-tolerant implementations of PUF circuits on FPGAs, through fault detection and fault recovery.

Chapter "Fault Tolerance in 3D-ICs" discusses fault tolerance in three-dimensional IC (3D ICs). Systems based on emerging technologies like Internet of things and beyond von Neumann architectures can be produced in large scale

only if they are resilient-aware, cost-effective and secure. The resilient and cost-effective solutions can be achieved by incorporating fault tolerance techniques at the architectural level of the system design is one of the plausible solutions. The choice of various fault tolerance techniques gives the designers a freedom to incorporate these in the early stage of the design and in turn leading to high yield and reliable architectures. Three-dimensional ICs with through-silicon via (TSV) is one of the emerging technologies consisting of vertical interlayer communication instead of long horizontal wires, results in the reduction of interconnect length and thus can improve the system performance. However, reliability and yield are major concerns that hinder resilient and cost-effective solutions for 3D-IC design. These can be addressed by incorporation of fault tolerance techniques.

Online detection of cyber-attacks on IoT devices is usually extremely challenging due to limited battery and computational resources available in these devices. An alternate approach is to shrink the attack surface in order to reduce the threat of attack. This would require that the device undergo more stringent security tests before deployment. Formal verification is a promising tool that can be used to not only detect potential vulnerabilities but also provide guarantees of security. Chapter "Formal Verification for Security in IoT Devices" reviews several security issues that plague IoT devices such as functional correctness of implementations, programming bugs, side-channel analysis and HTHs. In each of these cases, the chapter discusses state-of-the-art techniques that use formal verification tools to detect the vulnerability much before the device is deployed.

In Chapter "SENSE: Sketching Framework for Big Data Acceleration on Low Power Embedded Cores", the authors address the ever-growing IoT-based big data processing and cognitive computing on mobile and battery-operated devices. Big data processing on low-power embedded cores is challenging due to their limited communication bandwidth and on-chip storage. Additionally, IoT and cloud-based computing demand low overhead security kernel to avoid data breaches. In this chapter, authors present, "LESS", lightweight encryption using scalable sketching techniques for data reduction and encryption. LESS is a heterogeneous framework which consists of three important kernels: (1) a sketching module for data reduction; (2) an accelerator for efficient sketch recovery using scalable and parallel reconstruction architecture and (3) a host processor to perform postprocessing. LESS framework can reduce data up to 67% with 3.81 dB signal-to-reconstruction error rate (SRER). One of the critical challenges in big data processing on embedded hardware platforms is to reconstruct the sketched data in real-time with stringent constraints on error bounds and hardware resources. The authors also explore orthogonal matching pursuit (OMP) algorithm for sketch data recovery and demonstrate performance of LESS framework on face identification application.

We hope this monograph would be valuable to researchers and practitioners alike, not only as a collection of pointers to cutting-edge research in this exciting field, but also as a resource that would encourage the readers to explore newer techniques on their own. Cyber-physical systems and IoTs promise to revolutionize

our lives, but before their promise is fulfilled, many open problems in the context of their security and reliability need satisfactory solutions. It is our sincere wish that this book acts a small step towards that goal.

Kharagpur, India — Rajat Subhra Chakraborty
Patna, India — Jimson Mathew
Skellefteå, Sweden — Athanasios V. Vasilakos
August 2018

Contents

Security and Trust Verification of IoT SoCs

Alif Ahmed, Farimah Farahmandi, Yousef Iskander and Prabhat Mishra

Abstract System-on-Chips (SoCs) are widely used in designing Internet-of-Things (IoT) devices. In order to ensure the security of IoT devices, it is crucial to guarantee the trustworthiness of SoCs. Verifying the trust in SoCs is a major challenge due to their long and globally distributed supply chain. Malicious components can be inserted in different stages of the design cycle. These malicious functionalities work as a backdoor to severely affect the security of the design by giving control of the system to adversaries. The threat creates a critical need for designing new validation approaches that are capable of identifying hidden Trojans. Existing validation techniques cannot efficiently activate and detect Trojans since Trojans are designed to be inactive most of the time and triggered using very rare events. For example, if an adversary wants to hide a Trojan in register-transfer level (RTL) designs, rare branches would be an ideal choice to host Trojans. In this chapter, we introduce a Trojan activation technique that utilizes an effective combination of symbolic simulation as well as concrete execution to identify Trojans that are hidden under rare branches and assignments. The technique is scalable as it considers one path at a time instead of considering the whole design. It uses satisfiability modulo theories (SMT) solvers to solve the path constraints in order to generate a valid test to explore a new path in the design. The exploration continues until all of the rare branches in the design are activated in the search for hidden Trojans.

1 Introduction

Designing modern System-on-Chips (SoCs) is a very complex process since a diverse set of computational-, storage- and communication-related Intellectual Properties (IPs) are integrated. Due to the time-to-market constraints, designing and verifying

A. Ahmed (✉) · F. Farahmandi · P. Mishra
Department of Computer and Information Science and Engineering,
University of Florida, Gainesville, USA
e-mail: alifahmed@ufl.edu

Y. Iskander
Advanced Security Research Group, Cisco Systems, San Jose, USA

R. S. Chakraborty et al. (eds.), *Security and Fault Tolerance in Internet of Things*,
Internet of Things, https://doi.org/10.1007/978-3-030-02807-7_1

all of these IPs cannot happen in-house. Companies and countries across the world are involved in designing and fabricating an SoC. These SoCs are widely used today in designing Internet-of-Things (IoT) devices. In order to ensure the security and reliability of IoT devices, it is crucial to guarantee the trustworthiness of SoCs. Unfortunately, the long and globally distributed supply chain of SoCs makes it very difficult to ensure that an adversary has not threatened the security and trustworthiness of the design. An adversary can introduce a wide variety of security threats including IP piracy, reverse engineering (which can cause cloning, re-marketing, and counterfeiting), and inserting malicious modifications (hardware Trojans) in the design.

Hardware Trojans are extra components that modify the original functionality of the design by opening a backdoor for an adversary to take control of the whole system or cause unintentional effects. Hardware Trojans create a serious concern since an active Trojan can severely affect the trustworthiness of a design by leaking secret information, granting access to the critical functionality of the design as well as causing a denial of service. A typical Trojan has two main parts: (i) triggering logic to activate the Trojan and (ii) a payload for propagating the effect of a triggered Trojan to the attack target. To escape detection from conventional validation techniques, hardware Trojans are designed as small circuitry that is triggered using very rare internal events. Therefore, Trojans are dormant most of the time and this stealthy nature makes it extremely difficult to activate and detect them [46].

Hardware Trojans can be classified based on their trigger and payload mechanisms. The trigger condition of a Trojan can be designed as a combinational or a sequential circuit [8]. Combinational triggers are dependent on a rare event of trigger nodes. For example, a set of rare nodes (gates) can be used as a trigger by utilizing an 'and' gate. On the other hand, sequential triggers depend on the occurrence of a specific sequence of unusual events on the internal nodes. For example, counters can be used as Trojans when they trigger a malfunction by reaching a particular value. Sequential Trojans are more challenging to activate and detect.

Hardware Trojans can also be categorized based on their payload mechanisms which can be either digital or analog. Digital payloads can affect a memory location or change the logic values of internal nodes at a specific time. On the other hand, analog payloads modify physical characteristics of a design by changing its power, current, noise margin, and performance. Examples of such payloads are the insertion of bridging faults as well as insertion of cascaded buffers or extra circuitry that does not change the functionality of the design.

Researchers have proposed a wide variety of techniques based on simulation-based validation to activate and detect hardware Trojans. These techniques can be broadly divided into three categories: (i) logic testing approaches, (ii) formal methods, and (iii) hybrid approaches that combine both of these approaches as shown in Fig. 1. In the rest of this section, we discuss existing Trojan activation and detection techniques using these methods.

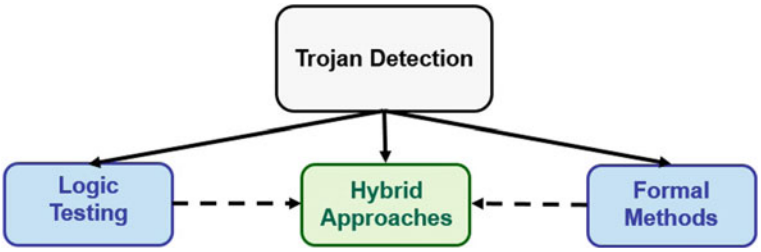


Fig. 1 Simulation-based validation approaches for Trojan detection and activation

1.1 Logic Testing

Logic testing is responsible for applying stimuli on primary inputs and observing the effect at the primary outputs or other observable points. To check whether a Trojan is hidden in the design, the Design-under-Test (DUT) is simulated using the generated tests and the values of the observable signals are compared with the expected values.

Hicks et al. [26] proposed an approach based on logic testing to detect unused circuits. The authors have shown that unused redundant components can be utilized by attackers to insert a malicious component in the design. The proposed approach tries to identify signal pairs that have the same sources and sinks by applying the verification tests to the design and simulate it. The algorithm deactivates the detected unused pairs to prevent hardware Trojans. However, the authors have shown in their subsequent work [45] that the algorithm may not be able to detect malicious circuits of other types.

FANCI approach is proposed by Waksman et al. [47] to identify rarely used logic circuits based on their boolean functionality and their control vales. The assumption is that hard-to-detect Trojans will be inserted into the nodes that rarely impacts the output functionality of the design. Therefore, they proposed an algorithm that constructs an approximate truth table for each of the internal signals to identify the effect of a particular signal on primary outputs. If a signal hardly affects primary output, that signal will be marked as a suspicious node. However, this method may mark even some safe nodes as suspicious since it may report 1–8% of the total signals in the design as potential malicious nodes. Therefore, using this method makes it hard to distinguish a Trojan-inserted design from a Trojan-free one. VeriTrust [49] is proposed to find uncovered circuits using conventional verification tests, and then it sets uncovered circuits as don't cares to identify redundant circuits as suspicious functions. Zhang et al. [50] (DeTrust) have introduced a type of Trojan whose triggers happen across different clock domains and FANCI cannot detect them and marks them as safe nodes. Another type of Trojan is introduced by DeTrust which VeriTrust cannot detect.

Automated Test Pattern Generation (ATPG) approaches are also employed to detect hidden Trojans. Full (100%) code coverage is targeted to detect hardware Trojans [51]. However, hardware Trojans may reside in the circuits that are tested for 100% code coverage. N-detect full scan ATPG is used to identify the nodes

whose functionality is hard to excite/propagate to observable points [5]. A guided equivalence checking as well as region isolation approaches are applied on hard-to-detect signals to localize the potential malfunctions. However, the strength of this approach is dependent on the capability of SAT solvers. SAT solvers may fail when encountering large and complex designs. A statistical test generation approach (MERO) has been proposed by Chakraborty et al. [9] to activate each of the rare nodes for at least N times in hoping for Trojan activation. The goal of this approach is to increase the probability of Hardware Trojans' trigger activation.

1.2 Formal Methods

There are several Trojan detection and localization techniques that rely on formal methods such as satisfiability solvers, symbolic algebra, and model checking to generate efficient tests to activate Trojans. Model checkers are used to check a design against its specification. Specification properties are primarily the expected behaviors of the design. These properties can be modeled as a collection of Linear Time Temporal Logics (LTL) [37]. The design is modeled as a set of states. To formally verify the safe behavior of the design, each property is checked through all possible states of the design implementation using model checkers. A counter-example is generated if the design fails to hold a property [6]. Since model checkers consider all of the design states, it is prone to state space explosion issue, especially when large designs and complex properties are involved. A Bounded Model Checking (BMC) is used to limit the design unfolding to a limited number of clock cycles. Since BMC does not check for all of the possible design states, it cannot formally prove the given property. However, BMC assumes that the designer knows the required number of clock cycles that a particular property should hold.

Rajendran et al. [40] have proposed a test generation technique for Trojan detection using BMC approach. They generate a set of security properties based on access privileges to critical data registers, address tables, or stack pointer of processors. The properties are then checked against the design using BMC to find unauthorized accesses. The BMC generates a test to activate the hidden Trojan when the given property does not hold in the design. However, the strength of this approach is dependent on the completeness of the security properties as well as the capability of the SAT solver used during BMC. GLIFT looks for confidentiality and integrity property violations [27]. The confidentiality property requires that any secure data may not enter the unsecured domain. Conversely, the integrity property requires that anything from unsecured domain may not enter the secured domain.

Equivalence checking is used to formally prove that different representations of a design display the same functionality—nothing more, nothing less. However, traditional equivalence checking methods may face state-space explosion issue when applied on large design. Using Gröbner basis theory enables formulation of the equivalence checking problem in the algebraic domain [25]. The design specification is first converted to a specification polynomial f_{spec}. The gate-level implementation is then converted to a set of implementation polynomial F, where each gate is modeled

as a polynomial. Polynomial division is then applied to reduce the specification polynomial f_{spec} over implementation polynomials F. The remainder should be zero for functionally equivalent designs. A non-zero remainder indicates that the specification differs from the implementation, and the design is not trustworthy.

The remainder not only expresses the outcome of the equivalence checking but also is beneficial in removing the Trojan. Any assignment which makes the total value of the remainder non-zero is a directed test (counter-example) that activates the Trojan. The directed test can be used to localize the source of error. Moreover, patterns and existing terms of the remainder provides valuable information to detect and correct the bug (gate-misplacement, as well as signal inversion are considered as fault models) [20–23].

1.3 Hybrid Approaches

Generating directed tests that could activate hidden hardware Trojans is exceptionally challenging since adversaries tend to implant hardware Trojans such that they are triggered under extremely rare input sequences. Model checking-based formal test generation approaches are effective for small designs. It is prone to state-explosion issue which limits its effectiveness on large or complex designs. Cruz et al. have proposed a test generation technique that combines the N-detect testing with ATPG as well as using model checkers [17]. Security properties are generated such that they target activation of rare nodes in the gate-level designs. The design is partitioned into two parts based on the location of the scan-chains. A set of constraints are generated using a model checker for the non-scan portion of the design, and then given to the ATPG tool. ATPG combines these constraints with the scan portion constraints for directed test generation. The proposed approach overcomes the limitation of both ATPG and model checking in gate-level partial-scan designs. However, such approach is not applicable to higher level of abstractions such as RTL.

In the rest of this chapter, we will discuss a promising hybrid test generation technique for Trojan activation. This technique is called concolic testing which interleaves concrete simulation with symbolic execution for test generation. Section 2 provides an overview of the concolic testing method. Section 3 surveys different concolic testing methods. Section 4 describes an automated concolic testing framework for generation of directed tests to detect hardware Trojans. Section 5 compares this test generation method with the related approaches outlined in Sect. 3. Finally, Sect. 6 concludes the chapter.

2 Background: Concolic Testing

The idea of concolic testing was introduced in the software domain [24, 43]. Liu et al. first applied this on hardware designs [30]. Typical flow of a concolic testing framework is shown in Fig. 2 [4]. In concolic testing, the source code is first instru-

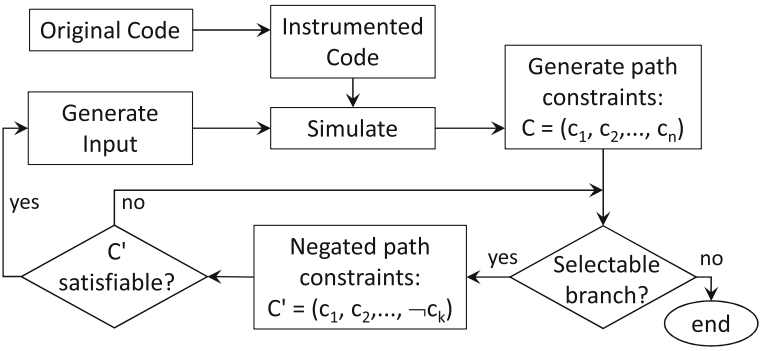


Fig. 2 Typical flow for a concolic testing engine [4]

mented with monitoring statements. This enables us to trace the execution path of the simulation. Initially, the design is simulated with a random set of input. In the subsequent iterations, the input is systematically generated to guide the simulation towards uncovered portions of the design. To do so, the logical expressions in an execution path is converted to constraints (C in Fig. 2). An uncovered branch from the execution path is then selected to be explored next by negating the constraint representing that ($\neg c_k$). This modified constraint along with the rest (C' in Fig. 2) is then solved by a constraint solver. If it is solvable, then we get an input set for which the execution path will go through that uncovered branch—improving coverage. Otherwise, a different branch is selected. The process continues until there is no selectable branch. Other criteria, such as time-limit or coverage target can also be used to end the concolic testing. This is a semi-formal method which interleaves concrete and symbolic execution. Unlike formal methods like bounded model checking, concolic testing is not susceptible to state explosion issue. The main reason is—concolic testing explores a single execution path at a time instead of trying to explore all possible paths at the same time. Concolic testing is applicable on larger designs because of this advantage.

3 Prior Works on Concolic Testing

Godefroid et al. introduced concolic testing as a tool called "DART" [24]. DART was capable of generating tests for bug localization in software. Sen et al. proposed jCUTE and CUTE which added support for multi-threading and few more programming languages [42, 43]. The concept of concolic testing is later applied to hardware designs by Liu et al. [30]. Above mentioned methods employ depth-first search (DFS) for selecting an alternate branch for exploring different paths of the design. Exhaustive search strategies like depth-first and breadth-first are thorough, but can

only cover a small portion of the overall search space within a limited time budget. This is known as *path explosion* problem.

Many search strategies have been proposed to avoid the path explosion issue. Liu et al. recently proposed an enhanced version of the DFS based technique [32]. In their approach, they have used the concept of state caching. During concolic testing, previously explored states are cached using compressed bitmap encoding. If such state is encountered again, then all the branches from that particular path are skipped. A random branch selection strategy is proposed in [7]. Because of the random nature of the selection, it displayed inconsistent performance across benchmarks. A control-flow graph (CFG) directed strategy is proposed in the same article. CFG is used here to assign distance values to each of the nodes which represent the distances from the target branch. During the branch selection procedure, an uncovered branch with the minimum distance is selected. Liu et al. proposed another CFG based technique named "HYBRO" [31]. HYBRO statically determines the control dependency of all branches. A branch is only selected during concolic testing if one of its dependent branches is still uncovered. Qin et al. proposed an improvement which supports arrays with dynamic references [39]. Ahmed et al. recently proposed a technique to control the thoroughness of search [4]. Using this strategy, if searching in an area does not improve branch coverage after a certain number of iterations, the search is moved to a different region and the procedure is repeated. Researchers have proposed several other coverage guided strategies such as CarFast [36]. CarFast selects branch depending on the number of uncovered branches that will be covered by the selection. Li et al. proposed a technique where the least frequently traversed branch is selected [29]. Context-Guided Search (CGS) proposed by Seo et al. keeps a cache of recently traversed sub-paths [44]. Branches that are the part of the cached sub-paths are skipped during the selection procedure.

So far, we discussed the methods which try to maximize the branch coverage. However, sometimes the goal is not to cover all branches but to cover a specific branch. This type of scenario occurs during debugging an error or while trying to generate tests for hard to reach corner cases. While previous methods may still work for such directed test generation, they usually take much longer to finish [3]. Directed test generation is covered extensively in existing literatures [12–15, 28, 33, 34, 38]. Researchers in [10, 11, 18] have utilized Symbolic Backward Execution (SBE) technique. In SBE, the design is symbolically unrolled starting from the target. Unrolling is continued until an entry point (usually primary inputs) is reached. However, SBE is difficult to apply on designs with data-dependent loops or non-linearity, and does not scale well with design size. Zamfir et al. reduced the search-space complexity by evaluating which edges in the CFG are essential to reach the target node. Unfortunately, these techniques are only suitable for software designs and do not consider the concurrent execution model of hardware designs. Ahmed et al. recently proposed a directed concolic testing method which is applicable to hardware designs [3]. As Trojan triggers usually reside in rare branches, this directed test generation technique is suitable for Trojan activation. Given a list of suspicious rare branches, the proposed method can quickly generate tests to cover these branches. The list of suspicious

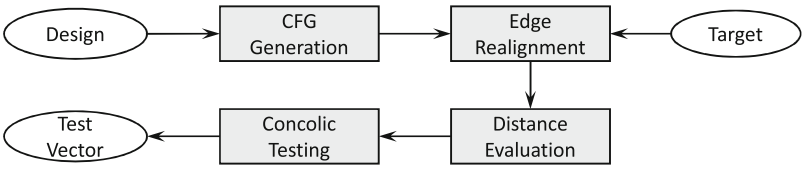


Fig. 3 Overview of the directed test generation framework [3]

branches can be obtained by running statistical probability analysis software such as TPC [2], or by simple random simulations. The remainder of this chapter describes this test generation technique in detail.

4 Directed Test Generation Using Concolic Testing

Figure 3 presents the overview of the directed test generation method proposed by Ahmed et al. [3]. Here, the input is an RTL design and a $target$ branch or statement. The objective is to generate a test vector for covering that target. Rest of the section will describe the major steps of this test generation method.

4.1 CFG Generation

Hardware designs require concurrent computation model. For example, all of the $always$ blocks in Verilog and $process$ in VHDL are executed concurrently. For the ease of explanation, '**process**' will be used from now on to describe concurrent blocks. Although these processes are executed concurrently, execution is sequential within a single process. Thus, the standard procedure for finding CFG in a sequential program can be applied to extract CFGs of individual processes. CFGs are consist of multiple *basic blocks* connected together. Basic blocks do not contain any branching, so if the first statement of the block is executed, then the rest will also be executed. A basic block can be connected to multiple other basic blocks, known as the $successor$ and $predecessor$ blocks. Suppose after finishing the execution of block P, the control is transferred to block Q. Then P is the predecessor of Q, and conversely, Q is the successor of P. Figure 4 shows an example Verilog code. As it contains two processes, two CFGs are produced. For the sake of simplicity, suppose that line l_x is contained within basic block bb_x. According to the definition, successor of bb_6 are bb_9 and bb_7. Predecessor of bb_6 is bb_4. Exit block is represented by the gray circle.

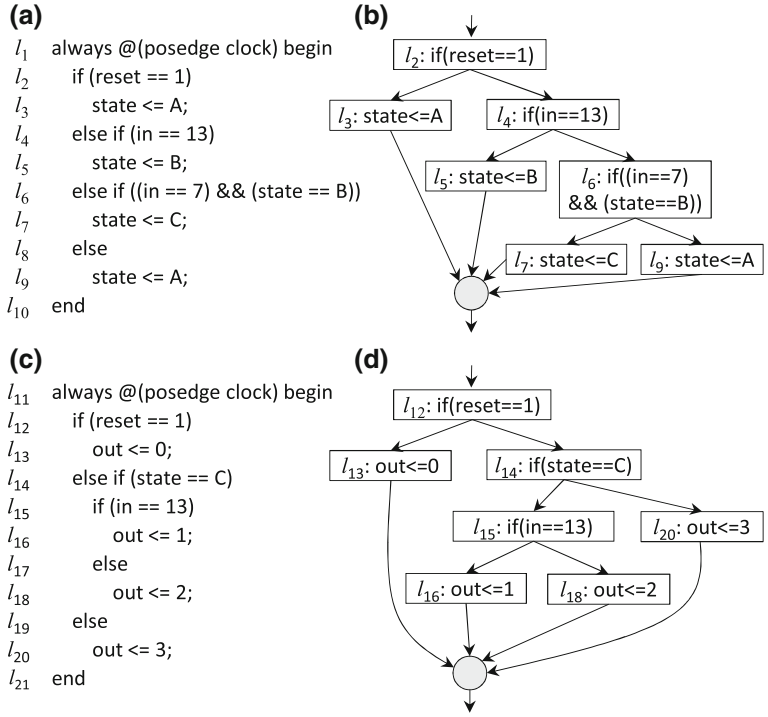


Fig. 4 Example Verilog code with two processes [3]. **a** Process 1 code, **b** process 1 CFG, **c** process 2 code, **d** process 2 CFG. The left path is taken if the condition is true

When the target statement is only reachable with inputs spanning multiple clocks, the design must be unrolled for at least those many cycles. This unrolling can be done by cascading several single-cycle CFGs. Making the exit block predecessor of the entry block produces the same result.

4.2 Edge Realignment

In the CFG generation phase, an independent CFG is generated for each process. As there are no edges between the CFGs, dependencies among processes are not correctly reflected. Consider an example where l_{15} is the target (Fig. 4). The condition $if(state == C)$ must be true in process 2 to reach l_{15}. However, the only assignment that can make this condition true is l_7 in process 1. This relationship information is not available in the independent CFG scheme. An edge realignment procedure is applied to resolve this issue. This realignment procedure modifies the edges of the

CFGs to correctly reflect the flow of the program across multiple clock cycles and multiple processes. For a better explanation, related terms are defined first:

Strict variable: Strict variables are the variables to which only concrete values are assigned [3]. out is a strict variable in the example of Fig. 4. Here, l_{13}, l_{16}, l_{18} and l_{20} are the assignments to variable out. Concrete values are assigned to out in all of these variables. Variable $state$ is another strict one.

Dominator: In a CFG, node d is a dominator of node p as all the paths from entry block to p go through d. l_{12} and l_{14} are dominators of l_{15} in the example of Fig. 4.

Immediate dominator: Suppose D is the set of all dominators of node p. Then the immediate dominator of p is defined as the node in D which does not dominate any other nodes in D. It can also be seen as the closest dominator of p. As an example, immediate dominator of l_9 is l_6.

Whether the execution path will go through a basic block or not will depend on its guard condition. Edge realignment procedure re-connects basic blocks to the blocks that can satisfy this condition. This procedure is shown in Algorithm 1. Major portion of the work is done by the function $update\_edge(bb_t)$. This function recursively aligns the edges, starting from bb_t, and going backward. Initially bb_t contains the target statement given by the user (main procedure, line 4). Guard condition of bb_t is first expanded (l_g) inside $update\_edge()$ function (line 3). We explain how the expansion works later. Two scenarios can occur at this point: (i) l_g contains no strict variable, or (ii) it contains at least one. If it does contain a strict variable, then the current predecessor of bb_t is replaced by the blocks that can satisfy the guard condition l_g (line 5–15). The $update\_edge()$ is called recursively on the satisfying blocks in this case. On the other hand, if l_g does not contain any strict variables, then we try to realign its immediate dominator (line 16–18). To prevent the unnecessary assessment of previously traversed block, a $visited$ flag is used. This algorithm does not modify the successor edges, because these edges are not used in the subsequent phases.

As an example, consider the CFGs of Fig. 4. Edge realignment procedure will be carried out on these CFGs. Here, l_{16} is the target statement. Thus, $update\_edge()$ function will be applied on b_{16} first. $l_{15} : in = 13$ is the guard condition of l_{16}. This condition falls into the second category as it does not contain any strict variables. According to Algorithm 1, l_{15} becomes the new target because it is the immediate dominator of l_{16}. $l_{14} : state = C$ is the guard condition of l_{15}.

Unlike the first guard condition, this condition contains the strict variable $state$. Among the assignments to $state$, only $(l_{14} \wedge l_7)$ can be satisfied by the constraint solver. Therefore, the current predecessor of l_{15} is removed, and the satisfying basic block bb_7 becomes the new predecessor. The whole procedure is then repeated with l_7 being the new target and continues until realignment can be done. Figure 5 shows the final CFG after the edge realignment is finished.

Algorithm 1 Edge Realignment [3]

Input: CFG, Target Statement l_t
Output: Realigned CFG
1: **for all** basic block, $bb \in CFG$ **do**
2: $bb.visited \leftarrow false$ // initialization
3: **end for**
4: Target basic block, $bb_t \leftarrow basic\_block(l_t)$
5: $update\_edge(bb_t)$
6: **return**

$update\_edge(bb_t)$
1: **if** bb_t is valid **and** $bb_t.visited$ is $false$ **then**
2: $bb_t.visited = true$
3: $l_g \leftarrow$ expanded guard condition of bb_t
4: $V_s \leftarrow$ set of strict variables in l_g
5: **if** V_s is not empty **then**
6: $bb_t.predecessors \leftarrow \emptyset$
7: **for all** variables $v \in V_s$ **do**
8: **for all** assignments l_a to v **do**
9: **if** $satisfiable(l_g \wedge l_a)$ **then**
10: $bb_a \leftarrow basic\_block(l_a)$
11: add bb_a to $bb_t.predecessors$
12: $update\_edge(bb_a)$
13: **end if**
14: **end for**
15: **end for**
16: **else if** $bb_i \leftarrow idom(bb_t)$ **then**
17: $update\_edge(bb_i)$
18: **end if**
19: **end if**

4.3 Distance Evaluation

After the edge realignment step, the modified CFG reflects the dependencies among processes. This CFG is utilized to assign a distance value to each of the basic blocks. Basic blocks that are closer to the target have a lower distance value. Distance assignment is done using the Breadth First Search (BFS) strategy, as shown in Algorithm 2. Initially, the target block's distance is 0, and all the other blocks have a distance of ∞ (line 1–4). Then, BFS is applied to the CFG (line 5–14). The search starts from the target node and goes backward (from target to root) using the predecessor edges. Distance is increased by one for each level of BFS (line 10). A simple queue is used to realize the BFS strategy. Figure 5 illustrates the final distance values for the target l_{16}.

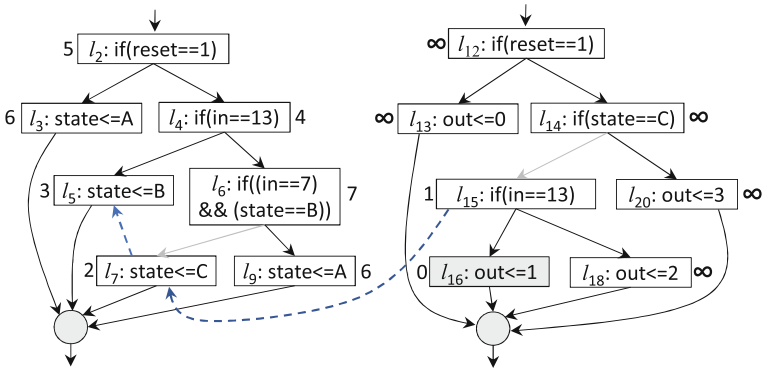


Fig. 5 CFG of the example shown in Fig. 4 after edge realignment and weight assignment. Here target is l_{16}. Omitted edges are shown using light solid arrows. New edges are depicted as blue dashed arrow [3]

Algorithm 2 Distance Evaluation

Input: CFG, Target Statement l_t
Output: CFG with distance values
1: **for all** basic block, $bb \in CFG$ **do**
2: $bb.distance \leftarrow \infty$
3: **end for**
4: $bb_t.distance \leftarrow 0$
5: Queue for distance evaluation, $Q \leftarrow \{bb_t\}$
6: **while** Q not empty **do**
7: $bb_t \leftarrow Q.pop()$
8: **for all** $bb_p \in bb_t.predecessors$ **do**
9: **if** $bb_p.distance > (bb_t.distance + 1)$ **then**
10: $bb_p.distance \leftarrow (bb_t.distance + 1)$
11: $Q.push(bb_p)$
12: **end if**
13: **end for**
14: **end while**
15: **return**

4.4 Test Generation

Concolic testing is used in this phase for test generation. In the first iteration of concolic testing, the design is simulated with a random input vector. In the subsequent iterations, distance values are used to guide the execution towards the target.

The concolic testing procedure is given in Algorithm 3. At the beginning of each iteration, the algorithm checks if the target is covered by the current execution path τ (line 2–5). The procedure simply returns the input vector if the target is covered. Otherwise, an alternate branch adjacent to τ is selected for exploration. In the case of multiple alternate branches, the branch with the lowest distance value will be selected first. If more than one branch has the same lowest distance, then the

Algorithm 3 Concolic Testing

Input: CFG with distance values
Output: Test vector
1: Input vector, $I \leftarrow random()$
2: Execution path, $\tau \leftarrow simulate(I)$
3: **if** target statement covered in τ **then**
4: **return** I
5: **end if**
6: $B \leftarrow$ list of branches adjacent to τ
7: $sort\_by\_distance(B)$
8: **for all** branch $b \in B$ **and** $is\_selectable(b)$ **do**
9: $C \leftarrow$ path constraints upto b
10: $I \leftarrow constraint\_solver(C)$
11: **if** I is valid **then**
12: decrease priority of b
13: **goto** 2
14: **end if**
15: **end for**
16: **return** $invalid$

branch with the lower clock cycle will be prioritized. This sorting is abstracted by the function $sort\_by\_distance()$ (line 7). Validity of the selected branch is checked by the function $is\_selectable()$ (line 8). This function ensures that selecting the branch will not repeat a previously traversed execution path. Next, constraints leading to the newly selected branch is given to the constraint solver (line 9–10). If the constraints are satisfiable, then the solution input vector is used for the next iteration (11–14). Otherwise, the next branch from the sorted list is evaluated. The process continues until no satisfiable branch remains or the target is covered. The algorithm discussed so far greedily selects branch with the lowest distance. To avoid repeated selection of the same branch, the distance value is increased each time a branch is selected (line 12).

Table 1 shows the iterations for the example of Fig. 5. Here, clock 0 contains the reset sequence, which is omitted to increase clarity. The first column in the table shows the iteration. Next three columns show the execution path at different clock cycles. The fifth column shows the candidate branches sorted by their priority. The final column shows the selected branch. As we can see in the example, a lower priority branch can be selected if the other candidates fail the validity check of $is\_selectable()$ function. In this specific example, the concolic testing reaches the target after three iterations.

Table 1 Execution path and selected branches during concolic testing iterations

Iteration	Execution path at			Sorted branch	Selected branch
	Clock 1	Clock 2	Clock 3		
0	l_2, l_4, l_6, l_9	l_2, l_4, l_6, l_9	l_2, l_4, l_6, l_9	l_{15}, l_7, l_5	l_5
	l_{12}, l_{14}, l_{20}	l_{12}, l_{14}, l_{20}	l_{12}, l_{14}, l_{20}	...	
1	l_2, l_4, l_5	l_2, l_4, l_6, l_9	l_2, l_4, l_6, l_9	l_{15}, l_7, l_5	l_7
	l_{12}, l_{14}, l_{20}	l_{12}, l_{14}, l_{20}	l_{12}, l_{14}, l_{20}	...	
2	l_2, l_4, l_5	l_2, l_4, l_6, l_7	l_2, l_4, l_6, l_9	l_{16}, l_{15}, l_7	l_{16}
	l_{12}, l_{14}, l_{20}	l_{12}, l_{14}, l_{20}	$l_{12}, l_{14}, l_{15}, l_{18}$	...	
3	l_2, l_4, l_5	l_2, l_4, l_6, l_7	l_2, l_4, l_6, l_7	–	–
	l_{12}, l_{14}, l_{20}	l_{12}, l_{14}, l_{20}	$l_{12}, l_{14}, l_{15}, l_{16}$		

5 Experiments

Following experiments are ran on a four-core Intel Core-i7 4.0 GHz processor and 16 GB of RAM [3]. OpenCores, ITC99, and TrustHub benchmarks are used for evaluation [1, 2, 16, 41]. Benchmarks from different sources ensured varying characteristics. Most of these benchmarks have rare branches, especially the TrustHub benchmarks contain Trojans with extremely difficult to activate branches. The Icarus Verilog Target API [48] is used as a Verilog parser for the concolic testing framework. Yices is used as the constraint solver [19].

5.1 Evaluation of Scalability

The main advantage of concolic testing approach is its scalability compared to the formal methods. It is demonstrated in this section by comparing the concolic testing framework against a bounded model checking tool named EBMC [35]. A custom version of AES cipher is used as the base design. The number of rounds in the cipher is slowly incremented to increase the design complexity. The benchmarks for this experiments are named as *cb__aes__xx*, with *xx* referring to the number of rounds in the benchmark. The rounds are sequentially cascaded, and the activation condition is made to depend on all the rounds. This ensures that increasing the number of rounds will increase the trigger activation difficulty.

Table 2 presents the comparison results. Here, the first eight rows are used to demonstrate the test generation time and memory consumption with the increasingly complex design. As we can see, memory consumption grows exponentially for EBMC. It exceeds the available 16 GB memory after 20 rounds. On the other hand, the memory requirement increases linearly for [3]. Runtime is similar for both approaches.

Table 2 Scalability comparison with model checking tool EBMC [3]

Benchmark	Unroll cycles	Lines of code[a] (k)	EBMC [35]		Directed [3]	
			Time (s)	Mem (MB)	Time (s)	Mem (MB)
cb__aes__01	5	33	1.27	179.4	0.51	55.3
cb__aes__05	10	167	11.47	1450.3	4.03	244.3
cb__aes__10	15	334	33.17	4130.6	14.47	502.4
cb__aes__15	20	501	70.78	8041.2	32.14	778.2
cb__aes__20	25	668	110.13	13202.8	86.03	1085.5
cb__aes__25	30	886	–	–	150.54	1405.3
cb__aes__30	35	1003	–	–	243.02	1780.3
cb__aes__35	40	1169	–	–	371.23	2112.7
cb__aes__10	15	334	33.17	4130.6	14.47	502.4
cb__aes__10	20	334	42.72	5361.8	15.33	520.2
cb__aes__10	25	334	53.78	6628.8	16.25	542.6
cb__aes__10	30	334	64.12	7871.5	17.22	563.7
cb__aes__10	35	334	74.32	9119.5	18.47	582.5
cb__aes__10	40	334	84.57	10361.3	19.36	608.0

[a]Counted after hierarchy flattening

In the last six rows of the table, design complexity is kept constant while the unroll cycles are slowly increased from 15 to 40. For EBMC, memory consumption and time requirement are increased by 150.8% and 155% respectively. For concolic testing, on the other hand, memory consumption and time requirement are increased by 21 and 33.8% only. These results indicate that the concolic testing scales better than the bounded model checking based approaches.

5.2 Evaluation of Coverage

Coverage is compared against two concolic testing methods—one with uniform coverage goal (QUEBS [4]) and the other being directed concolic testing method (CFG-Directed, which is a naive extension of [7] to apply on RTL designs). For ITC99 and OpenCore benchmarks, target branches are selected in two ways—*rare* and *random*. For *random* type, five reachable branches are chosen at random, and then average results are given. For *rare* target selection, benchmarks are simulated with random inputs for one million cycles, and the branch which is covered the least number of times is selected. TrustHub benchmarks contain Trojans with very hard to activate trigger conditions. Triggers are used as targets for these benchmarks.

Table 3 presents the comparison results. If a test vector is not generated within two thousand iterations, it is considered as a failure. As we can see, both CFG-directed and

Table 3 Performance and coverage comparison [3]

Benchmark	Unroll cycles	Target	QUEBS [4]		CFG-Directed[a]		Directed [3]	
			Time (s)	Iter	Time	Iter	Time (s)	Iter
b06	10	Rand	0.02	13.6	0.02	9.4	0.01	3
		Rare	0.02	20	0.01	7	0.01	5
b10	30	Rand	0.11	245.4	0.01	4.4	0.01	1.4
		Rare	0.13	301	0.01	5	0.01	1
b14	30	Rand	0.61	564	0.01	8.4	0.01	3.6
		Rare	0.90	814	0.01	21	0.01	1
i2c	10	Rand	–	–	1.62	356.4	0.57	21.2
		Rare	–	–	4.09	1123	0.98	40
OR1200 ICache	50	Rand	0.23	155	0.13	27.2	0.02	5.4
		Rare	0.34	224	0.20	34	0.02	9
AES_T1000	10	Trojan	–	–	4.67	1	3.88	1
AES_T1100	10	Trojan	–	–	19.62	7	11.80	4
wb__conmax T200	10	Trojan	–	–	–	–	13.36	1
wb__conmax T300	10	Trojan	–	–	–	–	11.06	1

[a]Extension of [7] for RTL designs

QUEBS failed for some targets with the given iteration limit. This result is expected from QUEBS, because it is designed with overall coverage in mind. Compared to CFG-directed [3] improved the number of iterations by 16.8x on average (without considering the failed cases of wb__conmax). Overall, the proposed method in [3] managed to cover targets with varying characteristics within a smaller number of iterations and was able to quickly activate rare Trojan triggers.

6 Conclusion

It is crucial to detect hardware Trojans in SoCs in order to design secure IoT devices. However, it is difficult to generate tests to activate stealthy Trojans. Logic testing and formal methods are not often suitable for such activation, with design size and rarity of trigger condition being the limiting factor. This chapter discussed a hybrid test generation by combining concrete simulation with symbolic execution to activate hardware Trojans. The presented method avoids the state explosion faced by formal methods by exploring one execution path at a time. Overall, this method managed to quickly generate tests to activate rare Trojans, whereas traditional test generation approaches failed.

References

1. Opencores website (2017). https://www.opencores.org
2. TrustHub website (2017). https://www.trust-hub.org
3. Ahmed, A., Farahmandi, F., Mishra, P.: Directed test generation using concolic testing on RTL models. In: 2018 Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1538–1543. IEEE (2018)
4. Ahmed, A., Mishra, P.: QUEBS: Qualifying event based search in concolic testing for validation of RTL models. In: 2017 IEEE 35th International Conference on Computer Design (ICCD), pp. 185–192. IEEE (2017)
5. Banga, M., Hsiao, M.S.: Trusted RTL: trojan detection methodology in pre-silicon designs. In: 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 56–59. IEEE (2010)
6. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of BDDs. In: Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, pp. 317–320. ACM (1999)
7. Burnim, J., Sen, K.: Heuristics for scalable dynamic test generation. In: ASE, pp. 443–446 (2008)
8. Chakraborty, R.S., Narasimhan, S., Bhunia, S.: Hardware trojan: threats and emerging solutions. In: 2009 IEEE International High Level Design Validation and Test Workshop, HLDVT 2009, pp. 166–171. IEEE (2009)
9. Chakraborty, R.S., Wolff, F., Paul, S., Papachristou, C., Bhunia, S.: Mero: a statistical approach for hardware trojan detection. In: Cryptographic Hardware and Embedded Systems-CHES 2009, pp. 396–410. Springer (2009)
10. Chandra, S., et al.: Snugglebug: a powerful approach to weakest preconditions. In: SIGPLAN, vol. 44, pp. 363–374 (2009)
11. Charreteur, F., Gotlieb, A.: Constraint-based test input generation for java bytecode. In: ISSRE, pp. 131–140 (2010)
12. Chen, M., Mishra, P.: Functional test generation using efficient property clustering and learning techniques. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**(3), 396–404 (2010)
13. Chen, M., Mishra, P.: Property learning techniques for efficient generation of directed tests. IEEE Trans. Comput. **60**(6), 852–864 (2011)
14. Chen, M., Qin, X., Koo, H.-M., Mishra, P.: System-Level Validation: High-Level Modeling and Directed Test Generation Techniques. Springer Science & Business Media (2012)
15. Chen, M., et al.: Automatic RTL test generation from systemC TLM specifications. TECS **11**, 38 (2012)
16. Corno, F., et al.: RT-level ITC'99 benchmarks and first ATPG results. IEEE Des. Test Comput. **17**(3), 44–53 (2000)
17. Cruz, J., Farahmandi, F., Ahmed, A., Mishra, P.: Hardware trojan detection using ATPG and model checking. In: International Conference on VLSI Design (2018)
18. Dinges, P., Agha, G.: Targeted test input generation using symbolic-concrete backward execution. In: ASE, pp. 31–36 (2014)
19. Dutertre, B., De Moura, L.: The Yices SMT solver. 2:1–2 (2006). Tool paper. http://yices.csl.sri.com/tool-paper.pdf
20. Farahmandi, F., Huang, Y., Mishra, P.: Trojan localization using symbolic algebra. In: 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 591–597. IEEE (2017)
21. Farahmandi, F., Mishra, P.: Automated test generation for debugging arithmetic circuits. In: DATE, pp. 1351–1356 (2016)
22. Farahmandi, F., Mishra, P.: Automated debugging of arithmetic circuits using incremental gröbner basis reduction. In: 2017 IEEE 35th International Conference on Computer Design (ICCD), pp. 193–200. IEEE (2017)
23. Farahmandi, F., Mishra, P.: FSM anomaly detection using formal analysis. In: 2017 IEEE 35th International Conference on Computer Design (ICCD), pp. 313–320. IEEE (2017)

24. Godefroid, P., et al.: DART: directed automated random testing. In: ACM SIGPLAN Notices, vol. 40, pp. 213–223 (2005)
25. Guo, X., Dutta, R.G., Jin, Y., Farahmandi, F., Mishra, P.: Pre-silicon security verification and validation: a formal perspective. In: Proceedings of the 52nd Annual Design Automation Conference, p. 145. ACM (2015)
26. Hicks, M., Finnicum, M., King, S.T., Martin, M.M., Smith, J.M.: Overcoming an untrusted computing base: detecting and removing malicious hardware automatically. In: 2010 IEEE Symposium on Security and Privacy (SP), pp. 159–172. IEEE (2010)
27. Hu, W., et al.: Detecting hardware trojans with gate-level information-flow tracking. Computer 44–52 (2016)
28. Koo, H.-M., Mishra, P.: Functional test generation using design and property decomposition techniques. ACM Trans. Embed. Comput. Syst. (TECS) **8**(4), 32 (2009)
29. Li, Y., et al.: Steering symbolic execution to less traveled paths. In: ACM SIGPLAN Notices, vol. 48, pp. 19–32 (2013)
30. Liu, L., Vasudevan, S.: STAR: generating input vectors for design validation by static analysis of RTL. In: HLDVT, pp. 32–37 (2009)
31. Liu, L., Vasudevan, S.: Efficient validation input generation in RTL by hybridized source code analysis. In: DATE, pp. 1–6 (2011)
32. Liu, L., et al.: Scaling input stimulus generation through hybrid static and dynamic analysis of RTL. TODAES **20**, 4 (2014)
33. Lyu, Y., Qin, X., Chen, M., Mishra, P.: Directed test generation for validation of cache coherence protocols. IEEE Trans. Comput. Aided Des. Integr, Circuits Syst (2018)
34. Mishra, P., Dutt, N.: Specification-driven directed test generation for validation of pipelined processors. ACM Trans. Des. Autom. Electron. Syst. (TODAES) **13**(3), 42 (2008)
35. Mukherjee, R., et al.: Hardware verification using software analyzers. In: ISVLSI, pp. 7–12 (2015)
36. Park, S., et al.: Carfast: achieving higher statement coverage faster. In: FSE, p. 35 (2012)
37. Pnueli, A.: The temporal semantics of concurrent programs. Theor. Comput. Sci. **13**(1), 45–60 (1981)
38. Qin, X., Mishra, P.: Directed test generation for validation of multicore architectures. ACM Trans. Des. Autom. Electron. Syst. (TODAES) **17**(3), 24 (2012)
39. Qin, X., Mishra, P.: Scalable test generation by interleaving concrete and symbolic execution. In: VLSID, pp. 104–109 (2014)
40. Rajendran, J., Vedula, V., Karri, R.: Detecting malicious modifications of data in third-party intellectual property cores. In: Proceedings of the 52nd Annual Design Automation Conference, p. 112. ACM (2015)
41. Salmani, H., et al.: On design vulnerability analysis and trust benchmarks development. In: ICCD, pp. 471–474 (2013)
42. Sen, K., Agha, G.: CUTE and jCUTE: concolic unit testing and explicit path model-checking tools. In: CAV, pp. 419–423 (2006)
43. Sen, K., et al.: CUTE: a concolic unit testing engine for C. In: ACM SIGSOFT Software Engineering Notes, vol. 30, pp. 263–272 (2005)
44. Seo, H., Kim, S.: How we get there: a context-guided search strategy in concolic testing. In: FSE, pp. 413–424 (2014)
45. Sturton, C., Hicks, M., Wagner, D., King, S.T.: Defeating UCI: building stealthy and malicious hardware. In: 2011 IEEE Symposium on Security and Privacy (SP), pp. 64–77. IEEE (2011)
46. Tehranipoor, M., Koushanfar, F.: A survey of hardware trojan taxonomy and detection. IEEE Des. Test Comput. **27**(1) (2010)
47. Waksman, A., Suozzo, M., Sethumadhavan, S.: Fanci: identification of stealthy malicious logic using boolean functional analysis. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, pp. 697–708. ACM (2013)
48. Williams, S.: Icarus Verilog (2006). http://iverilog.icarus.com/
49. Zhang, J., Yuan, F., Wei, L., Liu, Y., Xu, Q.: Veritrust: verification for hardware trust. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **34**(7), 1148–1161 (2015)

50. Zhang, J., Yuan, F., Xu, Q.: Detrust: defeating hardware trust verification with stealthy implicitly-triggered hardware trojans. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 153–166. ACM (2014)
51. Zhang, X., Tehranipoor, M.: Case study: detecting hardware trojans in third-party digital IP cores. In: 2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 67–70. IEEE (2011)

Low Cost Dual-Phase Watermark for Protecting CE Devices in IoT Framework

Anirban Sengupta and Dipanjan Roy

Abstract Intellectual property (IP) core providers are increasingly aware of the need to protect their investment from either counterfeit/forgery or illegal ownership. This chapter presents a novel low cost dual phase watermarking methodology during high level synthesis (HLS) for IP core protection of vendor. Robust vendor signature is embedded in two subsequent phases of high level synthesis to form an integrated watermark. We present a dual-phase watermarking methodology that embeds a multi-variable double phase watermarking during high level synthesis for application specific IPs (application specific integrated circuits) that incurs zero delay and register overhead as well as minimal hardware overhead. The dual-phase watermarking approach yields average reduction of embedding cost of 6% (which includes average area reduction of 7% and average latency reduction of 4%) when compared to two recent HLS based watermarking approaches for application specific IPs. Additionally, the approach also achieves stronger proof of authorship compared to two recent HLS based watermarking approaches.

1 Introduction

Internet-of-Things (IoT) represents interconnection (communication system) of smart devices, sensors, computing devices etc (including consumer electronics) through modern network technologies. IoT is playing an essential role in home electronics, from entertainment to smart home control. In the domain of IoT, driven by consumer electronics (CE) hardware, the importance of Electronics Design Automation (EDA) is pivotal. In the modern era of EDA for CE device, surging complexity of design is out-pacing the design productivity. The significance of reusable Intellectual Property (IP) [1–3] core for CE hardware is to cope up with these complex design

A. Sengupta (✉) · D. Roy
Discipline of Computer Science and Engineering, Indian Institute of Technology Indore, Indore, India
e-mail: asengupt@iiti.ac.in

D. Roy
e-mail: phd1501201007@iiti.ac.in

R. S. Chakraborty et al. (eds.), *Security and Fault Tolerance in Internet of Things*, Internet of Things, https://doi.org/10.1007/978-3-030-02807-7_2

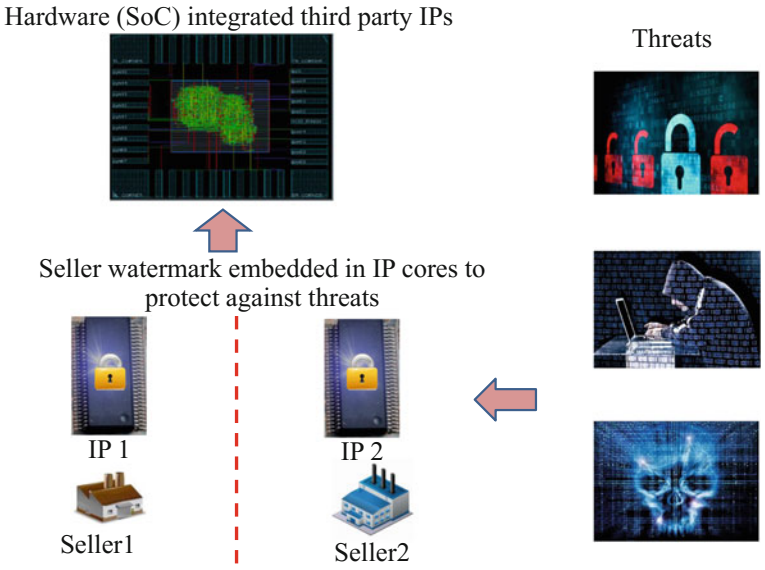


Fig. 1 Overview of IP core threats and its protection mechanism

requirements by reducing design time and enhancing design productivity. This makes IP core a promising and popular solution in the industry. However, with the increase in globalization and competition between the IP sellers/vendors, typical attacks such as IP counterfeiting/cloning, ownership abuse are escalating. Protection of IP core is essential to advance the benefits of reusable IP core [2–12]. However, while protecting an IP core correct functionality and accuracy must be retained. Ownership of IP ownership may be achieved by activating and deactivating each IP core by the system designer [13], through a reversible data hiding approach [14] or by information hiding techniques through steganography [15, 16]. Implanting a invisible owner's signature (known as watermark) is also one of the effective and promising option to protect a reusable IP core against ownership abuse and IP infringements (refer Fig. 1).

2 Overview of IP Core Protection Through Watermark

In this sub-section, we discuss some of the well-known powerful watermarking methodologies along with their differences with the presented dual-phase watermarking approach. Several watermarking techniques as discussed in [1, 11, 17–19] are embedded at lower design abstraction level. For example side channel based watermarking technique is discussed in [17], where, the vendor watermark is inserted into the netlist and bitstream of an IP design. An in-synthesis IP core watermarking

process is proposed in [11]. Though these approaches are useful however they do not protect designs at higher abstraction levels and incur more implementation complexity. Further [1, 18] is only applicable for Field-programmable Gate Arrays (FPGAs), not for Application Specific Integrated Circuits (ASICs). Additionally, both [18, 19] do not embed multi-phase watermarking during High-level synthesis (HLS) of IP design. Watermarking has been applied at higher design abstraction level also, for example, authors in [2, 3] insert watermarking in register assignment phase of HLS. Specifically, in [2] a dual variable encoding scheme (sequence of '0' and '1') is proposed for watermarking whereas in [3] a multi variable encoding scheme (sequence of 'i', 'I', 'T' and '!') is proposed for watermarking. Both of these approaches add additional edges in colored interval graph to achieve single-phase watermark (i.e. during register assignment). Thus both, robustness and tamper resistance of [1–3, 11, 18, 19], are significantly less than a multi-phase HLS based watermark [20].

Several multi-level hierarchical watermarking techniques are also discussed in [21–23]. These techniques embed watermark in multiple design abstraction levels which offer strong proof of ownership and high degree of tamper resistance, but does not explicitly implant dual-phase watermarking constraints at architectural level (during high level synthesis steps). Further, aforesaid approaches may also incur higher implementation complexity and design overhead as the watermarks are embedded in multiple independent levels.

2.1 Motivation of Embedding Dual Phase Watermark for IP Protection at Behavioral Level

This discussion in this chapter is motivated by the fact that embedding watermark at higher design abstraction (e.g. behavioral level) may be more advantageous than embedding watermark at lower design abstraction (e.g. gate level or layout level). This is because embedding watermark at higher design abstraction enables design protection in subsequent lower levels (as watermark constraints embedded at higher level propagate with design synthesis). Moreover, the embedding process incurs lesser implementation complexity. In addition to that, embedding watermark in dual phase within same abstraction level not only ensures more robustness, lower overhead and higher tamper resistance watermark but also increases difficulty of reverse engineering, compared to only single phase watermark or multiple design level based watermark designs [20].

The main focus of this chapter is to present a multi-variable dual phase (register assignment and scheduling) watermarking methodology embedded at architectural level for protecting against abuse of IP vendor's ownership. The additional design constraints due to watermark is embedded at architectural level in two different phases of high level synthesis i.e. functional unit assignment and scheduling. It must be noted that the presented dual-phase watermark embedding process embeds the watermark in same abstraction level but in multiple phases, however without incur-

ring much overhead and complexity [20]. Moreover, it is capable to achieve strong proof of ownership and high degree stronger robustness as it distributes watermarking constraints across two independent design phases compared to single phase. Though it is a well known fact that insertion of watermark may result in design overhead with respect to hardware area, execution delay, power etc., nevertheless, the design overhead obtained by other HLS-based watermark [2, 3] is larger than the presented dual-phase watermark due to encoding rules devised. Experimental results validate the reduction in overhead and increase in robustness compared to [2, 3], as highlighted in [20].

The remaining parts of this chapter is structured as: Sect. 2 explains the related work. Section 3 illustrate the dual-phase watermarking methodology while Sect. 4 demonstrate a motivational example. Section 5 reports the results and analysis, and conclusion in Sect. 6.

3 Dual-Phase Watermarking Methodology

3.1 Problem Formulation

From a given application in the form of data flow graph (DFG) and user specified hardware configuration $(X_i) = N(R_1), N(R_2), \ldots, N(R_D)$, design a watermarked IP core solution where $N(R_D)$ is the number of hardware of type R_D.

Threat Model: The methodology presented in this chapter protects a reusable IP core from following threats: ownership abuse and IP counterfeit/forgery. Thus, possible attacks of a watermarked IP are: illegal claim of an IP and partial/complete removal of an IP watermark.

Target Platform/Technology: The dual-phase watermarking technique is easily adaptable to any modern EDA tool. Any hardware description language (HDL) based EDA tool can be merged with the aforesaid approach.

3.2 Dual-Phase Watermark Encoding

This sub-section discusses the presented dual-phase watermark encoding process. Figure 2 presents a overview of the dual-phase watermarking approach. In the approach we assume that the IPs are imported from two different third party IP vendors because multi-vendor based IPs during system on chip (SoC) integration is a common practice in the industry. In several published literatures [24–26], the concept of two IP vendors for SoC design have been used. The dual-phase watermark is embedded during scheduling and hardware assignment phases of HLS [20]. The IP design in terms of assignment and scheduling phases are depicted through two different tables (a) "*non-critical operations* ($\mu_m > 0$)" table and (b) "*hardware*

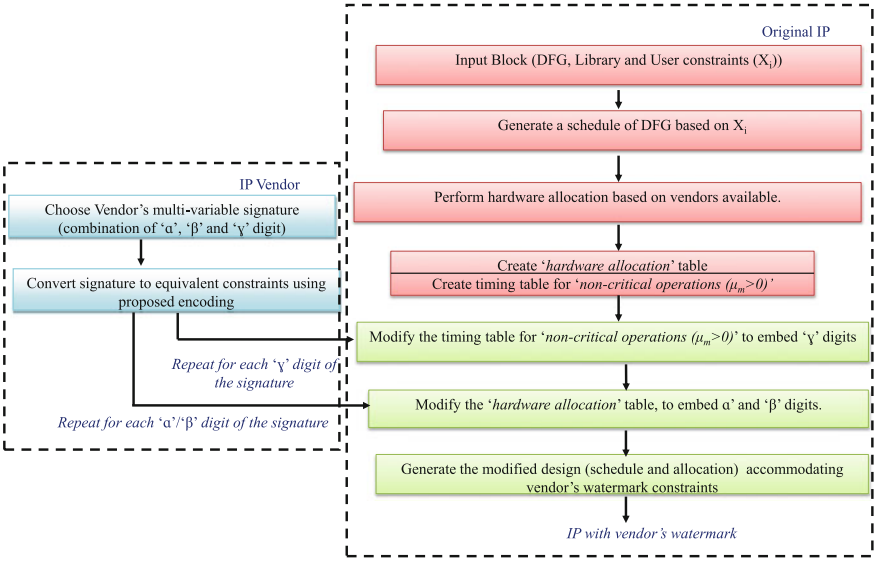


Fig. 2 IP core protection at behavioral level using dual-phase watermark [20]

assignment" timing table, where μ_m indicates the mobility of an operation. This dual-phase watermarking approach is comprises of three different digits viz. 'α', 'β', 'γ' where each digit indicates a specific encoding. A vendor's signature as a combination of 'α', 'β' and 'γ' is converted into its respective watermarking constraints based on the encoding rule of each digit, which is then subsequently embedded as additional constraints in the design. For example, 'α' and 'β' digit will embed a watermark by modifying the *hardware assignment* table and 'γ' will embed the vendor watermark by changing the "*non-critical operations* ($\mu_m > 0$)" table. The detailed encoding rule of each digit is defined as follows [20]:

- 'α' = Even operation and odd operation of odd control step will be assigned to hardware of vendor 2 (V2) and vendor 1 (V1) respectively.
- 'β' = Even operation and odd operation of even control step will be assigned to hardware of vendor 1 (V1) and vendor 2 (V2) respectively for.
- 'γ' = Shift an operation into its immediate next control step (cs) if the operation is in non-critical path with highest mobility.

In the dual-phase watermark encoding rule, no portion of the existing IP design requires to be hidden. Instead, the existing scheduling and hardware assignment of the IP design is locally altered to accommodate the signature encoded digits in the form of watermark. More specifically, in the scheduling phase, the operations of the non-critical path are locally shifted based on the encoding rule of γ as explained later in Tables 2 and 4. Next, in the hardware assignment phase the existing vendor assignment (allocation) is modified based on the encoding rule of α and β. As explained later Tables 1 and 3, due to signature digits of α and β, the existing hard-

Table 1 Hardware assignment table (before implanting watermark)

ODD C.S.	Operation no.	1	2	7	8	9	12	13
	Assigned hardware	M2	M1	A1	M1	M2	M2	A2
EVEN C.S.	Operation no.	3	4	5	6	10	11	14
	Assigned hardware	M1	M2	A1	A2	M2	A2	A1

Table 2 Timing table for non-critical operations (before implanting watermark)

Operation no.	2	3	4	6	8	7	10	13
Control step no.	1	2			3		4	5

Table 3 Modified hardware assignment table (after implanting watermark)

ODD C.S.	Operation no.	1	2	7	8	9	12	13
	Allocated hardware	M1	M2	A1	M1	M2	M2	A2
EVEN C.S.	Operation no.	3	4	5	6	10	11	14
	Allocated hardware	M2	M1	A1	A2	M2	A2	A1

Table 4 Modified timing table for non-critical operations (after implanting watermark)

Operation no.	2	3	4	6	7	10	13	8
Control step no.	1	2			3	4	5	

ware assignment before watermark (opn 1 to M2 and opn 2 to M1) is modified to a different hardware assignment (opn 1 to M1 and opn 2 to M2).

In the approach, watermark length can be controlled by varying the vendor signature strength i.e. the number of digits in the watermark embedded in an IP design depends on the size of the signature provided by the vendor. Moreover, the size of the design is also important to ensure all the watermarking constraints corresponding to the chosen signature are embedded. For example, a small size design can not accommodate a large size signature (watermark length). However, a large size design can accommodate various signature strengths. Thus to ensure that successive synthesis do not result in same watermark, signature strength should be accordingly chosen, e.g. for a small size design a very large size signature is undesirable.

The above encoding process is very useful as it enables to covertly insert watermark signature into an IP core during its design process [20]. Since the utility of multi-vendor hardware assignment model during IP core design is obvious, hence it provides a vehicle for the dual-phase watermarking approach to hiddenly embed the watermarking constraints into it. The watermarking constraints based on encoding

thus can be conveniently implemented in a hidden form. Thus an attacker (without the knowledge of encoding rules) will not have any hint where/how the watermark is embedded. The dual-phase watermarking approach does not aim to obscure an IP design to provide security, it only embeds vendor's signature secretly into the design (without disturbing IP functionality and obfuscating any IP design information) to protect the owner, in case of ownership conflict. Thus even if an attacker knows the complete design, its embedded watermark remains invisible. This is because our watermark embedded does not add any extra design component, logic or feature, but only performs local alteration of existing scheduling and hardware assignment, such that it does not hamper IP functionality or appearance. Further, since we employ local hardware re-assignment by exploiting multi-vendor assignment concept and local movement of non-critical path operation with mobility, to embed watermark, thus the dual-phase watermarking approach incurs low design overhead with respect to execution delay and design area. Additionally, the watermark generated through multi-variable encoding satisfies desirable properties such as minimal embedding cost, fault tolerance (as constraints are exclusive and distributed in nature), resiliency against threats (as specified in threat model) and low creation and detection time.

3.3 Process for Embedding Dual-Phase Watermark in IP Design

The following steps are used to embed dual-phase watermark [20]:

1. Schedule the DFG based on **list scheduling** algorithm and user provided hardware configuration.
2. Perform hardware assignment on the schedule DFG.
3. Create the "*hardware assignment*" table for all operation and timing table of "*non-critical operations* ($\mu_m > 0$)" to represent the IP design before embedding watermark.
4. Based on the operation number in each cs. sort the operations in increasing order.
5. Take the signature of the vendor as a combination of 'α', 'β', 'γ' digit's only.
6. For each occurrence of 'γ' digit shift/move an operation of non-critical path by scanning from cs 1 onward (without repeating) such that:
    a. In the immediate next cs. the operation should not has a child.
    b. Shifting/moving of the operation should not violate the user provided resource configuration constraints.
    c. The operation has the maximum mobility value will get priority to resolve conflict (if conflict arises between two or more operations).
7. For each occurrence of 'α' and/or 'β' hardware reassignment is performed in sorted order as per the encoding rules.
8. To represent a dual-phase watermarked IP core design, modify the "*hardware assignment*" table and "*non-critical operations* ($\mu_m > 0$)" table generated in step 3 based on steps 6 and 7.

In the dual-phase watermarking approach we have used classical resource constraint list scheduling algorithm published in traditional HLS papers such as [27–30] where priority function (e.g. mobility based, number of successor operations based) is used to resolve resource contention or conflict among operations. Similar techniques have been used in GAUT tool [31] where authors have used a modified list scheduling using bit width in addition to mobility. Further, contemporary algorithms as soon as possible (ASAP) scheduling has also been used in LegUp tool [32]. Thus the dual-phase watermarking approach uses the realistic list scheduling algorithm adopted from classical sources. Moreover, in watermark embedding process (in [20]), the 1st three steps are pre-watermark stage of the approach, where first step uses list scheduling to generate a schedule (refer to Sect. 5.1 that shows a motivational example of a valid schedule generated using list algorithm). Subsequently step 4 onward watermark embedding process begins where operations are sorted according to their sequence numbers (i.e. name ordering). This sorting is necessary to accommodate the watermark constraints by local alteration of scheduling (only non critical path) and hardware assignment. Therefore, operations are sorted only for inserting signature, not for generating a regular valid scheduling. For the sake of brevity, all the features of authors HLS tools have not been included while demonstrating dual-phase watermarking. However, our HLS methodology/tools (published in [26, 33–36]) combines many features (used in other HLS tools) such as data pipelining, loop pipelining, initiation interval, loop unrolling, loop folding, tree height transformation, logic transformation, redundant operation elimination, loop invariant code motion etc. Thus for preserving succinctness, the dual-phase watermarking method has not been demonstrated using above features.

3.4 Signature Detection

The signature detection for original owner of dual-phase watermarking approach can be achieved in two steps:

1. *Inspection*: The objective of inspection is to collect the relevant information from an IP design (hardware description language (HDL) files) such that the presence of watermark can be identified by only a knowledgeable user (of encoding rules). For instance in the dual-phase watermarking approach inspection can be performed to collect the information of "*non-critical operations* ($\mu_m > 0$)" timing and "*hardware assignment*" details from the source code of hardware description language.
2. *Verification*: The verification of signature is required to validate the presence of vendor signature in the reverse engineered IP design. In order to perform this, vendor's signature needs to be decoded (converted to constraints) using signature encoding rules (mentioned in Sect. 3.2). Finally, the presence of decoded constraints are verified in the reverse engineered design. The design flow of devised signature detection process is presented in Fig. 3.

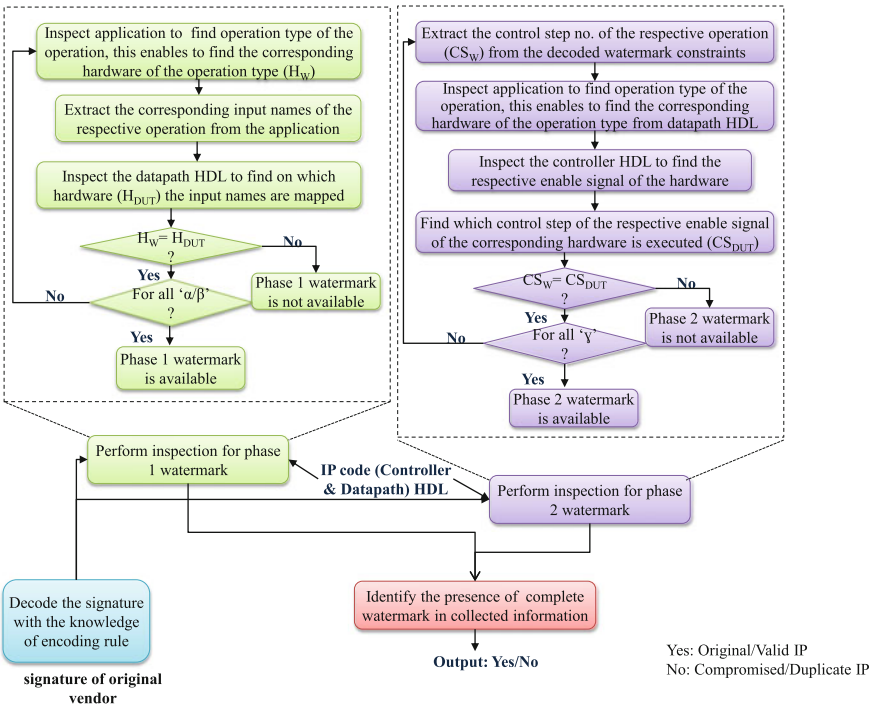


Fig. 3 Signature detection process of dual-phase watermarking approach

4 Motivational Example for Dual-Phase Watermarking Approach

4.1 Motivational Example for Dual-Phase Watermarking Scheme

Figure 4 shows a list scheduled DFG of MESA benchmark based on 2 adders and 2 multipliers which is provided as user input. In control step 1 the number of ready to schedule multiplication operations are 4 however, the number of multipliers available are 2 (according to user resource constraints). In the dual-phase watermarking approach this type of resource conflict is resolved using mobility based list scheduling. As operation 1 and operation 2 has lesser mobility than operation 3 and operation 4 they are scheduled in control step 1. Similarly, for other resource conflicts the operation having lesser mobility gets higher priority to schedule in a control step. Two instances are obtained for each hardware type, by importing one from vendor 1 (V1) other from vendor 2 (V2). The initial (before embedding watermark) IP schedule performs random hardware assignment. The corresponding operation numbers (1–14) of each functional unit appears in the left and the assigned hardware type of

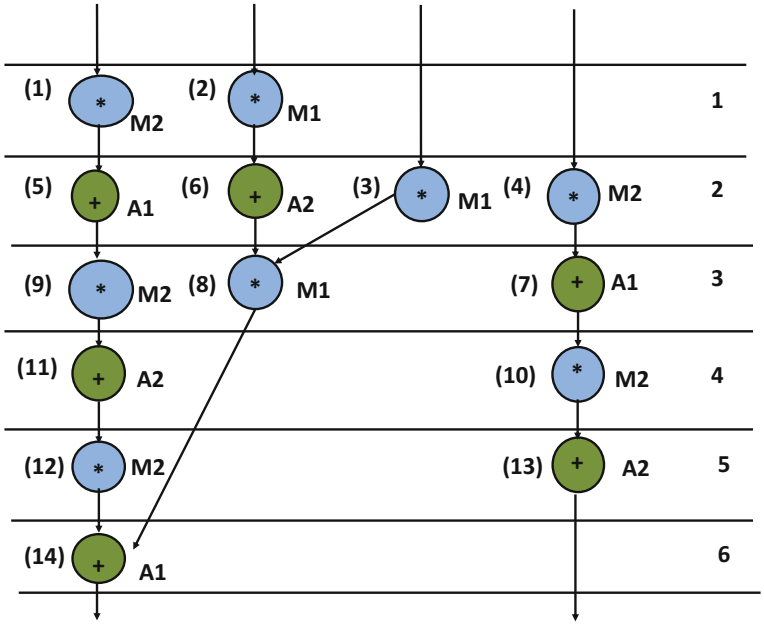


Fig. 4 Scheduled DFG of MESA using 2 adders and 2 multipliers with random hardware assignment before implanting any constraints

each functional unit appears in the right (Fig. 4). For instance, the first operation of cs 1 is marked as operation number (1). 'M2' indicates the multiplier assigned to this operation is imported from vendor type 2. The operations and its corresponding assigned hardware is presented in Table 1. The first and third row of the table represent the operation # of odd cs and even cs respectively and the second and fourth row shows the corresponding assigned hardware. Next step is to create a table containing details of timing information for IP design (before embedding watermark). *Note: We do not include timing info of critical path operations as watermark is not inserted there*. The table consists of list of all non-critical path operations and their respective cs number. The first and second row of the table indicate the operation number and the corresponding cs number respectively. In the table, based on mobility, operations of same cs are reported in sorted order. Next step is selecting a unique vendor signature provided as watermark. Assuming: "$\gamma\gamma\alpha\beta\beta\alpha\alpha$".

The dual-phase watermarking methodology inserts watermarking constraints during scheduling and hardware assignment phases of HLS respectively. According to the encoding rule, a single encoded digit 'γ' shall move an operation of non-critical path with higher mobility (refer Fig. 4) to its immediate next cs. Now as per the rule 6 in Sect. 3.3 operation number 8 is the first eligible operation to move in cs 4 and then subsequently to cs 5 due to occurrence of two consecutive 'γ' digits. The reason is that, other operations of non critical path viz. 2, 3, 4, 6, 7 and 10 does not satisfy the rule 6, while operation number 13 is not eligible because lack of

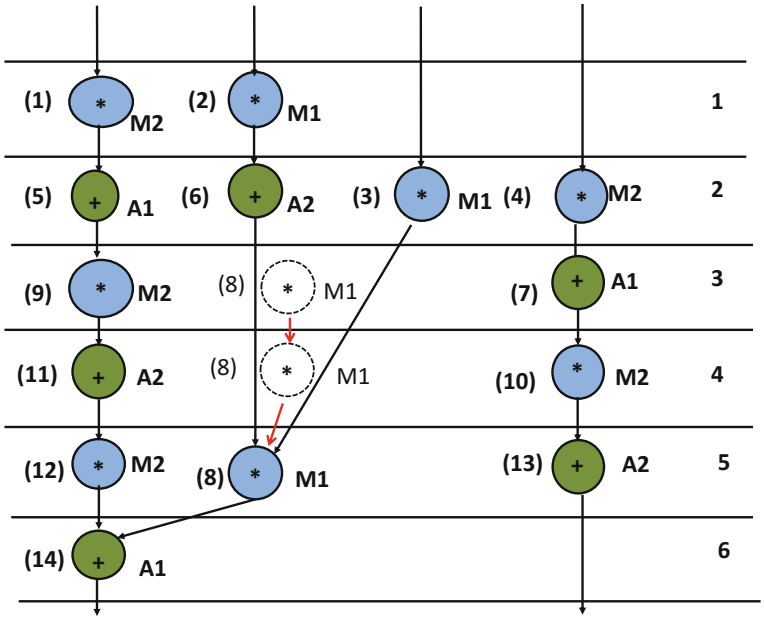


Fig. 5 Scheduled DFG after implanting first-phase of watermark ('γ' digits)

another 'γ' digit in signature chosen. After inserting two 'γ' digits (according to the sample vendor signature) the modified scheduled DFG with first phase watermarking constraints embedded is shown in Fig. 5. The aforesaid scheduled DFG after embedding 'γ' is further used to embeds watermark in the hardware assignment phase to insert 'α' and/or 'β'. Now according to the chosen signature, the 3rd digit of the watermark is α. Therefore, due to this α re-assignment of hardware M1 to operation 1 is performed. Similarly, the 4th digit i.e. β e-assignment of hardware M2 to operation 3 is performed. This process continues for other signature digits. The modified "*hardware assignment*" table and "*non-critical operations* ($\mu_m > 0$)" table after implanting watermark are present in Table 3 and Table 4 respectively. The final dual-phase watermark implanted schedule is presented in Fig. 6. The overhead due to scheduling phase is nil, while hardware assignment phase in minimum. Further, register overhead is nil.

4.2 Properties of Generated Watermark

A watermark embedded in an IP core design must comprise of several desirable properties. The watermarking methodology discussed in this chapter is capable to generate those desirable properties. We discuss these properties achieved below:

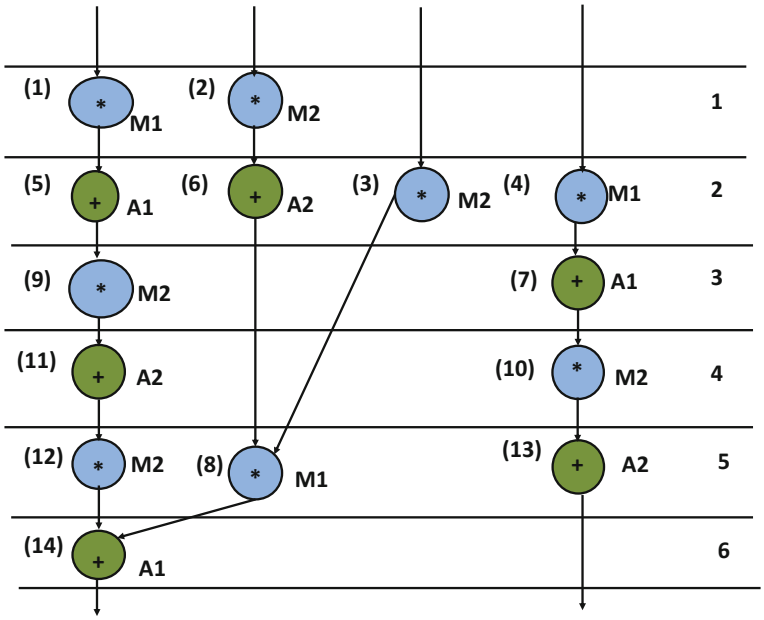


Fig. 6 Scheduled DFG after embedding dual-phase watermark ('α','β','γ' digits)

1. *Low embedding cost*: The watermark generated through dual-phase watermarking scheme incurs low area and latency overhead. Additionally zero register overhead is imposed.
2. *Resiliency against threats*: The watermark is embedded in two different phases using multi-variable encoding scheme. This makes the watermark strong and robust against typical threats such as false claim of ownership, IP counterfeit/forgery and watermark tampering.
3. *Tamper Tolerance*: As the watermark is embedded in two phases and distributed over the complete design, the ownership remains intact even after any tampering or partial removal of watermark (i.e. if the watermarking constraints of phase 1 are removed by an attacker, the ownership can be still proven by the watermarking constraints of phase 2 or vice versa). Additionally, partial removal could also occur if some watermarking constraints of a specific phase is only removed. In such a scenario, the dual-phase watermarking approach is also capable to detect watermark as the remaining watermark constraints are still distributed in the design.
4. *Watermark creation and detection time*: The dual-phase watermarking is fast such that the creation time is very less i.e. in the order of few milli-seconds. Besides, it is simple for a genuine entity (with complete knowledge of encoding rules) to detect the watermark while tough for an outsider.

5 Results and Analysis

In subsequent discussion of this chapter, we term a pre-watermark design (before embedding any watermark constraints) as baseline design. The dual-phase watermarking solution, related approaches [2, 3] and baseline design, all have been implemented in Java and run on AMD processor. Subsequent subsections present the comparative results with [2, 3]. Following are the major reasons for choosing [2, 3] for comparison:

1. Both [2, 3] have integrated watermark in one of the three major phases (i.e. scheduling, hardware and register assignment, binding) of HLS prior to datapath and control unit generation. However, [11] has not integrated watermark in one of these phases, but has embedded in IP control unit. Since the dual-phase watermarking approach has also embeds watermark in two of these major phases of HLS [20] it is more closer to [2, 3] from comparison perspective.
2. Additionally, [2, 3] targets HLS for application specification integration circuits (ASICs), however, [11] targets HLS for field programmable gate array (FPGA). Since the approach also targets HLS for ASICs, it includes similar platform as [2, 3], compared to [11].
3. Finally, the approach in [11] suffers from a limitation that lack of sufficient "temporally free" output slots result into erroneous watermarking process. Whereas, [2, 3] do not suffer from any such type of limitation like the dual-phase watermarking approach.

A 15 nm technology based on NanGate library [37] is used to calculate the design area and execution latency of each design. HLS benchmarks are adopted from [38, 39] has been used in the dual-phase watermarking approach in this chapter.

5.1 Typical Attack Scenarios

In this section of the chapter, we discuss various type of attack scenarios. Assuming, entity 'X' is the owner of a watermarked design (D_w) which entity 'Y' has brought from 'X'. In such a case the following threats may arises [2, 20]:

5.1.1 Unintended Signature Extraction

Entity 'Y' may try to identify an existing information in the watermarked design (D_w) through inspection or reverse engineering and claims as his watermarking constraint. Subsequently, 'Y' can then demand his ownership claiming design contains his signature. In such a case when the design contains both 'X's signature (original) and 'Y's signature (unauthorized) then, the entity having stronger and more meaningful watermark will be considered as the real owner.

5.1.2 Unauthorized Signature Insertion

Entity 'Y' may try to insert his own signature into D_w and claim for ownership. In that scenario, the newly generated watermarked design will contain 'Y's signature on the top of the 'X's signature. In such contradiction, 'X' can establish his authorship easily as 'X's design only contains his signature, whereas, 'Y's design contains signatures of both 'X' and 'Y'.

5.1.3 Tampering Original Signature

Entity 'Y' may try to remove the watermark (partially or fully) from D_w by performing some alterations to create an unauthorized design. However, as dual-phase watermarking methodology distributes additional constraints throughout the complete design in two different phases of HLS, therefore it is extremely difficult to tamper all the watermarking constraints in the design.

5.2 Strength of Protection and Tamper Tolerance Ability

Table 5 presents the probability of coincidence (P_c) of the dual-phase watermarking technique [2, 3]. Probability of coincidence [20] is defined as the possibility of generating the same hardware assignment and scheduling as the one obtained after embedding watermark. The metric for the approach is defined below:

$$P_c = \left(\frac{1}{\prod_{i=1}^{D} N(R_i)}\right)^{w} * \left(\frac{1}{\prod_{j=1}^{G} Mob(O_j)}\right) \tag{1}$$

where w is the total count of α/β digit available in the signature, $N(R_i)$ is number of hardwares of hardware type R_i and D is total types of hardwares. $Mob(O_j)$ indicates the mobility of the non-critical operation j (O_j) and G is the total count of γ digits available in the signature. Probability of coincidence (P_c) for related work [2, 3] is based on information provided in their paper. Table 5 shows the dual-phase watermarking approach have stronger credibility of authorship as lower P_c value is achieved compared to [2, 3]. Further,higher the signature size, stronger the proof of ownership (as P_c decreases).

Table 6 indicates the resilience of the dual-phase watermarking approach, [2, 3] against tampering i.e. tamper tolerance ability. More the watermarking constraints embedded throughout the design, more the difficulty for an attacker to tamper and remove it completely. In [2, 3], watermarking constraints are embedded as extra edges in the colored interval graph. However, in many cases all the watermarking constraints corresponding to the signature strength does not get added as extra edges due to presence of default edges in the design. In other words, number of actual (effective) watermarking constraints added to the design is much lesser than signature strength

Table 5 Comparison of (P_c) as proof of ownership for dual-phase watermarking approach [2, 3] for signature strength: 80 digits

Benchmarks	Probability of coincidence (P_c)		
	Dual-phase	[2]	[3]
2D-ARF	$5.8 * 10^{-76}$	$5.7 * 10^{-73}$	$5.7 * 10^{-73}$
DCT (8-tap)	$1.4 * 10^{-79}$	$5.7 * 10^{-73}$	$5.7 * 10^{-73}$
2D-DWT	$8.9 * 10^{-60}$	$1.2 * 10^{-56}$	$1.2 * 10^{-56}$
EWF	$3.2 * 10^{-60}$	$6.8 * 10^{-49}$	$6.8 * 10^{-49}$
IDCT (8-tap)	$3.3 * 10^{-76}$	$5.7 * 10^{-73}$	$5.7 * 10^{-73}$
MPEG-2 motion vector	$6.2 * 10^{-95}$	$2.0 * 10^{-92}$	$2.0 * 10^{-92}$
1D-JPEG-IDCT (8-tap)	$2.9 * 10^{-89}$	$4.6 * 10^{-87}$	$4.6 * 10^{-87}$
MESA feedback points	$7.3 * 10^{-96}$	$4.6 * 10^{-80}$	$4.6 * 10^{-80}$
MESA interpolate aux	$2.1 * 10^{-116}$	$1.3 * 10^{-101}$	$1.3 * 10^{-101}$
MESA matrix multiplication (4×4)	$1.2 * 10^{-83}$	$1.5 * 10^{-83}$	$1.5 * 10^{-83}$

Table 6 Comparison of tamper tolerance as proof of resilience for dual-phase watermarking approach [2, 3] for signature strength: 80 digits

Benchmark	Actually embedded watermarking constraints			% of actually embedded watermarking constraints		
	Dual-phase	[2]	[3]	Dual-phase	[2]	[3]
2D-ARF	56	40	48	70	50	60
DCT (8-tap)	80	56	64	100	70	80
2D-DWT	51	41	48	64	51	60
EWF	48	27	29	60	34	36
IDCT (8-tap)	80	54	64	100	68	80
MPEG-2 motion vector	80	56	56	100	70	70
1D-JPEG-IDCT (8-tap)	80	55	58	100	69	73
MESA feedback points	80	56	58	100	70	73
MESA interpolate aux	80	50	57	100	63	71
MESA matrix multiplication (4×4)	80	62	66	100	78	83

applied. Thus an attacker has to put much less effort in identifying the signature. On the contrary, for dual-phase watermarking approach, watermarking constraints are embedded in scheduling and assignment phases of the design as local alterations. This technique reduces chances of default constraints being present in the design, thus enabling more watermarking constraints to be effectively added to the design. Thus an attacker has to put much higher effort in identifying the signature by reverse engineering. Therefore, the tamper tolerance ability of the dual-phase watermarking approach is higher than [2, 3].

Table 7 Comparison of dual-phase watermarking approach with baseline with respect to area, latency, embedding cost and cost overhead percentage (%)

Benchmarks	Resource configuration	Area (μm^2)		Latency (ns)		Cost		Cost overhead %
		Baseline	Dual-phase	Baseline	Dual-phase	Baseline	Dual-phase	Dual-phase
ARF	5(+), 3(*)	191.10	209.19	2.67	3.11	0.77	0.87	12.98
DCT	6(+), 3(*)	250.87	263.45	3.95	4.19	0.80	0.84	5
DWT	2(+), 4(*)	162.79	165.94	1.98	2.08	0.78	0.81	3.85
EWF	3(+), 2(*)	184.81	197.39	3.24	3.82	0.85	0.95	11.76
IDCT	5(+), 3(*)	246.15	253.23	3.77	4.16	0.78	0.83	6.41
MPEG	3(+), 8(*)	280.76	287.05	2.44	2.59	0.73	0.76	4.11
JPEG	5(+), 5(*)	747.90	756.55	14.90	15.92	0.72	0.76	5.56
MESA-FP	4(+), 7(*)	370.41	380.63	4.88	4.94	0.71	0.74	4.23
MESA-IA	8(+), 8(*)	644.87	667.68	9.24	9.62	0.65	0.68	4.62
MESA-MM	4(+), 4(*)	526.12	534.77	9.52	9.96	0.71	0.73	2.82

5.3 Embedding Cost Comparison and Design Overhead Analysis

This sub-section discusses the embedding cost evaluation and design overhead analysis. Embedding cost is calculated in terms of area and latency of a watermarked design. Design overhead analysis is performed in terms of design area, design latency and embedding cost.The embedding cost is evaluated based on the following function [3]:

$$C_f(X_i) = w_1 L_T / L_{max} + w_2 A_T / A_{max} \tag{2}$$

where, $C_f(X_i)$ is the embedding cost of the solution for hardware configuration X_i, L_T and A_T indicates total execution latency and total hardware area of the watermarked design. L_{max} and A_{max} indicates maximum execution latency and hardware area, w_1 and w_2 are user specified weight factor,both of them is set as 0.5 to provide equal weightage.

The comparison of dual-phase watermarking approach with the baseline design in terms of area, latency and embedding cost is reported in Table 7. Further embedding cost overhead percentage (%) of the dual-phase watermarking approach compared to baseline design is reflected in the last column of the aforesaid table. As evident from aforesaid table the dual-phase watermarking approach incurs minimal overhead compare to a baseline design (no watermark).

Table 8 reports the comparative analysis between [2, 3] and the dual-phase watermarking approach in terms of design latency, design area and embedding cost of a watermarked design. For most of the benchmarks reductions in design parameters such as area, latency and embedding cost is achieved with respect to [2, 3]. Table 8 also reports the reduction % achieved by the dual-phase watermarking

Table 8 Comparison of dual-phase watermarking approach with [2, 3] with respect to (w.r.t) area, latency, embedding cost and reduction percentage (%)

Benchmarks	Area (μm^2)			Area reduction (%)		Latency (ns)			Latency reduction (%)		Cost			Cost reduction (%)	
	[2]	[3]	Dual-phase	W.r.t [2]	W.r.t. [3]	[2]	[3]	Dual-phase	W.r.t. [2]	W.r.t. [3]	[2]	[3]	Dual-phase	W.r.t. [2]	W.r.t. [3]
ARF	225.71	223.35	209.19	7.32	6.34	3.11	3.11	3.11	0	0	0.92	0.90	0.87	5.44	3.33
DCT	290.98	288.62	263.45	9.46	8.72	4.51	4.51	4.19	7.09	7.09	0.94	0.92	0.84	10.64	8.70
DWT	182.37	180.01	165.94	9.01	7.82	2.43	2.43	2.08	14.40	14.40	0.93	0.92	0.81	12.90	11.96
EWF	209.19	204.47	197.39	5.64	3.85	3.89	3.89	3.82	1.80	1.80	0.99	0.98	0.95	4.04	3.06
IDCT	280.96	278.40	253.23	9.87	9.04	4.34	4.34	4.16	4.14	4.14	0.91	0.89	0.83	8.79	6.74
MPEG	309.85	309.85	287.05	7.36	7.36	2.77	2.77	2.59	6.50	6.50	0.81	0.81	0.76	6.17	6.17
JPEG	783.29	783.29	756.55	3.41	3.41	16.52	16.52	15.92	3.63	3.63	0.79	0.79	0.76	3.80	3.80
MESA-FP	403.44	403.44	380.63	5.65	5.65	4.95	4.95	4.94	0.20	0.20	0.77	0.77	0.74	3.90	3.90
MESA-IA	701.50	701.50	667.68	4.82	4.82	9.74	9.74	9.62	1.23	1.23	0.70	0.70	0.68	2.86	2.86
MESA-MM	554.44	554.44	534.77	3.55	3.55	10.13	10.13	9.96	1.68	1.68	0.75	0.75	0.73	2.67	2.67

Table 9 Comparison of storage hardware between baseline, dual-phase watermarking approach [2, 3] (for 80 digits watermark size)

Benchmarks	Resource configuration	# of storage hardware			
		Baseline	Dual-phase	[2]	[3]
ARF	5(+), 3(*)	8	8	9	8
DCT	6(+), 3(*)	8	8	9	8
DWT	2(+), 4(*)	5	5	7	6
EWF	3(+), 2(*)	4	4	6	5
IDCT	5(+), 3(*)	8	8	10	9
MPEG	3(+), 8(*)	14	14	14	14
JPEG	5(+), 5(*)	12	12	12	12
MESA-FP	4(+), 7(*)	21	21	21	21
MESA-IA	8(+), 8(*)	48	48	48	48
MESA-MM	4(+), 4(*)	24	24	24	24

approach compared to [2, 3] with respect to latency, area and implementation cost. Firstly, reductions compared to [2, 3] have been achieved because both approaches embed the complete signature sequence during register assignment phase of HLS (resulting into higher chances of register overhead incurring more area/latency). Secondly, both aforesaid approaches do not leverage upon the promising concept of multi-vendor model (with different area/delay values) for hardware assignment. Using a single vendor for hardware assignment throughout the design may result in higher design area/delay.

Table 9 reports the number of register required for baseline, dual-phase watermarking approach [2, 3]. It is observed that register overhead of the dual-phase watermarking approach is **zero** for all the tested benchmarks compared to [2, 3], thus demonstrating that the dual-phase watermarking approach requires lower registers than [2, 3].

6 Conclusion

A dual-phase (embedded in same design abstraction level) watermarking methodology during high level synthesis has been presented in this chapter. The aforesaid approach is based on a new embedding scheme (that implants watermark during scheduling and assignment without change in functionality) and signature encoding mechanism. The dual-phase watermarking approach does not add/reduce any new design component while inserting watermark. The dual-phase watermarking methodology achieved an average reduction of design area, design latency and embedding cost of 7% (min = 3.41% and max = 9.87%), 4% (min = 0% and max = 14.40%) and

6% (min = 2.67% and max = 12.90%) respectively compared to two similar [2, 3] HLS based watermarking approaches. Further, stronger proof of authorship is also achieved compared to [2, 3]. In addition, the dual-phase watermarking approach is more robust, scalable, tamper tolerant and easily adaptable to any modern CAD tool.

One of the important aspects of future research could be geared towards development of stronger encoding mechanisms that would encompass more variables to increase tamper tolerance capability. Moreover, development of multi-phase watermark could be an important problem to investigate for the future that could increase the robustness of author credibility further. Additionally, another aspect of improvement could be refining the signature strength such that an optimal balance between strong tamper tolerance and low embedding time could be achieved. Finally, impact of watermark creation time on different type of applications such loop based, non-loop based and nested loop based data flow graphs could be analyzed in future research.

Acknowledgements Under grant no. 22/730/17/EMR-II CSIR has supported this work financially. The authors would like to thank CSIR for the same.

References

1. Castillo, E., Meyer-Baese, U., Garca, A., et. al.: IPP@HDL: efficient intellectual property protection scheme for IP cores. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **15**(5), 578–591 (2007)
2. Koushanfar, F., Hong, I., Potkonjak, M.: Behavioral synthesis techniques for intellectual property protection. ACM Trans. Des. Autom. Electron. Syst. **10**(3), 523–545 (2005)
3. Sengupta, A., Bhadauria, S.: Exploring low cost optimal watermark for reusable IP cores during high level synthesis. IEEE Access J. **4**(99), 2198–2215 (2016)
4. Fernandez, M., Soriano, M., Cotrina, J.: Tracing illegal redistribution using errors-and-erasures and side information decoding algorithms. IET Inf. Secur. **1**(2), 83–90 (2007)
5. Yuan, L., Qu, G., Ghouti, L., et. al.: VLSI design IP protection: solutions, new challenges, and opportunities. In: Proceedings of the 1st NASA/ESA Conference on Adaptive Hardware and System (AHS), pp. 469–476 (2006)
6. Abdel-Hamid, A.T., Tahar, S., Aboulhamid, E.M.: A public-key watermarking technique for IP designs. In: Proceedings of the Conference on Design, Automation and Test in Europe, vol. 1, pp. 330–335 (2005)
7. Roy, J.A., Koushanfar, F., Markov, I.L.: EPIC: ending piracy of integrated circuits. In: Proceedings of the Design, Automation and Test in Europe (DATE), pp. 1069–1074 (2008)
8. Yu, T., Zhu, Y.: A new watermarking method for soft IP protection. In: Proceedings of the International Conference on Consumer Electronics, Communications and Networks (CECNet), pp. 3839–3842 (2011)
9. Nie, T., Zhou, L., Li, Y.: Hierarchical watermarking method for FPGA IP protection. IETE Tech. Rev. **30**(5), 367–374 (2013)
10. Ziener, D., Teich, J.: Power signature watermarking of IP cores for FPGAs. J. Signal Process. Syst. **51**(1), 123–136 (2008)
11. Le Gal, B., Bossuet, L.: Automatic low-cost IP watermarking technique based on output mark insertions. Des. Autom. Embed. Syst. **16**(2), 71–92 (2012)
12. Wu, Y.-T., Shih, F.Y.: Digital watermarking based on chaotic map and reference register. Pattern Recognit. **40**(12), 3753–3763 (2007)

13. Alkabani, Y., Koushanfar, F., Potkonjak, M.: Remote activation of ICs for piracy prevention and digital right management. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 674–677 (2007)
14. Ni, Z., Shi, Y.-Q., Ansari, N., Su, W.: Reversible data hiding. IEEE Trans. Circuits Syst. Video Technol. **16**(3), 354–362 (2006)
15. Marvel, L.M.: Information hiding: steganography and watermarking. In: Javidi, B. (ed.) Optical and digital techniques for information security. Advanced sciences and technologies for security applications, vol. 1, pp. 113–133. Springer, New York, NY, USA (2005)
16. Cox, I.J., Miller, M.L., Bloom, J.A., Fridrich, J., Kalker, T.: Digital Watermarking and Steganography. Morgan Kaufmann, San Mateo, CA, USA (2007)
17. Kufel, J., Wilson, P.R., Hill, S., et. al.: Sequence-aware watermark design for soft IP embedded processors. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **24**(1), 276–289 (2016)
18. Jain, A.K., Yuan, L., Qu, G., et. al.: Zero overhead watermarking technique for FPGA designs. In: Proceedings of the 13th ACM Great Lakes Symposium on VLSI, pp. 147–152 (2003)
19. Cui, A., Qu, G., Zhang, Y.: Ultra-low overhead dynamic watermarking on scan design for hard IP protection. IEEE Trans. Inf. Forensics Secur. **10**(11), 2298–2313 (2015)
20. Sengupta, A., Roy, D., Mohanty, S.P.: Triple-phase watermarking for reusable IP core protection during architecture synthesis. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. pp. 1–1 (2017). https://doi.org/10.1109/TCAD.2017.2729341
21. Rashid, A., Asher, J., Mangione-Smith, W. H., et. al.: Hierarchical watermarking for protection of DSP filter cores. In: Proceedings of the IEEE 1999 Custom Integrated Circuits Conference (Cat. No. 99CH36327), San Diego, CA, pp. 39–42 (1999)
22. Charbon, E.: Hierarchical watermarking in IC design. In: Proceedings of the IEEE 1998 Custom Integrated Circuits Conference (Cat. No. 98CH36143), Santa Clara, CA, pp. 295–298 (1998)
23. Cui, A., Chang, C. H., Zhang, L.: A hybrid watermarking scheme for sequential functions. In: IEEE International Symposium of Circuits and Systems (ISCAS), Rio de Janeiro, pp. 2333–2336 (2011)
24. Rajendran, J., Zhang, H., Sinanoglu, O., et. al.: High-level synthesis for security and trust. In: Proceedings of the IEEE 19th International On-Line Testing Symposium (IOLTS), pp. 232–233 (2013)
25. Karri, R., Rajendran, J., Rosenfeld, K., et. al.: Trustworthy hardware: identifying and classifying hardware trojans. Computer **43**(10), 39–46 (2010)
26. Sengupta, A., Bhadauria, S., Mohanty, S.P.: TL-HLS: methodology for low cost hardware trojan security aware scheduling with optimal loop unrolling factor during high level synthesis. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **36**(4), 655–668 (2017)
27. Thomas, D.E., Lahnese, E.D., Walker, R.A., et. al.: Algorithmic and Register-Transfer Level Synthesis: The System Architects? Workbench. Kluwer Academic Publisher (1990)
28. Heijligers, M.J.M., Cluitmans, L.J.M., Jess, J.A.G.: High-level synthesis scheduling and allocation using genetic algorithms. In: Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '95). ACM, New York, NY, USA (1995)
29. Zoltan, B.: Scheduling algorithms for high-level synthesis. ACAM Sci. J. **5**(1–2), 48–57 (1996)
30. Hwang, C.T., Lee, J.H., Hsu, Y.C.: A formal approach to the scheduling problem in high level synthesis. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **10**(42), 464–475 (1991)
31. Coussy, P., Chavet, C., Bomel, P., et. al.: 'GAUT: a high-level synthesis tool for DSP applications' in High-Level Synthesis: From Algorithm to Digital Circuit. Springer, Germany, Heidelberg (2008)
32. Canis, A., Choi, J., Aldham, M., et. al.: LegUp: an open-source high-level synthesis tool for FPGA-based processor/accelerator systems. ACM Trans. Embed. Comput. Syst. **13**(2), 1–27 (2013)
33. Sengupta, A.: Exploration of kc-cycle transient fault secured datapath and loop unrolling factor for control data flow graphs during high level synthesis. IET Electron. Lett. **51**(7), 562–564 (2015)
34. Sengupta, A., Roy, D.: Protecting an intellectual property core during architectural synthesis using high-level transformation based obfuscation. IET Electron. Lett. (2017). https://doi.org/10.1049/el.2017.1329

35. Mishra, V.K., Sengupta, A.: Swarm inspired exploration of architecture and unrolling factors for nested loop based application in architectural synthesis. IET Electron. Lett. **51**(2), 157–159 (2015)
36. Sengupta, A., Mohanty, S.P.: High-level synthesis of digital circuits in the nanoscale, mobile electronics era. In: IET Book: Nano-CMOS and Post-CMOS Electronics: Circuits and Design, pp: 219–261 (2016). e-ISBN: 9781785610004
37. NanGate 15 nm open library (2016). http://www.nangate.com/?page_id=2328
38. Express Benchmarks (2016). http://www.ece.ucsb.edu/EXPRESS/benchmark/
39. Mohanty, S.P., et. al.: Low-Power High-Level Synthesis for Nanoscale CMOS Circuits. Springer Science+Business Media, LLC (2008). https://doi.org/10.1007/978-0-387-76474-0

Secure Multicast Communication Techniques for IoT

Subho Shankar Basu and Somanath Tripathy

Abstract With the advent of Internet of Things (IoT) and the wide number of applications that it is being applied on, eventually it will surpass the present size of the Internet. But with these number of devices serving so many applications and being reachable remotely over the Internet, they become equally prone towards attacks and vulnerabilities. Hence, efficient and secure mechanisms tailored to such portable devices are to be designed. Most importantly the protocols running on these devices need to satisfy the basic security requirements while consuming minimum resources in terms of memory, bandwidth and power. Also as these devices will be in millions there is an increasing need to design multicast security mechanisms as many of the applications require it. Till date there has been limited contribution towards multicast security with approaches made mainly based on extending the DTLS protocol, which certainly has a number of drawbacks. In this chapter, we discuss major requirements of secure group communication and present different secure multicast communication techniques emphasizing an interesting and effective approach called S-CPABE.

1 Introduction

Internet of Things (IoT) is being talked about across all corners of the world. It has the real potential to dynamically transform the future in a way we live our everyday life. Devices of daily use will have the ability to compute and communicate with its surroundings and integrate with the Internet. This will lead to these objects to sense the environment, take decisions on their own and communicate with the rest of the

S. S. Basu (✉) · S. Tripathy
IIT Patna, Patna, India
e-mail: subho.basu@gmail.com

S. Tripathy
e-mail: som@iitp.ac.in

R. S. Chakraborty et al. (eds.), *Security and Fault Tolerance in Internet of Things*,
Internet of Things, https://doi.org/10.1007/978-3-030-02807-7_3

world. In a sense all these devices will be independent of one another and have the sufficient intelligence to co-exist in the same world. As a result they will assist in all possible ways in different applications that we come across and help in an efficient and economic lifestyle. As for example cars will become driver-less and have the real time knowledge of which route to take depending on the traffic conditions. They will have the ability to communicate with each other and literally will never meet with any accidents. They will also know the best parking options available to them on a dynamic basis and know when to re-fuel or recharge at the optimal time and station. In a word, with the introduction of computation, memory and communication in all such devices, they will begin to behave optimally in all scenarios which would never be possible by normal human beings. As a result humans will have the possibility to obtain absolute optimality with a greater precision at a much reduced cost.

But, with so many possibilities, there are also other parameters which need to be thought of and security is one of them. These devices bridge all types of distance and transparencies as they can sense and gather information, compute on the data and also send the information to the other parts of network. So there could be equal number of vulnerabilities that could spring up in this environment. Say for example a medical wearable device sends a critical heart beat sensor data to a central system. If somehow this device gets maliciously affected and sends wrong information of the heart beats or introduces an intentional delay in sending the data, the concerned patient would be at a life risk. There could be tons of security issues rising from these contexts and hence securing the devices and the network as a whole is of utmost importance in the IoT scenario.

Today's Internet is reasonably secure with the use of security mechanisms and protocols, across systems. Starting from application level security across transport level and down to layer 2 security principles, there is the possibility to provide multi level security mechanisms for the traditional Internet devices. But the significant difference among the traditional Internet and the IoT devices lies in the amount of resources they have. Normal Internet devices like PCs, tablets and smart-phones can make use of reasonable amount of computation, memory and communication because they have no critical constraints on power consumption and can be power plugged or recharged frequently as they are within human reach. On the contrary, IoT devices will be in billions scattered everywhere and it would not be possible for them to have human maintenance and sometimes even reach (remote mountains, forests for example.) So essentially their lifetime is limited to a single non-rechargeable battery. Another constraint comes with their requirement to have low manufacturing costs which requires them to have small form factors leading to smaller processors, memory and radio chipsets. So due to these two basic constraints of power consumption and form factor, the IoT devices are incapable of running heavy algorithms and protocols, whether be it for a normal network stack or for security purposes. Hence the protocols running on them has to be lightweight. The challenge boils down to designing efficient lightweight network and security protocols for these IoT devices so that as far as possible they can maintain the same level of communication and security requirements with the traditional Internet but with the constraints of power and

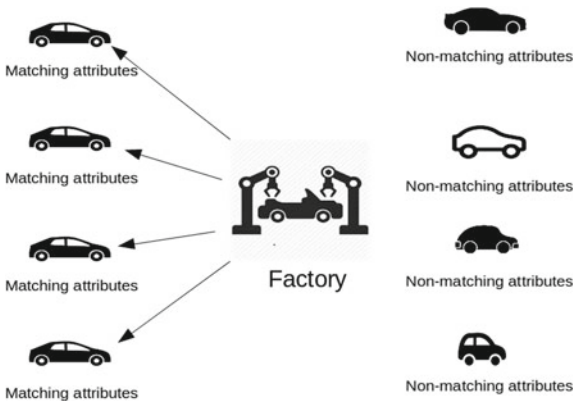


Fig. 1 Firmware upgrade in a batch of cars through secure multicast

form factor. And particularly this is a challenging task in security as in general there are lots of computations and communications happening in traditional protocols.

The requirements of security are also different in different contexts, and many applications require multicast or broadcast security mechanism. Particularly in the context of IoT as there will be cheap sensor devices scattered everywhere with fetching environmental data from time to time, addressing them individually might not make too much sense. Thus there should be efficient multicast security protocols to be able to run on these devices. Consider a simple example as in Fig. 1 for the case of a firmware upgrade in a batch of cars from a certain company manufactured in a certain year. In this case the mechanism has to be secure so that the update happens correctly. The upgrade requirement is targeted for a set of cars with some particular attributes in terms of the manufacturing year, the generation, engine model, etc. This can of course be done one at a time in a unicast fashion but will require a lot of time and energy. An efficient way would be to transmit a multicast message for the destined cars. So we see the necessity of multicast security as an increased need in the IoT scenario which facilitates secure communication among the members in a group.

2 Secure Group Communication

In a typical group communication system there are five different operations namely Initiate, Join, Leave, Partition and Merge [1]. First of all it is required to form a group with a group initiation with the Initiate operation. Then this group is visible to different members who might show interest in joining the group through the Join operation. This joining procedure might also require some authentication process with a group controller. A member might also want to leave a group at any instant of time through the Leave operation. Thus the joining and leaving of a member into or from a group is highly dynamic. With each change in the group dynamics

the group members have to have a fresh re-keying mechanism to preserve forward and backward secrecy. It might also happen that a group is partitioned into multiple subgroups and each subgroup has their own set of keys. The subgrouping is performed through the Partition operation. Subgrouping increases efficiency in the sense that sometimes there may be limited communication among some members of the group so that they do not need to get involved in the bigger group. Members forming a very stable sub-group are thus not affected by members from unstable groups with frequent join and leave. But eventually different groups may be combined back to a bigger group through the Merge operation. Different types of network have different necessities for group communication. It is thus important to choose the appropriate group communication mechanism for better efficiencies in terms of re-keying factors, computations and power consumption due to communications. The major security requirements and services in group communication are as follows.

- **Group Key Management**

The central importance in a group communication system is the Group Key Management (GKM). The security of the entire group depends on the strength of the group key management, based on the cryptographic principles and the protocols used in the design. There can be various protocols for the design. The most common architecture is to have a group controller which generates the group key and distributes that to the individual members securely. This process has to be repeated for each re-keying when a member joins or leaves the group. It could be better for systems where there are not too many members joining or leaving the system frequently. The other way of designing the group key is where all the members contribute together to generate the group key, which is rather more complex. But whatever the process might be GKM is required to meet the following needs.

  – The key generation should be robust and secure.
  – The key should be distributed to only the valid members in the group and in a secure way.
  – Each time a member joins or leaves the group, re-keying of the group key is necessary. Those keys which are dependent on the group key should also be re-computed.
  – The re-keying should be equally secure.
  – It should be computationally hard for an intruder to generate the current group key.

In addition to that, the group key should also have a lifetime of its own so that even if there has not been a necessity for a re-keying due to group dynamics, the re-keying should be done if the time exceeds the lifetime of the group key.

- **Group authentication**

In a group communication system, authentication of both the members participating in the group as well as the messages sent are necessary. This leads to the prevention of identity related attacks like spoofing. Normally a certificate from a certifying authority is used to prove the identity of a user and a message in the system. Expired

certificates are also compiled into a revocation list which is then distributed to notify all the members.

- **Group authorization and access control**

In a unicast communication system with access control, the recipient having the decryption key has full control normally referred to as a single access privilege. But in a group communication system different members of the group may have different access privileges referred to as the multiple access privilege. Thus members who have the correct access privileges for a flow can only access the content.

- **Group accounting and non-repudiation**

All group transactions need to have an accounting measure so that there could not be the possibility of non-repudiation problems. An entity within the group is normally assigned the task of maintaining this accountability, so that in case of a dispute it can derive the corresponding proofs of a sender or a message. In general the group signature and the member certificate are used to authenticate the sender and the message.

- **Group privacy and anonymity**

Messages sent within a group need to have the information about the sender and the message be possibly anonymous, so that no extra information could be derived from a message by an intruder. Different encryption techniques can be used to achieve the same.

- **Group message integrity and confidentiality**

Message integrity within a group confirms that a message has not been modified in the transit from the sender to the receiver. In a group communication this is highly important since multiple members in the group have the decryption key. In the event that one of the members are compromised, it can modify the original message and transmit a fresh version. Message integrity can be achieved by hashing techniques. Confidentiality is also important and signifies that only the authorized members within the group have the possibility to make out meaningful data from a message. Different encryption mechanisms can be used to ensure the confidentiality of a message within a group.

- **Group survivability and availability**

It is important for a group to sustain its survivability against DoS and flooding attacks from an attacker. An attacker may target the group controller with flooding messages so as to limit the legitimate members to reach the controller. Group availability ensures that an authorized member of the group can always communicate within the group by using restricted group resources. Any violation of the principles should be promptly detected and appropriate actions are to be taken. For example the impact of a flooding attack can be mitigated by authenticating sources that generate flooding packets with an early detection of massive number of packets from a single source. Once such an attack has been detected further communication with that source can be immediately dropped.

3 Secure Group Communication Approaches

A number of approaches for secure group communication have been proposed by various authors. But the requirement and the motivation of these protocols and mechanisms revolves around the basic concepts of the group key formation, re-keying mechanisms and trying to optimize the performance as per the network and communication patterns. A few of them are discussed here. Also we discuss later how the Attribute Based Encryption mechanism is used in our approach S-CPABE [2]. S-CPABE is so versatile in terms of flexibility and maintaining knowledge information from past on the nodes that it can perform in a highly optimized fashion, since the target for IoT applications is minimizing power consumption on the end nodes.

Some key hierarchy based principles have been proposed to optimize keying requirements. DeCleene et al. [3] proposed an algorithm where the operational area has been partitioned into multiple sub-areas. Each of the sub-areas has their own area key, but the whole operational area has a network wide common group key. The area key is used to encrypt the group key. This is useful in a mobile network scenario so that nodes that are less mobile in a particular area can make use of the same area key for a longer period compared to the nodes that are frequently moving between areas with the requirement to change the area key most often. The concept has been further extended to keep a history of the area keys, so that when a node re-enters a previously visited area, the corresponding area key is re-used instead of generating a fresh one to reduce time and computation. A topology matching key management technique has been proposed by Sun et al. [4] where a tree structure of the keys are managed according to the cells in a wireless network. The re-keying happens when a node moves from one cell to another and managed by a key distribution centre (KDC). The communication overheads are low and shows a reduction of as much as 80% as compared to topology independent key management (TIKM) schemes. Gupta and Cherukuri proposed three schemes namely single session key (SSK), different session key (DSK), and a combination (HYBRID) [5]. In the SSK scheme the members are given the same session key, while in the DSK scheme the members hold different session keys based on their locations. The HYBRID scheme is a mix of the SSK and the DSK so that members that are within the same location for a sufficient long time hold the same session key whereas members that change their location more frequently hold different session keys. Westerhoff et al. [6] proposed a decentralized scheme called MOMBASA, which is a multicast based approach to achieve low latency for member handoffs with minimum packet loss and secure protocol operations. Each of the member nodes registers themselves with a Mobility-Enabling Proxy (MEP) and communicates with the MEP in a unicast manner. The MEP is in charge of communicating in a multicast manner with the group. This proxy mechanism of MOMBASA shows that it is protected against various attacks by using three security components namely packet filtering at the access network boundaries, deployment of a AAA infrastructure and limiting DoS attacks.

We present two Secure group communication mechanisms in the subsequent sections. Both the approaches use the secret sharing mechanism of Shamir, due to its flexibility in a multicast environment.

3.1 ECC-Based Secure Group Communication

The scheme proposed by Porambage et al. [7] is a group security scheme based on an Elliptic Curve Cryptography (ECC) variant. Here the initiator is a member of the group and starts the communication, and assumes that it has all the identities of the members in the group for whom the message is destined. This scheme operates through the following steps.

- Step 1 The initiator I selects the sensors with their ids which will together form the multicast group, by defining N the group size, and U = $\{U_1, U_2, \ldots, U_{(n-1)}\}$, the set of receivers. It then generates a random number $r_i \in Z_p^*$ and calculates $R_i = r_i.G$ where G is the base point generator with prime order p. The broadcast message is then created using the initiator's public key $Q_i = d_i.G$, R_i and U, where d_i is its private key and U is the set of receivers. The message $\{Q_i, R_i, U\}$ is signed and broadcast to the entire network, announcing the initiation of the multicast communication.
- Step 2 When a receiver receives the broadcast message, it verifies from the set U whether it is one of the recipients. If it is then starts processing the message further otherwise drops it. The signed message also determines the integrity of the message. Each of the receivers j then generate a random number $r_j \in Z_p^*$ and along with R_i calculates $R_{ij} = r_j.R_i$ and $R_j = r_j.G$. Then R_{ij}, the private key of j d_j, and the initiator's public key Q_i are jointly used to generate the secret EC point $S_j = d_j.Q_i + R_{ij}.U_j$ then computes $Auth_j = \mathrm{h}(S_j||R_{ij}||U_j)$ and sends $\{R_j, Q_j, Auth_j, U_j\}$ to I.
- Step 3 The receivers respond to the initiator in a unicast fashion. If there is a loss of response from any of the receivers after a retransmission timeout, the initiator re-sends the message again. After the initiator receives the message from the responder U_j, it calculates the EC point S_j^* from $R_{ij}^* = r_{ij}^* G$ (here, $r_{ij}^* = r_i r_j mod p$), $S_j^* = d_i.Q_j + R_{ij}^*$. The initiator I then also checks for $Auth_j = h(S_j^*||R_{ij}^*||U_j)$. If the verification is successful it proceeds to the next step, otherwise it discards the message and re-sends the same multicast initiation message to the particular nodes. In the event that the request fails again, the particular node is removed from the multicast group.
- Step 4 The EC point $S_j = (x_j, y_j)$ as calculated by I for all the group members is encoded as the point (u_j, v_j), where $u_j = h(x_j)\ and\ v_j = h(y_j)$. A random value k is then generated as the multicast group key. Then the initiator interpolates the polynomial f(u) of degree (n − 1) by using Lagrangian interpolation method. The polynomial f(u) passes through n points $(0, k)$, $(u_1, V_1), \ldots, (u_{n-1}, v_{n-1})$ and generates (n − 1) additional points $P_1, P_2, \ldots, P_{(n-1)}$. The new auth code is then

calculated as $Auth = h(k||r_i||P_1||\ldots||P_{(n-1)})$. The initiator then broadcasts the message $Auth$, $P_1, \ldots, P_{(n-1)}$ with the digital signature.

- Step 5 On receiving the second broadcast message, the responder verifies the signature and finds (n − 1) points. The responder U_j uses S_j to compute (u_j, v_j), and having those n points enables it to reconstruct the polynomial f(u) and derive the group key k by Lagrangian interpolation method with k = f(0). U_j also verifies whether $Auth = h(k||r_i||P_1||P_2||\ldots||P_{(n-1)})$, and if correctly verified the key is authenticated.
- Step 6 Each node then sends an acknowledgement with h(k, Q_j) to the initiator and this confirms that the nodes have the authenticated multicast group key k.

According to the description of the problem, a multicast communication can be replicated by using multiple unicast communications, but that is not very effective. The approach of S-CPABE uses the basics of IP multicast for multicast communication, but for security it uses CPABE encryption to encrypt packets. So essentially all the receivers which indeed satisfy the access structure as defined at the source will only be able to decrypt the packet. In raw CPABE, each individual receiver decrypts the packet by using the decryption functionality, which for constrained devices is not practical. In some way there should be a way to refrain these devices from performing heavy cryptographic operations like in CPABE. Moreover in an IoT environment there will be lots of nodes which will satisfy the access structure in the same manner due to similar device characteristics and attributes. This will lead to the decryption phase of the CPABE for these devices to be exactly similar and hence wastes power due to redundant scenario. And since the decryption phase is quite cryptographic heavy, the cumulative power requirements due to redundancy will be quite huge in a relative aspect. This additive power consumption in an IoT environment with hundreds and thousands of nodes is a huge figure and needs to be considered seriously. Next we describe in details a related approach of using Shamir's secret sharing using Attribute Based Encryption (ABE) technique that we have used to design the mechanism of S-CPABE.

3.2 S-CPABE *for Secure Multicast Communication*

In this section we present *S-CPABE* (Segregated CPABE) which segregates CPABE to push the intensive computations to the gateway for reducing the computational burden from low-power end devices. S-CPABE uses ABE cryptographic mechanisms, particularly Ciphertext Policy Attribute Based Encryption (CPABE), so for better readability Attribute Based Encryption is precisely described next.

3.2.1 Background of ABE Mechanisms

This section briefly touches the basics of Attribute Based Encryption (ABE) mechanism for better understanding of the underlying concepts of S-CPABE. ABE cryptographic algorithms are a generalized form of Identity Based Encryption (IBE) [8] where they have an identity associated to each and every member taking part in the secure message transfer. As for example an email-id or a unique personal identification number, etc. can be utilized as the unique identity. The public key of the recipient is derived from the public parameters publicized globally along with the individual identity. Hence essentially it is not required to transfer the public key of the receiver before the actual encrypted message is transmitted. An ABE mechanism requires each receiver to have a set of attributes which defines the characteristics or the identity of it. In essence there exists two types of ABE, named CPABE [9] and KPABE [10]. CPABE has been chosen in this work according to which the sender encrypts the packet using the public parameters and an access structure which needs to be satisfied by the receiver for the decryption. Every recipient possesses a private key generated by the Private Key Generator (PKG) and this key is utilized to decrypt the message, with the condition that it indeed satisfies the access structure which the sender has sent. The entire concept is rooted on Shamir's secret sharing mechanism [11]. The key at the root in the access structure is broken down recursively down to the leaves at the sender's side. Conversely at the receiver's side the mechanism is reversed and the root secret is re-generated bottom-up from the leaves recursively by the usage of Lagrange's polynomial interpolation. The entire process consists of four main phases namely Setup, Keygen, Encrypt and Decrypt as discussed in [9].

Setup(k)—(P, MSK): In this phase, the PKG takes the security parameter k as input and outputs a public parameter P and a master secret key MSK. The public parameter P is multicasted to all nodes taking part in the group communication, while the MSK is preserved as a secret at the PKG only.

Keygen(MSK, S)—D: This phase enables the PKG to authenticate the individual members of the group. After that it takes as input the MSK and the set of attributes S possessed by the member, and outputs a private key D. PKG then sends this D to the corresponding member over a secure channel.

Encrypt(P, T, Msg)—CT: In this phase, the sender outputs the ciphertext CT, using the public parameter P, the access structure T which should be satisfied by the target receiving nodes, and the message Msg.

Decrypt(P, CT, D)—Msg: In this phase, first of all a member needs to satisfy the access structure T. If satisfied it takes as input the public parameter P, the ciphertext CT and its private key D to retrieve the original message Msg back.

An end-to-end security is achieved in CPABE where the end nodes are solely responsible for the decryption and none of the intermediate nodes in the pathway are responsible or capable to decrypt correctly. Hence trivially it makes good sense if these decryptions are pushed off to the gateway, as the gateway is much powerful to perform computation intensive operations. But it does not satisfy the requirement for end-to-end security unless the protocol has been modified correctly. So in essence what has been proposed is a partial decryption approach in the gateway by segregating

the CPABE decryption mechanism. The gateway gets the responsibility to handle major part of the decryptions but should fail to perform the end-to-end decryption. The data obtained from partial decryption is passed on to the end nodes, which are then required to complete the remaining part of the decryption. In this way it prevents the gateway from retrieving the plain text (message or key) to maintain end-to-end security, and at the same time takes off the burden from the end nodes to perform the major redundant computations thus saving energy.

S-CPABE uses all the operations including Setup, Keygen, Encrypt of CPABE [9]. Only the Decryption phase has been divided into two phases to be carried out (a) partly by the proxy or gateway and (b) completed by the end nodes in the last step. The operations and calculations has been discussed as follows.

Setup: This phase is carried out by taking as input the security parameter and it outputs the public parameter PK and the master secret key MK. Typically $PK = G_0, g, h = g^{\beta}, f = g^{1/\beta}, e(g, g)^{\alpha}$ and MK is (β, g^{α}). e is the Bilinear mapping and G_0 is a bilinear group of prime order p with generator g. α and β are two random numbers in Z_p, the set of positive integers.

Keygen(MK,S): The Keygen phase enables the PKG to generate the private key for every group member by the usage of the master key MK and the set of attributes S. The key is computed as

$$\begin{aligned} SK = (D = g^{(\alpha+r)/\beta}, \\ \forall j \epsilon S : D_j = g^r . H(j)^{r_j}, \\ D'_j = g^{r_j}). \end{aligned} \tag{1}$$

Here r and r_j are random numbers in Z_p and H is a function which maps an attribute to an element in the group.

Encrypt(PK,M,T): The sender then encrypts the message M by using the public parameter PK and the access structure T, as follows

$$\begin{aligned} CT = (T, \widetilde{C} = Me(g, g)^{\alpha s}, C = h^s, \\ \forall y \epsilon Y : C_y = g^{q_y(0)}, \\ C'_y = H(att(y))^{q_y(0)}). \end{aligned} \tag{2}$$

Here s is the root secret of the access tree, $q_y(0)$ is the secret at node y, and Y is the set of leaves in T. All the other symbols including $\widetilde{C}$, C_y, C'_y etc. are various representations for mathematical expressions used in the entire process.

Decryption Phase I—Partial decryption at the gateway:

The same mathematical construct as that of CPABE [9] is followed in S-CPABE for the decryption. The function is defined as:

$$\begin{aligned} DecryptNode(CT, SK, x) &= \frac{e(D_i, C_x)}{e(D'_i, C'_x)} \\ &= e(g, g)^{r q_x(0)}. \end{aligned} \tag{3}$$

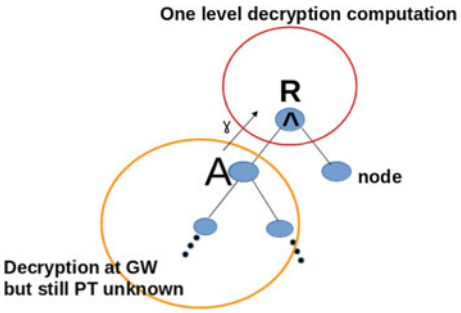


Fig. 2 Segregation in S-CPABE

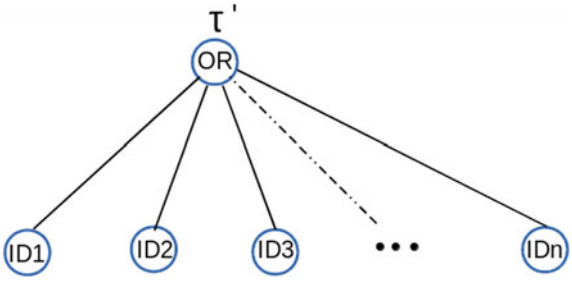


Fig. 3 Access tree for CT^1

The DecryptNode function only operates successfully if it satisfies the node attributes in the access structure, otherwise returns $\perp$. It is to be noted that for this function, the outcome for each node is the same expression value $F_x = e(g, g)^{rq_x(0)}$. In phase I, we continue performing this on the gateway, until we reach one level away from the root, namely the node A on the left hand side of the access tree as shown in Fig. 2. Thus the gateway can recursively compute F_A as

$$F_A = \prod_{Z \epsilon S_x} F_z^{\Delta_i, S'_x(0)}, i = index(z),$$
$$S'_x = \{index(Z) : Z \epsilon S_x\}$$
$$= e(g, g)^{rq_A(0)}. \quad (4)$$

As this phase ends, F_A is transferred securely to the end nodes as $\gamma = (CT, CT^1)$ by another CPABE level (CPABE-2). The ciphertext CT^1 is as in Eq. 2 with corresponding access structure τ' as shown in Fig. 3 which would be the logical OR of the node-ids of all the authorised nodes. This is to prevent decryption by the unauthorized end nodes.

Decryption Phase II—Completion of the last decryption at the end nodes:

The targeted end nodes satisfying the access tree τ' then decrypt CT^1 to obtain F_A. Moreover, each of the destined end nodes satisfy the attribute "node" in Fig. 2 so that they satisfy the access structure after having obtained F_A. This attribute called "node" just acts as a marker for the last end node to do the final decryption.

It just separates this node from other intermediary nodes. Hence every end node has the attribute "node", but it is absent in all of the intermediary devices whether gateways, proxies, routers, etc. This ensures preserving the requirement for end-to-end security just like CPABE. The nodes can therefore perform the only final step of the decryption computation of CPABE at the root node R resulting in significant reduction in computation.

$$\begin{aligned} F_R &= \prod_{Z \epsilon (A,node)} F_z^{\Delta_i, S'_x(0)} \\ &= \prod (e(g,g)^{rq_{parent(z)}index(z)})^{\Delta_i, S'_x(0)} \\ &= (e(g,g)^{rq_R(1)})^{\Delta_1 S_1(0)} \cdot (e(g,g)^{rq_R(2)})^{\Delta_2 S_2(0)} \\ &= e(g,g)^{r[q_R(1)\Delta_1 S_1(0) + q_R(2)\Delta_2 S_2(0)]} \\ &= e(g,g)^{rq_R(0)} \\ &= e(g,g)^{rs}. \end{aligned} \tag{5}$$

The final group key or message M is computed as

$$\begin{aligned} &\widetilde{C}/(e(C,D)/F_R) \\ &= \widetilde{C}/(e(h^s, g^{(\alpha+r)/\beta})/e(g,g)^{rs}) = M. \end{aligned} \tag{6}$$

It is to be noted that each end node executed this process, earlier once more to obtain F_A from CT^1. Thus, two one-step CPABE decryptions are necessary for each end node to obtain the plaintext (group key).

3.2.2 Operation Principle of S-CPABE

The proposed S-CPABE operations for secure multicast in IoT are as shown in Fig. 4 and briefly explained as follows:

- In the deployment phase before providing any service, the nodes with their attribute set, gets registered to the gateway (proxy).
- For the security in the group, the user generates a random key and encrypts it using S-CPABE. The server can then use this key for further encrypted messages destined for the user. Similarly, for sending sensitive data server group, the user sends the key encrypted by CPABE. The messages are then encrypted using this random key and a symmetric encryption algorithm. The access structure needs to be formed as A AND node, where "node" is the attribute that is present only in the end nodes for the last step decryption, but not in the gateway. So the gateway can only partially satisfy the access structure till node A, which can be further nested recursively down the access tree.
- The gateway does the partial decryption to one level until the root and sends it to the group of authorised servers/nodes for the last step decryption.
- The end nodes then perform the last step decryption to get the key. This key is then the same key used by the sender and receiver for further encryption and decryption purposes.

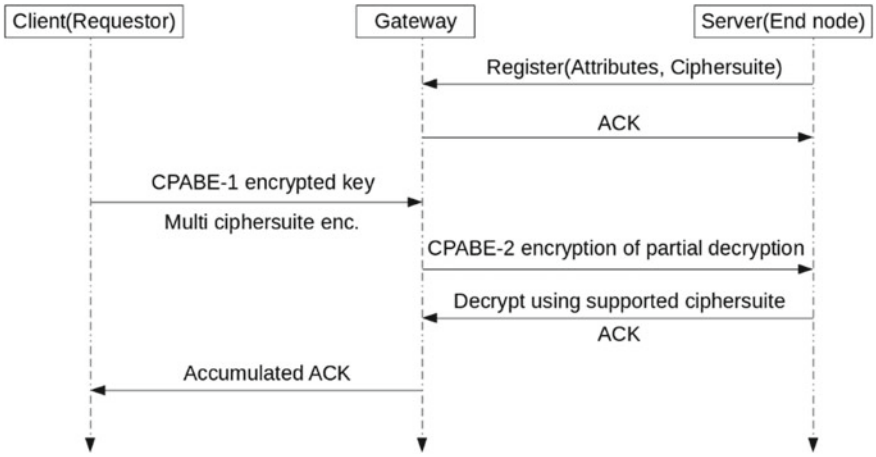


Fig. 4 Flow diagram of S-CPABE

3.2.3 Cost of Key Management in S-CPABE

To manage keys for S-CPABE is rather simple. In ABE based systems, nodes just need to contact the PKG once to get their private key. So the PKG can be entirely removed from the system if there are no new nodes joining the group. This first initialization can be done during the network bootstrapping phase using any protocol, or a pre-shared key (PSK) approach can be considered without the loss in security or generality. The actual encryption and decryption phase is even more lightweight as compared to other asymmetric protocols, as in S-CPABE the receivers need not distribute their public keys to the senders at the start of the process. The sender can derive the public key of the receiver from the public parameter PK and the access structure, thereby reducing the number of messages for the public key distribution. Also the sender does not need a detailed knowledge apriori of the number of destined receivers, their specific identities, certificates, etc. No states are required to be maintained unlike DTLS based systems. The sender can generalize a target group with their attribute set and the access structure, making it very much suited for multicast IoT environments.

3.2.4 Security of S-CPABE

The proposed architecture is based on the idea of segregating CPABE, and hence the security goes in hand with that of original CPABE [9].

- *Curious Gateway*: The architecture considers that the gateway is a trusted one and performs all the methodologies as described, but also curious to decipher the plaintext (original group key or message). It fails to do so just from the CT as

essentially it does not satisfy the access structure shown in Fig. 2 because it lacks the attribute "node".

- *Curious Revoked node*: It is simple to deny access to a subgroup of nodes that are not targeted for a particular message. As shown in Fig. 3, the gateway generates the access tree using the authenticated targeted nodes but not including the revoked ones which can also have the "node" attribute, but ensuring that they do not satisfy the sender's access structure. Thus the revoked nodes fail to obtain F_A since they cannot satisfy the access structure τ'. Therefore, the revoked nodes are unable to retrieve the plaintext.
- *Curious New node*: S-CPABE also manages easily new nodes joining the group. A new attribute (att) is assigned to this new node. The sender creates the access tree like the one in Fig. 2 except that the sub tree to the right hand side of the root node would be modified to (node OR att). This enables the new node to retrieve the plain text further. But since this new node neither satisfies the access structure in Fig. 2 nor Fig. 3, it is thus unable to decrypt the previous messages.
- *Authentication*: S-CPABE inherently supports authentication as the PKG takes the sole responsibility to authenticate the node and delivering its private key. The only difference with respect to CPABE is that the gateways also need to authenticate themselves so as to ensure that the forwarded messages from the gateway are also genuine ones.

3.2.5 Computational Analysis

A platform has been formed to study the performance of S-CPABE and compare it with traditional CPABE. The end nodes have been considered as individual Linux machines with identical hardware and software capabilities and different implementations have been compared on the same setup. Hence without loss of generality it can be assumed that if the proposed model performs better in comparison to others in the PC environment, then it is also expected to hold true in constrained environments. As the primary concern is the overhead for decryption for the end nodes, the experiment focuses on the segregated decryption mechanism of S-CPABE as compared to other protocols. As expected from the theoretical overview, S-CPABE shows better results as compared to CPABE. The results show that the approach of S-CPABE is indeed an indication that ABE mechanisms can indeed be run on constrained devices if dealt with properly.

The hardware setup consists of a PC with Intel Core i3 1.7 GHz CPU and 4 GB RAM running Ubuntu 14.04 LTS. For the softwares, cpabe and libbswabe toolkit [9] have been used, running on top of the pbc and gmp libraries. Valgrind utility of Linux [12] has been used for measuring performance aspects. The main optimizations for S-CPABE was done in the bswabe_dec function in the file core.c. The main intention was to see the efficiency gain in the decryption strategy in the end nodes.

Table 1 displays the results from valgrind on the test setup for CPABE operations for Setup, Keygen and Encrypt. The latency is represented in (Million) instructions and the memory in (Kilo) bytes. PKG performs the setup and keygen phases only

Table 1 Valgrind test results for CPABE

Phases	Latency (in instructions)	Memory (in bytes)
Setup(PKG)	105.5 Mi	8.336 KB
Keygen(PKG)	2 attrs = 220.3 Mi	12.9 KB
	4 attrs = 440.4 Mi	14.73 KB
	6 attrs = 661.3 Mi	16.68 KB
Encrypt(Sender)	3 nodes (a and b) = 160.8 Mi	13.30 KB
	7 nodes (a and b)	
	v (c and d) = 267.9 Mi	15.34 KB

Table 2 Valgrind comparative results

Protocols	Latency (in Instructions)	Memory (in Bytes)
DTLS	200 Mi (Server) ‖ (Client) 180 Mi ‖ 20 Mi	223 KB (almost the same for server and client)
CPABE	ormid50ˆ node dec = 158.4 Mi 5 node dec = 255.8 Mi	54.45 KB 18.12 KB
S-CPABE	Std. 3 node dec = 155.7 Mi	16.97 KB
tinydtls	Server = 156 Ki Client = 201.9 Mi	360 Bytes < 10 KB 2.859 KB < 10 KB

once. The PKG is considered to be unconstrained, so the memory and latencies for these two phases are not of importance to the constrained devices. The main focus of this paper has been on the decryption phase as this is the major drive for IoT use cases where multicast messages are pushed from a non-constrained device to a bunch of constrained devices.

Table 2 shows the metrics for decryption times and memory requirements for S-CPABE and CPABE. The tunnel setup times and memory requirements for DTLS and tinydtls have also been shown here. The comparison needs to be done carefully as DTLS is a symmetric key mechanism whereas ABE is an asymmetric one. DTLS has the notion of creating and maintaining tunnels between the source and destination to exchange key parameters. DTLS handshake is time taking and requires heavy bandwidth until the formation of the tunnel and establishment of DH key, but afterwards the mechanism is lightweight as it is symmetric. So DTLS protocol is handshake-heavy but computation-light. On the contrary, ABE mechanisms follow a reverse approach. They are based on public key encryption mechanisms and are thus computation-heavy but requires no handshake as in DTLS and thus communication-light. The focus was to keep both the approaches on the same plate as far as possible in order to get a good comparison.

3.3 Discussion

S-CPABE provides a very flexible and efficient model as compared to multicast DTLS. First of all for DTLS needs as many tunnels as the number of members in the group, and these tunnels need to be maintained till the very end until the member leaves the group. This apparently heavy task is totally absent in S-CPABE. A change in the group members results in transmission of the new group secret key individually through the tunnels to each end devices, and for frequent changes in group members this is a huge task. In S-CPABE the single multicast message with a new access structure is sufficient to meet the need. Next for the TGK (TEK Generation Key) distribution DTLS requires a handshake, but for S-CPABE a one time tunnel is sufficient or even a pre-shared key approach should be fine. This reveals the flexibility of S-CPABE over DTLS. In a multicast IoT environment it is difficult to address each and every node individually in the network. Rather, it is much easier to classify nodes in the network and send a message destined for them. ABE mechanisms give the intuitive sense to this approach rather than DTLS. DTLS being a stateful protocol is not scalable but ABE mechanisms being stateless, S-CPABE is highly scalable. Mobility handling is also difficult in DTLS due to the overhead of the maintenance of the tunnels from the group controller. But S-CPABE has not any of those requirements and thus suits perfectly to IoT mobility use cases. In DTLS the group controller also becomes highly vulnerable as it contains all the session keys of the individual members. But in ABE mechanisms although the PKG generates the private keys for the individual members, it can vanish entirely once no more members intend to join the network, making the system much more secure. The latency and memory requirements discussed previously also verifies the fact that in overall S-CPABE performs better than multicast DTLS.

4 Conclusion

Multicast security is a basic necessity for IoT devices and applications, assuming the big figures for the number of IoT devices in the future. But it has always been a challenge to design efficient multicast security protocols for IoT. Traditional multicast security mechanisms are quite complex and they do not have the resource constraints like in IoT devices. Existing solutions try to make use of the DTLS protocol in a multicast fashion, but retains the same drawbacks of DTLS. Different requirements for secure multicast communication are highlighted in this chapter. Two different approaches for achieving multicast security have been discussed. Both the schemes use the concept of Shamir's threshold secret sharing. The merging of the flexibilities of ABE security mechanisms for multicast on one hand and reducing the computations on the end nodes on the other hand gives a perfect notion of achieving a multicast secure environment for the resource constrained IoT devices. The idea has been implemented in S-CPABE and compared with DTLS. The results indeed show that

the constraint parameters are well within the memory and computation constraints required by a typical IoT node. The approach is also independent of any physical layer technology used and fits perfectly for upcoming LPWAN technologies for IoT. ABE mechanisms are a perfect fit for multicast scenarios and opens the opportunity for future works to optimize the mechanisms to suite for IoT environments.

References

1. Sakarindr, P., Ansari, N.: Security services in group communications over wireless infrastructure, mobile ad hoc, and wireless sensor networks. IEEE Wirel. Commun. **14**(5) (2007)
2. Basu, S.S., Tripathy, S.: Securing multicast group communication in IoT enabled systems. IETE Tech. Rev. (2018) (accepted)
3. DeCleene, B., Dondeti, L., Griffin, S., Hardjono, T., Kiwior, D., Kurose, J., Towsley, D., Vasudevan, S., Zhang, C.: Secure group communications for wireless networks. In: Military Communications Conference, MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force, vol. 1, pp. 113–117. IEEE (2001)
4. Sun, Y., Trappe, W., Liu, K.R.: A scalable multicast key management scheme for heterogeneous wireless networks. IEEE/ACM Trans. Netw. **12**(4), 653–666 (2004)
5. Gupta, S.K., Cherukuri, S.: An adaptive protocol for efficient and secure multicasting in IEEE 802.11 based wireless lans. In: Wireless Communications and Networking, WCNC 2003, vol. 3, pp. 2021–2026. IEEE (2003)
6. Westerhoff, L., Reinhardt, S., Schafer, G., Wolisz, A.: Security analysis and concept for the multicast-based handover support architecture Mombasa. In: Global Telecommunications Conference, GLOBECOM'04, vol. 4, pp. 2201–2207. IEEE (2004)
7. Porambage, P., Braeken, A., Schmitt, C., Gurtov, A., Ylianttila, M., Stiller, B.: Group key establishment for enabling secure multicast communication in wireless sensor networks deployed for IoT applications. IEEE Access **3**, 1503–1511 (2015)
8. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. In: Advances in CryptologyCRYPTO 2001, pp. 213–229. Springer (2001)
9. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy, SP'07, California, USA, pp. 321–334. IEEE (2007)
10. Wang, C.-J., Luo, J.-F.: A key-policy attribute-based encryption scheme with constant size ciphertext. In: 2012 Eighth International Conference on Computational Intelligence and Security (CIS), Guangzhou, China, pp. 447–451. IEEE (2012)
11. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
12. Seward, J., Nethercote, N., Fitzhardinge, J.: Valgrind, an open-source memory debugger for x86-gnu/linux (2004). http://www.ukuug.org/events/linux2002/papers/html/valgrind

An Adaptable System-on-Chip Security Architecture for Internet of Things Applications

Atul Prasad Deb Nath, Tamzidul Hoque, Sandip Ray and Swarup Bhunia

Abstract Modern-day System-on-Chip (SoC) security architectures designed for smart connected devices, such as Internet of Things (IoT) and automotive applications, are often confined by two crucial design aspects: in-field configuration and low overhead. Due to the restrictions posed by these design aspects, it is extremely difficult to develop a robust and adaptable architecture for SoC security policies in IoT and automotive platforms. Security policies, on the other hand, are of critical significance as they implement the confidentiality, integrity, and availability requirements of diverse on-chip security assets. During the complex and often a long life of a system, security requirements evolve, giving rise to the need of adapting security policies. Existing SoC architecture and design flow do not provide the flexibility for easy adaptation of SoC security policies based on emerging threats or security requirements. To address these design constraints and subsequent limitations, a novel security architecture and CAD flow is proposed in this work for efficient implementation of diverse security policies. The adaptable architecture and associated CAD flow enable *hardware patching* through a reconfigurable security policy engine that can be seamlessly and securely upgraded in-field to address unanticipated attacks and update new security requirements. The infrastructure of the proposed security framework is build with three primary building blocks. First, a centralized Reconfigurable Security Policy Engine (RSPE) is introduced to implement and upgrade policies in-field without comprehensive changes in the architecture. Second, a set of smart security wrappers are developed for efficient extraction of security critical event information and avoidance of communication bottleneck. Third, the on-chip debug instrumentation i.e. the Design-for-Debug (DfD) infrastructure is employed with minimal modification for extensive access to an arbitrary number of signals of

A. P. Deb Nath (✉) · T. Hoque · S. Ray · S. Bhunia
Department of Electrical and Computer Engineering, University of Florida, Gainesville, USA
e-mail: atulprasad@ufl.edu

T. Hoque
e-mail: thoque@ufl.edu

S. Ray
e-mail: sandip@ece.ufl.edu

S. Bhunia
e-mail: swarup@ece.ufl.edu

R. S. Chakraborty et al. (eds.), *Security and Fault Tolerance in Internet of Things*,
Internet of Things, https://doi.org/10.1007/978-3-030-02807-7_4

the SoC. A suitable CAD framework is also proposed along with the architecture to systematically implement diverse security policies. The result analysis shows that the architecture provides a high level of adaptability with minimal overhead in terms of power, area, energy, and performance. Hence, the security architecture is highly suited for SoC in IoTs and automotive systems operating in a rigid boundary of performance and energy profiles.

1 Introduction

The era of Internet-of-Things (IoT) is characterized by an unprecedented growth of smart devices and connected systems. The growth is expected to continue at an even faster rate, with estimates of 50–75 billion connected devices by 2020. The estimations show that the number might run into trillions within the next decade [1]. Such rapid growth surpasses any other sector in the history of human civilization. Security is clearly a critical component of computing devices targeted towards IoT applications. In particular, IoT devices are provided access to a tremendous amount of sensitive assets. These assets mandate protection against any kind of illegal access and modification. The nature of these assets can vary from confidential personalized information of the consumers to critical design collateral. For instance, contacts, health information, location, records of financial transactions, etc. can be categorized as sensitive end-user assets residing in the SoC. Similarly, firmware information, fuses, cryptographic artifacts, digital rights management (DRM) keys, defeature bits, design-for-debug, etc. are crucial architectural assets of the SoC.

A system-on-chip (SoC) lies at the heart of most IoT devices. The SoC is built upon numerous pre-defined hardware modules known as Intellectual Property (IP) blocks. The IPs are placed on an interconnect fabric that supports standardized communication and coordination. Each IP come with a specific functionality and aids the overall system functions of the SoC. The security assets of the SoC are distributed over different IPs during the process. Consequently, the process of protecting the assets is challenging and often requires implementation of complex and ambiguous system level security policies [2–5]. In most of the cases, the security policies are implemented as an amalgamation of the requirements imposed by SoC designers and IP vendors. These policies go through multiple phases of refinement during the design cycle in an ad-hoc manner. Due to the lack of systematic approach and continuity of the process toward the final phases of the design cycle, it is quite challenging for the designers to develop well-defined security architectures before the final product launch. Consequently, the products fail to address emerging threats and cannot be updated for novel security vulnerabilities.

On the contrary, SoC *security architectures* i.e. the mechanisms to ensure protection of sensitive assets from malicious, unauthorized access, constitute a crucial component of modern System-on-Chip (SoC) designs. There has been significant work on robust SoC security architectures, both in academic research and in industrial arena [2, 3, 6, 7]. However, they are difficult to apply to IoT applications. From

an architectural perspective, the IoT and automotive devices mainly come with two critical design and security constraints: (1) in-field configuration and update of security requirements; and (2) operation within the periphery of minimal energy and high performance. In-field security configuration is crucial because of the long in-field device life, (e.g., an automotive system may have a lifetime of over a decade), which invariably implies the need to modify security requirements when the device is in operation.

Traditional system design approaches have relied on software or firmware implementation for functionality that is anticipated to require significant in-field updates. For instance, most automotive microcontroller units (MCU) include an over-the-air update architecture. Unfortunately, software implementations of diverse critical system functionality may incur significant overhead in terms of energy and area cost, making them unsuitable for many IoT applications such as wearables, implants, and infrastructure devices. Even for high-end systems such as automotive electronic control units, software implementations are often not viable: these systems must enforce aggressive real-time requirements for detecting in-field security violations and performing mitigatory actions, which are difficult to enforce with software. Consequently, there is a strong need to develop a *hardware-based* security architecture that can provide in-field configuration while permitting enforcement of performance, energy, and real-time constraints.

The security architecture presented in this chapter is developed from rudimentary building blocks to mitigate the major challenges posed by IoT design constraints. The CAD flow of the architecture is highly compatible with existing hardware implementation tools. The overall framework allows the mapping of an arbitrary number of policies and facilitates a seamless update of security policies in post-silicon stages as well as in field operations. Simultaneously, it facilitates the policy implementation process while keeping the energy and performance overheads within satisfactory levels. The framework introduces a novel approach of hardware "patchability" to alter and update SoC requirements via system and IP level security policies.

The security architecture is primarily composed of three building blocks. The first building block is a plug-and-play, adaptable *infrastructure IP* named Reconfigurable Security Policy Engine (RSPE). The purpose of RSPE is to implement and enforce the System-on-chip security policies. It works as the security brain of the architecture and contains some or parts of the system level policies. The second building block is a set of standardized smart *security wrappers* that extracts security critical information from various IPs and delivers the information to the policy engine. The third and final building block of the architecture is an interface between the security wrappers and the local *debug modules* of the IPs. The integration of the debug infrastructure allows increased access to the required signals of the SoC in terms of observability and controllability. The role of each of the components in policy implementation is discussed in detail in the subsequent sections of the chapter. The chapter illustrates how the security architecture is developed from bottom up using the building blocks to facilitate policy implementation. More importantly, the work demonstrates how security policies can be adapted, modified, and upgraded or patched based on the emerging threats during in-field operation of IoT devices and automotive systems.

The CAD flow associated with the architecture demonstrates a streamlined method for compiling SoC level security policies by exploiting the commercial off the shelf EDA mapping tools. The results obtained from illustrative policies are presented in this work to show that the proposed framework delivers superior performance compared to software-based or micro-controlled implementation of security policies in terms of energy, power, and area overhead.

Experimental results show that this approach is viable as a foundation for patchable hardware. The implementation improves over a traditional micro-controller-based implementation of security policies by almost an order of magnitude in performance and energy costs.

The remainder of the chapter is organized as follows. An essential background on SoC security policies is outlined in Sect. 2. The motivation for proposing the novel approach is highlighted in Sect. 3. The proposed security architecture is elaborately discussed in Sect. 4. Section 5 discusses the CAD flow used to automatically map and update security policies in the RSPE. The experimental results are illustrated and analyzed in Sect. 6. The conclusion and relevant future work is discussed in Sect. 7.

2 Security Assurance in SoC

2.1 Security Policies

Modern-day system-on-chips (SoCs) contain a significant number of security critical assets. The assets must be protected from illegal access and malicious alteration. The security requirements for these assets can be defined in terms of confidentiality, availability, and integrity. SoC Security policies are primarily implemented to protect these assets by mapping the security requirements into *actionable* design constraints. The design constraints are implemented by SoC integrators or IP providers to realize the protection mechanisms in the architecture. The examples of typical SoC security policies are as following:

- *Example 1*: Any data transmitted by the crypto engine during boot time should not be controllable and observable by any unsolicited IP other than the target IP.
- *Example 2*: A on-chip storage unit with secret keys can be accessed for silicon validation. However, the storage accessibility should be restricted after chip production.

Example 1 illustrates the confidentiality requirement of the SoC assets. On the other hand, example 2 shows an instance where the integrity constraint is enforced.

These security requirements are enforced into actionable design constraints by the security policies. The policies provide secure means of accessing the SoC assets. The criticality of accessing the assets can vary based on the time of access (e.g. boot time or normal mode of execution) and different the stages of SoC life cycle (e.g. design, manufacturing, production, supply-chain, etc.).

A short introduction to the various classes of SoC security policies is given in the subsequent sections. A detailed elaboration of security policies is beyond the scope of this article.

Further details on the policies can be obtained by going through the cited bibliography provide at the end of this article.

Access Control: Access control policies employ the security requirements for accessing the assets of the SoCs into design attributes. In particular, these policies control how the assets stored in different IPs for the SoC are accessed by other requesting *agents*. An *agent* in the system can be hardware or software components of the IP blocks. The aforementioned examples show how these policies define the access control. Access control policies are vital for any SoC as they form the basis for other relevant policies involving integrity, information flow, secure boot, etc. [8–10].

Information Flow: It is possible to infer the value of secure assets by accessing the information indirectly. Such inference can be done by indirectly observing the information or snooping the active computation or ongoing communication between the IPs. The primary purpose of information flow policies is to prevent such violation of the security in the SoC. Examples of typical information flow policies are as following:

- Key Obliviousness: The low-security IPs connected in a standard network-on-chip (NoC) should not be able to extract the cryptographic keys by snooping the information flow of the crypto engine.

Due to the complexity of modern SoC, it is difficult to analyze the information flow tracking of secure assets across multiple IPs. Also, it is equally challenging to implement information flow policies. These policies often require sophisticated protection mechanisms and advanced mathematical arguments for correctness. The policies might involve complexity results from information security. Information flow policies are typically employed for critical assets that demand very high confidentiality [11–13].

Liveness: Liveness policies are implemented to ensure ceaseless system operation throughout the execution time without any stagnation. Common examples of liveness policies involve the timely generation of ACK (acknowledgment) or NACK (negative-acknowledgment) signals in response to REQ (request) signals. Any deviation from such handshaking protocol might lead to deadlock or livelock situation and consequent violation of system availability requirements [14].

Time-of-Check versus Time of Use (TOCTOU): The goal of implementing Time-of-Check versus Time of Use (TOCTOU) policies is to verify the identity of the *agent*, e.g. an IP, accessing an asset based on the prior access privilege given to the entity. As the name implies, the policy cross-checks the identity of the IP at the time of use with the information stored during the time of check. Firmware upgrade can be a critical example of TOCTOU as it is of utmost importance to verify that the update installed on the chip is actually the update which has been permitted earlier via secure authentication mechanism [3, 15].

The primary focus of the aforementioned security policies is to ensure a secure integration of third-party IPs procured from individual vendors. Hence, the scope of the policies falls within the periphery of SoC integration and its design characteristics. The underlying assumption, however, is that the IPs themselves are trustworthy and do not contain any malicious entity. The threat model of these policies considers external attacks originating from software or the SoC interface. Malicious alterations introduced inside the IPs like hardware Trojans or back-doors are beyond the scope of the threat model. Therefore, the proposed framework is highly applicable for SoC designs based on in-house IPs rather than untrusted third-party IPs.

2.2 Security Architecture: The Current State of the Practice

Security architectures are designed with the primary purpose of implementing security requirements into actionable design constraints. Unfortunately, the present state of the practice is highly reliant on human insights and creativity. The current approach of designing SoC security architecture usually includes developing a baseline architecture definition and refining it repeatedly through the following steps:

- Development of threat models to identify potential threats to the existing architecture definition.
- Refinement of the architecture with preventive and mitigation strategies to ensure invulnerability against identified threats.

The baseline is usually developed from legacy architectures available for existing products and gradually adapted for policies defined through the SoC design exploration. For every asset located inside the SoC, the architect must identify the following:

- Who can access the asset?
- What kind of access is permitted by the policies?
- What points in the system execution or product development life-cycle such access requests can be granted or denied?

The process of answering the aforementioned questions can be highly complex and tedious for many reasons. As many assets are defined in the IP blocks of the SoC during system execution, the process of ensuring integrity, confidentiality, and availability of these assets becomes challenging. The current industrial practice lacks any disciplined mechanism for implementing these policies: the SoC designers exploit different features of the architecture and artifacts of the system (e.g., fuses, firmware, etc.) to apply the policies as design constraints. In recent times, the architects are utilizing Trusted Execution Environments (TEE) to facilitate the implementation of access control requirements in modern SoCs. TEEs are comprised of a combination of hardware support (e.g., secure operating modes, virtualization), and software mechanisms (e.g., context switch agents, integrity check) which can be used to enforce isolation of sensitive data when necessary for access control [16].

3 Motivation for Adaptable Architecture

3.1 Case Study: An Attack on Confidentiality

To demonstrate the vulnerability of current practices to confidentiality attacks, we consider a violation scenario involving a trusted IP and a third-party rogue IP. We assume that the trusted hardware block is a crypto engine, namely IP `A`, and the untrusted module is a third party IP, namely IP `B`, with generic functionalities. Ideally, in a typical SoC design cycle, the rules defining how the assets of IP `A` will be accessed by IP `B` are supposed to be set in the risk assessment phase. However, it is challenging for SoC engineers to finalize the security requirements in a single phase of the design process. Due to the incessant flow of new information and knowledge across various stages of the design flow, the designers keep reservations for changes towards the later stages of SoC design. Consequently, the policy implementation process gets extended to multiple phases and goes through numerous changes and refinements until the product reaches the final stage. As a result, it becomes very challenging for the design engineers to come up with definitive policies with specific constraints for every asset and their corresponding information flow policies for critical IPs like crypto core IP `A` and untrusted third party IPs like IP `B` at launch time of the product i.e. time `0`. Moreover, the resource boundaries compel the architects to integrate numerous IPs in the network-on-chip (NoC) with the cryptographic core. The process also involves eventual marking of the IPs as *safe* which permits them to observe the crypto keys being transmitted in the communication fabric. Being located on the same network-on-chip (NoC) with the cryptographic engine, the untrusted or rogue IP `B` can be exploited by attackers at a later time of operation i.e. time `t` of product lifecycle to violate the information flow policy and snoop on the crypto keys. For example, it might be possible for the attacker to revoke *Key Obliviousness* property by exploiting the malicious IP `B` to infer the keys of the crypto core. Thus, the attacker can snoop data from the cryptographic engine through the low security interconnect fabric and obtain access to the security assets. Unfortunately, remedial actions for the aforementioned scenario depicting a confidentiality breach of the design is not accommodated by the current security measures. The current security architectures offer no in-field or adaptive solution to mitigate such attack on confidentiality and usually include the rigorous task of upgrading the firmware or OS kernel of the SoC.

3.2 Case Study: An Attack on Integrity

In this section, a scenario is outlined to depict an attack on the integrity of security assets. Similar to the prior case study, this attack scenario includes a malicious/rogue third-party IP. In this case, the adversary attempts to launch a code injection attack through the rogue IP. A common example of such attack would be the exploitation of direct memory access (DMA) to overwrite code segments via code injection. Unfor-

tunately, there are many instances where the DMA-protected regions can be accessed through the loopholes of poorly developed policies. For instance, the System Management Interrupt (SMI) handlers can be exploited by the attacker to write to an internal address of System Management RAM (SMRAM). SMRAM is the reserved part of DRAM and usually utilized by SMI handlers of BIOS. The attacker can execute a code injection attack instance by exploiting the malicious IP and exploit SMI handlers to corrupt secure memory. Such instance can lead to severe consequences like malicious control over the address range of secure memory contents [17]. The existing mitigation techniques fail to provide an adaptive remedy for the aforementioned attack. The current practice includes extensive testing of SMI handlers for pointer validation vulnerabilities. Also, the preventive measures to thwart similar attacks via DMA include overhauling every memory access points in the system execution, identifying memory access requests to DMA-protected regions, and setting up mechanisms to bar DMA requests to all protected accesses.

3.3 Current Challenges with Adaptable Architecture

Modern SoCs are designed with a multitude of IPs. Consequently, the security requirements of these IPs are pretty diverse. The current norm of security policy implementation includes distributing the policies across the entire SoC. Moreover, the lack of standardization in policy implementation often leads to adoption of ad hoc approaches and reliance on manual and creative insights of SoC designers. Hence, the present practice poses major obstacles in developing systematic ways of in-field policy update backed by full-fledged security architecture. Although limited reservations are made for in-field update of few security aspects, the identification process of these security critical artifacts requires an early start and usually extend through several phases of the SoC life cycle. Existing methodologies of implementing security policies cannot accommodate mitigation measures in response to novel and emerging threats. Usually, the process of updating security policies to meet novel security requirements mandates comprehensive changes in architecture and implementation. When unanticipated modifications are required in practice today, they are handled primarily through designer ingenuity, typically making use of a combination of firmware updates, complex defeaturing, etc. Given the complexity of these approaches, it is unsurprising that the process itself can introduce security loopholes leading to untrustworthy, vulnerable, and even functionally fragile products.

Inspired by the limitations of the current practice in developing security requirements and architectures, we advocate an adaptable security architecture for SoCs. An adaptable/patchable architecture is capable of overcoming these limitations by securely upgrading, and adapting rectified security requirements in a seamless manner in field operations. Through a flexible and centralized security policy engine, it is possible to provide in-field patch of security requirements for emergent threats with minimal architectural changes and without incurring significant area, energy, or performance overhead.

4 A Generic Adaptable Infrastructure

In order to develop a generic hardware patchable security architecture, a key observation is that the security policies can be represented by a standard sequence of commands. These commands can be employed to analyze the behavior pattern of the IPs in the SoC and designed to respond accordingly to enforce the security policies. It is also important to implement these commands in a separate, centralized IP that can communicate with the different IPs in the SoC design through a standardized interface protocol. Figure 1 shows a high-level overview of a centralized security architecture where an infrastructure IP communicates with other IPs to ensure system security. The objective of the communication is to identify the security state of the system, determine if a requested access for a sensitive asset at the current state violates a security requirement and if so, take mitigation measurements. Given the above observation, the following three critical architectural components have been identified that can together implement SoC security policies in post-silicon phases and during field operations [18].

4.1 Design Choices

The centralized architecture of the policy engine is one of the key design aspects of the Reconfigurable Security Policy Engine (RSPE). It facilitates the process of implementing and reusing RSPE as a single infrastructure IP block. The fundamental motivation for a centralized security architecture is driven by the need for a central infrastructure for comprehending, analyzing, upgrading, and validating intricate system level security policies. The contemporary practice of SoC security assurance involves implementing security policies *sprinkled* across multiple IPs in a distributed manner. Such nature of policy implementation gives rise to enormous complexity

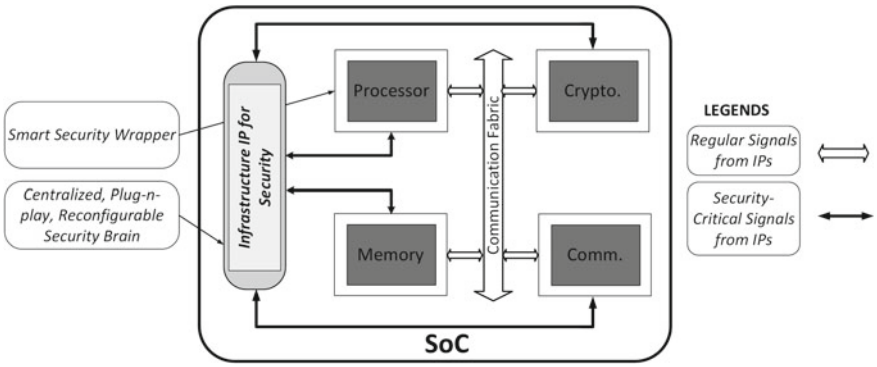


Fig. 1 A high-level illustration of the SoC architecture is presented in the figure with the proposed RSPE (reconfigurable security policy engine)

in analyzing, modifying, and upgrading the policies, especially, based on emerging threats and vulnerabilities. Hence, a centralized architecture e.g. RSPE for security policy update can be an efficient solution to the existing limitations. One caveat in a central policy engine based security architecture, however, is the bottleneck in system performance caused by all the IPs communicating with the RSPE. To address the issue, a distinct design feature, namely *smart* security wrappers, is employed in the proposed security architecture. The purpose of the *smart* security wrapper is to extract security critical information from the ongoing transaction of the SoC and forward the information to the centralized RSPE. Another crucial design choice is the selection of a reconfigurable fabric to implement the policy engine stemmed from the need to configure and update security policies in-field to meet customer security requirements as well as to protect the design and assets from novel emerging threats. There are possibilities that the process of updating the bitstream of the FPGA-based RSPE itself can fall victim to malicious attacks. Such attacks can be prevented by deploying proper security measures for remote patch installation on the RSPE. Figure 2 demonstrates how it is possible to patch security policies in-field through an FPGA based implementation. The countermeasure adopted for RSPE is discussed in a later section of the chapter.

4.2 Centralized Reconfigurable Security Policy Engine

The Reconfigurable Security Policy Engine (RSPE) can be deemed as a *security brain* in the architecture. It provides an interface to program security policies for the SoC. The key functionalities of the policy engine are to analyze various security critical events obtained from the security wrappers, determine the security status of the system based on the analyzed information, and initiate (*request) and (*disable) signals for the corresponding IPs of the SoC. The detailed architecture of the proposed

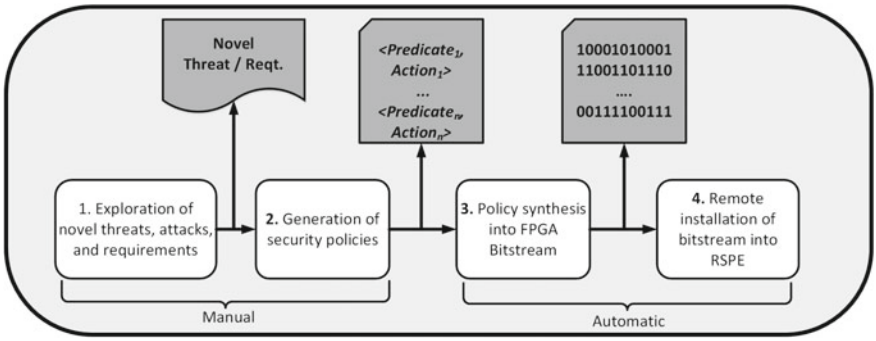


Fig. 2 The figure depicts how the software flow of the proposed framework accommodates the manual detection of security threats and automates the process of the policy implementation

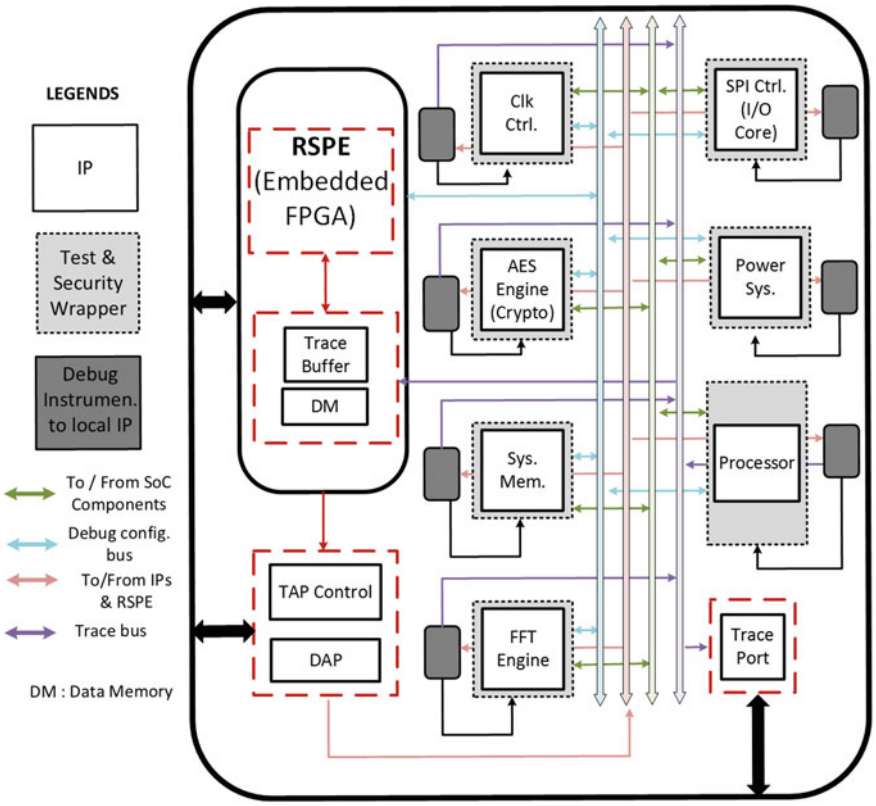


Fig. 3 The figure shows an elaborate illustration of the security architecture: it contains all the building blocks of the architecture, namely, RSPE (the centralize security brain that implements the policies), the smart security wrappers (an extension for existing security wrappers to detect security critical signals), and the design-for-debug instrumentation (local debug modules customized to get access to an extended number of critical signals)

RSPE is illustrated in Fig. 3. The primary components of the policy engine include a security buffer and policy engine module. The purpose of the security buffer is to deliver the IP-specific events obtained from the smart security wrappers to the policy engine. The policy engine modules is designed to serve the analytical and computational purposes of the RSPE and enforces the security policies.

4.2.1 Security Buffer

The security buffer is interfaced with the RSPE through a *buffer controller*. The controller helps the policy engine to analyze the buffer frames. The buffer storage is implemented through a standardized scheme of static segmentation. The length of the variable vary depending on the volume of the meta-data. The event logs are made accessible to the policy engine by the buffer ports. A buffer controller is employed

to control these ports. The buffer controller maintains the congruency in different speeds of read and write operations, sizes of segments, and frequency of events among security wrappers and the policy engine.

4.2.2 Policy Engine

We implement the policy engine on an embedded FPGA. The key to mapping policies to FPGA-based policy engine is the observation that a security policy engine can be represented as a specialized finite state machine that interacts with the IPs of the system through a highly standardized protocol. The state machine for a specific policy enforcement can be explained by the following steps: (1) receipt of events from the various security wrappers; (2) computation of the system security state; and (3) asserting/de-asserting various output control signals. In the policy engine micro-architecture, the data buffer stores the corresponding event frames from different IP blocks, the corresponding buffer control logic flags the appearance of "new events" to the policy execution engine. Consequently, a new input event frame read from the buffer memory coupled with the consequent computations would signify the definition, requirement, and implementation of a security policy. This permits a systematic extraction of the state machines of security policies from the software/firmware implementation, which can then be converted to a typical hard-logic state machine for the chosen embedded FPGA architecture.

4.3 Smart Security Wrappers

4.3.1 IP Security Wrappers

The primary purpose of the smart security wrappers is to extract security critical events occurring at the time of inter-IP communication and forward the information to the RSPE. Any naive attempt to obtain information pertaining to all the events and transactions taking place in the SoC through data, control, and status signals would result in a communication bottleneck with prohibitive routing overhead. An efficient way of overcoming the limitation is identifying the security critical events and exploiting the selected events to extract the information for the policy engine. Such *smartness* is incorporated in the security wrappers implemented in the framework while delivering a standard frame-based communication interface between the IPs and the RSPE. As the proposed changes are template-based, these can be easily integrated to the current practice of IP implementation.

One challenge about the extraction of security critical events is the effective segregation of signals of interest from the non-critical signals. In this framework, the challenge is addressed by meticulous classification of IPs and signals according to their security aspects. For examples, *Memory IPs* such as memory controllers, flash control logic, cache controllers, Direct Memory Access (DMA) modules, etc. are

engaged in operations that involve access to various memory hierarchies. Similarly, processor cores usually include general purpose CPUs, GPUs, and other cores controlled by microcode or firmware e.g., UART controller, Ethernet, audio cards, video cards, etc. Based on the functionality of each of the IP classes, the security critical IPs can be categorized and prioritized accordingly. The security critical events can also be classified in a similar fashion. For instance, some events in the processor core signify the beginning and end of system critical applications or processes like computations for generating exceptions, interrupts generated by system controllers etc. The metadata for each of these events contains adequate information for the policy engine. For an event requesting a DMA access, the metadata will contain access details that can be analyzed from page size, DMA burst sizes, and memory address range, etc. These metadata are forwarded to the RSPE by the security wrappers, often based on the requests made by the policy engine itself. In addition to the standard events, there IP specific events used in this framework that highlights the security critical events belonging to each of the IPs. Moreover, the proposed architecture allows extensive flexibility for the SoC integrator as it is possible to map IP specific events in the wrappers based on custom security requirements.

4.3.2 Security Wrapper Implementation

The proposed security wrappers deliver the security critical event information in a standard format through a frame-based design. The wrappers can be instantiated for corresponding events in individual IPs. The architecture of a typical smart security wrapper is illustrated in Fig. 4. The major components of the security wrapper include an activity monitor logic (to check whether the IP is in active mode), a detector for the event types and a buffer storage for storing the event metadata. A correspondence with

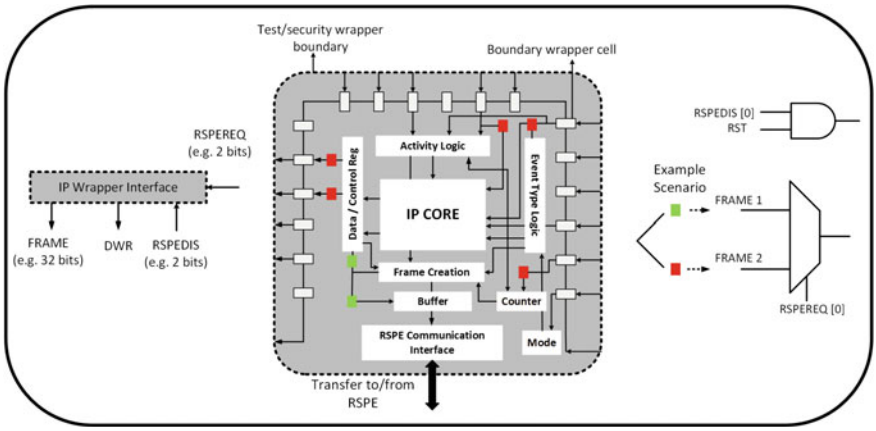


Fig. 4 Security wrapper architecture: the purpose of the security wrappers in to provide a standardized, frame based communication between the IPs of the SoC and the RSPE

local clock domain might be required for some events. The wrappers are also designed with configurable registers. These registers can be configured by the RSPE at boot time to select a particular set of events which require modification. The frame-based interface aided the process of establishing a standard communication mechanism between the wrappers and the RSPE. The functionality of the RSPE is generally confined to two major actions. Firstly, RSPE can send a *disable* signal to block an IP transaction in case of a suspected security policy violation. The application of the *disable* signal can vary over a range of granularities depending on the implemented policies. Secondly, RSPE can initiate a *request* signal to request additional data or status of control signals when additional information is required for enforcing policies. The boundary scan interface of the IP, which is already present in the architecture, is exploited to transfer data in a parallel shift or access mode. This helps to meet the higher bandwidth demands required by some specific functional security validation processes.

4.4 Design-for-Debug Interface

4.4.1 Debug-Aware IP Security Wrapper

The process of making the security wrappers *smart* is facilitated by the local Design-for-debug (DfD) modules of each IP in the SoC. DfD modules are inserted in most of the modern architectures for debug purposes. In this framework, the debug modules, corresponding to each of the IPs, are exploited to extract an extensive number of security critical signals and implement an arbitrary number of policies based on the security requirements. The utilization of DfD modules for security policy implementation should not come at the cost of any hindrance during the regular operation of debug mode. The exploitation of DfD modules without hampering their regular functionality requires IP-level modification of the DfD logic and efficient adaptation of the smart security wrappers [19]. Figure 5 illustrates the required modification for the usage of the debug infrastructure for security policy implementation. In particular, a separate port is required to transfer security critical data to the RSPE. Such separate port is introduced in the architecture to avoid the re-purposing of existing debug trace port and bus. The IP specific events of interest are programmed in the local DfD modules by the RSPE. The programming is facilitated by the configuration register interface of corresponding DfD modules. RSPE must be capable of identifying corresponding events from communication frames sent by the security wrapper as the DfD modules are configurable at runtime to detect related or disparate security events. The interface is standardized throughout all DfD module and security wrapper pairs by tagging the event information with corresponding addresses of the configuration registers. The standardization, however, comes at the cost of the overhead of additional registers in the IPs especially in cases where only a few events are detected by the debug logic. The trace packet generation of the local DfD modules can be disabled during the RSPE access to any leakage power when the

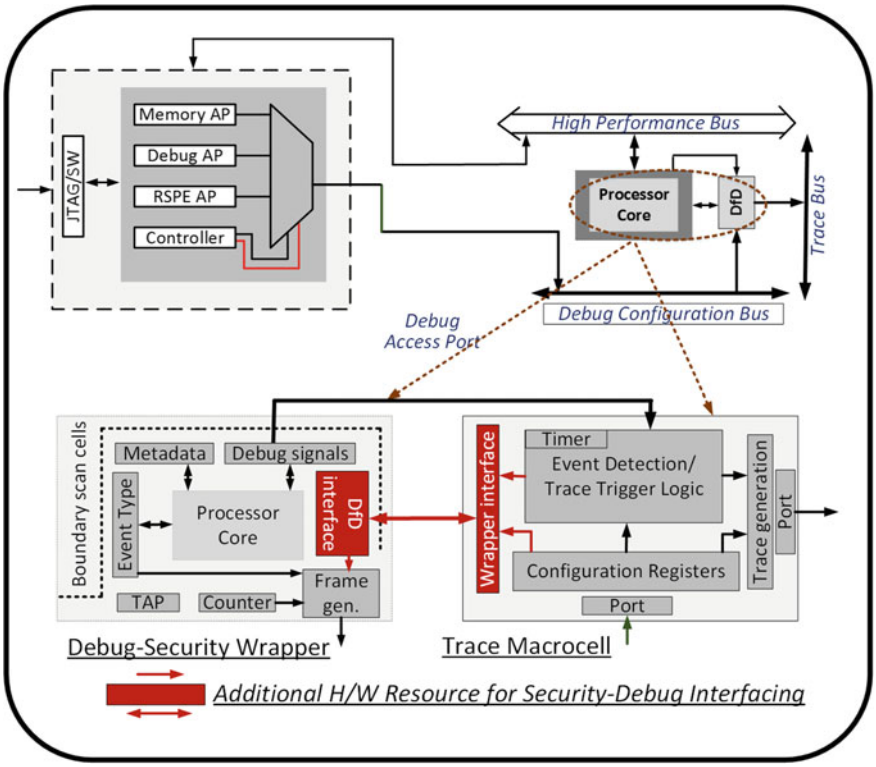


Fig. 5 Additional hardware resources for interfacing local DfD with IP security wrapper

debug infrastructure is not in use. Apart from security critical event triggering and observation of the signal relevant information, it is possible to re-purpose the DfD control hooks during the design of the SoC and at the time of in-field upgrade or patch.

4.4.2 RSPE-Debug Infrastructure Interface

The local debug modules must be configurable by the RSPE to identify the security critical events and enforce appropriate control. The communication interface between the on-chip debug instrumentation and RSPE is depicted in Fig. 6. RSPE utilizes the Debug Access Port (DAP) and configuration bus to program the trace cells. Due to the abundance of configuration registers and corresponding logics of local DfD modules, the policy engine can program the trace cells to monitor any relevant security critical signal. The programming is done during the boot time and the configuration interface is turned off during regular mode of operation to save leakage power dissipation.

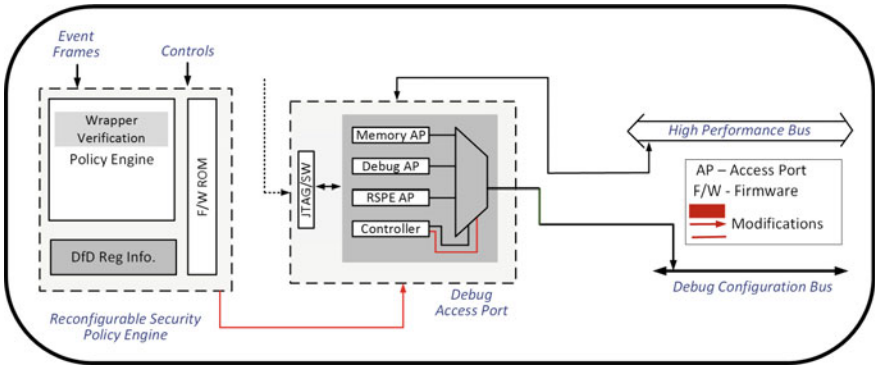


Fig. 6 Interfacing RSPE with on-chip debug

In case the RSPE fails to configure the local DfD for monitoring all security relevant events at boot phase, the DAP can be powered on through the power management module at runtime.

5 Overall Flow and Major Steps

5.1 CAD Framework for Security Policy Synthesis

A systematic and standardized method for synthesizing diverse security policies into the proposed Reconfigurable Security Policy Engine (RSPE) is presented in this section. The primary features of the framework are as following:

- It is possible to synthesize arbitrary security policies in an automatic manner using the CAD framework if the policies are presented in the specified tuple format of predicates and actions.
- The CAD flow implements the security policies in an energy optimum way while meeting the performance requirements through exploration of the design space.
- Using the presented framework, it is possible to perform incremental mapping of security policies through partial reconfiguration. It aids the process of in-field upgrade and patching of security policies.
- The framework is compatible with conventional FPGA synthesis flow and capable of utilizing commercial off-the-shelf application mapping tools.

The steps of mapping security policies into an embedded FPGA fabric is illustrated in Fig. 7. Two novel steps are introduced (highlighted as blue colored action items in the flowchart) into the front-end flow to integrate the proposed framework to conventional FPGA synthesis flow. Through the newly introduced steps, a pre-compilation phase is integrated into the flow for parsing the security policies. An

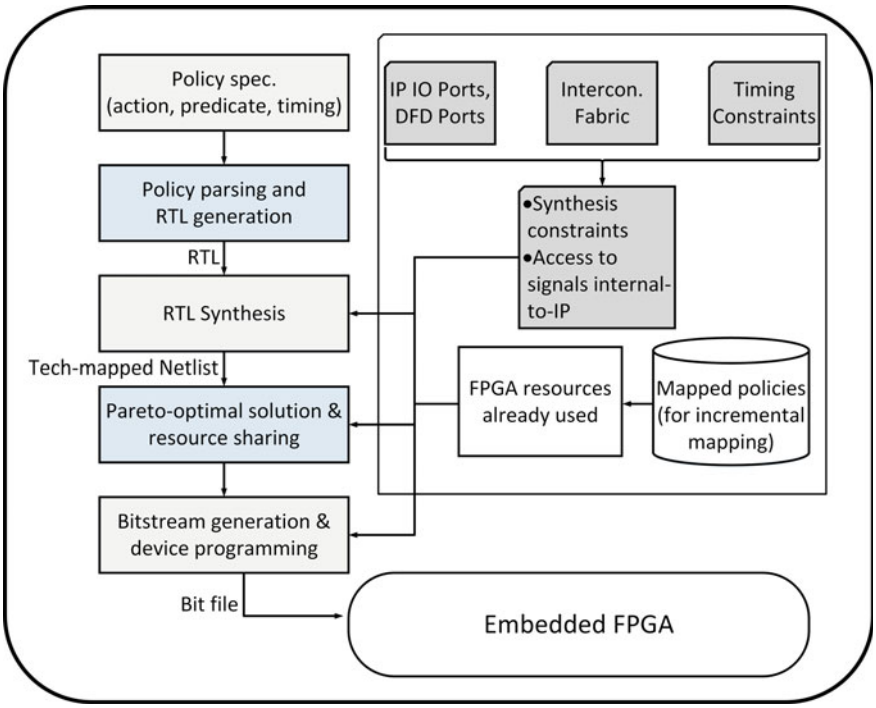


Fig. 7 An illustration of the mapping process of diverse SoC security policies on embedded FPGA-based reconfigurable security policy engine

RTL description of the policies is also created through these two steps. In the proposed flow, the security policies are mainly presented in a specific format of 3-tuples: ⟨**timing, predicate, action**⟩. A short description of each of the tuples are given below:

Timing:
The timing tuple of the policies signifies two major aspects of security. First, the timing information can state the mode of operation of the SoC. For example, it helps to determine if the SoC is being run on user mode or supervisor mode. Second, it helps to define the timing of SoC operation with respect to the global clock of the system. For instance, policy 1 and 3 of Table 1 exploits timing information to determine the mode of operation of the DLX microprocessor (e.g. user mode) as well as the entire SoC (e.g. debug mode).

Predicate:
The predicate tuple works as the pre-condition for any policy enforcement process. The predicate information is obtained from IP-specific observable signals or specific properties of the IP interconnect. The predicate tuple is usually derived from the logical operations among multiple security critical signals that are observable. For the first policy enumerated in the Table 1, the predicate part of the policy is derived

Table 1 This table shows a set of representative security policies. These policies are implemented on the micro-controlled policy engine and the embedded FPGA-based reconfigurable security policy engine

Policies	Tuple: predicate	Tuple: action	IPs involved
P# 1	(Operational mode: user) and (read or write request to memory by any IP)	Read or write requests within valid address ranges are allowed	Any IP with access to system memory
P# 2	(Operational mode: supervisor) and (read or write request by any IP)	Read requests within shared address ranges are allowed; no write request is allowed	Any IP with access to system memory
P# 3	(Operational mode: debug) and (the trace cells of local debug modules are busy and power unit is busy)	Firmware update of power module or changes in configuration register of SPI module is not allowed	Power management unit and SPI module
P# 4	(Operational mode: user) and (request for updating the inst. memory)	Instruction memory update is not allowed	Processor of the SoC
P# 5	Active crypto mode	Any request to access system memory or interrupts from other IP cores are not allowed	Crypto engine and other IPs of the SoC

by the logical OR operation of multiple signals. The type of the signals can be like memory read or write request by the user, memory read or write request by an IP, etc. The pre-condition for the application of action part of the policy is formed by the logical AND operation between the timing and predicate information.

Action:
In the 3-tuple format, the third tuple is known as the action tuple. It is applied in any policy when the joints conditions imposed by timing and predicate tuple is true. The usual approach for applying the action tuple involves asserting or de-asserting certain signal or a set of signals. The process might involve checking a set of variables as well. For instance, in policy number 1 of the Table 1, a check is required to see whether the read or write request falls within the address range for the corresponding IP.

The three-tuple policies are parsed to generate equivalent Verilog RTL code. The RTL conveys the information of the I/O ports of all IPs constituting the SoC, the test and debug ports, and the interconnect fabric. The inputs and outputs for the RTL modules are analyzed to determine the observable signals and controllable signal along with the timing information. The inputs of the modules are deemed as the controllable signals required for the enforcement of a security policy. Similarly, the outputs of the module are considered as observable signals for any given policy. In the parsing process, policies without state elements are represented through the “assign”

statement in RTL. For policies that require the application of a state machine, a separate "always" block is introduced in the RTL code.

Resource Optimization:
Once the RTL module containing the security policies is developed, it is synthesized into the reconfigurable fabric of FPGA exploiting commercial off-the-shelf synthesis tools. We employ the principle of pareto-optimality to generate a set of mapping solutions while considering the restrictions posed by timing, resource, and other relevant constraints. The resource sharing approach is exploited to get an optimized area and energy requirement when many policies are being mapped within the boundary of resource constraints. In this architecture, the new sets of policies are mapped in an incremental fashion along with the prior sets of policies. To obtain such incremental mapping, the existing resource usage information is utilized and employed with the partial mapping feature of commercial FPGA mapping tools. Consequently, it is possible to opt for a specific mapping methodology based on the energy or area preferences for custom products. The preferred solution can be selected from a set of efficiently designed mapping solutions.

5.2 Authentication and Remote Upgrade

In-field *patchability* of security policies is one of the primary features of RSPE. RSPE prevents the exploitation of existing policy loop-holes by in-field upgrades. The upgrade is provided based on emergent threats and security requirements. Consequently, secure remote upgrade of RSPE patch is equally important. In this work, a challenge-response based authentication mechanism is employed to protect the remote RSPE patch installation from threats originating from malicious Trojans or other back-door vulnerabilities. The authentication scheme relies on the keys generated by an SRAM-based Physically Unclonable Functions (PUFs). The SRAM PUF exploits the intrinsic process variation of the chip and generates unclonable keys for secure authentication. SRAM PUFs require minimal efforts for implementation as there is no need for additional circuitry. Moreover, being a weak PUF, it is very cost efficient for resource constraint devices like IoT and automotive systems. As the keys are only generated at power-up period of the chip and do not require on-chip key storage, it is extremely difficult for the attacker to get access to the PUF keys. Hence, the SRAM PUF-based authentication protocol aids secure remote authentication as well as in-field *patch* upgrade of the policy engine. For instance, the security threats related to Time-of-Check versus Time of Use (TOCTOU) attacks are mitigated by single-threaded policy update procedure i.e. an overlapping *patch* request isn't be allowed to interrupt an on-going *patch* [15]. It should be noted that the threat model of the work addresses possible external attacks via software and SoC interfaces. The security issues related to third-party IPs, due to lack of trust verification measures, however, are excluded from the threat model. In particular, the scope of the implemented policies is limited to the IP integration characteristics of the SoC.

6 Results and Discussion

To evaluate the proposed framework, an SoC model is developed from bottom-up. The representative model is constituted of several crucial IPs that are prevalent in industrial SoCs. The building blocks of the SoC include a 32-bit pipelined DLX microprocessor core, a 128b AES crypto core, a 128b FFT engine, a representative memory controller, a clock controller, a power unit, and a Serial Peripheral Interface (SPI) IP. The model has 32 KBs of central memory. Due to difficulties of obtaining open-source SoC models for research purpose, we acquired the IPs from http://www.Opencores.org (an open-source online repository for hardware IPs) and developed our own illustrative SoC model.

To extend the baseline SoC model with our proposed architecture and framework, we developed two versions of the security policy engine. These versions include a micro-controlled policy engine built on a DLX μP core and a Reconfigurable Security Policy Engine (RSPE) implemented on FPGA fabric. A set of 10 illustrative policies is implemented on both versions of the security policy engine. The types of the policies include access control, liveness, secure boot, and secure information flow. Some representative policies are depicted in Table 1. The policies are functionally verified using Altera ModelSim simulation platform. A library of 32 nm node technology is used to obtain the area, performance, and power consumption results. The target platform for design implementation was a commercial off-the-shelf FPGA development board named Cyclone V by Altera.

The implementation of a diverse number of arbitrary security policies mandates the security policy engine's access to required security critical signals of the SoC. We have performed an accessibility analysis of the proposed policy engine on the representative SoC with 3 different versions of the model. The first and second versions are consist of standard test wrappers (IEEE P1500 boundary scan-based wrapper) and the smart security wrappers (an extension of IEEE P1500 boundary scan-based wrapper and debug wrappers), respectively. In the final version of the design, we combined the smart security wrappers with on-chip Design-for-Debug (DfD) instrumentation (similar to ARM coresight) to enhance the observability and controllability over required signals. Table 2 provides the estimated number of observable and controllable signals in different versions of the proposed design, namely *TW* (the test wrapper-based design), the *SW* (the smart security wrapper integrated policy engine), and *DfD* (the Design-for-Debug Infrastructure integrated with smart security wrappers). The comparative observability and controllability results of the signals in all three versions are presented in Table 2. In particular, % *Increment* column of Table 2 represents the increment of controllability and observability in DfD interfaced design. From the results, it is evident that DfD integration increases the potential number of observable and controllable signals of each IP blocks by a vast magnitude. For instance, the percentage increment in observability is as high as 171.43% (for an illustrative memory controller block) in the extended version of the design with DfD instrumentation compared to the smart security wrapper based design. DfD infrastructure enhances the observability over the signals of DLX μP

Table 2 The table illustrates an estimated number of controllable and observable signals in different phases of the design

Design specification					
	IP type	Test wrapper	Security wrapper	Design-for-debug	% increment by DfD over SW
Observable	DLX Up	5	547	772	41.13
	AES	5	386	776	101.04
	SPI	n/a	104	161	54.81
	Mem. Cont.	5	224	608	171.43
	FFT	5	134	218	62.69
	Total	20	1395	2535	81.72
Controllable	DLX uP	1	142	255	79.58
	AES	1	107	188	75.70
	SPI	n/a	75	144	92.00
	Mem. Cont.	1	104	187	79.81
	FFT	1	76	156	105.26
	Total	3	504	930	84.52

Table 3 The table shows an estimated number of SoC security policies that can be implemented in various case scenarios: the number of policies are dependent on the number of observable and controllable signals which are represented in the table as predicate and action tuples, respectively

Possible number of arbitary security policies				
Tuple type	Test wrapper	Security wrapper	Design-for-debug	% increment by DfD over SW
2 predicate, 1 action	570	490,046,760	2,987,015,850	5.10
4 predicate, 1 action	14,535	7.91857E+13	1.59645E+15	19.16
8 predicate, 1 action	377,910	1.757E+23	3.89E+25	220.42
8 predicate, 2 action	377,910	4.42E+25	1.81E+28	407.94

core by 41.13%. As for controllable signals, the percentage increment is 79.58% for the DLX μP. The maximum increment in controllability is up to 105.26% for the FFT module.

To further illustrate the notion of adaptability and flexibility in in-field policy implementation, we prepared Table 3 based on the controllability and observability information obtained by comparing different designs. Table 3 shows the feasibility of implementing an arbitrary number of policies by obtaining access to the extended number of controllable and observable signals through the debug instrumentation. The results clarify the initial assumption that local debug modules help to implement a significantly higher number of policies compared to other versions of the design that lack the DfD integration. The estimated number of security policies is calculated based on the standard policy framework proposed in this work. Apart from the timing

information, the primary policy-determining components of the framework are *predicate* tuples and corresponding *action* tuples. The *predicate* and *action* tuples of the policies are represented as *P*, and *A* respectively in Table 3. For each of the IP blocks constituting the SoC model, the observable signals are considered as the predicates for security policies. The controllable signals of the IPs are deemed as the action tuples that can be exploited to implement an arbitrary number of policies for the SoC. Possible implementation scenarios of maximum number of arbitrary policies is analyzed under different cases e.g. 2 predicates, 1 action; 4 predicates, 2 actions; 8 predicates, 2 actions, etc. The debug instrumentation based design provided better results due to enhanced accessibility to security critical signals.

A comparison of overhead results between the FPGA-based policy engine and the micro-controlled version is illustrated in Table 4. Testbenches are developed to exercise a set of 10 security policies for specific case scenarios. For the DLX μP based implementation, the execution of each policy involved between 15 and 20 instructions. The dynamic and static power analysis is performed by the observing the signal activities caused by the application of representative testbenches. The voltage and thermal models considered for the calculations are obtained from the standard 32 nm technology library. The total power is determined by adding the dynamic and static power values obtained through the simulation. Furthermore, the total energy is measured by multiplying the total power results by the latency values obtained for 10 policies.

In the FPGA-based design, the critical parameters like latency, die area, total power, total energy are calculated using the values obtained by the Altera Quartus tool. Once the design is compiled via Quartus tool, the partial and fully utilized arithmetic logic modules (ALMS) are multiplied with their respective areas to get the total die area. The dynamic and static power calculation is done by observing the signal activity triggered by the same set of testbenches utilized for the micro-controlled version of the policy engine. An estimation of the net leakage power for the embedded FPGA is made by multiplying the total logic utilization factor with the static power reported by the tool for the entire FPGA chip.

It is evident from Table 4 that the FPGA-based RSPE is more energy efficient (around 5.02X) than the DLX μP based policy engine. Furthermore, the MCU implementation takes on average 5.5x time compared to FPGA based design to execute

Table 4 A comparative analysis among the different aspects of design overhead i.e. area, performance, power, and energy is presented in the table. It shows the overhead of the DLX microprocessor based policy engine and embedded FPGA based reconfigurable policy engine module

	Die area (μm^2)	Clock frequency (MHz)	Cycle count 10 policies	Total latency (μs)	Dynamic power (mW)	Static power (mW)	Total energy (nJ)
DLX μP	0.724	203	210	1.04	14.27	63.48	80.86
FPGA	1.06	138	26	0.189	64.9	20.43	16.13
Ratio	**0.68**	1.47	8.07	**5.49**	0.22	3.11	**5.02**

Table 5 Execution result of each policy on the FPGA based reconfigurable security policy engine

Security policy no.	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Energy (nJ)	1.865	1.842	1.851	1.876	1.861	1.839	1.846	1.85	1.868	1.875
Latency (ns)	21.74	14.48	7.24	21.74	21.74	14.48	21.74	14.48	21.74	7.24
Resources (ALMs)	5465	4065	3260	5465	4065	4065	5465	4065	5465	3260

these policies. As many IoT devices operate under strict energy constraints and often require real-time security protection and mitigation, the superior performance and energy efficiency of the FPGA implementation are of critical benefits compared to micro-controlled architecture. Obviously, these comparative values might vary with the choice of different processors, but we expect the relative trends to hold for most scenarios. In any case, these experiments point to the trade-offs and analysis that must be performed in practice in selecting the deployment platform (MCU vs. embedded FPGA) for any target SoC design with custom security requirements. Note that the area cost in the reconfigurable fabric from the experiments is about 1.47 times that of the MCU implementation. The higher area cost in FPGA arises because DLX μP is a simple RISC core and its area is typically low compared to other microprocessor cores such as MinuteIA, or ARM used in practical MCU implementations.

To provide a sense of the energy cost in FPGA for individual policy implementation, the energy consumed by each is presented in Table 5. Standard testbenches are applied to execute each policy in an isolated manner. The energy costs incurred by various policies fall in a close range. The occurrence of such similar energy costs happened primarily due to the fact that the policies enforce similar access controls. Consequently, the policies require similar computation on each of the IP frames. Besides, the duration of all the policies varies over 3 cycles (2 cycles for reading corresponding event frames once the buffer flag is raised and 1 cycle for executing the policy based on the security status). Obviously, in practice, the policies may involve multiple cycles of execution. Furthermore, several policies might get executed together, resulting in significant overlapping of performance and energy costs. Nevertheless, the results provide a relative comparison of the FPGA implementation of various policies, and, in practice, may suggest targets for optimizations. A comparative analysis of area overhead for microcontrolled and FPGA-based policy engine implementation is presented in Table 6. The table shows the area overhead introduced by the addition of micro-controlled and FPGA-based policy engine in the entire SoC model. Though the overhead of the RSPE seems high (i.e. 20–30% approx.) for the given SoC model, the overhead is reasonably low when compared to realistic industrial SoCs like Apple A6 and Qualcomm Snapdragon 800. The results indicate that the FPGA-based RSPE has 0.68x higher area overhead compared to the micro-controlled policy engine. However, the total area overhead of the FPGA-based RSPE is less than 5% in case of industrial SoCs.

Table 6 A realistic comparison on area overhead of RSPE in system-on-chips

SoC	Original area (μm^2)	μC design overhead (%)	FPGA design overhead (%)
SoC model	13.1×10^6	21.7	30.74
Apple A6 (APL0598)	96.71×10^6	2.92	4.26
Qualcomm Snapdragon 800	118.3×10^6	2.39	3.49

7 Conclusion

SoC Security architectures designed for IoT and automotive applications need to be adaptable, especially in the face of evolving requirements and uncertainties over their long lifetime. This chapter considers existing limitations of such adaptability and demonstrates a viable *patchable hardware* implementation through a centralized Reconfigurable Security Policy Engine, a set of smart security wrappers, and on-chip Design-for-Debug (DfD) infrastructure. In the proposed architecture, the policy engine acts as the centralized security brain, the smart security wrappers extract critical signals from IP blocks of varying structural properties, and the DfD modules provides extensive observability and controllability over the required signals. The work also introduces an automatic synthesis flow to facilitate the mapping process of an arbitrary number of SoC security policies. The proposed CAD flow is compatible with existing commercial tools. Significantly higher access to required signals and capability to implement a vast number of arbitrary security policies are distinct properties of the design. Moreover, the proposed hardware-software framework facilitates the policy implementation at the cost of minimal overhead. The proposed work is highly cost-efficient compared to conventional software-based implementation in terms of performance, power, and energy. For a set of illustrative policies, it is shown that an order of magnitude reduction can be obtained in cost of performance, power, and energy overhead. Future work will involve evaluating the architecture on industrial SoC models with test chip fabrication.

References

1. Evans, D.: The internet of things—how the next evolution of the internet is changing everything. White Paper, Cisco Internet Business Solutions Group (IBSG) (2011)
2. Sastry, M.R., Schoinas, I.T., Cermak, D.M.: Method for enforcing resource access control in computer system. US Patent 20120079590 A1 (2012)
3. Krstic, S. et al.: Security of SoC firmware load protocol. HOST (2014)
4. Li, X. et al.: Sapper: a language for hardware level security policy enforcement. ASPLOS (2014)
5. Rushby, J.: Noninterference, transitivity, and channel-control security policies. SRI International, Computer Science Laboratory (1992)

6. ARM: Building a secure system using trustzone technology. ARM Limited (2009)
7. Basak, A., Bhunia, S., Ray, S.: A flexible architecture for systematic implementation of SoC security policies. IEEE ICCAD (2015)
8. Miettinen, M., Heuser, S., Kronz, W., Sadeghi, A.-R., Asokan, N.: Conxsense: automated context classification for context-aware access control. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, pp. 293–304. ACM (2014)
9. Conti, M., Crispo, B., Fernandes, E., Zhauniarovich, Y.: Crêpe: a system for enforcing fine-grained context-related policies on android. IEEE Tran. Inf. Forensics Sec. **7**(5), 1426–1438 (2012)
10. Hull, R., Kumar, B., Lieuwen, D., Patel-Schneider, P.F., Sahuguet, A., Varadarajan, S., Vyas, A.: Enabling context-aware and privacy-conscious user data sharing. In: 2004 IEEE International Conference on Mobile Data Management, 2004. Proceedings, pp. 187–198. IEEE (2004)
11. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy, IEEE, pp. 11–11 (1982)
12. Ray, S., Yang, J., Basak, A., Bhunia, S.: Correctness and security at odds: post-silicon validation of modern SoC designs. In: DAC (2015)
13. Amtoft, T., Bandhakavi, S., Banerjee, A.: A logic for information flow in object-oriented programs. ACM SIGPLAN Notices **41**(1), 91–102 ACM (2006)
14. Alpern, B., Schneider, F.B.: Recognizing safety and liveness. Distrib. Comput. **2**(3), 117–126 (1987)
15. Borisov, N., Johnson, R., Sastry, N., Wagner, D.: Fixing races for fun and profit: how to abuse a time (2005)
16. Ray, S., Jin, Y.: Security policy enforcement in modern soc designs. In: 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Nov 2015, pp. 345–350
17. Loucaides, J., Furtak, A.: A new class of vulnerability in SMI handlers of BIOS/UEFI firmware. In: The 15th Annual CanSecWest Conference (CanSecWest) (2015)
18. Nath, A.P.D., Ray, S., Basak, A., Bhunia, S.: System-on-chip security architecture and cad framework for hardware patch. In: 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), Jan 2018, pp. 733–738
19. Basak, A., Bhunia, S., Ray, S.: Exploiting design-for-debug for flexible SoC security architecture. In: DAC (2016)

Lightweight Fault Tolerance for Secure Aggregation of Homomorphic Data

Nektarios Georgios Tsoutsos and Michail Maniatakos

Abstract Homomorphic encryption constitutes a powerful cryptographic method that enables data aggregation in distributed applications over large datasets, such as electronic voting, electronic wallets, secure auctions, lotteries and secret sharing. At the same time, as attack trends move towards the lower levels of the computation stack and new threats continue to emerge, the lack of trust in contemporary computing paradigms keeps increasing. Since, homomorphic encryption helps preserve the confidentiality of sensitive information, it offers a powerful countermeasure against contemporary and future privacy threats, while allowing meaningful processing even though the data remains unreadable. Nevertheless, when homomorphic primitives are mapped to hardware circuits to improve performance, they become vulnerable to random faults and soft errors since homomorphic operations are malleable by construction and do not provide any explicit assurance towards data integrity. In this chapter, we present a fault tolerance methodology that protects homomorphic aggregation circuits through concurrent detection of random errors in homomorphic ALUs and encrypted values stored in memory. Our approach establishes the theoretical foundations to extend residue numbering to additive homomorphic operations, which enables lightweight fault detection with detection rates of more than 99.98% for ALU operations, and 100% for clustered faults and single bitflips in memory values. Using an efficient modular reduction algorithm, our method incurs a performance overhead between 3.6 and 8%, for a minimal area penalty.

N. G. Tsoutsos (✉)
University of Delaware, Newark, DE 19716, USA
e-mail: tsoutsos@udel.edu

M. Maniatakos
New York University, Abu Dhabi, UAE
e-mail: michail.maniatakos@nyu.edu

R. S. Chakraborty et al. (eds.), *Security and Fault Tolerance in Internet of Things*,
Internet of Things, https://doi.org/10.1007/978-3-030-02807-7_5

1 Introduction

As computing devices become more ubiquitous every day, users rely on them for storage and processing of their sensitive data [36]. From health information and banking records, to location services and file backups, computers are the virtual containers of the digital footprints of millions of users. At the same time, distributed computing paradigms such as wireless sensor networks and grid computing have become increasingly popular, as they allow efficient and scalable computation compared to monolithic dedicated machines. Likewise, from electronic voting [11] and electronic wallets, to electronic auctions and lotteries [15], as well as secret sharing and fingerprinting protocols [25], data aggregation remains a prevalent application on large datasets. Nevertheless, there is no shortage of concerns regarding the privacy guarantees offered by contemporary computing platforms, especially in case sensitive data are outsourced to untrusted third parties [8].

In several cases, there is financial incentive for the hosts of distributed computing to violate user privacy and extract identifiable information (e.g., to develop targeted advertisement campaigns). Moreover, as commercial cloud service providers are compromised [2, 33], mass surveillance instances are reported in whistle-blower cases [26], and private companies are compelled to digitally sign insecure software [22], the debate on the importance of user privacy becomes even more intense. Furthermore, as attack trends keep moving towards lower levels in the computation stack [4, 40, 41, 45, 49], the portrayal of the hardware as the computation root-of-trust continues to be challenged.

Nevertheless, an important observation is that privacy threats are a concern *only if the data is unencrypted* [43]. In case encryption algorithms are used for protection, the risk of privacy violations is minimized, as any leaked data would be unreadable. This is particularly useful, as it is not always possible to foresee the nature of attacks that will emerge in the future, while encryption offers oblivious protections against generic privacy threats. Hence, using cryptographic primitives such as *homomorphic encryption* [14] that enables meaningful manipulation of encrypted values (such as data addition or multiplication), it is possible to return control of data privacy back to the end users, while the data is being processed.

Even though homomorphic encryption presents multiple benefits with respect to privacy, contemporary implementations in software incur prohibitive performance overheads. One direct approach to mitigate these performance concerns would be to use custom hardware units that evaluate homomorphic operations natively [44]. Integrated circuit (IC) implementations, however, are vulnerable to different classes of random faults that may compromise the integrity of encrypted results. In particular, since homomorphic encryption is malleable by construction [24], without dedicated countermeasures it may not be possible to detect whether a homomorphic ciphertext output has a fault or not. Moreover, since common homomorphic schemes are based on special properties of modular arithmetic, fault tolerance is not a trivial challenge, and traditional methods such as Residue Number Systems (RNS) [16] are not directly applicable. In addition, since encrypted arguments have very large bitsizes (often,

thousands of bits), integrity techniques like parity checks and error-correcting codes *yield unacceptable overheads* with respect to bit redundancy.

Notably, since common homomorphic operations are naturally compatible with modular arithmetic, it is possible to extend the inherent mathematical properties of RNS and create a lightweight fault tolerance framework. Specifically, the methodology introduced in this chapter is tailored to arithmetic logic units (ALUs) for homomorphic data aggregation, it incurs minimal overheads with respect to integrity metadata, and it is theoretically founded on modular arithmetic properties. Our approach enables concurrent detection of random errors in ALU operations and encrypted arguments stored in memory, and it is realized using a *fast modular reduction* procedure that exploits the properties of *Mersenne prime* moduli to drastically improve runtime performance.

The rest of this chapter is organized as follows: in Sect. 2 we elaborate on background notions, while in Sect. 3 we present our assumed threat model. Our theoretical arguments on adapting residue numbering for modular arithmetic are discussed in Sect. 4, and in Sect. 5 we present our algorithm for fast modular reductions. Our main methodology on fault tolerance is presented in Sect. 6, while in Sect. 7 we discuss our experimental setup and evaluation results. Finally, related work is discussed in Sect. 8, and the chapter concludes in Sect. 9.

2 Preliminaries

2.1 Cryptographic Schemes and Homomorphic Encryption

In modern cryptography, a pair of encryption and decryption algorithms (also called *encryption scheme*) is used to map *plaintexts* into unreadable *ciphertexts*, based on a secret key. One necessary property that encryption algorithms should offer is the computational indistinguishability of ciphertexts compared to truly random values [24]. Essentially, it should be impossible to define an algorithm with polynomial-time complexity that discriminates an encrypted value from a truly-random sequence of bits. In effect, ciphertext data look like random data, and this property offers an effective solution against privacy threats by rendering the data inaccessible to unauthorized entities. Encryption schemes can either be *symmetric*, when the same secret key is used for both decryption and encryption, or *asymmetric*, when different keys are used for encryption and decryption (called *public* and *private* keys respectively).

Two essential features of encryption schemes are the *diffusion* and *confusion* properties [37]. The latter warrants that a sequence of ciphertext bits depends on multiple bits of the secret key. Likewise, the former warrants that a minimal modification in a plaintext (such as flipping one bit) would yield a significantly different ciphertext under the same encryption key (i.e., multiple ciphertext bits are affected). Naturally, diffusion would also hold in the inverse direction, so a minimal modification in a ciphertext would yield a significantly different plaintext under the same decryption key.

In addition to confusion and diffusion, one characteristic property of encryption algorithms is *malleability*, which describes the potential to manipulate an encrypted value in a meaningful way. Specifically, if an encryption scheme is malleable, it is possible to transform any ciphertext to a new one that decrypts to a plaintext message with a known relationship to the message encoded in the original ciphertext. Depending on the application, malleability could be a desirable or entirely undesirable property, and there exist encryption algorithms that are non-malleable by design (e.g., [31]).

At the same time, there exist malleable encryption schemes with *homomorphic* properties, which enables *meaningful computations* directly over already encrypted values [32]. In particular, homomorphic encryption schemes support applying known arithmetic operations directly over ciphertexts so that the decryption of the result would be equal to the output after applying a related arithmetic operation on the corresponding plaintexts (i.e., as if encryption was not involved in the process). Given two plaintext messages m_1 and m_2, the homomorphic property can be formally expressed using the following formula:

$$\text{Decrypt}[\text{Encrypt}[m_1] \circ \text{Encrypt}[m_2]] = m_1 \diamond m_2, \tag{1}$$

where $\diamond$ and $\circ$ are associated arithmetic operations.

For example, in both the ElGamal [12] and the RSA schemes [34], the modular multiplication of two ciphertexts is homomorphic to the modular multiplication of the corresponding plaintexts. Such schemes are called *multiplicative homomorphic*, as the $\circ$ operation applied on ciphertexts yields a preimage of the multiplication of their plaintexts. Likewise, in the Paillier cryptosystem [30] and the exponential ElGamal variant [10], the modular multiplication of two ciphertexts is homomorphic to the modular addition of the corresponding plaintexts. Such schemes are *additive homomorphic*, as the $\circ$ operation applied on ciphertexts yields a preimage of the addition of their plaintexts.

Both additive and multiplicative schemes are further classified as *partially homomorphic encryption (PHE)* schemes, as only one homomorphism (i.e., either multiplication or addition of plaintexts) is allowed [14]. Until a decade ago, all known homomorphic encryption schemes were partially homomorphic; since 2009, however, and the invention of the first *fully homomorphic encryption (FHE)* scheme, it is possible to have both additive and multiplicative homomorphisms at the same time [9, 13, 17, 38, 39, 46].

2.2 Paillier Additive Homomorphic Encryption

The Paillier cryptosystem is among the first efficient PHE schemes that enables homomorphic addition [30]. The security of this scheme is reduced to the *decisional composite residuosity assumption*, which asserts that:

Given a composite number N and an integer z, it is *hard* to decide whether there exists y such that

$$z \equiv y^N \pmod{N^2}. \tag{2}$$

The Paillier cryptosystem is further classified as a probabilistic asymmetric encryption scheme and it is formally defined as follows:

Let u and v be two large primes of equivalent length, randomly and independently chosen of each other. Let $N = u \cdot v$ be the product of these primes and $\lambda = \text{LeastCommonMultiple}(u-1, v-1)$; the bitsize of N is the *security parameter* of the cryptosystem. Let g be a random integer in $\mathbb{Z}^*_{N^2}$ so that $\mu = (L(g^\lambda \pmod{N^2}))^{-1} \pmod{N}$ exists (i.e., the order of g is greater than zero and also divided by N), where the $(\cdot)^{-1}$ power denotes modular multiplicative inversion (over modulus N) and $L(x) = \frac{x-1}{N}$. The private key is (λ, μ), while the public key is (N, g).

The encryption operation is defined as follows: Let m be a message to be encrypted, with m in $\mathbb{Z}_N$, and let r be a random integer in $\mathbb{Z}^*_N$; the randomness of r is essential, as it ensures the *semantic security* of the probabilistic Paillier scheme [19]. Then, the encryption function of any message m is defined as the unique mapping between a ciphertext $c \in \mathbb{Z}^*_{N^2}$ and values r and m:

$$c = \text{Encrypt}[m] = g^m \cdot r^N \pmod{N^2} \tag{3}$$

and the decryption function of any ciphertext c is defined as:

$$m = \text{Decrypt}[c] = L(c^\lambda \pmod{N^2}) \cdot \mu \pmod{N}. \tag{4}$$

In Paillier, the complexity of the decryption algorithm is equivalent to one modular exponentiation operation, which makes the scheme practical. This is not always the case with additive homomorphic schemes: for example, the exponential ElGamal cryptosystem [10] incurs high decryption overheads, as the decryption complexity reduces to evaluating discrete logarithms, which is intractable in general.

The homomorphism of Paillier is formally defined as:

$$\begin{aligned} \text{Decrypt}[\text{Encrypt}[m_1] \cdot \text{Encrypt}[m_2] \pmod{N^2}] = \\ \text{Decrypt}[r_1^N g^{m_1} \cdot r_2^N g^{m_2} \pmod{N^2}] = \\ \text{Decrypt}[(r_1 r_2)^N g^{m_1+m_2} \pmod{N^2}] = m_1 + m_2 \pmod{N}, \end{aligned} \tag{5}$$

so decrypting the result of the modular multiplication of the encryptions of two messages would be equal to the modular addition of these messages. The latter is a significant result, as this will be the basis for secure homomorphic aggregation.

2.2.1 Paillier Ciphertext Size and Range Expansion

To guarantee asymptotic security against decryptions without knowing the private key (e.g., prevent recovery of the private key by factorizing the public key N), the bitsize of N must be adequately large (usually, thousands of bits long) [18]. By definition of Paillier encryption, ciphertexts are reduced to modulus N^2, so their typical bitsize is twice as large as the bitsize of N. Thus, if N is 1024 bits long, Paillier ciphertexts will be 2048 bits long respectively. Moreover, for a 1024-bit long N, the ciphertext bitsize is always 2048 bits, regardless of the corresponding plaintext bitsize (i.e., ciphertexts have a fixed bitsize for *any* valid plaintext, even if that plaintext is a 64-bit, a 32-bit, or even a 1-bit long value). In effect, the plaintext range is expanded to a fixed ciphertext range.

2.2.2 Paillier Cryptosystem Using Multi-prime Keys

If the calculation of modulus N uses more than two prime factors, the key-pair generation procedure would be modified as follows:

Let $\{p_1, p_2, \ldots, p_k\}$ be a set of k primes so that at least two of these are generated randomly and independently of one another, with equivalent and adequately large bitsizes. In this case, the public modulus equals $N = \prod p_i$ with $1 \leq i \leq k$, while the private parameter λ equals LeastCommonMultiple$(p_1 - 1, p_2 - 1, \ldots, p_k - 1)$.

The asymptotic security of the multi-prime cryptosystem holds, as it would be computationally intractable to compute λ without knowing the prime factors of N. In fact, given that at least two prime factors of N are random and adequately large, factorizing N would be intractable.

3 Assumed Threat Model for Fault Tolerance

This section elaborates on the different threat scenarios that we deem possible, and the various errors we strive to mitigate. The application in our scope is secure data aggregation implemented using additive homomorphic encryption, assuming without loss of generality that Paillier [30] is our baseline cryptosystem.[1] In Paillier, ciphertext modular multiplications are homomorphic to plaintext modular additions so that implementing homomorphic aggregation applications in hardware requires ALUs for *modular multiplication* (e.g., a Montgomery modular multiplication ALU is implemented in [28]). Since a typical bitsize of Paillier ciphertexts would be 2048 bits, the ALUs implementing modular multiplications are expected to consume thousands of cycles before generating an output, and also incur considerable silicon area

[1] As mentioned in Sect. 2.1, another additive homomorphic scheme is the exponential ElGamal [10], yet its decryptions require computing discrete logarithms. In addition, the Damgård-Jurik-Nielsen cryptosystem [11] generalizes the additive homomorphic property of Paillier.

overheads. Thus, there is a non-negligible probability of *permanent* or *transient* faults impacting ALU operations, which is a threat for the *integrity* of homomorphic outputs. In our scope, we consider scenarios where these random faults may impact an arbitrary number of steps associated with the computation of the ALU output.

Along with the modular multiplication ALUs, our threat model further considers memory soft errors affecting cells that store ciphertext data. Specifically, we consider (a) Multi-Bit Upsets (MBUs) where clusters of up to four consecutive bits may be cleared or set together (i.e., clustered faults as in [27]), and (b) Single Event Upsets (SEUs). These faults may impact the integrity of the homomorphic arguments during memory storage, and eventually cause corrupted outputs in case such erroneous ciphertexts are loaded for processing. Our threat model assumes a non-negligible probability for these soft errors, considering that the silicon area occupied by memory cells can be large. On the contrary, we assume that soft errors in individual registers within functional units have negligible probability.

4 Adapting Residue Numbering for Modular Multiplication

Residue Number Systems offer a powerful and popular method for detecting faults in standard ALU operations [23]. Instead of using conventional *digits* whose weights are defined by their position, a residue number system can represent an integer as a tuple of weightless residue values based on a predetermined set of *moduli* [16]. The latter supports carry-free and parallelizable multiplication and addition, as each residue value can be computed independently. Moreover, residue values allow concurrent detection of errors during multiplication or addition in an ALU: the residue values of each ALU input are multiplied or added together and that result is compared with the corresponding residue of the ALU operation output to verify equality.

In this chapter, one of our main objectives is fault detection for modular multiplication ALUs, by adapting the same efficient and simple residue numbering checks that are applicable to standard ALUs. Nevertheless, residue number systems are not directly compatible with complex arithmetic operations such as divisions, strict inequalities, or modular multiplications, and implementing these operations usually requires highly complex algorithms [1]. Furthermore, applying a standard residue-based fault detection check directly to a modular multiplier yields an invalid result.[2]

To address this limitation of residue numbering in the context of verifying modular multiplications, we observe that the public parameter N in Paillier is defined as a product of different primes (Sect. 2.2). Thus, to allow correct residue-based checks, an appropriate solution would be to define the set of moduli used for the residue number

[2]For example, if we use RNS modulus 7 to reduce the output of modular multiplication $20 \times 30 \pmod 9$, we have: $((20 \times 30) \bmod 9) \bmod 7 = (600 \bmod 9) \bmod 7 = 6 \bmod 7 = 6$. This is different from: $(((20 \bmod 7) \times (30 \bmod 7)) \bmod 9) \bmod 7 = ((6 \times 2) \bmod 9) \bmod 7 = (12 \bmod 9) \bmod 7 = 3$.

system with the primes composing the public modulus N in Paillier. Still, if the modulus N can be factorized using exactly two distinct prime factors v and u (i.e., N equals the product of v and u), disclosing either prime as part of the residue numbering moduli set essentially reveals Paillier's private key that should remain secret. Our observation is that, in order to enable fault detection using lightweight residue-based checks while preserving cryptographic security, we can introduce another integer z as a factor of the public modulus. To ensure correctness, the new factor z of modulus N should also be a prime, similar to the RSA cryptosystem configuration with multiple primes [5]. Even though z can be publicly disclosed, the prime factors v and u must always be kept secret (Sect. 2.2.2).

In the rest of this section, we offer theoretical foundations for adapting residue number systems in modular multiplication operations.

Representation: In a residue number system, non-negative integers are uniquely represented as tuples of residue values. Each residue value corresponds to one predefined modulus from a set of positive integers forming the *basis* of the numbering system. More formally, the residue numbering basis is defined as a tuple of k positive moduli $(m_1, m_2, \ldots, m_k)$ so that:

$$gcd(m_i, m_j) = 1, \quad \forall i, j \in \{1, 2, \ldots, k\}, i \neq j \tag{6}$$

(i.e., all moduli in the basis are pairwise coprimes). Then, for a given basis, any non-negative integer value X smaller than $M = \prod m_i$ is uniquely represented as a tuple of k integer residues $(x_1, x_2, \ldots, x_k)$ so that $x_i = X \bmod m_i$.

Uniqueness: The representation of non-negative integers using a residue number system is always unique as required by the following theorem (based on the Chinese Remainder Theorem).

Theorem 1 *Given a residue numbering basis of pairwise coprime positive moduli $(m_1, m_2, \ldots, m_k)$, then for any integer $X \in [0, M)$, where $M = \prod m_i$ for $1 \leq i \leq k$, the set of residues $X \bmod m_i$ is unique.*

Proof If the statement of the theorem did not hold, there would exist two distinct integers A and $B \in [0, M)$ with identical residue number system representations. Hence, for all i we would have $A = c_i \cdot m_i + a_i$, $B = d_i \cdot m_i + b_i$, $c_i \neq d_i$ and $a_i = A \bmod m_i = B \bmod m_i = b_i$, so $A - B$ would be a multiple of m_i:

$$A - B = (c_i - d_i) \cdot m_i, \quad \forall i \text{ and } c_i \neq d_i. \tag{7}$$

The latter also denotes that $A - B$ is a multiple of $lcm(m_1, m_2, \ldots, m_k)$, which is equal to M, given that all m_i are pairwise coprimes. Nevertheless, if $A - B$ is a multiple of M, then it is impossible for both A and B to be in the interval $[0, M)$ (*contradiction*). □

Simple Arithmetic: Using a residue number system, it is possible to apply carry-free multiplication, addition or subtraction directly over the residues of two input

values, and compute the residue number system representation of the output of the corresponding arithmetic operation. Formally, if the residue number system representation of inputs A and B is $(a_1, a_2, \ldots, a_k)$ and $(b_1, b_2, \ldots, b_k)$ respectively, then the residue number system representation of the output $X = (A \diamond B) \bmod M$ would be $(x_1, x_2, \ldots, x_k)$, where $x_i = (a_i \diamond b_i) \bmod m_i$, $M = \prod m_i$, and $\diamond$ corresponds to multiplication, addition or subtraction.

Nested Reductions: As we prove in the following theorem, under certain conditions it is possible to compute modular reductions that are *nested*. This result is particularly useful in order to compute a residue representation of a value that has *already been reduced* to a given modulus (such as a ciphertext reduced modulo N).

Theorem 2 *If N and z are positive integers so that z divides N, then for any non-negative integer X we have:*

$$(X \bmod N) \bmod z = X \bmod z. \tag{8}$$

Proof Let $a = X \bmod N$ and $b = a \bmod z = (X \bmod N) \bmod z$. Therefore, it holds that $X = k \cdot N + a$ for some integer k and $a = j \cdot z + b$ for some integer j. If we combine the two equalities, we observe that $X = k \cdot N + j \cdot z + b$. Given that N is divided by z, there should exist an integer m such that $N = m \cdot z$; hence,

$$X = k \cdot m \cdot z + j \cdot z + b = z \cdot (k \cdot m + j) + b. \tag{9}$$

If the previous equation is divided by z, a remainder b will be produced:

$$X \bmod z = (z \cdot (k \cdot m + j) + b) \bmod z = b. \tag{10}$$

Considering that $X \bmod z = b$ and $(X \bmod N) \bmod z = b$ using our definition of b, we conclude that $(X \bmod N) \bmod z = X \bmod z$, when $X \in \mathbb{Z}_{\geq 0}$, and $z, N \in \mathbb{Z}_+$ so that N is divided by z. □

Modular Multiplication: The result of Theorem 2 is fundamental in our fault tolerance methodology, since it can be applied to modular multiplication ALUs operating on encrypted values that have *already been reduced* to a public modulus N.[3]

Corollary 1 *If N and z are positive integers and N is divided by z, then for any pair of non-negative integers X and Y the following equation holds:*

$$(X \cdot Y \bmod N) \bmod z = ((X \bmod z) \cdot (Y \bmod z)) \bmod z.$$

Proof Using the result of Theorem 2, we have that:

$$(X \cdot Y \bmod N) \bmod z = X \cdot Y \bmod z. \tag{11}$$

[3]Notably, if the divisibility conditions outlined in the Corollary are not satisfied, multiplying the residues of arguments X and Y would return an inaccurate residue value for the output product.

Let $a = (X \bmod z)$, $b = (Y \bmod z)$ and $c = (X \cdot Y \bmod z)$. In this case, there exist some integers x and y such that $X = (x \cdot z + a)$ and $Y = (y \cdot z + b)$. Applying these expressions for X and Y, the product $X \cdot Y$ equals:

$$X \cdot Y = (x \cdot z + a) \cdot (y \cdot z + b) = x \cdot y \cdot z^2 + x \cdot z \cdot b + y \cdot z \cdot a + a \cdot b, \quad (12)$$

$$X \cdot Y = z \cdot (x \cdot y \cdot z + x \cdot b + y \cdot a) + a \cdot b. \quad (13)$$

If we set a new integer $w = x \cdot y \cdot z + x \cdot b + y \cdot a$, Eq. 13 is transformed to $X \cdot Y = w \cdot z + a \cdot b$ so that

$$X \cdot Y \bmod z = (w \cdot z + a \cdot b) \bmod z = a \cdot b \bmod z, \quad (14)$$

by observing that $z \cdot w$ is divided by z (i.e., $a \cdot b$ and $z \cdot w + a \cdot b$ are *congruent* modulo z). Finally, if we combine Eqs. 11 and 14, we obtain the result promised in the Corollary. □

5 Fast Modular Reductions Using Mersenne Primes

Following the analysis in Sect. 4, one essential condition to use residue number systems is the ability to reduce integer inputs using a fixed modulus. Notably, modular reduction implementations that support non-fixed (arbitrary) moduli could introduce considerable overheads, impacting the runtime performance of a system. In particular, since these reductions require integer division of an arbitrary input (i.e., the dividend) with the corresponding non-fixed modulus (i.e., the divisor), this overhead could be significant for Paillier ciphertext inputs in the order of 2048 bits. Although fast modular reduction methods have been reported in the literature (e.g., [6]), the runtime cost is a linear function of the dividend bitsize.

With respect to verifying modular multiplications using a residue number system, our key observation is that the modulus z in Corollary 1 *does not need to be random*, except for the correctness condition to be a prime number that divides the Paillier modulus N (Sect. 4). As we demonstrate, a judicious selection of modulus z to be a prime number that is smaller by exactly 1 from a power of 2 allows significant efficiency improvements when an arbitrary input is reduced modulo z. An integer z with such special property is actually a *Mersenne prime* [35], which is formally expressed as $M_p = 2^p - 1$ so that both p and M_p are prime numbers.

To complement our fault-tolerance methodology, we present an efficient modular reduction algorithm that is applicable in case Mersenne primes $2^p - 1$ are used as the reduction modulus. Specifically, given that any integer value can be represented with radix-2^p digits, we can compute a reduction modulo a Mersenne prime $M_p = 2^p - 1$ by first evaluating the *summation* of all these digits and then reducing the sum modulo M_p, as illustrated in Algorithm 1. Such special-case reductions can be very fast, as their runtime overhead is not a function of the bitsize of the input, but instead a

Algorithm 1 Fast Reduction Modulo a Mersenne Prime M_p

Input: X, p so that both p and $2^p - 1$ are primes
Output: $Remainder$
1: **procedure** FASTMOD(X, p)
2: $M_p \leftarrow 2^p - 1$
3: **while** $X > 2 \cdot M_p - 1$ **do**
4: $Acc \leftarrow 0$
5: **while** $X \neq 0$ **do**
6: $Acc \leftarrow Acc + X \& M_p$ ▷ Add a masked radix-2^p digit to Acc
7: $X \leftarrow X \gg p$ ▷ Shift X right by p bits
8: $X \leftarrow Acc$ ▷ Recursion over all radix-2^p digits of Acc
9: **if** $X \geq M_p$ **then**
10: $X \leftarrow X - M_p$
11: **return** $Remainder = X$

linear function of the number of digits in the input, expressed in radix-2^p (i.e., the number of p-bit input blocks). The correctness of Algorithm 1 is demonstrated in the following theorem.

Theorem 3 *Let p be a prime and $M_p = 2^p - 1$ a Mersenne prime. Then, for any positive radix-2^p integer X in the form:*

$$X = \sum_{i=0}^{\lfloor \log_{2^p} X \rfloor} a_i \cdot (2^p)^i, \tag{15}$$

where 2^p is the numbering system base *of the representation and a_i are the digits of integer X in that base, the reduction of X modulo M_p equals:*

$$X \bmod M_p = \left(\sum_{i=0}^{\lfloor \log_{2^p} X \rfloor} a_i \right) \bmod M_p. \tag{16}$$

Proof Let $a'_i = a_i \bmod M_p$ and $b'_i = (2^p)^i \bmod M_p$ be the reductions modulo M_p of each digit a_i and each base power $(2^p)^i$ respectively. In this case, using the definition of *congruence* [24], we observe that a_i and a'_i are congruent modulo M_p for each i, and the same property is true for each $(2^p)^i$ and b'_i as well. Leveraging the properties of congruence, it is possible to apply standard arithmetic rules with respect to multiplication and addition over congruent integers. Hence, for each $i \geq 0$, it is possible to apply the reductions modulo M_p *distributively* to individual digits and base powers:

$$(a_i \cdot (2^p)^i) \bmod M_p = (a'_i \cdot b'_i) \bmod M_p. \tag{17}$$

The same observation holds for each product term in the summation of Eq. 15:

$$X \bmod M_p = \left(\sum_{i=0}^{\lfloor \log_{2^p} X \rfloor} a_i \cdot (2^p)^i \right) \bmod M_p = \left(\sum_{i=0}^{\lfloor \log_{2^p} X \rfloor} \left(a_i \cdot (2^p)^i \right) \bmod M_p \right) \bmod M_p. \tag{18}$$

Similarly, for each $i \geq 0$, it is also possible to distribute the reductions modulo M_p to the individual (2^p) factors composing each base power $(2^p)^i$ (when $i = 0$, we assume that the *empty product* equals 1):

$$(2^p)^i \bmod M_p = \left(\prod_{1}^{i} (2^p \bmod M_p) \right) \bmod M_p. \tag{19}$$

Combining Eqs. 17, 18 and 19, with our definitions of a'_i and b'_i, we obtain the following equation:

$$X \bmod M_p = \left(\sum_{i=0}^{\lfloor \log_{2^p} X \rfloor} \left(a'_i \cdot \prod_{1}^{i} (2^p \bmod M_p) \right) \right) \bmod M_p. \tag{20}$$

In the last equation, it is possible to simplify the product terms since we know that $2^p \bmod M_p = (M_p + 1) \bmod M_p = 1$, using the definition of M_p. Lastly, based on our definition of a'_i, we observe that $(\sum a'_i) \bmod M_p = (\sum (a_i \bmod M_p)) \bmod M_p$, which is equal to $(\sum a_i) \bmod M_p$ for each i, after applying standard rules of arithmetic. Hence, we can simplify Eq. 20 to Eq. 16 as promised. □

6 Lightweight Error Detection for Random Faults

One important concern towards mitigating integrity threats in homomorphic aggregation applications is the ability to detect errors attributed to random faults. Typically, these faults can be caused by low energy neutrons from cosmic radiation interacting with the insulator layers of integrated circuits (ICs), as well as by high energy cosmic neutrons, or can be attributed to alpha particles emitted due to impurities in IC packaging [3]. The following sections discuss our approach that enables efficient detection of such random faults in additive homomorphic ALUs and in memories hosting ciphertext arguments. An overview of our fault tolerance methodology for modular multiplication ALUs is illustrated in the block diagram of Fig. 1.

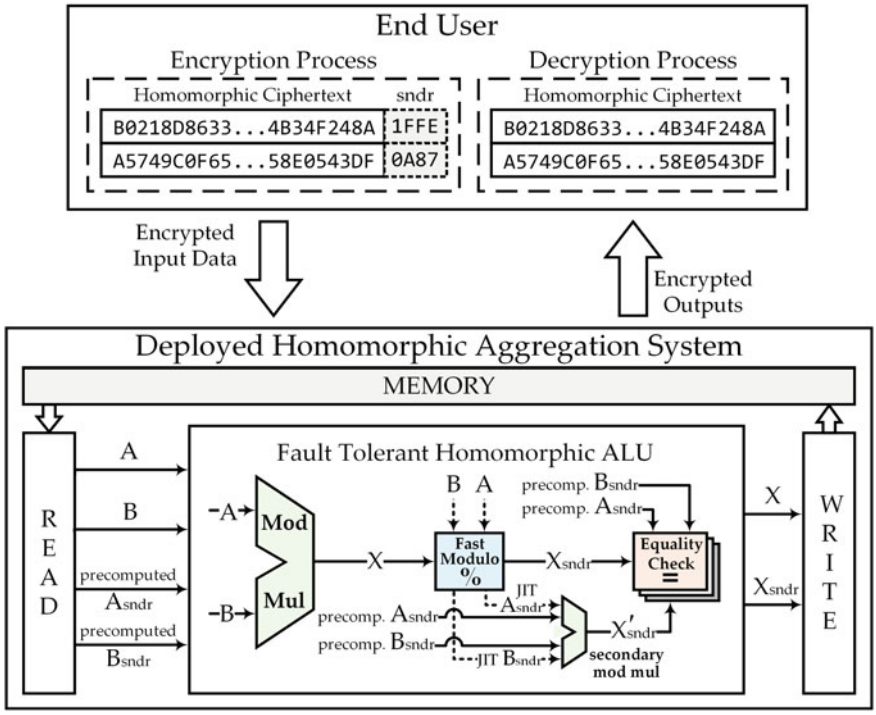


Fig. 1 Abstract view of a homomorphic aggregation system featuring an additive homomorphic ALU with fault tolerance and redundant equality checks. In typical aggregation applications (e.g., secure electronic voting), the end users encrypt their sensitive data using homomorphic encryption and send the corresponding ciphertexts to a remote system that performs homomorphic addition; when the final results are evaluated, the end users download the ciphertext outputs and decrypt them locally

6.1 Detecting Errors in Modular Multiplication ALUs

To enable the detection of random errors in the output X of modular multiplication ALUs, we employ our results for adapting residue number systems in modular multiplication (Sect. 4) and construct special homomorphic *syndromes*. Each syndrome is defined as the residue $sndr = X \bmod M_p$ of an integer value X, using a predefined Mersenne prime M_p as the modulus. Notably, such reductions are efficiently computed using additions of p-bit blocks, as illustrated in Algorithm 1. Moreover, as discussed in Sect. 4, a necessary condition for both correctness and security is that Paillier's public parameter N is factorized to $k > 2$ primes, where one of them is a Mersenne prime M_p.[4]

[4]In this case, the public parameter N of the cryptosystem is the product of all k prime numbers, while the private parameter is *Carmichael's* λ *function* of N (Sect. 2.2.2). We remark that the reduction modulus for homomorphic additions of two ciphertexts is N^2 [30].

Algorithm 2 Modular Multiplication with Error Detection

Input: A, B, N, M_p so that N is divided by M_p
Output: (Modular Multiplication Result, Updated Syndrome, Fault Status)
1: **procedure** MODMUL-ED-ALU(A, B, N, M_p)
2: $X \leftarrow A \cdot B \bmod N^2$
3: $X_{sndr} \leftarrow X \bmod M_p$
4: $A_{sndr} \leftarrow A \bmod M_p$ ▷ For Just-In-Time syndromes
5: $B_{sndr} \leftarrow B \bmod M_p$ ▷ For Just-In-Time syndromes
6: $X'_{sndr} \leftarrow A_{sndr} \cdot B_{sndr} \bmod M_p$ ▷ Parallelizable step
7: **if** $X'_{sndr} = X_{sndr}$ **then** ▷ Redundant equality comparison
8: **return** $(X, X_{sndr}, Correct)$
9: **else**
10: **return** $(\perp, \perp, Faulty)$

In our case, we assume with no loss of generality that N comprises $k = 3$ prime factors, specifically v, u and M_p; hence, since one of the factors is a Mersenne prime, we can leverage Theorem 2 and Algorithm 1 to evaluate the syndrome of ALU result X very efficiently. As already mentioned, in case N does not have a third factor, adapting residue numbering to enable fault detection would entail revealing either v or u as the modulus for residue numbering, since the latter should also be a divisor of N. Nevertheless, this alternative would undermine the security of the cryptosystem, which requires that at least two sufficiently large prime factors of N must remain secret at all times (i.e., factorizing N should be prevented).

Using our construction for homomorphic syndromes, it is possible to assert (with high probability) that a random error has not occurred in a homomorphic operation, by computing the quantity $X \bmod M_p$ for an ALU output X and verifying that it is equal to the *expected syndrome*. Nevertheless, this correctness check would not be practical unless we can compute the expected syndrome for the ALU output based on the syndromes corresponding to the two ALU inputs. Indeed, for ALU inputs A and B and ALU output $X = A \cdot B \bmod N^2$, it is possible to apply the result of Corollary 1 and verify the following equality for error-free outputs:

$$X \bmod M_p = (A_{sndr} \cdot B_{sndr}) \bmod M_p, \tag{21}$$

where $A_{sndr} = A \bmod M_p$ and $B_{sndr} = B \bmod M_p$. Specifically, as summarized in the error detection procedure of Algorithm 2, knowing A and A_{sndr}, as well as B and B_{sndr} for a given ALU operation, we are able to apply Eq. 21 and verify if X is correct. In general, we can pre-compute offline the initial value of each syndrome, store it along with the corresponding ciphertext in encrypted memory, and update it at runtime to the syndrome matching the ALU output after a homomorphic operation. In particular, as discussed in the next section, pre-computing and storing each syndrome in encrypted memory allows efficient detection of random memory errors as well. Conversely, if only ALU error detection is in scope, we can compute each syndrome in a *Just-In-Time (JIT)* manner, as soon as the corresponding ciphertext input is fetched for ALU processing (so no additional memory is needed for syndrome storage).

As we observe from Eq. 21, computing the expected syndrome of the ALU output requires a second modular multiplication of two input syndromes. In this case, and given that the primary ALU module is allocated for the modular multiplication of ciphertext inputs A and B using modulus N^2, additional ALU hardware is required for the modular multiplication of A_{sndr} and B_{sndr} (each having a much smaller bitsize compared to A and B) using modulus M_p (i.e., not N^2). Notably, if we have separate hardware units, it is possible to parallelize the two modular multiplications of Algorithm 2 (i.e., Steps 2 and 6), so we can mask the overhead of syndrome multiplication with the main homomorphic addition operation of A and B. On the other hand, a prerequisite for calculating X_{sndr}, which is necessary for syndrome verification (Step 3 in Algorithm 2), is the computation of ciphertext X (Step 2). At the same time, in case the fault-tolerant modular multiplication (Algorithm 2) is pipelined, the syndrome computation of Step 3 can be parallelized with *the next* modular multiplication in the pipeline (i.e., Step 2); if an ALU error is eventually detected, then the pipeline is "flushed".

Typically, in case the equality check of Step 7 in Algorithm 2 fails, we consider the following (not mutually exclusive) possibilities: (a) an error occurred at the equality check, (b) an error occurred during the modular multiplication of two syndromes within the secondary ALU, (c) an error occurred during the calculation of a syndrome from its corresponding ciphertext (i.e., within a Mersenne reduction module), (d) an error occurred during the homomorphic addition of two ciphertexts in the primary ALU. If we assume that such errors are attributed to transient faults, it is possible to handle such exceptions by repeating the primary and secondary ALU operations with freshly-calculated syndromes. Similarly, to mitigate *undetectable errors* in equality comparisons (Step 7), it is important to employ redundant equality checking units. Additionally, although our methodology enables the detection of arbitrary many random errors at the output of the primary homomorphic ALU, there is still a residual risk of errors that *escape detection* if the syndrome of a faulty ALU result *collides* with the syndrome of the correct result.[5] Specifically, undetectable collisions are possible when the syndrome X_{sndr} corresponding to a faulty output $X^{(f)}$ is the same as the expected syndrome $X'_{sndr} = A_{sndr} \cdot B_{sndr} \bmod M_p$ (for example, if the faulty result $X^{(f)}$ is congruent to the correct result X so that $X^{(f)} = X + k \cdot M_p$ for some integer $k \neq 0$).

To maximize the efficiency benefits of our error detection methodology, the bitsize of the selected Mersenne prime M_p should be significantly smaller that the bitsize of N^2 (usually by one to two orders of magnitude). Given that our selection of modulus M_p defines the bitsize of our homomorphic syndromes, this also determines the runtime and area overheads of our secondary modular multiplication ALU that computes the expected syndrome in Step 6 of Algorithm 2. As we report in Sect. 7, our experimental evaluation shows that selecting Mersenne prime M_{19} or M_{31}

[5] An undetectable collision is also possible if one error has occurred in the primary ALU output, but it is ultimately masked due to an additional (very precise) error in the computation of either the expected syndrome (i.e., Step 6 in Algorithm 2) or the actual syndrome of the result (i.e., Step 3 in Algorithm 2). In our analysis, we consider that this scenario has negligible probability.

would be sufficient to enable more than 99.999% fault coverage. In particular, the probability of an escaped error that is attributed to syndrome collisions would be ${M_p}^{-1}$. Thus, in case the bitsize of M_p is overly small, the escaped error probability increases; conversely, if the bitsize of M_p is excessively large, this would increase the storage requirements for pre-computed syndromes that are paired with ciphertext arguments in encrypted memory. We remark that a secondary modular multiplication ALU for 31-bit or 19-bit syndromes requires 34 or 22 clock cycles respectively, while the runtime overhead for one homomorphic addition of 2048-bit ciphertext inputs in the primary ALU is 2051 clock cycles [42]; generally, the runtime overhead of homomorphic addition is a linear function of the argument bitsize.

6.2 Detecting Errors in Encrypted Memories

In typical homomorphic aggregation applications, encrypted arguments are stored within memory modules while being processed. Since memory cells are continuously susceptible to soft errors, one fundamental objective is the ability to detect the occurrence of these errors in order to mitigate them (for example, the application can reload the corrupted ciphertexts from redundant memory modules or permanent storage). A common protection mechanism against soft errors is the use of *Error Correcting Code (ECC)* memories (e.g., [50]), which are able to mitigate a limited number of bit errors within each data word. Nevertheless, a shortcoming of this approach is that error-correcting memories can incur non negligible delay and area overheads, since they require additional dedicated ECC circuits. In addition, the number of parity bits required for ECC grows logarithmically with the bitsize of the protected value, so this approach does not scale gracefully, given that the bitsize of each homomorphic ciphertext is in the order of 2048 bits.

As the previous subsection discusses, one alternative for obtaining the syndromes of the homomorphic ALU inputs would be to pre-compute the residue of each ciphertext and fetch it from memory. In particular, each pre-computed syndrome can be stored along with its corresponding ciphertext and form a (ciphertext, syndrome) tuple in memory; these tuples are updated at runtime when intermediate ALU outputs are written back to memory. This option is beneficial in our methodology, as it also allows the detection of random errors in memory *at no further cost*. Specifically, when a ciphertext and its corresponding syndrome are fetched from memory at runtime, we can assert their integrity by calculating the residue modulo M_p of the ciphertext (Algorithm 1) and compare this result with the pre-computed syndrome. Indeed, by checking if the fetched syndrome matches the calculated residue, it is possible to correctly detect the occurrence of random errors in either the ciphertext or its syndrome with probability $1 - {M_p}^{-1}$. Assuming there are redundant checks to prevent equality errors in every comparison, this probability embodies escapes due to collisions, as well as reduction errors.

Conversely, in case no actual soft errors have occurred in memory, any discrepancy in the equality comparisons is considered a false positive, which can be attributed

to errors during residue calculations. The latter is indistinguishable from true positives, so it is handled as a regular error. Moreover, considering that the modular multiplications required for homomorphic aggregation are at least one order of magnitude slower compared to our efficient reductions using Mersenne prime moduli (as supported by our results in Sect. 7.2), the proposed integrity checks for encrypted memory arguments could be executed within the existing reduction units using time-multiplexing. Hence, the memory integrity overhead would be masked (i.e., we can have *delay-free* detection of memory errors), since we can perform the required reductions in parallel to the primary homomorphic ALU operations.

6.2.1 Complete Coverage for Single Memory Bit-Flips

Depending on the number and nature of memory errors, one important remark is that our residue-based detection methodology can offer higher fault coverage, compared to the probabilistic estimate of $1 - {M_p}^{-1}$ mentioned in the previous paragraphs. In particular, since it is algebraically impossible for two congruent values (modulo M_p) to have a Hamming distance equal to 1, our fault coverage would always be 100% when we consider single bit-flips.

If we assume that that X is the original ciphertext in memory and $Y = X \oplus 2^j$ is the faulty ciphertext after a single bit-flip at bit position j of X, then the corresponding single-bit error would be masked if and only if:

$$|X - Y| = k \cdot M_p, \quad \text{for some } k \in \mathbb{Z}_{>0}, \tag{22}$$

which is a congruence relation between X and Y. In our methodology, however, we observe that Eq. 22 cannot be true, since for $x_i \in \{0, 1\}$ and $j \geq 0$ we have:

$$X = \sum_{i=0}^{\lfloor \log_2 X \rfloor} x_i \cdot 2^i \equiv x_j \cdot 2^j + \sum_{i=0, i \neq j}^{\lfloor \log_2 X \rfloor} x_i \cdot 2^i, \tag{23}$$

$$Y = X \oplus 2^j = (1 - x_j) \cdot 2^j + \sum_{i=0, i \neq j}^{\lfloor \log_2 X \rfloor} x_i \cdot 2^i, \tag{24}$$

$$|X - Y| = \left| x_j \cdot 2^j - (1 - x_j) \cdot 2^j \right| = 2^j \quad \forall x_j \in \{0, 1\}. \tag{25}$$

Therefore, in order to have a masked single bit-flip, there should exist integers $k > 0$ and $j \geq 0$ so that $2^j = k \cdot M_p$. In the trivial case where $j = 0$, the equation does not hold for any $k > 0$, as 1 cannot have M_p as a factor. Similarly, if $j > 0$, by the fundamental theorem of arithmetic [21], 2^j should have a *unique prime factorization*; since the latter is a power of 2, it cannot have prime M_p as a factor. As a result, $2^j \neq k \cdot M_p$ for all $k > 0$ and $j \geq 0$, so the fault coverage for single bit-flips is exactly 100%.

6.2.2 Complete Coverage for Clustered Faults

Complete fault coverage can also be shown algebraically for clustered faults of up to 4 bits (as in [27]). In this case, the unique factors of the absolute difference in Eq. 25 are primes 2, 3, 5 and/or 7. Hence, any Mersenne prime M_p with $p > 3$ (i.e., any M_p other than $M_2 = 3$ and $M_3 = 7$) is not a factor either, so the fault coverage is 100% as well.

7 Experimental Evaluation

We evaluate the effectiveness of our lightweight fault-tolerance methodology using Monte Carlo simulations in Python 2.7, and we assess the area and delay overheads of our ALU and memory error detection modules by implementing their designs in Verilog. All simulations are executed on two servers, each comprised of 64 GBs RAM and an 8-core Xeon E5-2650 processor that can execute 16 Python threads at 2.00 GHz; our RTL designs are synthesized using Xilinx XST 14.7 targeting a Kintex xc7k160t-3 FPGA. In addition, our fault model makes the following assumptions:

(a) Regarding the modular multiplication ALU, we consider faults during any iteration of the sequential steps composing one homomorphic operation.
(b) Regarding soft errors in encrypted memory, we consider Single Event Upsets (SEUs) and Multiple Bit Upsets (MBUs) as in [27], where up to 4 consecutive bits are cleared or set together (i.e., clustered faults).

7.1 Fault Coverage Results

ALU and Memory Error Detection: In Table 1 we present the average error detection rates for ALU protection. Our results for different Mersenne prime moduli M_p verify the theoretically expected value of $1 - {M_p}^{-1}$, as discussed in Sect. 6.1. In addition, we evaluated our error detection method for SEUs and MBUs in encrypted memory (according to the aforementioned fault model) by performing 10^{10} Monte Carlo simulations. Our results for Mersenne primes M_{13}, M_{17}, M_{19} and M_{31}, as well as encrypted values of 1024 bits in size, show that the number of escaped faults in our simulations is exactly zero (i.e., we achieve 100% coverage). This result is actually expected due to the use of Mersenne prime moduli, since single bit-flips and clustered faults cannot produce a value *congruent* to the original one, as discussed in Sects. 6.2.1 and 6.2.2.

Table 1 Average ALU error detection rates after 10^{10} Monte Carlo simulations, for different Mersenne primes

Mersenne prime	Average error detection rates (%)
M_{13}	99.98769900
M_{17}	99.99227200
M_{19}	99.99980970
M_{31}	99.99999995

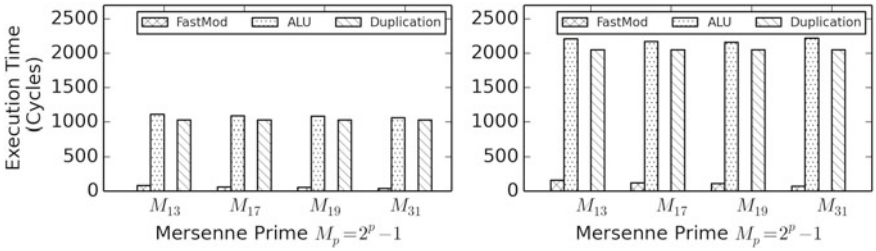


Fig. 2 Execution delay overhead in cycles for our fast reduction unit and fault tolerant ALU, compared to a modular multiplier with resource duplication, for argument sizes of 1024 bits (left) and 2048 bits (right) and different Mersenne primes

7.2 Area and Delay Overhead Results

In typical homomorphic aggregation scenarios, the fast Mersenne reduction as well as the ALU and memory error detection need to be part of the deployed homomorphic aggregation system, as they are required to enable *concurrent error detection*. Since the overhead of memory error detection corresponds (by construction) to that of fast modular reductions (e.g., by time-multiplexing the same functional units), our primary focus is the ALU and the modular reduction unit. In order to assess the area and delay overhead of our methodology, we instantiated a fast reduction unit (Algorithm 1) and a fault-tolerant modular multiplier (Algorithm 2) in Verilog, and evaluated their performance. For comparison, we used a fault tolerant implementation of a traditional modular multiplication ALU that uses resource duplication as its protection strategy.[6] Our evaluation results for argument sizes of 1024 and 2048 bits (commonly used in homomorphic operations), as well as four different Mersenne primes, are presented in Figs. 2 and 3 for delay and area overheads respectively.

[6]Our modular multiplication ALUs need to support any *cyclic group*, not only binary extension fields.

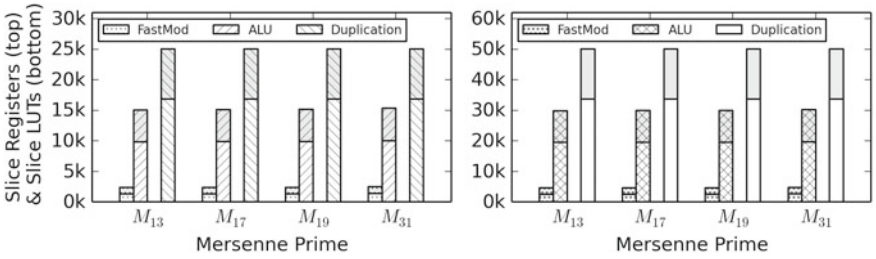


Fig. 3 Area overhead in terms of FPGA Slice LUTs (bottom bars) and Slice Registers (top stacked bars) of our fault tolerant ALU and fast modular reduction module, compared to a modular multiplier with resource duplication, for argument sizes of 1024 bits (left) and 2048 bits (right) and different Mersenne primes

7.2.1 Delay Overhead Comparison

In terms of delay overhead, our results show that the required number of clock cycles for fast Mersenne reduction is approximately equal to the bitsize ratio between the input argument and the used Mersenne prime. For example, the reduction of a 1024-bit input using M_{13} would take 82 cycles, while using M_{31} decreases this overhead to 37 cycles (trading longer syndromes for fewer execution cycles). The delay of the fault tolerant ALU equals the sum of the fast Mersenne reduction delay and the modular multiplication delay (which requires at least as many cycles as the argument bitsize). Thus, for M_{13} and argument size of 1024 bits, the fault tolerant ALU requires 1109 cycles, while for M_{31} one operation would require 1064 cycles. Compared to a duplicated modular multiplier, which requires 1027 cycles for 1024-bit arguments (assuming that the redundant modules work in parallel), the *additional* delay overhead of our fault tolerant ALU is between 3.6 and 8% for different Mersenne primes.

This additional overhead is generally attributed to the non-parallelizable steps of Algorithm 2, where the syndrome calculation in Step 3 must follow the modular multiplication that generates the ALU result (Step 2). Our results for fast Mersenne reductions, however, show that these reductions can be performed about 12–28 times faster compared to a standard modular multiplication. Therefore, in practice, this allows us to mask the delay overhead of many such (time-multiplexed) reductions, by executing them in parallel with modular multiplications.

7.2.2 Area Overhead Comparison

In terms of area overhead, our synthesis results are presented in Fig. 3. The results indicate that our implementation of a fault tolerant ALU incurs an additional area overhead between 20 and 23% for 1024-bit values, or between 19 and 21% for 2048-bit arguments. The latter is primarily attributed to the additional fast modular reduction and syndrome multiplication modules, and it is about 5× smaller compared to the 100% additional area overhead in case of resource duplication.

To achieve error detection for the encrypted memory, additional metadata of size equal to the used Mersenne prime bitsize should be stored along with each encrypted memory value. Since the stored encrypted value sizes are in the order of thousands of bits (e.g., 1024 or 2048), adding metadata in the order of a couple dozen bits makes our approach a proper alternative to ECC, especially considering that our lightweight methodology can detect an arbitrary number of errors with high probability (bounded by $1 - {M_p}^{-1}$, where M_p is the used Mersenne prime). Furthermore, it should be noted that ECC implements error detection within the memory module ICs, while our approach uses resources of the homomorphic ALU; due to the aforementioned delay overhead masking by the primary modular multiplications, and contrary to ECC solutions, our approach does not impact memory performance at all. Additionally, considering that ECC typically uses 8 parity bits for every 64 data bits [50], our methodology (which requires between 13 and 31 bits in total for 1024-bit and 2048-bit values) is more efficient in terms of extra storage requirements.

8 Related Work

In the area of reliable computation, several error detection methods have been proposed in the past. Traditional approaches, such as resource duplication, M out-of N majority vote (e.g., triple modular redundancy) or time redundancy, can be effective, but incur undesirable area or delay overheads. In addition, error codes, such as Berger, Bose-Lin, BCH, Reed-Muller, Hamming, or Cyclic Redundancy may offer robust detection of random memory errors [20, 29], but are incompatible with homomorphic aggregation by their construction: they either cannot support additive homomorphic operations like modular multiplication, or can only be ported to multiplication over specific fields like *binary extension fields* (e.g. [7]), which is not generally the case in homomorphic aggregation.

Moreover, random memory errors can be mitigated using ECC memory modules, where protections are implemented within memory ICs. Still, considering that such memories typically use 8 parity bits for every 64 data bits [50], storage efficiency can be lower, compared to the practical syndrome sizes in our approach. In addition, ordinary *modular checksums* (e.g. [47]) would require keeping track of the non-modular representations and the number of reductions, but such intermediate values are not generated as part of the homomorphic aggregation, since they could leak side channel information about the protected secrets.

With respect to efficient modulo operations, the authors of [6] propose an improved hardware implementation of modular reductions that requires $O(n - m)$ steps, where n is the argument size and m is the modulus size in bits. Their approach is more attractive when the modulus size in very big, while in our methodology we purposefully select relatively small Mersenne primes to reduce the storage requirements for syndromes. Furthermore, the authors of [48] discuss improved reduction algorithms for arbitrary moduli, but their approach uses lookup tables that require continuous memory accesses and may leak side channel information in addition to affecting

performance. In the presented approach, modular reductions are optimized, since we do not require arbitrary moduli (except in the primary homomorphic aggregation ALU).

9 Concluding Remarks

In this chapter, we present a lightweight fault tolerance methodology intended for error detection in homomorphic aggregation applications that are based on additive homomorphic encryption. Our approach is based on residue numbering extensions, modular arithmetic, as well as the intrinsic properties of homomorphic encryption algorithms. In modular multiplication ALUs, our methodology can achieve at least 99.98% fault coverage for only 3.6–8% additional runtime overhead, while our additional area overhead is about 5× smaller, compared to standard fault tolerance solutions such as resource duplication. Furthermore, in terms of memory error detection (which comes at no additional cost), the presented methodology can achieve 100% fault coverage for single bit-flips and up to four clustered faults in encrypted memories, providing a suitable alternative to generic VLSI fault tolerance methods (such as ECC and resource redundancy) ported to homomorphic aggregation applications.

References

1. Bajard, J.-C., Didier, L.-S., Kornerup, P.: An RNS montgomery modular multiplication algorithm. IEEE Trans. Comput. **47**(7), 766–776 (1998)
2. Barron, C., Yu, H., Zhan, J.: Cloud computing security case studies and research. In: World Congress on Engineering, pp. 1287–1291 (2013)
3. Baumann, R.: Soft errors in advanced computer systems. IEEE Des. Test Comput. **22**(3), 258–266 (2005)
4. Becker, G.T., Regazzoni, F., Paar, C., Burleson, W.P.: Stealthy dopant-level hardware Trojans. In: Cryptographic Hardware and Embedded Systems Workshop, pp. 197–214 (2013)
5. Boneh, D., Shacham, H.: Fast variants of RSA. CryptoBytes **5**(1), 1–9 (2002)
6. Butler, J.T., Sasao, T.: Fast hardware computation of x mod z. In: Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), pp. 294–297. IEEE (2011)
7. Chiou, C.-W., Lee, C.-Y., Deng, A.-W., Lin, J.-M.: Concurrent error detection in montgomery multiplication over $GF(2^m)$. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **89**(2), 566–574 (2006)
8. Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R., Molina, J.: Controlling data in the cloud: outsourcing computation without outsourcing control. In: Cloud Computing Security Workshop, pp. 85–90. ACM (2009)
9. Chung, K.-M., Kalai, Y., Vadhan, S.: Improved delegation of computation using fully homomorphic encryption. In: Advances in Cryptology—CRYPTO 2010, pp. 483–501. Springer (2010)
10. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Advances in Cryptology—EUROCRYPT'97, pp. 103–118. Springer (1997)

11. Damgård, I., Jurik, M., Nielsen, J.B.: A generalization of Paillier's public-key system with applications to electronic voting. Int. J. Inf. Secur. **9**(6), 371–385 (2010)
12. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory **31**(4), 469–472 (1985)
13. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Archive **2012**, 144 (2012)
14. Fontaine, C., Galand, F.: A survey of homomorphic encryption for nonspecialists. EURASIP J. Inf. Secur. **2007**(1), 26–35 (2007)
15. Fouque, P.-A., Poupard, G., Stern, J.: Sharing decryption in the context of voting or lotteries. In: International Conference on Financial Cryptography, pp. 90–104. Springer (2000)
16. Garner, H.L.: The residue number system. IRE Trans. Electron. Comput. EC-8(2):140–147 (1959)
17. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. thesis, Stanford University (2009)
18. Giry, D., Bulens, P.: Cryptographic Key Length Recommendation [Online] (2017). https://www.keylength.com/
19. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984)
20. Gorshe, S.S.: Concurrent Error Detection. Ph.D. thesis, Oregon State University (2002)
21. Hardy, G.H., Wright, E.M.: An Introduction to the Theory of Numbers. Oxford University Press (1979)
22. Hodson, H.: Apple vs FBI: first salvo in the information war. New Sci. **229**(3062), 24–25 (2016)
23. Jenkins, W.: The design of error checkers for self-checking residue number arithmetic. IEEE Trans. Comput. **100**(4), 388–396 (1983)
24. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. CRC Press (2008)
25. Kuribayashi, M., Tanaka, H.: Fingerprinting protocol for images based on additive homomorphic property. IEEE Trans. Image Process. **14**(12), 2129–2139 (2005)
26. Lyon, D.: Surveillance, Snowden, and big data: capacities, consequences, critique. Big Data Soc. **1**(2), 1–13 (2014)
27. Maniatakos, M., Michael, M.K., Makris, Y.: Multiple-bit upset protection in microprocessor memory arrays using vulnerability-based parity optimization and interleaving. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **23**(11), 2447–2460 (2015)
28. Mclvor, C., McLoone, M., McCanny, J.V.: Fast Montgomery modular multiplication and RSA cryptographic processor architectures. In: Asilomar Conference on Signals, Systems and Computers, vol. 1, pp. 379–384. IEEE (2003)
29. Mitra, S., McCluskey, E.J.: Which concurrent error detection scheme to choose? In: International Test Conference (ITC), pp. 985–994. IEEE (2000)
30. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology—EUROCRYPT'99, pp. 223–238. Springer (1999)
31. Paillier, P., Pointcheval, D.: Efficient public-key cryptosystems provably secure against active adversaries. In: Advances in Cryptology—ASIACRYPT'99, pp. 165–179. Springer (1999)
32. Prabhakaran, M., Rosulek, M.: Homomorphic encryption with CCA security. In: Automata, Languages and Programming, pp. 667–678. Springer, Berlin, Heidelberg (2008)
33. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Computer and Communications Security (CCS), pp. 199–212. ACM (2009)
34. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
35. Robinson, R.M.: Mersenne and fermat numbers. Proc. Am. Math. Soc. **5**(5), 842–846 (1954)
36. Salim, F., Haque, U.: Urban computing in the wild: a survey on large scale participation and citizen engagement with ubiquitous computing, cyber physical systems, and internet of things. Int. J. Hum. Comput. Stud. **81**, 31–48 (2015)
37. Shannon, C.E.: Communication theory of secrecy systems. Bell Syst. Tech. J. **28**(4), 656–715 (1949)

38. Smart, N., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. IACR Cryptol. ePrint Archive **2009**, 571 (2009)
39. Stehlé, D., Steinfeld, R.: Faster fully homomorphic encryption. In: Advances in Cryptology—ASIACRYPT, pp. 377–394. Springer (2010)
40. Tsoutsos, N.G., Konstantinou, C., Maniatakos, M.: Advanced techniques for designing stealthy hardware trojans. In: Design Automation Conference (DAC), pp. 1–4 (2014)
41. Tsoutsos, N.G., Maniatakos, M.: Fabrication attacks: zero-overhead malicious modifications enabling modern microprocessor privilege escalation. IEEE Trans. Emerg. Top. Comput. **2**(1), 81–93 (2014)
42. Tsoutsos, N.G., Maniatakos, M.: HEROIC: Homomorphically EncRypted One Instruction Computer. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1–6 (2014)
43. Tsoutsos, N.G., Maniatakos, M.: Trust no one: thwarting "heartbleed" attacks using privacy-preserving computation. In: Symposium on VLSI (ISVLSI), pp. 59–64. IEEE (2014)
44. Tsoutsos, N.G., Maniatakos, M.: The HEROIC framework: encrypted computation without shared keys. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **34**(6), 875–888 (2015)
45. Tsoutsos, N.G., Maniatakos, M.: Cryptographic vote-stealing attacks against a partially homomorphic e-voting architecture. In: International Conference on Computer Design (ICCD), pp. 157–160. IEEE (2016)
46. Van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Advances in Cryptology—EUROCRYPT, pp. 24–43. Springer (2010)
47. Walter, C.D.: Data integrity in hardware for modular arithmetic. In: Cryptographic Hardware and Embedded Systems (CHES), pp. 204–215. Springer (2000)
48. Will, M.A., Ko, R.K.L.: Computing mod without mod. IACR Cryptol. ePrint Archive **2014**, 755 (2014)
49. Yang, K., Hicks, M., Dong, Q., Austin, T., Sylvester, D.: A2: analog malicious hardware. In: IEEE Symposium on Security and Privacy (S&P). IEEE (2016)
50. Yoon, D.H., Erez, M.: Memory mapped ECC: low-cost error protection for last level caches. In: International Symposium on Computer Architecture (ISCA). ACM (2009)

An Approach to Integrating Security and Fault Tolerance Mechanisms into the Military IoT

Zbigniew Zieliski, Jan Chudzikiewicz and Janusz Furtak

Abstract Security and dependability are the most crucial challenges for the IoT implementation into the military domain. Insufficiently secured military IoT can provide adversary possibility of manipulation or disruption of data transmitted between units or even taking control of or disabling automated systems. On the other hand, if we already apply certain security mechanisms, IoT network should work reliably, even with faults arising out of unreliable hardware, and harsh or even hostile physical environment. In the paper it was pointed out that security mechanisms and fault-tolerant techniques to be effective in military applications should be tightly integrated. An approach for integrating security techniques on the access layer and the fault-tolerant techniques which are based on faulty (or misbehaving) sensor nodes diagnosis and reconfiguration was proposed. Presented solutions for securing the military IoT network ensure strong nodes authentication within network clusters and securing data transmissions between sensor nodes (SN) and gateways with the use of COTS IoT platforms equipped with TPM modules. Fault diagnosis (or detection) is based on the comparison method within network clusters. The method for determining effective diagnosable structures within clusters was presented. An experimentally constructed network called SFTN was build to demonstrate the proposed approach. Also, some results from a study of the experimental network in Cooja simulator were conducted.

Keywords Fault-tolerant system · Fault-diagnosis · Internet of Things · Security

This work is an extended version of the paper *Integrating Some Security and Fault Tolerant Techniques for Military Applications of Internet of Things* [1].

Z. Zieliski (✉) · J. Chudzikiewicz · J. Furtak
Faculty of Cybernetics, Military University of Technology, ul. W. Urbanowicza 2, 00-908 Warszawa, Poland
e-mail: zbigniew.zieliski@wat.edu.pl
URL: http://www.wat.edu.pl/

J. Chudzikiewicz
e-mail: jan.chudzikiewicz@wat.edu.pl

J. Furtak
e-mail: janusz.furtak@wat.edu.pl

R. S. Chakraborty et al. (eds.), *Security and Fault Tolerance in Internet of Things*, Internet of Things, https://doi.org/10.1007/978-3-030-02807-7_6

1 Introduction

The Internet of Things (IoT) is rapidly growing a world-wide technology, mainly focused on civilian applications. These include smart city initiatives, wearable devices for near real-time health monitoring, smart homes and buildings, smarter vehicles, etc. Moreover, there is a possibility that application of IoT in the military domain will bring significant benefits and change the picture of future military operations.

The adaptation of the IoT concept in the military domain (Military Internet of Things—MIoT) emphasizes on connectivity of military things (e.g.: military vehicles, weapons, materials, sensors, soldier devices, etc.) without any human participation [2–4]. The MIoT is most frequently used in safety critical real-time applications. MIoT systems have to deal with faults arising out of unreliable hardware, limited energy, connectivity interruption, and harsh or even hostile physical environment e.g. radiation. These findings highlight the need to consider a number of factors including security, and fault tolerance techniques while designing the MIoT application, in order to achieve MIoT robustness against attacks and resilience against faults typically considered in the area of fault-tolerant system design. For military domain, the system's capability to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents is the most important.

The main issues in MIoT systems are security and dependability. Security has been described via attributes of confidentiality, integrity, and availability. Dependability is typically understand as a systems ability to avoid service failures that are more frequent and more severe than is acceptable. Integrating dependability and security is a challenging and ongoing effort [5]. In the work [6] the schema and taxonomy of dependable and secure computing was proposed. This scheme shows that the main threats for dependability and security are faults, errors, and failures. One of the basic ways to improve dependability and security is fault tolerance. This directly lead us to the fact that in the case of military systems, the notions of security and fault tolerance should be treated in an integrated manner. The expected number of device failures will grow linearly with the population of MIoT. Highly constrained devices will be the most vulnerable, and malicious entities will seek to control at least some devices either directly or indirectly. In this context, fault tolerance is indispensable to assure service reliability. Also some security-enhancing services such as remote node validation in MIoT should work in reliably manner. Therefore, to improve security, it will be required to ensure fault-tolerance of nodes involved in strong validation.

The concern is that insufficiently secured MIoT can provide adversary possibility of manipulation or disruption of data transmitted between units or even taking control of or disabling automated systems. Security within IoT has been extensively investigated and plenty of solutions have been proposed as for instance [7–10]. These works concern security in various areas of applications of the IoT as smart-home [7], smart-cities [8] etc. To provide end to end secure communication some solutions were proposed, one of them based on hardware implementation of the IPSec, which was elaborated in [9]. Another important problem is to secure the range of devices in

the MIoT. MIoT devices have serious security implications as they may contain sensitive or valuable data making them attractive to various forms of adversary attack. The work [11] introduced the Object Level Protection concept for Military IoT which deals with securing data at the application layer, but does not concern the threats that may occur in the data link layer. To address this issues the implementation of security functionalities at the link layer was conducted as in [10] for IEEE 802.15.4-compliant IoT devices. We solved this problem with the use of asymmetric cryptography of TPM (Trusted Platform Module)[1] to support the processes of securing transmitted data and storage of sensors data resources [12].

Fault tolerance in distributed systems (also in Wireless Sensor Networks—WSN) have been the subject of active research in the past few decades. Recently, fault tolerance and management in WSNs have drawn researches' attention [13–15]. Some approaches, energy efficient detection schemes and algorithms for failure detection, faulty sensor identification have been proposed [14]. The mutual testing method [16] at the processor level was adopted among them, in which each processing element (sensor node) is capable of testing its neighbors [15, 17].

The main goal of the work is to integrate some security solutions for the WSN link layer with the use of pertinent commercial off the shelf (COTS) technologies and fault-tolerant techniques. The basis of the approach is a method of fault detection with applying comparison mutual nodes testing for diagnosis of misbehaving nodes within MIoT network clusters and cryptographic mechanisms for strong objects authentication and the data transmission protection. Cryptographic mechanisms are provided by the TPM modules installed in distinguished cluster nodes. The experimental setup, aimed to enable investigation of the key research problems for implementing effective solutions into the military domain was developed.

The main goal of the paper is to integrate security solutions for the WSN link layer with the use of pertinent commercial off the shelf (COTS) technologies and fault-tolerant techniques. The basis of the approach is the use of cryptographic mechanisms provided by the TPM module for strong object authentication and protection of the data transmission between sensor nodes. The experimental setup, aimed to enable investigation of the key research problems for implementing effective solutions into the military domain was developed.

The rest of the paper is organized as follows. In Sect. 2 overall structure of SFTN (Secure and Fault-Tolerant Network) for MIoT was described, and the basic terms and certain properties of the logical network structure were explained. The general approach to fault-tolerant schemes for the MIoT and fault detection method based on the comparison tests and determining so called simplified comparison graph was presented in Sect. 3. In Sect. 4 the security solutions for strong authentication and securing data transmissions on the network layer were proposed. In Sect. 6 selected components of the network implementation in the Cooja environment are presented. The simulation results of SFTN in the Cooja were presented in Sect. 6. Some solutions

[1] TPM is an implementation of a standard developed by the Trusted Computing Group [18]. This module is designed to support the cryptographic procedures and protocols that can be used for securing data [19].

proposed for the experimentally constructed network called SFTN were conducted in Sect. 7. In Sect. 8 some concluding remarks are given.

2 Preliminaries

The architecture of the considered MIoT network is based on a layered model [4]. In this model, the following layers can be distinguished: sensor layer, access layer, network layer, service layer, and application layer. The basis of the experimental MIoT network (called SFTN—Secure and Fault-Tolerant Network) is the multi-layer architecture of the MIoT network.

It was assumed that SFTN is part of MIoT. In the SFTN network, objects cooperate with each other. Each of these objects (called sensor node—SN) is equipped with many sensors (S) and a device for wireless data transmission. An example of a sensor node (SN) can be a wireless communication module including microcontroller-processing the data that is at the disposal of a soldier, but a role of the sensors (S) can be played by wearable sensors installed on a soldier's uniform. Additionally, it was assumed that MIoT is divided into disjoint groups of objects (clusters). Each cluster will include a limited number of objects and will represent the organizational structure and communication capabilities between objects. The structure of such a SFTN network is shown in Fig. 1.

The actions performed at lower layers may be divided into two parts: data acquisition and short-range wireless transmission. Data acquisition part collects various information through the use of RFID, sensors and other technologies.

Each cluster object is able to perform similar actions. Cluster objects are mobile, and wireless connections are used to exchange data. Each cluster object must also perform the following functions:

- acquire and pre-process the data from sensors in which the sensor node is equipped;
- authentication of objects in the cluster;
- securing data transmission between the nodes sensor and between the clusters;
- performing diagnostic procedures and cluster reconfiguration procedures.

Each cluster object contains elements of the sensor layer and the access layer.

In the sensor layer there are sensors (S) which transmit the read data to the sensor node (SN). The sensor nodes pre-process the received data. This processing may include: aggregating, averaging, creating the data structures or other processing needed to transfer the data to another sensor node or to another cluster. Data transmission between the SN and the S is bidirectional. It is assumed that sensors (S) do not directly send data between each other.

SN work on the access layer. Data transmission between the SNs is secured using cryptographic methods. The SN are able to perform the same functions, in particular, can play the Master role in the security domain and/or can serve as a Gateway in the domain of transmitting sensory data. Trusted Platform Module (TPM) is an obligatory element of each SN.

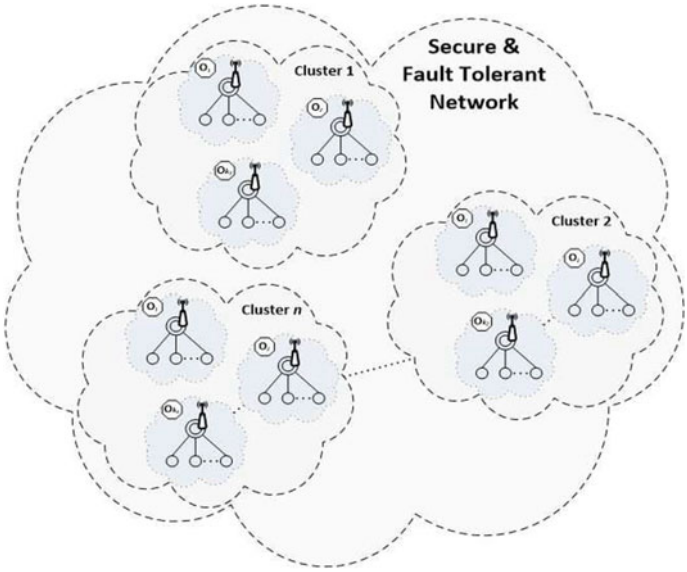


Fig. 1 The structure of Secure and Fault-Tolerant Network (SFTN)

In each cluster exist two interpenetrating domains. The first domain is called a security domain. In the domain the nodes exchange data used to authenticate the nodes. In this domain are also implemented the diagnostic procedures. The second domain is the domain of sensor data transmission from each SNs to another one, which plays a Gateway role in the domain. The Gateway node is also responsible for securing of the data exchange to other clusters.

Definition 1 The logical structure of sensors network is described by coherent ordinary graph $G = \langle E, U \rangle$ (E—a set of sensors, U—a set of bidirectional direct wireless link—data transmission channels).

Denote by $d\left(e,\ e'|G\right)$ the distance from node e to e' in G; it is the length of the shortest chain connecting node e with the node e'.

Let $r\left(e|G\right) = max_{e' \in E(G)} d\left(\left(e,\ e'\right)|G\right)$ be the greatest distance from the node $e \in E\left(G\right)$ to another node of the set $E\left(G\right)$, and $r\left(G\right)$, and $D\left(G\right)$ (respectively) denote the radius and the diameter of a graph G i.e. $r\left(G\right) = min\left\{r\left(e|G\right) : e \in E\left(G\right)\right\}$ and $D\left(G\right) = max\left\{d\left(e',\ e''|G\right) : \left\{e',\ e''\right\} \subset E\left(G\right)\right\}$ If $r\left(e|G\right) = r\left(G\right)$ then the node e is called the central node of the network G.

Denote by $E^{(d)}\left(e|G\right) = \left\{e' \in E\left(G\right) : d\left(e, e'|G\right) = d\right\}$ for $d \in \{1, \ldots, D\left(G\right)\}$ and by

$$\varsigma\left(e|G\right) = \left(\varsigma_1\left(e|G\right), \ldots, \varsigma_r\left(e|G\right)\right) \text{ for } \varsigma_d\left(e|G\right) = \left|E^{(d)}\left(e|G\right)\right|. \tag{1}$$

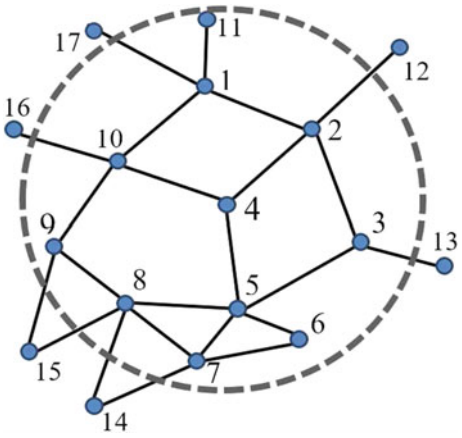


Fig. 2 An example of the cluster structure G (ECS)

Definition 2 [20] Let $\varphi(e|G) = \sum_{e' \in E(G)} d\left(e, e'|G\right)$ for $(e \in E(G))$ be the attainability of the node e in the network G.

Using (1) we have

$$\varphi(e|G) = \sum_{d=1}^{r(e|G)} d\,\varsigma_d(e|G). \tag{2}$$

Let us consider the exemplary network structure presented in the Fig. 2. We focus our analysis on the subnet of the network indicated by the dotted circle which represents the exemplary cluster structure G (ECS).

We can determined $\varsigma(e|G)$ using (1) and $\varphi(e|G)$ using (2) for the structure G.

The central cluster node can be determined on the base of minimal largest distance $r(e|G)$ of nodes or attainability of nodes. For the ECS (Fig. 2) the central node is node e_4, because $\varphi(e_4|G) = 18$ and it is the min. value for the structure G.

Definition 3 Let $T\langle E, U^*\rangle$ be the dendrite i.e. such coherent acyclic partial graph of G that:

$$\exists\left\langle e', e''\right\rangle \in U \Rightarrow \left\langle e', e''\right\rangle \in U^* \Leftrightarrow$$
$$\Leftrightarrow \left[\left(d\left(e_i, e'\right)\right) \neq d\left(e_i, e''\right) \wedge d\left(e', e''\right) \neq 1\right]$$
$$for\ r(e_i) = min_{e \in E(G)} r(e).$$

The dendrite T is a base to determine routing table and the diagnostic structure for G.

3 Faults Diagnosis Techniques Within SFTN

The purpose of fault-tolerant techniques (FTT) is to use available means to detect, identify, and isolate possible sensor faults, actuator faults, processing elements faults, communication links and system faults. In the design of a fault-tolerant system the fault model should be defined.

- ***The Fault Model***

At the sensor layer of the MIoT all nodes components can be divided into two groups. The first group consists of nodes with a processor/microcontroller, storage subsystem and power supply infrastructure. The second group can include sensors and actuators. It is known that all components from the first group are much more reliable that the ones from the second. That means they have much lower rate of failures [13]. However simultaneous occurrence of microprocessor's and sensor faults couldn't be excluded. Faulty nodes and its sensors should be diagnosed and isolated from the system.

At the access layer of the SFTN we assume communication links faults, communication nodes (base station, gateway faults, etc.). Further we present some extension of our approach proposed in the work [17].

- ***Sensor Level Fault Diagnosis***

Fault diagnosis is performed in two phases—diagnosis of processing nodes (microcontrollers) which reliability state of the processing elements is diagnosed and subsystem level phase where health of all the sensors/actuators is determined.

According to approach called MM method [16], the diagnosis is performed by sending the same input to pairs of nodes and comparing their responses. Based on the collective results of comparisons, one may be able to claim the faulty/fault-free status of the nodes in the SFTN network.

- ***System-Level Fault Diagnosis***

At the system level, diagnose of the communication nodes and the communication links has to be performed. In the work [17], at the system level, we proposed communication links faults and communication nodes faults diagnosis based on distributed agents (similar to SNMP agents).

- ***The Method of Fault Detection Within Network Clusters***

Diagnosis is a stage where the exact occurrence of a fault should be attributed to a specific node (SN_i or S_j). We assume the diagnosis of the SFTN is performed with a use of the MM* method and consists of conjunction of all possible results of comparison tests for the logical structure G of the network. Three sensor nodes take part in every comparison: a comparator, which orders neighboring sensor nodes e' and e'' the same task and checks whether the results of the task are identical and the sensor nodes $\{e', e''\}$—a comparison pair.

Let E^1 and E^0 denote, accordingly, the set of faulty as well as fault-free network sensor nodes, and $d(\psi, E^1)$—result of the comparison test ψ for the set of faulty sensor nodes E^1. We assume $d(\psi, E^1) = 0$ means that the difference between

obtained results from both compared sensor nodes not exceed given threshold ϑ_k and $d(\psi, E^1) = 1$—in the contrary.

Definition 4 [16] A sensor network is *t—diagnosable* if each sensor node can be correctly identified as fault-free or faulty based on a valid collection of comparison results, assuming that the number of faulty nodes does not exceed a given bound t.

According to the work [17] the sensor network with a logical structure G is t_d—*detectable* if for any fault situation there exist at least one comparison test ψ such that $d(\psi, E^1) = 1$ provided that the number of faulty nodes does not exceed t_d i.e. $\left|E^1\right| \leq t_d$.

The method of the fault detection in the SFTN is based on the network clustering and determining t_d—*detectable* structures within clusters.

In [17] we also introduced simplified comparison graph (SCG) as a graph $G(\langle G; E^{\bullet}\rangle)$, where the set of comparators $E^{\bullet}$ will be distinguished with bold nodes, a pair of antisymmetric arcs will be represented by the edge, and arcs leading to nodes which aren't comparators will be represented by dotted lines. The arc $< e', e'' >$, where $e', e'' \in E^{\bullet}$, denotes that the node e'' (a comparator) is an element of a comparable pair of comparator e'. The example of the SCG is presented in the Fig. 3b.

On the base of the properties of the SCG graph (proved in the work [17]) we modified the procedure for determining a diagnostic structure of the cluster G_i in the following way. The procedure consists of 3 main phases:

1. Designating a central cluster node on the base of minimal: $r(e \mid G_i)$ or $\varphi(e_i|G)$ of the nodes G (see Sect. 2);
2. For the selected cluster central node determine dendrite;
3. Based on a designated dendrite building a diagnostic structure (SCG).

- ***The Method of Determining Diagnostic Structure***

The method consists of three phases.

1. *Designating a central cluster node*

In the first stage, the node which meets one of the conditions: $r(e_i|G) = min_{e\in E(G)} r(e|G)$ or $\varphi(e_i|G) = min_{e\in E(G)} \varphi(e|G)$ is chosen as a *central cluster node*.

2. *Determine a dendrite*

In the second phase the dendrite T (satisfying the condition $d_{max}(e_i|T) = r(e_i)$) is determined. Based on the presented method, the algorithm for determining the central node and the dendrite was developed.

3. *Transformation the dendrite to the diagnostic structure*

The explanation of the transformation of the dendrite into a diagnostic structure will be presented in the example of the structure from Fig. 2. According to proposed approach node e_4 is appointed as the central cluster node. Next, the dendrite presented in the Fig. 3a) is determined. On the basis of the obtained dendrite the diagnostic structure (SCG) can be obtained. Transformation of the dendrite into the diagnostic structure is very simple and it involves the use of two simple rules:

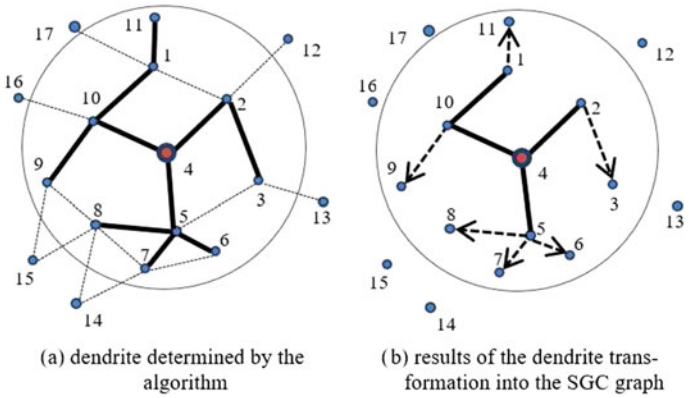


Fig. 3 Illustration of the procedure of diagnostic structure determining for ECS structure G from Fig. 2

1. Each node of the dendrite with a degree that equals at least 2 (the number of neighboring nodes) becomes a comparator;
2. The edges connecting nodes of degree one with the comparators should be replaced by dotted arcs directed to the comparable nodes.

From the properties of *t—diagnosability* [16] and properties of SCG we can state that obtained structure is 1—*diagnosable* and at least $(|E^{\bullet}| - 1)$—*detectable*, where $E^{\bullet}$ denotes the subset of comparators. For the presented example (Fig. 3b) the set of comparators $E^{\bullet} = \{1, 2, 4, 5, 10\}$ and the set of comparison test $\Psi = \{(1; 10, 11), (10; 9, 1), (10; 1, 4), (4; 2, 5), (4; 5, 10), (2; 3, 5), (5; 4, 6), (5; 7, 8)\}$.

4 The Security Solutions in the Network Layer

There is a security domain (SD) in each cluster. This domain includes all sensor nodes (SN) of the cluster. Security mechanisms are applied inside this domain. The most important element of the domain is one of the SNs, which plays the role of Master in the domain (M node). This SN is elected from among domain nodes during domain creation. The remaining nodes of the domain play the role of replicas of Master (R node). R nodes can take over the role of Master if diagnostic procedures detect a malfunction of M node.

The SN becomes a full member of the domain after the SN registration in the domain has been properly completed and then the registered SN plays the Replica role in the domain. During this procedure, credentials are established for each registered SN. These data are stored on one side in the resources of registered SN, and on the second hand, in the resources of the M node. For this purpose, to allow each R node to be able to take over the role of Master, after registration of the SN in the domain in the

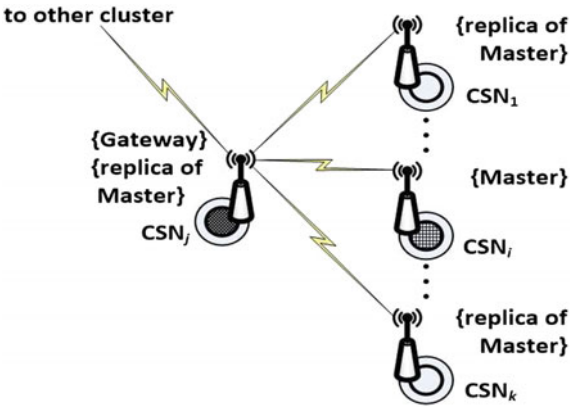


Fig. 4 Logical topology of cluster network (security domain)

resources of this node is created a copy of the resources of the security domain. While performing the above-described activities to protect data transmission between SN, asymmetric cryptography is used only in the first step, but symmetric cryptography is used in other steps.

The sink node for the domain is a node, which plays the Gateway role (*G* node) (Fig. 4). The transfer of sensor data from each SN to sink node is cryptographically protected using a common symmetric key for the domain. This encryption key is obtained from the M node during the registration procedure of each SN node in the security domain.

To avoid several types of routing attacks inside SFTN, we adopted a solution in which SN authentication in the domain is enforced before starting the creation of the routing table. This means that only authenticated nodes can exchange the route data.

Cryptographic security is applied to protect the sensitive data stored in resources of SN, and also to protect the exchange of data between SNs of the domain in the procedures of exchanging the authentication data, in diagnostic procedures and procedures for transmission of the sensor data. All cryptographic procedures performed in the domain are supported by the TPM module. The TPM module is an obligatory element of each SN.

5 Simulation of the SFTN

In the Cooja simulator, the main SFTN modules have been implemented (Fig. 5):

- *Routing*—creates a routing table for cluster nodes,
- *DiagS-Builder*—determines the diagnostic structure,
- *Self-Diagnostics*—performs a comparative test for a specific diagnostic structure and identifies defective nodes,

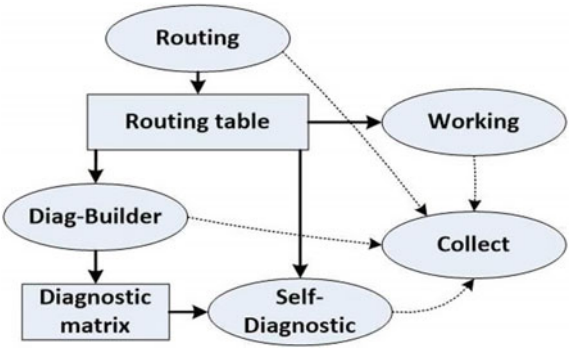


Fig. 5 Diagram of dependencies between SFTN modules implemented in Cooja

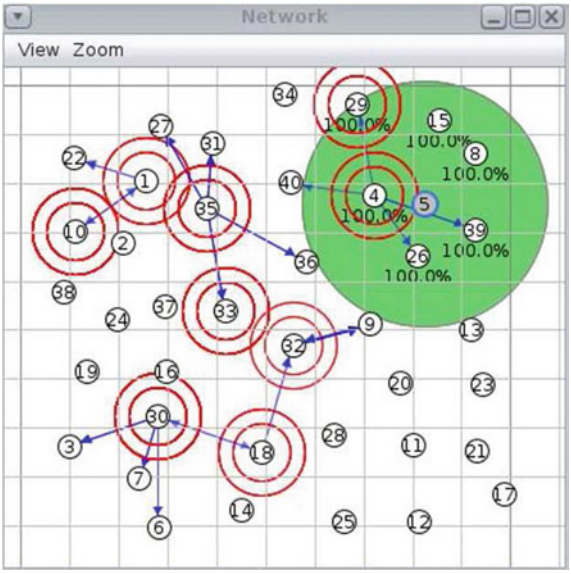


Fig. 6 An exemplary network implemented in the Cooja simulator

- *Working*—emulates the basic node functions such as: (data sensing, processing and transmitting to a gateway),
- *Collect*—collects information about events, statistics and sends them to the simulation module to illustrate in the *collect-view*.

The routing table contains, based on the number of hops as a metric route, routes to other nodes in the cluster. Based on this assumption, in the Cooja simulator, a software was developed to determine the routing table for the network shown in Fig. 6. The range of the transmitters of individual nodes is limited to 25 m.

The functions of radius determination and reachability for each node were implemented in Cooja simulator. The values of these parameters were determined for the example network, which took about 69 (simulation time). Then, according to

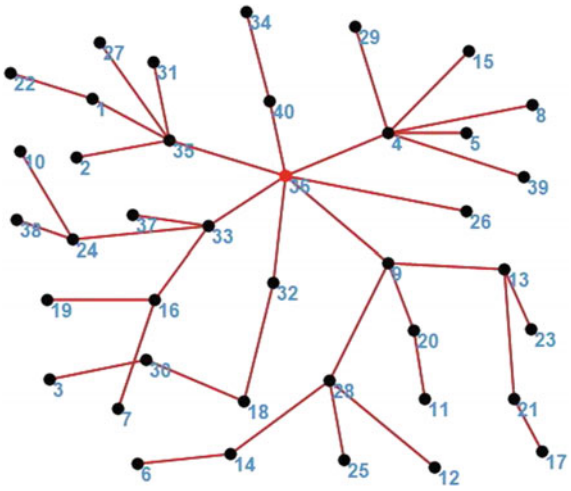


Fig. 7 An example of the dendrite for an investigated network

the adopted method, the central node, dendrite and a diagnostic structure for the exemplary network were determined.

Figure 7 shown the dendrite determined for the exemplary network, in which the central node is the node number 36.

6 Results of the SFTN Simulation

The efficiency of determining the diagnostic structure for the exemplary network structure was measured in the Cooja simulator. The following parameters were measured:

- Average power consumption across all nodes.
- The energy consumption in the central node.
- The average number of sent and received packets.
- The average time of determine the diagnostic structure.

Measurement was performed for a network structure that was degraded. The network structure degradation was carried out by removing five randomly selected nodes in every stages. Table 1 presented the order of removing nodes for each stages.

In the Cooja simulator, the rime protocol on the Skymote modules has been implemented. We are using Powertrace profile to estimate power consumption in every network nodes. Energy consumption was calculated on the basis of the following formula:

$$Power = \frac{Energest\_Value * Current * Voltage}{RTIMER\_SECOND * Runtime}$$

Table 1 The order of removing nodes from the SFTN for each stage

Number of nodes in network	Numbers of deleted nodes
40	–
35	4; 10; 21; 30; 34
30	3; 11; 15; 22; 24
25	6; 29; 33; 38; 39
20	7; 8; 17; 20; 27
15	5; 12; 19; 23; 31

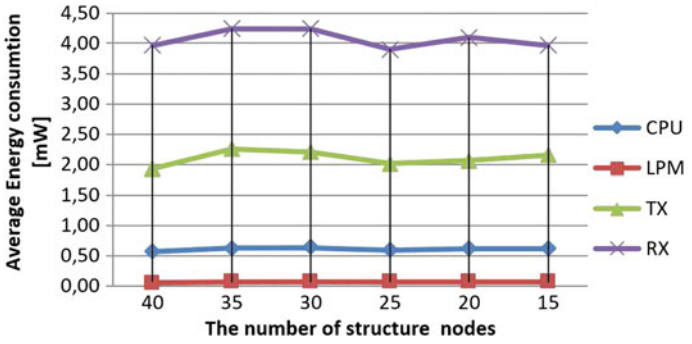


Fig. 8 The average energy consumption for the four operating modes

where:

Energest_Value—value determined by Powertrace in Cooja;

Current—energy consumption in the four highlighted modes of Skymote mote:

- CPU (active mode)—600 μA;
- LPM (Low Power Mode)—3 μA;
- TX (radio transmitter)—21 mA;
- RX (radio receiver)—23 mA.

Voltage—3, 6 V;

RTIMER_ SECOND—ticks per second for rtime (protocol implementation) and equal 32768;

Runtime—value is determined from powertrace_ start(CLOCK_ SECOND * 10) and equal 10.

In the Fig. 8 the average energy consumption in network nodes for different network structures were presented. As we can see, regardless of the degree of network degradation, the average energy consumption for Skymote modes remains at this similar level.

The Fig. 9 shows the energy consumption in the central node (node number 36) of the exemplary network subject to the degradation process. As we can see, for a structure with 25 nodes, there has been a decrease in energy consumption by the

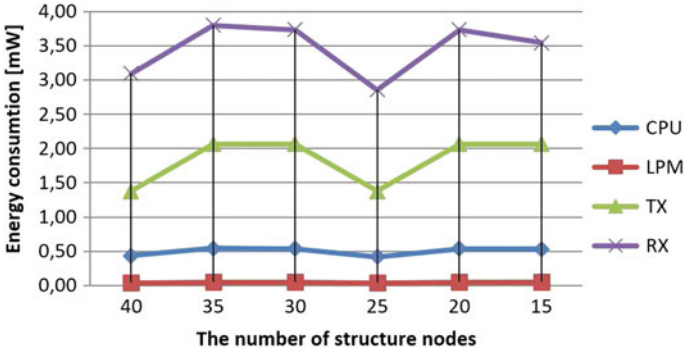


Fig. 9 The energy consumption in the central node (36) for the four operating modes

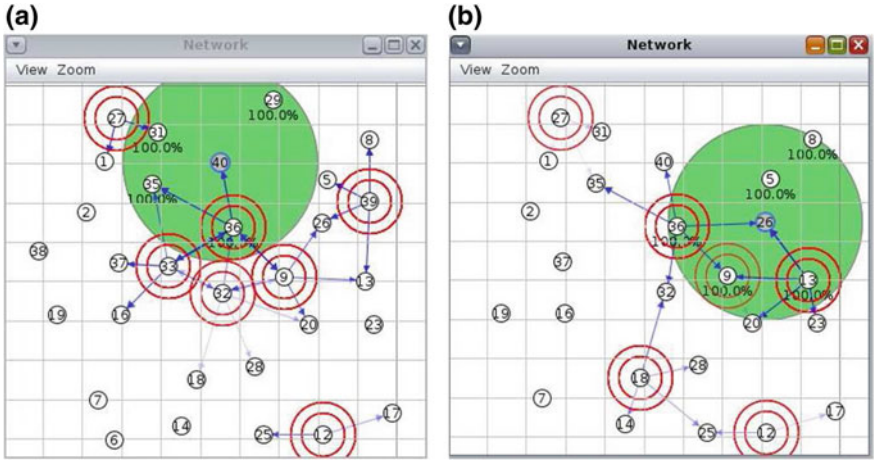


Fig. 10 The structures with the number of 30 nodes (**a**) and 25 nodes (**b**)

transmitter and receiver modules of the central node. A similar anomaly, albeit with a lower force, also occurs for the average energy consumption (Fig. 8). The cause of this anomaly is to remove from the structure with 30 nodes, the nodes 6; 29; 38; 39 (see Fig. 10b), which were extreme nodes in the network on the number of nodes 30 (see Fig. 10a).

In the Fig. 11 shown the average number of sent and received packets by the nodes of the exemplary network subject to the degradation process. As the network structure degrades, the number of packets received by each node decreases in a linear way, with a fixed number of packets sent. This is due to the reduction the number of nodes evenly.

The average time of determining the diagnostic structure is shown in Fig. 12. Determination of the diagnostic structure was the fastest for the structure with the

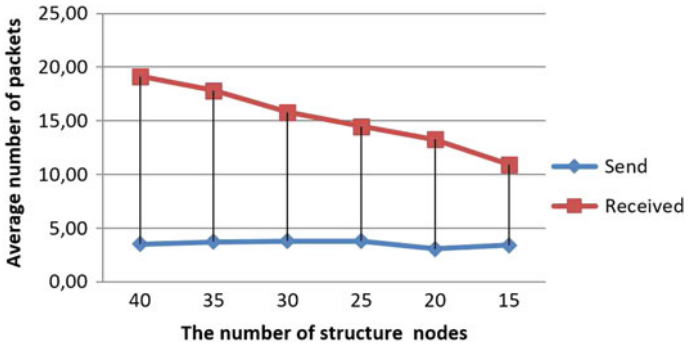
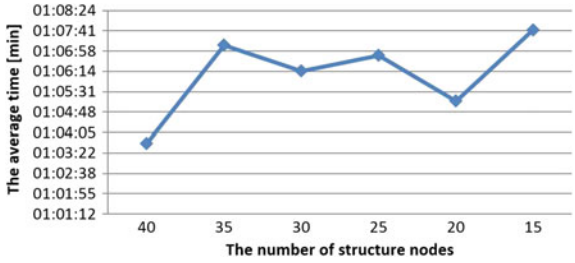


Fig. 11 The average number of sent and received packets

Fig. 12 The average time of determine the diagnostic structure



largest number of nodes, while the slowest process was in the structure with the lowest number of nodes. The reason for this is the decrease the number of neighboring nodes as the structure degrades, which forces multi-hop forwarding of messages.

7 An Experimental Model of Fault Tolerant Network for MIoT

Figure 13 shown the hardware implementation of the one cluster of the SFTN. The constructed model contains four elements (in the Fig. 13, marked as A, B, C and D), each of which consists of an SN node (indicated in Fig. 13 through 1) and three S nodes (marked in Fig. 13 through 2).

The model was constructed with the use of the following main components: Arduino Mega2560R3—a base module for build Collecting Sensor Node and Sensor Node; XBee 1mW Wire Antenna Series 1 or Series 2—wireless communication module; TPM—detachable part of hardware component of Atmel I2C/SPI Demonstration Kit; Temperature and Humidity Sensor DTH11.

In the SN node's software architecture shown in Fig. 14, the Secure Software Module (SSM) is used to secure the transmission carried out between SN nodes through the XBee.

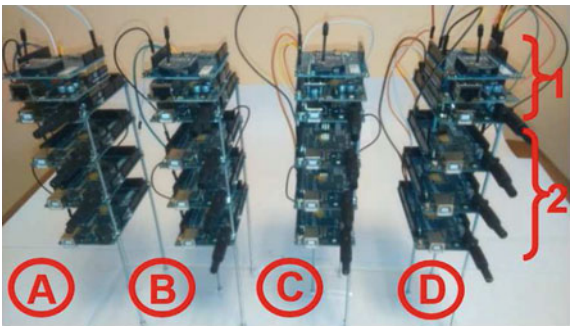


Fig. 13 Laboratory model of cluster of EFTNS

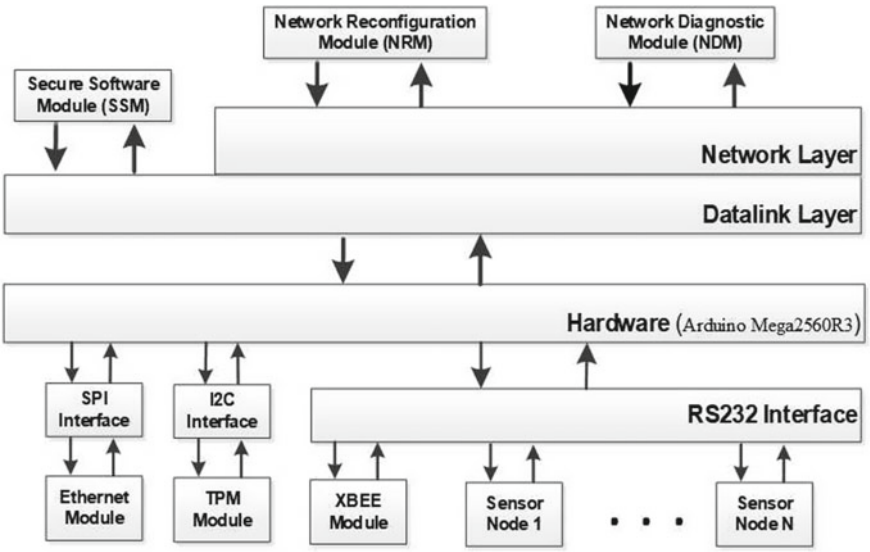


Fig. 14 The software architecture of Collecting Sensor Node

The tasks of this module include, among others, ensuring: authentication, integrity, and integrity of data, as well as authorization of nodes.

The process of auto-reconfiguration of the network after detecting errors in its structure is carried out by the Network Reconfiguration Module (NRM).

The Network Diagnostic Module (NDM) is used both as a communication interface for system level diagnostics and as a port for sensor level diagnostics.

Summary

Proposed solutions for securing the Military IoT network ensure strong nodes authentication within network clusters and securing data transmissions between Collecting Sensor Nodes (SN) and gateways with the use of COTS IoT platforms equipped with TPM modules.

Analyzing the obtained simulation results (Figs. 11 and 12), we can see that the number of nodes in the structure, as well as the number of incident nodes with each node, have a significant impact on the time of determining the diagnostic structure. Decreasing the number of structure nodes we decrease the number of packets received, while the number of packets sent remains at the same level.

Simulation studies confirmed that the solutions for network diagnostics based on the adoption of system-level comparison methods proposed in the paper have good efficiency with a little waste of network bandwidth.

References

1. Zieliński, Z., Chudzikiewicz, J., Furtak, J., Głebocki, P.: Integrating some security and fault tolerant techniques for military applications of Internet of Things. In: IEEE World Forum on Internet of Things, 12–14 Dec 2016, Reston, VA, USA (2016)
2. Zheng, D.E., Carter, W.A.: Leveraging the Internet of Things for a More Efficient and Effective Military, A Report of the CSIS Strategic Technologies Program (2015)
3. Suri, N., et al.: Analyzing the applicability of Internet of Things to the battlefield environment. In: Proceedings of ICMCIS Conference, Brussel (2016)
4. Yishi, L., Fei, J., Hui, Y.: Study on application modes of military Internet of Things (MIOT). In: Proceedings of 2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE), vol. 3, pp. 25–27 (2012)
5. Qian, Y., Joshi, J., Tipper, D., Krishnamurthy, P.: Information Assurance: Dependability and Security in Networked Systems, pp. 113–170. Morgan Kaufmann Publishers (2018)
6. Avizienis, A., Laprie, J.C., Randell, B., Landwhehr, C.: Basic concepts and taxonomy of dependable and secure comuting. IEEE Trans. Dependable Secur. Comput. **1**, 11–33 (2004)
7. Sivaraman, V., Gharakheili, H.H., Vishwanath, A., Boreli, R., Mehani, O.: Network-level security and privacy control for smart-home IoT devices. In: WiMob'15
8. Bohli, J.M., Skarmeta, A., Victoria Moreno, M., García, D., Langendörfer, P.: SMARTIE project: secure IoT data management for smart cities. In: International Conference on Recent Advances in Internet of Things (RIoT), Singapore, pp. 1–6 (2015)
9. Rao, M., Newe, T., Grout, I., Lewis, E., Mathur, A.: FPGA Based Reconfigurable IPSec AH Core Suitable for IoT Applications. In: CIT/IUCC/DASC/PICOM'15 (2015)
10. Altolini, D., Lakkundi, V., Bui, N., Tapparello, C., Rossi, M.: Low power link layer security for IoT: implementation and performance analysis. In: IWCMC'13 (2013)
11. Wrona, K.: Securing the Internet of Things a military perspective. In: Proceedings of 2nd IEEE World Forum on Internet of Things—WF-IoT'15 (2015)
12. Chudzikiewicz, J., Furtak, J., Zielinski, Z.: Secure protocol for wireless communication within Internet of Military Things. In: Proceedings of 2nd IEEE World Forum on Internet of Things—WF-IoT'16 (2016)
13. Koushanfar, F., Potkonjak, M., Sangiovanni-Vincentelli, M.A.: Fault-tolerance in sensor networks. In: Mahgoub, I., Ilyas, M. (eds.) Handbook of Sensor Networks, Section VIII, no. 36. CRC Press (2004)

14. Jiang, P.: A new method for node fault detection in wireless sensor networks. Sensors **9**, 1282–1294 (2012)
15. Chen, J., Kher, S., Somani, A.: Distributed fault detection of wireless sensor networks. In: DIWANS '06 Proceedings of the 2006 Workshop on Dependability Issues in Wireless Ad hoc Networks and Sensor Networks, pp. 65–72 (2006)
16. Maeng, J., Malek, M.: A comparison connection assignment for self-diagnosis of multiprocessor systems. Digest International Symposium on Fault Tolerant Computing, pp. 173–175 (1981)
17. Chudzikiewicz, J., Furtak, J., Zielinski, Z.: Fault-tolerant techniques for the Internet of Military Things. In: Proceedings of 2nd IEEE World Forum on Internet of Things—WF-IoT'16 (2016)
18. TPM Main Part 1 Design Principles. Specification Version 1.2. Revision 116, Trusted Computing Group, Incorporated (2011)
19. Kinney, S.: Trusted Platform Module Basics: Using TPM in Embedded Systems. Elsevier Inc, Embedded Technology Series (2006)
20. Chudzikiewicz, J., Zieliński, Z.: On some resources placement schemes in the 4-dimensional soft degradable hypercube processors network. In: Proceedings of DepCoSRELCOMEX Conference, Poland, pp. 133–143 (2014)

Fault-Tolerant Implementations of Physically Unclonable Functions on FPGA

Durga Prasad Sahoo, Arnab Bag, Sikhar Patranabis, Debdeep Mukhopadhyay and Rajat Subhra Chakraborty

Abstract Most of the faults in circuits or systems occur due to the unintentional but natural phenomenon (e.g. imperfection in manufacturing process or significant change in the working environment), and thus, these faults are often follow a pattern and comparatively easier detect than the intentional faults. In the context of secure design/system, the adversary (intentionally) injects some faults in the system to bypass the security protection or reveal secret information. Since the adversaries' fault injection objectives are often very subjective, it is difficult to find a pattern among the faults in a system, and this makes the fault detection and fault recovery difficult in a secure system. In this chapter, we discuss possible intentional faults in an emerging hardware security primitive, known as *Physically Unclonable Functions* (PUFs). We show how the faults vary over the PUF designs and its applications. In addition, we explain different fault detection circuits and fault recovery techniques which are specific to PUF designs and their implementations on FPGA platforms.

D. P. Sahoo
Bosch India (RBEI/ESY), Bengaluru, India
e-mail: dpsahoo.cs@gmail.com

A. Bag (✉) · S. Patranabis · D. Mukhopadhyay · R. S. Chakraborty
IIT Kharagpur, Kharagpur, India
e-mail: arnabbag@iitkgp.ac.in

S. Patranabis
e-mail: sikharpatranabis@gmail.com

D. Mukhopadhyay
e-mail: debdeep@cse.iitkgp.ernet.in

R. S. Chakraborty
e-mail: rschakraborty@cse.iitkgp.ernet.in

R. S. Chakraborty et al. (eds.), *Security and Fault Tolerance in Internet of Things*,
Internet of Things, https://doi.org/10.1007/978-3-030-02807-7_7

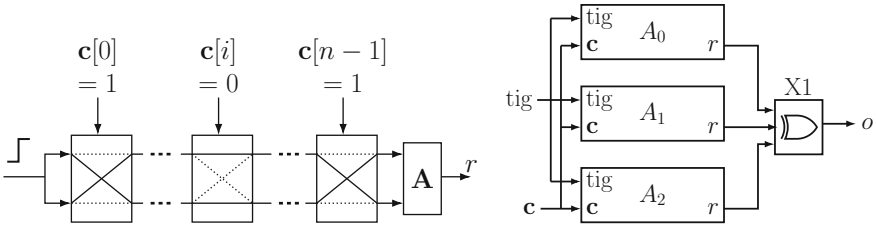
1 Introduction

Present day embedded devices with state-of-the-art security solutions for a vast number of applications, such as Internet of Things (IoT), have seen increased demand for reconfigurable and efficient hardware. Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs) are now widely used as reconfigurable hardware for designing cryptographic primitives. FPGA implementations for conventional cryptographic schemes such as secure storage and random number generators have been compromised by several side-channel based attacks. Therefore demand for alternative solutions has come up. *Physically Unclonable Functions* (PUFs) [18, 19] have emerged as pertinent solutions in this regard.

A silicon PUF is treated as Boolean mapping $\lambda : \{0,1\}^n \rightarrow \{0,1\}^m$, where the m-bit output (known as `response`) depends on the input (known as `challenge`) and extractable random manufacturing variations in the device. Note that variation in circuit behaviors (e.g. current, voltage, and delay profiles) are mostly analog in nature. A PUF circuit is used to extract device-specific variations and eventually, generates a binary response using efficient quantization technique. Arbiter PUF (APUF) [18] and Ring Oscillator PUF (ROPUF) [33] are two widely studied *Delay* PUF primitives, while SRAM PUF [14] is a popular candidate of *Memory* PUF.

By description, PUF is a physical device which provides an easy way to evaluate (by challenge response pair), but at the same time, it has an extremely unpredictable nature. Practically, it is almost impossible to clone a PUF physically, even with the detailed knowledge of the exact manufacturing process used for its production. The intrinsic random nature makes PUF an attractive alternative of conventional cryptographic primitives like random key generation and storage for secure keys. As it can be seen in case of traditional cryptography, where secure non-volatile memory elements were used to store secret keys, PUFs provide a method to generate the secret keys, instead of storing it. PUFs generate the random keys [20, 21] through a unique mechanism of challenge-response pairs (CRP) sequence which is extremely difficult to reproduce. Additionally, there are PUFs like delay-based PUFs [18, 33], which are extremely efficient when implemented on FPGA, this makes this PUF an ideal choice for the cryptographic applications like authentication and key generation. For other applications, XOR Arbiter PUF (APUF) is predominantly used for authentication [16, 24] purpose, and the Ring Oscillator PUF (ROPUF) family is popular for random number generation [21]. We provide a brief discussion of APUF and its design in the rest of the chapter. Readers who are unfamiliar with PUFs' functionality and PUF designs can have a basic understanding from it. To know about PUFs in more detail, readers may refer to [19, 30].

A block diagram of the classic APUF design is shown in Fig. 1a. The cascade of path-swapping switches is a core part of the APUF design. At the end of the



(a) APUF with n-bit challenge **c** (b) 3-XOR APUF

Fig. 1 Arbiter PUF design and its XOR composition. Here, A_0, A_1 and A_2 are three instances of APUF

cascade chain, there is an *arbiter*. The path-swapping switches are typically realized using a pair of 2:1 multiplexers. A voltage impulse is applied to the input node of the cascade. It propagates along two symmetrical paths constructed by the switches after application of control bits to the switches. The source of randomness in PUF, the individual path delays in each stage are randomly determined by effect of device level process variation of manufacturing process, thus making it unpredictable in nature. The accumulated delay differences of these individual stages primarily determine the output of the arbiter circuit at the output end. The arbiter is usually realized using an edge-triggered D Flip-Flop (DFF) or SR Latch. The control bit string, generally known as the *challenge* in literature, is applied to the cascade of switches (each bit of the string is applied to the corresponding switch in the cascade). The output, known as the *response*, is extracted from the output of the arbiter.

Besides the primitive PUF designs which are vulnerable to the machine learning based modeling attacks [28], there are many proposals for PUF compositions [31, 32] where primitive PUFs are combined to achieve a higher level of modeling resistance. Let us briefly introduce the XOR APUF design which is one of target PUF designs to be used for explaining the fault attacks. An x-XOR APUF consists of x APUF instances (a 3-XOR APUF is shown in Fig. 1b), and for a given challenge **c**, the 1-bit response is generated as the bitwise XOR of 1-bit responses of x APUF instances. It is shown in the literature [28] that the modeling resistance of x-XOR APUF increases with the increasing value of x.

PUFs have been predominantly subjected to several attacks based on machine learning methods where a mathematical model is created from partially observed challenge-response behaviour [27]. Recently, several other attack methodologies, especially physical attacks, have been reported in the literature [11–13, 29, 34, 35], employing techniques like power side-channel, laser-based fault injection, etc. for attack purpose. In this chapter, we confine the discuss to fault attacks on PUFs focusing on the following facts:

- Objectives of fault attacks and fault injection techniques for FPGA-based PUF implementations, and
- Fault detection and fault recovery techniques specific to the FPGA platform.

We start the discussion with different fault injection techniques and objectives of faults from attackers' perspective.

2 Fault Injection and Its Objectives

By definition, in a device/circuit, a fault is an aberration originated by manufacturing defects or post-manufacture interference from external sources. In cryptographic applications, fault injection can be done in a(n) invasive/semi-invasive way, and the goal is to extract some secret information related to the target cryptosystem such as the key. Fault can be injected into the device by various means—by introducing voltage spikes to the power supply, by introducing glitches to the clock supply, by applying high-intensity EM radiations, temperature variations, and even by applying short timed pulses of flash light or laser beam to the hardware (depackaged chip) [5, 15].

> The principle of fault attack is to induce intentional faults (unexpected behaviour) into cryptographic implementations, to reveal their internal secret states.

There are several fault attacks on PUFs reported in the recent literature [35], but there is a fundamental difference in fault attacks on PUFs. In traditional fault attacks on cryptosystems, the objective remains to recover any information related to the secret key or the secret key itself. In the case of PUF, fault attacks try to compromise its randomness, reliability and unclonability [22] properties mainly, reducing the strength of cryptographic primitives designed using it. We discuss few such attack methodologies below:

1. **Entropy reduction**. PUFs with responses having a low level of entropy are easy to reproduce, therefore impractical for secret key generation and random number generation for security applications. As described in [35], a back-side laser beam injection has been applied to forcefully convert the inverting stage to non-inverting stage of the RO, thus hindering the normal operation of the RO. As a consequence, this reduces the entropy of the output. Applications based on the ROPUF, for instance, key-generators or random number generators are susceptible to this type of attack.
2. **Accelerating ML based modeling**. Machine learning (ML) based attacks are already there in the literature for attacks on PUF. The problem is that ML-based attacks are either computationally expensive or infeasible. For instance, Ruhrmair [28], Tobisch [36] reported ML-based models of x-XOR PUF which become computationally infeasible for $x \geq 10$. However, authors of [35] reported a laser-based fault injection attack to tweak the structure of the XOR APUF which provided them with the access the CRPs of individual APUFs, allowing

them to model the entire XOR APUF. Authors of [7] have observed the correlation between the reliability of XOR APUF and the component APUFs. They exploited this reliability information of XOR APUF to model the individual APUFs. There are some fault attacks [7] which try to extract secret information from side channel leakage resulting from unreliable CRPs generated due to operating temperature variation. Other than these, semi-invasive attacks like the one described in [26], uses IC backside EM radiations to characterize RO PUFs.

3. **Denial of service**. PUFs like Loop PUF (LPUF) [10] and Bistable Ring PUF (BRPUF) [9] are mostly used for authentication purpose, and they mostly suffer from denial-of-service attacks. In a denial-of-service attack, an adversary tries to cause an authentication failure by injecting faults despite the verifier maintaining the database of the original fault-free PUF device. Similar attack methodology can be applied to LPUF also, where a laser beam shoot can invert the output of an inverter, stalling the oscillation of the oscillator loop. In the case of BRPUF instance, an adversary can modify an odd number structure to an even number of inverters, which makes the output unstable. If the faults are not permanent, the PUF device can be reverted back to its normal, fault-free state, but services depending upon the PUF operation will be disrupted for a certain time interval.

It is clear that, without suitable countermeasures, PUF based designs are deemed to be insecure against these threats. There are traditional countermeasures against these standard attack methodologies, but for PUFs, these traditional countermeasures pose significant challenges, especially the spatial and temporal redundancy [25] in the case of a design of fault-tolerant circuits involving PUFs. Two properties—randomness and unclonability, are the challenging ones, as described next:

- **Randomness**. A PUF instance can be uniquely characterized by a random set of CRPs. This implies that there is no particular or reference CRP that can be used for fault detection in PUF. It can not be guaranteed whether injecting a fault can improve or degrade the randomness of the PUF, making in extremely tough to identify any occurrence of a fault.
- **Physical Unclonability**. Infeasibility of physical cloning of PUFs implies that spatial redundancy can not be treated as a viable option for detecting faults. Temporal redundancy becomes ineffective for an adversary who introduces a *remnant fault* [15]. Once introduced, this fault, persists throughout the FPGA operation until is reconfigured, causing the circuit to generate same faulty output both in redundant and original phases.

Traditional spatial redundancy based fault-tolerant approaches are not applicable for PUF as the challenge-response behaviour of PUF instances are unique. Thus, the redundant copies of a PUF instance would have different behaviour.

In the next section, we discuss about various delay PUFs, especially about the design of novel fault-tolerant PUF circuits related to delay PUFs. These are considered robust against high precision laser pulse injection fault attacks [35].

3 Countering Fault Attacks on XOR APUF

To the best of our knowledge, the x-XOR PUFs instances are resistant against conventional machine learning attack. However, recently authors of [35] proposed two laser injections based fault attacks where they were able to convert an XOR PUF into a simple APUF design. An adversary can extend this attack by repeating x times to get responses of x individual APUF instances, and finally, develop a linear time model of an x-XOR APUF for any arbitrary x. Considering the exact location of the fault injections, these two attacks have been described in [35]:

- **Attack-I**. The Last switch (S_{n-1}) of APUF (cf. Fig. 4)—a component of XOR PUF is targeted for this attack. The adversary alters the logic of the S_{n-1} switch such that CLK input (cf. Fig. 4) of the arbiter always remains at '0', therefore producing a fixed value irrespective of the input challenge at the output of the arbiter. The adversary needs to modify all the PUFs except the one she wants to collect the CRPs. Refer to the example provided in Fig. 2 to get a better understanding of this attack.
- **Attack-II**. In this particular attack scenario, the final XOR circuit of the XOR APUF is targeted (cf. Fig. 3), typically implemented by LUT on FPGA. A laser pulse is injected to alter the contents of SRAM cells present in the target LUT to transform the XOR gate into a buffer for i-th input. Therefore, the response produced by i-th APUF will be available as XOR response. This is considered as a serious security issue as the basic assumption of the XOR APUF (that the internal inputs of XOR are secret) is violated in this case.

Both Attack-I and Attack-II have a common underlying philosophy, the adversary injects faults to alter the configuration memory of the programmable logic cell(s), and forcing to show different functional behaviour from its normal behaviour. We introduce two countermeasures against the attacks on the XOR PUF mentioned earlier. The basic idea is to incorporate an additional fault-checking circuit in the

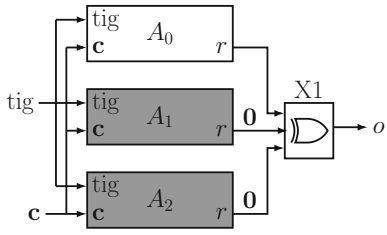
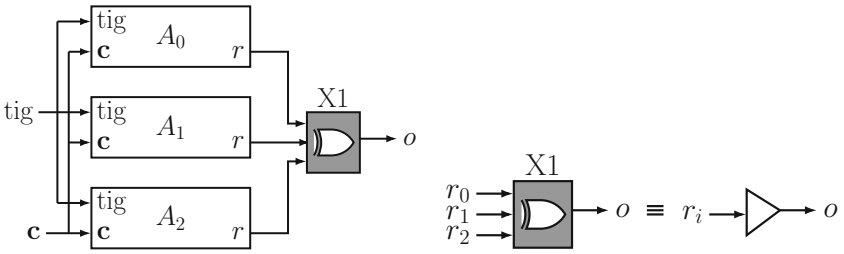


Fig. 2 Exemplification of fault Attack-I on 3-XOR APUF. Two highlighted APUF (gray color) blocks are faulty and producing constant '0' output regardless of applied challenge. The output of 3-XOR APUF is the same as response of APUF A_0



(a) Fault injected on XOR logic (b) XOR Logic behaves as buffer

Fig. 3 Demonstrating Attack-II on 3-XOR APUF

design of the XOR PUF to detect fault injection. The output of the XOR PUF is forced to 0 if a fault is detected; otherwise normal operation continues.

3.1 Countermeasure Design for Attack-I

Attack-I in [35] works by injecting a fault in the lower part of the PDL switch located at the S_{n-1}-th stage of an individual APUF instance, making the CLK of the switch to fixed at '0'. In other attack scenario, the D signal of the switch can be set at a fixed value of '0' or '1', forcing the arbiter to always sample the same value. We introduce an ingenious countermeasure to identify any modifications in the PUF circuit of these types. The strategy is to incorporate a 3-input fault detection logic (FDL) having an output T, as described in Fig. 4. A summary of the functional behaviour of the FDL circuit is provided in Table 1. We state the following theorem:

Theorem 1 *Each individual APUF circuit generates correct output if and only if $T = 1$.*

Proof Notice that, in the absence of any blockage (stuck-at-0 or stuck-at-1) on the path originating from S_0 and extending to S_{n-1} (implying all the intermediate switching circuits are properly functioning), the 'tig' signal (logic 0 or 1) propagates to the

Fig. 4 APUF with fault detection logic (FDL)

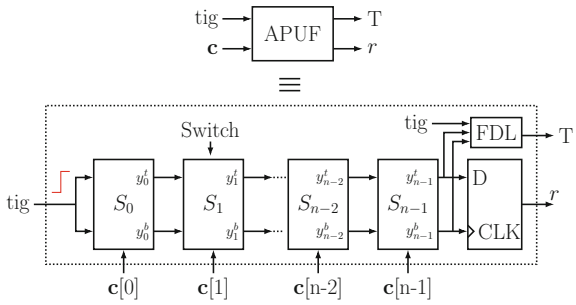


Table 1 Truth table of 3-input FDL in Fig. 4

Inputs			Output
tig	y^t_{n-1}	y^b_{n-1}	T
0	0	0	1
1	1	1	1
x	0	1	0
x	1	0	0

stage S_{n-1}, namely the outputs y^t_{n-1} and y^b_{n-1}. Otherwise, the 'tig' signal will not be able to propagate to the outputs y^t_{n-1} and y^b_{n-1}. Therefore, the circuit can function properly only if:

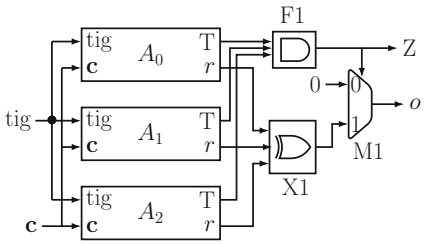
$$\text{tig} = 0,\ y^t_{n-1} = y^b_{n-1} = 0$$
$$\text{tig} = 1,\ y^t_{n-1} = y^b_{n-1} = 1$$

along with T = 0 for other conditions. It accurately conforms to the given truth table of 'T' provide in Table 1. This completes the proof of Theorem 1.

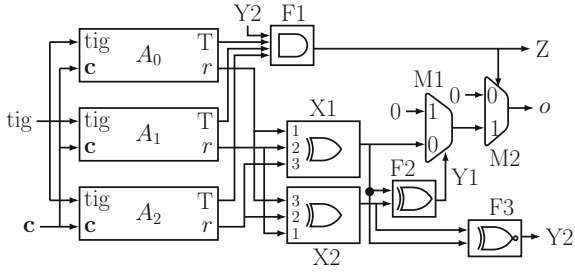
Ultimately, the complete XOR APUF combines the 'T' values output for each APUF instance, and a fault is detected if any of these instances produce output 'T = 0'. A representative illustration of the proposed countermeasure is provided in Figs. 1b and 5a depicting the 3-XOR APUF designs without FDL and with FDL, respectively. Attack-I can not be detected for the design in Fig. 1b, but it can be detected for the design in Fig. 5a, given that the AND gate (F1) used to combine 'T' outputs of APUFs has not been modified. The output o of XOR PUF will be '0' unless 'Z' is equal to '0', irrespective of the challenge applied. Otherwise, the XOR APUF produces correct output at the output o. After integrating the FDL circuit, it is comparatively harder to exploit the advantage of fault injection in XOR PUF.

3.2 Countermeasure Design for Attack-II

Fault Attack-II targets the XOR gate (X1) of XOR APUF [35], used to combine the outputs of the component APUFs (cf. Fig. 3). The corresponding LUT for the XOR function is altered to function as a buffer by performing a precise laser pulse injection. Our proposed countermeasure tries to detect the modification by incorporating a redundant XOR gate (X2) with *different input connection patterns*, as depicted in Fig. 5b. Observe that, an adversary needs to modify two LUTs of both the XOR gates to make those work as buffers for the same APUF instance to bypass this countermeasure. This is extremely difficult due to their input connection patterns. Additionally, an extra XOR (F2) gate is used to ensure that X1 and X2 generate the same output, to prevent attacks on any of X1 or X2. If output Y1 of F2, the con-



(a) Countering Attack-I



(b) Countering Attack-I & II

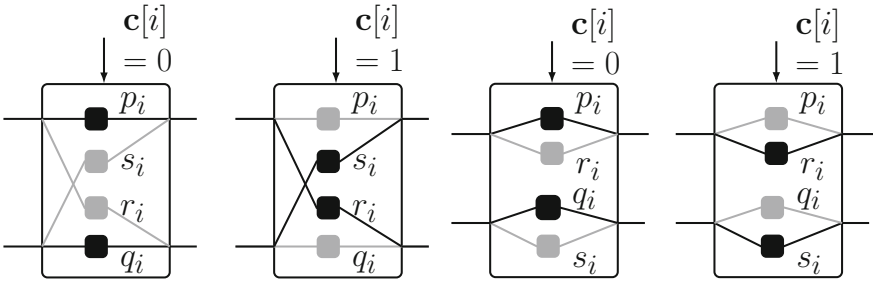
Fig. 5 3-XOR APUF with FDL to resist Attack-I and Attack-II

trol input of MUX M1, is '0', then the output of M1 is the output of XOR APUF, otherwise M1 always outputs '0' irrespective of the input challenge.

After all, the utility of the XNOR gate F3 in Fig. 5b can be described as follows. Notice that, if the F3 is absent, an adversary can alter X1 and F2 (X2 is not touched) in a way such that X1 behaves as a buffer and F2 is forced to output '0' always. Fortunately, F3 can detect this attack, as it is the complement gate of F2 and output of F3 is connected to the input of the fault detection logic F1. An adversary can attack the modified circuit only by altering both F2 and F3, which is difficult achieve due to their dissimilar functionalities. Redundant gates may be added to the PUF, multiplexed accordingly to produce the final output signal 'Z', this reinforces the fault tolerance of the design.

3.3 A Self-testable APUF Switch Design

The fault detection logic for APUF presented in Fig. 4 monitors only the inputs of an arbiter circuit to detect the fault. We next introduce an alternative fault detection scheme which uses a properly modified design of APUF switch that allows to detect any inconsistency in the expected switching activity. Figure 7a and 7b show the the LUT based construction of a PDL switch (cf. Fig. 6b) and classic switch (cf. Fig. 6a) of APUF, respectively. For clear discussion, we primarily focus on the necessary



(a) Classic APUF's i-th stage (b) PDL-based APUF's switching stage

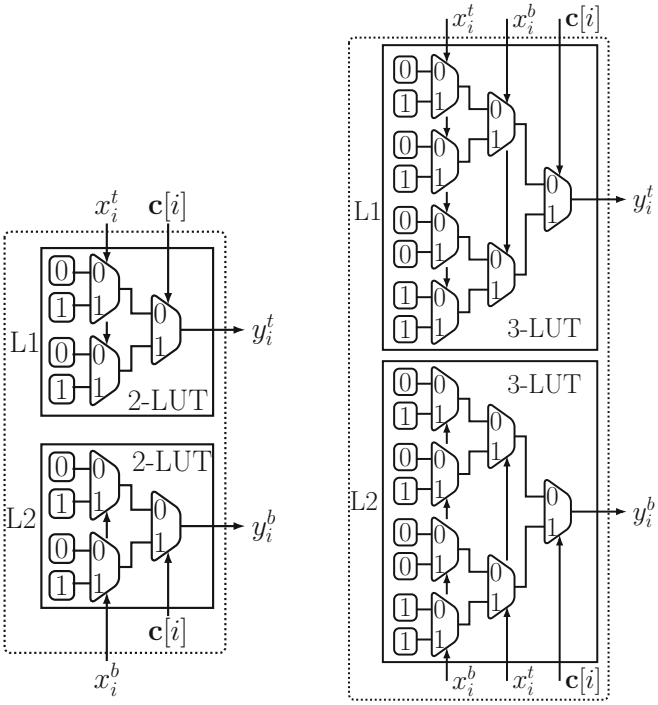
Fig. 6 Switching stages of classic APUF and PDL-based APUF [23]. Depending on the challenge bit $\mathbf{c}[i] \in \{0, 1\}$, a pair of delay elements is selected

Table 2 Truth table of G1 in Fig. 8

Inputs			Output
x_i^t	x_i^b	y_i^t	T_i
0	0	0	1
1	1	1	1
x	0	1	0
x	1	0	0

modifications of the PDL switch to make it self-testable. An identical approach can be may be considered for self-testable classic switch designs also.

Evidently, LUTs L1 and L2 have identical content, as can be observed from Fig. 7a. Therefore, these two should have equal responses for the same inputs. Thus any anomaly can be detected, as any change occurring in either L1 or L2 input, but not to the other one, can be easily detected by simple XNOR logic. Although XNOR logic can ensure that the contents of L1 and L2 are *same*, it can not guarantee that the contents are as per the depiction in Fig. 7a. This is the primary reason behind our motivation to develop a new countermeasure, as presented in Fig. 8. This countermeasure allows identifying any anomaly in its expected functional behaviour automatically. Notice that, in Fig. 7a, the contents of the SRAM cells accessed by x_i^t and x_i^b are identical with the values of x_i^t and x_i^b themselves, and these are the exact values that propagate to y_i^t and y_i^b, respectively. Therefore the self-testable design of APUF switch which we proposed, a logic module G1 and an XNOR gate (G2) are required. The XNOR gate is required for equality check of x_i^t, x_i^b and y_i^t. Functionality of G1 is briefly described in Table 2. If both G1 and G2 are '1', it can be guaranteed that contents of L1 and L2 are correct. Otherwise, an undesired modification might have been introduced in the circuit and it outputs '0'. An AND logic (G3) is employed to achieve this. It can be understood that it is possible to map the logic of each of the three functions G1, G2 and G3 to a single 4-bit LUT. Therefore, the self-testing APUF switch consumes only a single LUT in Xilinx 7-series FPGAs.



(a) PDL switch based on 2-LUT

(b) Classic switch based on 3-LUT

Fig. 7 APUF's i-th switch realization using LUT

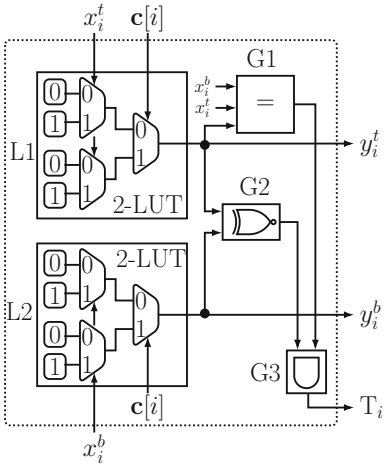


Fig. 8 Self-testable PDL switch using 2-LUTs

4 Countering Fault Attacks on ROPUF

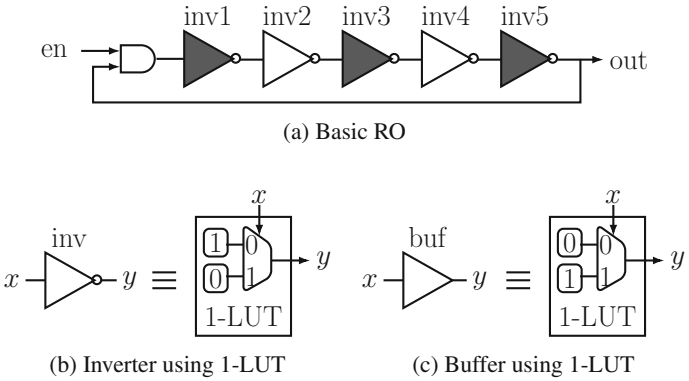
Recently, authors of [35] proposed a fault attack that aims to lower the entropy of the generated key by transforming one of a pair of ROs into a cascade of an even number of inverters by applying laser pulse injection and altering the content of the LUT of the last inverter. As a result, fault-free RO counters and faulty RO counters diverge significantly from their expected behaviour. Particularly, fault-free RO counter values are higher than the faulty RO, causing a biased distribution of the keys generated by the ROPUF. We introduce a countermeasure that can detect the presence of a fault and raises an alert signal to inform the system about the key generated by the faulty ROPUF.

A distinctive feature of ROPUF is that input challenges do not influence the functionality of individual RO. The challenge is used specifically to select the pair of RO only. The delay elements (inverters) in an RO are used to construct a loop, which is not in the case of APUF. Therefore, the presence of delay element in different stages makes it difficult to detect any modification while the RO is being evaluated. We discuss a naïve fault detection scheme for ROPUF and identify the problems this construction. Next, we improve it to develop a stronger countermeasure.

Figure 9b, c show the realization of an inverter and a buffer using 1-LUT. The following trivial facts can be observed from Fig. 9b, c:

1. The LUT content of inverter is the complement of buffer LUT, and if one changes an inverter to a buffer (or a buffer to an inverter), then it can be easily detected by comparing their output while the same input is applied to both of them.
2. Outputs of two inverters are the same when they have the same inputs.

The input to the RO inverters with odd indices (highlighted in black in Fig. 9a) are the same for each period of oscillation, and the same thing happens with the inverters

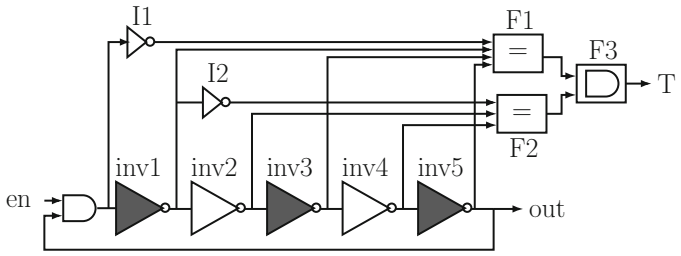


(a) Basic RO

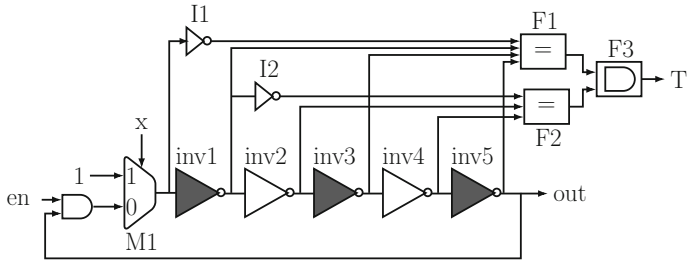
(b) Inverter using 1-LUT

(c) Buffer using 1-LUT

Fig. 9 Ring oscillator realized using 1-LUT. Inverters with odd index are highlighted with black color. Functional behaviour of F1 and F2 is identical as mentioned in Table 2, and they output '1' if and only if all inputs are same



(a) RO with FDL



(b) RO with FDL and an extra MUX

Fig. 10 Fault detection logic (FDL) with ring oscillator design. Black colour highlights the inverters with odd indices

with even indices. Therefore, a structural modification can easily be detected in RO just by checking the equality of the corresponding outputs of inverters with odd indices and similarly for even indices also. Figure 10a provides an illustration of RO design with these checkpoints. The design can operate free of faults considering that the value of 'T' should be 1 when at the end of an oscillating period, the output ('out') of RO changes its state (from $0 \rightarrow 1$ or $1 \rightarrow 0$). This circuit can any inverting to non-inverting input transformation (and vice-versa) and allowing adequate fault attack detection in [35]. If an adversary wants to bypass the countermeasure of Fig. 10a, she has to inject an additional fault in one of the two detection logic entities F1 or F2, or the final AND gate (F3), thus making the attack increasingly difficult.

One issue with the proposed countermeasure is that the 'T' signal requires constant monitoring and the end of every oscillation. The circuit can be modified to any adversary at any time before the evaluation of the RO circuit is completed. In general, this is an expensive task, considering that a 5–7 stage configuration RO has a frequency in the range of MHz. Still, we can logically assume that even if the adversary can change the circuit at any point in the middle of evaluation, it is highly improbable that she can not revert the system back before the evaluation is finished. Therefore it is adequate to check the structural integrity of the RO in pre-evaluation and post-evaluation phase. Including this, the integrity check is performed on the RO with an odd number of inverters, *not* on an RO consisting a feedback loop. This is acceptable considering that the feedback loop does not have any logic element.

However, it is necessary to check the integrity of the RO chain for both 0 and 1 input provided to the first inverter (inv1) in Fig. 10a. This countermeasure facility is realized by incorporating an additional MUX, as illustrated in Fig. 10b. If 'x' and 'en' are set to '0', then input of inverter 'inv1' is set to '0', whereas for 'x' equal to 1 and 'en' equal to either '1' or '0', input of 'inv1' is fixed to '1'. These two assignments are required for signals 'en' and 'x' for fault detection, while the following assignment is required:'x = 0' and 'en = 1' for normal operation of the RO.

5 Countering for Fault Attacks on Other Delay PUFs

This section is mainly devoted to possible fault attacks on two other varieties of delay PUFs—the Loop PUF and the Bistable Ring PUF. The countermeasure strategies for preventing such attacks also have been discussed.

5.1 Loop PUF

The Loop PUF can be considered as a variant of the delay PUF, comprising a connected sequence of delay elements forming a closed loop as illustrated in Fig. 11. The input signal of each delay element ('in' in Fig. 12) in a delay chain propagate to the output ('out' in Fig. 12) through any of the two possible paths. The path is defined by the control bit. The controller for the LPUF module generates N control words $(\mathbf{c}^1, \ldots, \mathbf{c}^N)$, each of length M-bits, extracted from an input challenge $\mathbf{c}$. Thus, for a given challenge c, N different permutations of the control words $(\mathbf{c}^1, \ldots, \mathbf{c}^N)$ generate an N bit response. For further reading, refer to [10].

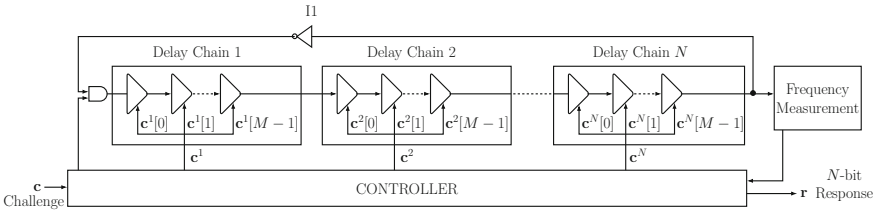
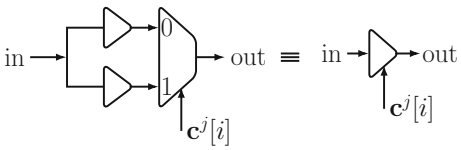


Fig. 11 Architectural overview of Loop PUF

Fig. 12 Non-inverting delay element of Loop PUF with control bit $\mathbf{c}^j[i]$, $i \in [0, M-1]$, $j \in [1, N]$



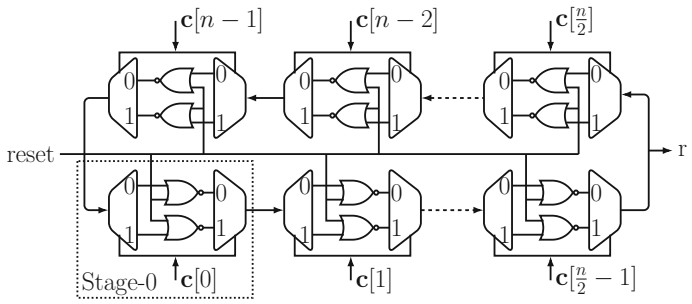


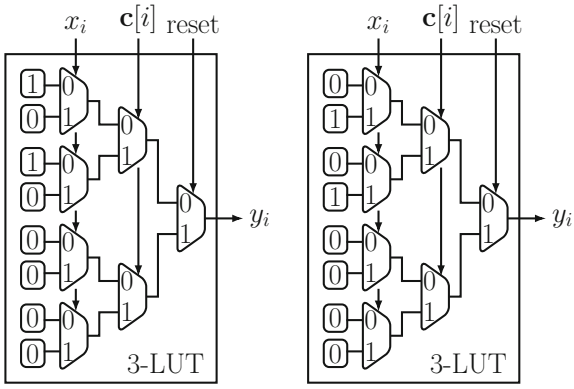
Fig. 13 Architectural diagram of BRPUF with n-bit challenge **c** and 1-bit response r

Now, we begin our discussion on possible fault attack strategies for LPUF implementation on FPGA. Precise fault injection strategies like laser pulses can be carried out on LPUF also, as in the attacks on XOR APUF and ROPUF presented in [35]. The objective of the attack we proposed to alter the behaviour of an LPUF in a way such that an authentication failure is encountered when the PUF is used for hardware authentication purpose. An adversary can use fault injection to alter the inverter (I1) of LPUF (cf. Fig. 11) to forcefully transform it into a non-inverting gate, inhibiting the oscillation in the input signal of the frequency measurement unit. Alternatively, fault injection can also be performed to transform a non-inverting delay element to an inverting one, but one has to ensure that inverter I1 is not modified. Therefore, in case of any of the attacks, the fault injection stops the oscillation making the output of the PUF fixed irrespective of the input challenge, causing authentication failure. If remnant fault injection has been performed, then the SRAM-based FPGA would allow the LPUF to revert to its initial fault-free state after reconfiguration, but the services relying upon the LPUF would be disrupted for the time period before reconfiguration.

We discuss possible countermeasure strategies for this attack. Similar to XOR APUF and ROPUF, additional logic is necessary to identify to any fault causing modification of LPUF architecture. This is shown in (cf. Fig. 11), all the non-inverting delay elements are homogeneous in nature and receive similar input during the oscillation period. Therefore a fault detection method similar to the ROPUF in Fig. 10 can detect any modification in a subset circuit formed by the non-inverting elements. Any modification done to the inverter I1 of LPUF can be identified by allowing no modification to the non-inverting delay elements, the inverter is functioning properly.

5.2 Bistable Ring PUF

A bistable ring (BR) is closed chain constructed using an even number of inverters. BR has only two ‘0’ or ‘1’, originating from the random process variation of inverters.



(a) Fault-free i-th stage (b) Faulty i-th stage

Fig. 14 A 3-LUT based realization of i-th stage of a BRPUF

The functionality of BR is exploited in Bistable Ring PUF (BRPUF) to generate 1-bit response. The compact representation proposed in [9] introduced the challenge-response mechanism. This compact representation comprises of two NOR gates, a MUX and a DMUX. The principal objective of the DMUX is to prevent the switching of output for both the NOR gates for a given challenge bit. However, in the case of FPGA implementations, DMUX can be avoided to realize the two NOR gates in a singe LUT, as depicted in Fig. 14a. To evaluate a BRPUF for a given challenge, we start by asserting 'reset' signal to '1' reseting all BRPUF stages' output to '0'. This is the initial and unstable state of a BR. After that, 'reset' is asserted to '0' and the BR reaches a stable state after the signal propagates through the loop (Fig. 13).

BRPUF instances are vulnerable to precise fault injections, as described in [35], and this can be clearly observed in Fig. 14a. If the attacker is able to alter the inverting stage of an BRPUF to a non-inverting one, as shown in Fig. 14b, the BR output would start oscillating and it would start working like an RO, unable to reach to a stable state. Therefore, fault injection can make a BRPUF unreliable. The fault detection scheme proposed for the ROPUF used in detection of fault for BRPUF can also used here as these have homogeneous stages. In addition to this, the authors of [35] also mentioned the fault injection-based modeling of XOR BRPUF. This can also be detected with the fault injection scheme proposed for APUF in Fig. 5b.

6 Fault Recovery Schemes

So far we have discussed about the proposed schemes for fault detection against fault attacks on XOR APUF and RO PUF. However, it is extremely important to recover the circuit back to its original fault-free state. Now, we explore the possibilities for efficient approaches for fault recovery of APUF and XOR APUF circuits. Other

delay based PUFs can also have similar methodologies for fault recovery. Broadly, FPGA-based designs have two main choices for fault recovery:

1. **Rollback**. Revert back to have the same timing performance of the original one.
2. **Random-sliding**. Replace the faulty PUF instance with a new one with different timing behaviour.

We shall see later that the second recovery scheme is more suitable than the other one. First, we identify that an adversary requires to restart the circuit x times to collect CRPs for each of the x APUF instances while performing fault assisted modeling attack on x-XOR APUF in [35]. We consider this assumption for our realizations of the *rollback* and *random-sliding* recovery schemes.

6.1 Rollback

On Xilinx FPGA, the faulty PUF instance can be reverted back to the original through two ways: *partial reconfiguration* (PR) [4] and *dynamically configurable LUT*. In PR, the original PR module bitstream of the PUF instance is stored in a memory, and later dynamic partial reconfiguration is performed without any external communication. In Xilinx FPGA, a user application can use the internal configuration access port (ICAP) [3, 17] to configure the application structure at run time. In our design, a *Fault Recovery Controller* (FRC) is employed to perform PUF reconfiguration upon fault detection in the PUF design.

Rollback can be performed in a different way also. Xilinx FPGAs contain configurable LUT (CFGLUT). These are widely used to design evolvable hardware (EH) [6, 8]. The contents of CFGLUTs can be modified dynamically in a synchronous way to change the functional behavior of the circuit. Our APUF design uses CFGLUT for the switches, which allows us to recover the fault modified switch. Xilinx provides HDL primitive CFGLUT5 [1, p. 106] for its 7 series FPGAs, and also for Spartan-6 FPGAs, that facilitates the instantiation of CFGLUT in the design and consumes a single 6-LUT in SLICEM (memory slice) [2, pp. 18–19] in FPGA. CGFLUT interface is shown in Fig. 15. This method is comparatively more efficient that the PR based scheme. This is mainly because of the homogeneity of the APUF switches, and corresponding CFGLUTs can be configured parallel with the same configuration file in 32 clock cycles only.

6.2 Random-Sliding

Figure 16a depicts 4-LUT based realization of two-variable function (f), and only i0 and i1 inputs of the LUT are used for f and inputs i2 and i3 remain unused. As shown by a coloured rectangle in Fig. 16a, we can see that i2 = 0 and i3 = 0, only four SRAM cells are used out of 16 SRAM cells. From Fig. 16a, we can observe that starting from

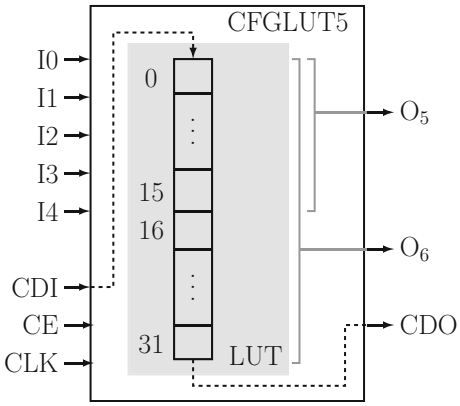


Fig. 15 Dynamically reconfigurable lookup table is primarily realized using CFGLUT5 HDL primitive. Other than LUT inputs I0, ..., I4 and outputs O_5 and O_6, it has a few reconfiguration interface with ports: *configuration data in* (CDI), *configuration data out* (CDO), *clock enable* (CE) and *configuration clock* (CLK). This primitive can be used to realize either a single 5-LUT with output O_6 or as two 4-LUTs using both the outputs O_5 and O_6

the top, the four consecutive SRAM cells of alternative blocks have the same value. This implies that the LUT would work as the function f irrespective of the value of i2 and i3 of 4-LUT. However, even if there is no change in functional property (input-output characteristic) of LUT, the timing behaviour of f can show different result because different propagation delay for different input-to-output combination [23] after random assignment of i2 and i3. Figure 16 also explains how LUT's SRAM cells are used to realize the function f for different assignments to i2 and i3 inputs of LUT at different locations. This is one to make the function f fault-tolerant. Depending upon the whether the output of f is faulty or not, we can try a different configuration of i2 and i3. Any configuration which produces non-faulty output and uses that configuration. In the worst case, if the entire LUT is detected as a faulty one, (partial) reconfiguration of the FPGA is required. This fault-tolerant function realization approach is referred as *Random Sliding*.

LUT based implementation of PDL switch is shown in Fig. 7a, which consists of two 2-LUTs. Therefore, if we realize this PDL switch with 6-LUT of Xilinx Artrix-7 FPGA, we are left with four unused LUT inputs. If the same assignment is done for the unused inputs for both the LUTs of PDL switch, then 2^4 switch configurations can be used as there are 16 possible assignments to four unused LUT inputs. Therefore, the random sliding notion can be extended to make PDL switches fault-tolerant. Afterwards, to have a fault-tolerant APUF design with fault detection circuit included this random-slid PDL switch can be used, as mentioned in Sect. 3. Observe that, one can reconfigure the FPGA with different configuration bits for random sliding for each switch, but we have used the same configuration bits to all unused LUT inputs

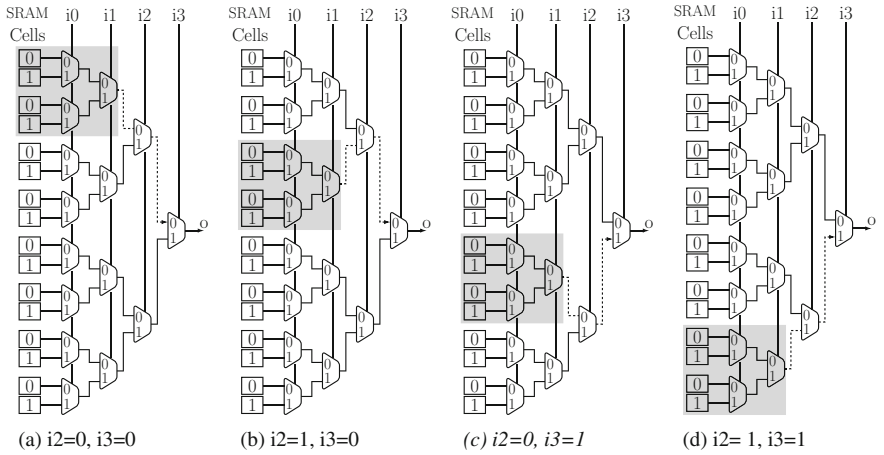


Fig. 16 A 2-variable Boolean function $f(\text{i0}, \text{i1})$ is implemented using 4-LUT. The circuit corresponding to $f(\text{i0}, \text{i1})$ shows four different timing behaviors for four different assignments for i2i3 $\in \{00, 10, 01, 11\}$

of PDL switches. Thus, 16 different configurations can be realized using the same hardware. Therefore this method does not require any additional hardware resource.

XOR APUF can be made robust against fault-assisted modeling attacks by employing random-slid PDL as discussed in [35]. To perform the XOR APUF attack in [35], the PUF circuits needs to be restarted x times to collect the CRPs of x APUFs in x-XOR APUF. Depending upon the random-slid configurations generation every time, whether randomly or not, the x-XOR PUFs become different before and after the restart. Therefore, even if an adversary is able to collect CRP data for APUF by restarting x-times, the collected data is not same XOR APUF. This increase the difficulty level for modeling the individual APUFs. If we can exploit the random-slid scheme with the fault detection scheme proposed by us, fault-tolerance can be ensured for the delay PUF design. In the presence of random sliding, the verifier can authenticate the PUF by maintaining a database of all configuration of the PUF. If the PUF is a legitimate one; then for some possible configuration, the responses should match the CRPs.

Note that the use of `Random-sliding` is not limited to the FPGA-implemented PUFs. It can be used in any FPGA-based design. The purpose of using `Random-sliding` may be to achieve fault-tolerant feature or to get different realizations of the same function.

7 FPGA Implementations and Hardware Overhead

We have implemented the original APUF and RO designs, including the fault-tolerant versions, to validate the proposed fault attack countermeasures. Xilinx Artix-7 FPGAs have been used as the target platform.

7.1 Implementation Details and Fault Simulation

Despite Artix-7 FPGA having 6-LUT blocks, the CFGLUT5 HDL primitive allows to realize at most 5-input function. Classic switches require two 3-LUTs and two 2-LUTs for PDL switch in APUF. Therefore, two (or three) LUT inputs, which remain unused can be utilized for random sliding. This implies CFGLUT5 can be used for dynamic configuration of the switch and random sliding both.

Combination of configurable LUT and random sliding has been used for fault recovery. We apply random sliding to a faulty instance to search for a fault free random-slid instance. If we do not find any fault-free instance, then dynamic reconfiguration is applied to CFGLUT5 based switches. A similar approach has been followed for implementation of ROs and ROPUF. CFGLUT5 instances are primarily used for realizing each stage of the 5-stage RO.

We have performed the simulation of fault injection process with the CFGLUTs, which allows changing the LUT configuration at run time. No complex designs were used to perform injections on PUFs implemented with CFGLUT5 primitive only. In this the fault simulation can be performed in a more precise and controlled way as it allows us to reinitialize the LUT with different configuration bits. The number of bits to be reconfigured and the bit position decides the number of clock cycles required for fault injection. This is mainly because reconfiguration takes place serially.

7.2 Post Placed-and-Route Simulation Results

A simulated output of the APUF behaviour (post place-and-route), with classic switching stages before and after the fault injection, is depicted in Fig. 17. Eight clock cycles are required (there are eight SRAM cells as shown in Fig. 7b) to configure the bottom part of the last APUF switch, to make its output '0' regardless of the input. 'cenB[15:0]' signal vector (for 16-stage APUF) in Fig. 17 controls the the reconfiguration of the bottom part of the switch. When 'cenB[i] = 1', the configuration is performed for bottom part of the ith switch. Fault injection is controlled by the vector 'cenB[15:0]'. 'tig = 1' and 'T = 1' implies that the response of APUF is valid. Observed in Fig. 17, that 'T' is not '1' even even if the trigger ('tig') is set to '1'. This indicates a fault in the APUF implementation. Also note that signals 'pathT'

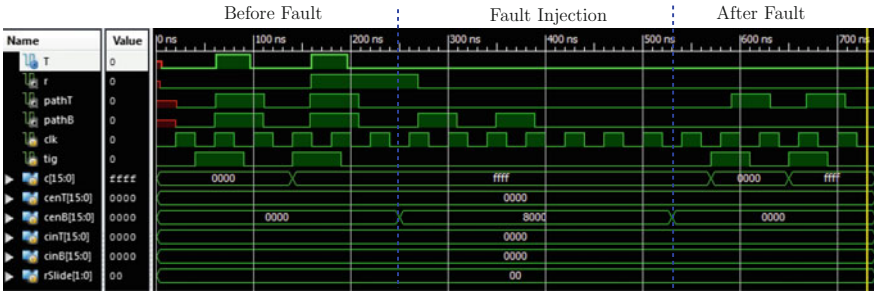


Fig. 17 Waveform of post place-and-route simulation 16-stage APUF (cf. Fig. 4) with classic switches. It depicts the pre and post fault injection behavior of APUF. This design is without self-testable switch. Here signal vector 'c' and signal 'r' is the challenge and corresponding response, respectively, and 'T' is the output of fault detection circuit in APUF

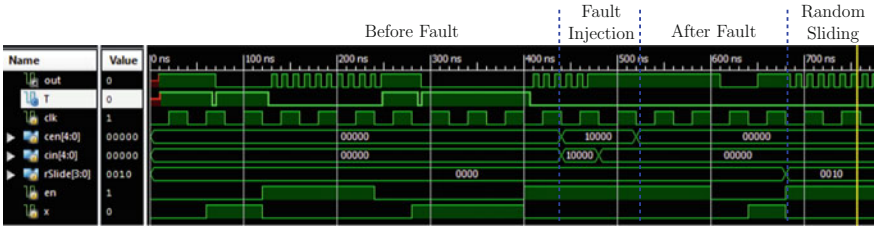


Fig. 18 Waveform of post place-and-route simulation of 5-stage RO with fault detection logic as shown in Fig. 10b. It depicts the pre and post fault injection behavior of RO. It also shows how RO is recovered by using the random-sliding with 'rSlide' signal vector. 'T' is the output of fault detection circuit in RO

and 'pathB' belong to the top and bottom terminals of APUF's last switch. After the completion of fault injection, 'pathB' is fixed to '0'.

A simulated behaviour (post place-and-route) for pre fault injection and post fault injection of a 5-stage RO with incorporated fault detection logic is presented in Fig. 18. CFGLUT5 instances have used predominantly to realize each stage of the RO. As mentioned previously, this particular design allows to detect fault in pre-evaluation and post-evaluation stage. If 'T = 1' is present at these checkpoints, then RO is considered as fault free fit for response generation. Otherwise the RO is in its faulty state, and must be recovered back to normal state. The vector 'cin[i]' in Fig. 18 is used as the configuration bit for the ith stage of RO. 'cen[i]' is used as the enable signal for configuring CFGLUT5 of ith stage. We have injected faults in the last stage ($i = 4$) of the 5-stage RO to make it as a non-inverting buffer.

Table 3 Summary of resource overhead of fault-tolerant design in Artix-7 FPGA

Design	Original design		FDL	
	LUT	FF	LUT	FF
APUF (Fig. 4)	$2n$	1	1	0
3-XOR APUF (Fig. 5b)	$6n + 1$	3	$3 + 6$	0
5-stage RO (Fig. 10b)	6	0	4	0

Note n implies number of stages present in APUF. FDL refers to fault detection logic, and it is a pure combinational circuit

7.3 Hardware Overhead

Table 3 provides a summary of resource overhead of the proposed countermeasures. One extra LUT is required to detect a fault in a single fault-tolerant APUF. As 3-XOR APUF is constructed using three fault-tolerant APUFs; three LUTs are required to transform the component APUFs into fault-tolerant ones. Additionally, six LUTs are needed to transform the 3-XOR APUF into a fault-tolerant one. This is depicted in Fig. 5b. For designing 5-stage fault-tolerant RO as in Fig. 10b, additional six LUTs are required to implement the logic for M1, I1, I2, F1, F2, and F3. However, (I1, F1) have been mapped to one LUT and (I2, F2) have been mapped to another one. Therefore, it requires just four LUTs of the FDL logic of an RO. Although separate LUT has been used to map M1, this can also be mapped to the same LUT of AND gate (just prior to the first stage) of the RO. Notice that, each LUT has dual dual output in Artrix-7 FPGAs. This allows for further logic resource optimization of the FDL.Our reported resource overhead includes only the PUF with FDL circuit in Table 3. However, additional logic resources are required to implement and incorporate the above mentioned fault-recovery methods.

8 Summary

We discussed various efficient fault-tolerant implementation of delay PUFs which are robust against a number of high precision fault attacks. We introduced fault-tolerant implementations for APUFs that incorporate additional fault detection logic (FDL) to identify the illegitimate modifications to the circuit in run time due to fault injections. This technique has been extended to develop a similar fault-tolerant implementation of XOR APUF. It also includes additional reinforcement for the terminal XOR gate to prevent modeling the PUF by collecting the individual response data. A fault-tolerant implementation for ROPUF also has been introduced. The proposed design counters the attacks targeted to individual RO components trying produce bias in the output. The technique can also identify any modification to the delay stages trying to halt the oscillations at the RO output. We also discussed about the possible fault attacks on two other delay PUFs—namely Loop PUF and Bistable Ring PUF.

We discussed about few prospective circuit modifications those can prevent such attacks. We also discussed two fault recovery schemes—the *Rollback* method and the *Random-sliding* method. A brief comparative discussion of their implementation overhead and efficiency has been provided. In the end, validation of correctness of the designed countermeasures were performed by performing precise fault injection on different delay PUF implementation on Artrix-7 FPGA platform. The assumption of absence of a fault in fault detection circuit is practically achievable, one might have to use redundant copies of the FDL logic, and it is possible because it is a deterministic circuit component. However, as PUF behavior is unique for each instance, fault-tolerance based redundancy can not be achieved for it.

References

1. Xilinx 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide for HDL Designs UG768 (v14.7) October 2, 2013. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/7series_hdl.pdf
2. Xilinx 7 Series FPGAs Configurable Logic Block User Guide UG474 (v1.7) November 17, 2014. http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf
3. Xilinx 7 Series FPGAs Configuration User Guide UG470 (v1.10) June 24, 2015. http://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf
4. Xilinx Partial Reconfiguration User Guide UG702 (v14.1) April 24, 2012. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug702.pdf
5. Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault injection attacks on cryptographic devices: theory, practice, and countermeasures. Proc. IEEE **100**(11), 3056–3076 (2012). https://doi.org/10.1109/JPROC.2012.2188769
6. Bartolini, D.B., Carminati, M., Cancare, F., Santambrogio, M.D., Sciuto, D.: HERA project's holistic evolutionary framework. In: Proceedings of IEEE International Symposium on Parallel & Distributed Workshops, pp. 231–238 (2013). https://doi.org/10.1109/IPDPSW.2013.110
7. Becker, G.T.: The gap between promise and reality: on the insecurity of XOR Arbiter PUFs. In: Proceedings of 17th International Workshop on Cryptographic Hardware and Embedded Systems (CHES) (2015)
8. Cancare, F., Bartolini, D.B., Carminati, M., Sciuto, D., Santambrogio, M.D.: On the evolution of hardware circuits via reconfigurable architectures. ACM Trans. Reconfig. Technol. Syst. (TRETS) **5**(4), 22 (2012). https://doi.org/10.1145/2392616.2392620
9. Chen, Q., Csaba, G., Lugli, P., Schlichtmann, U., ührmair, U.R.: The bistable ring PUF: a new architecture for strong physical unclonable functions. In: Proceedings of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 134 –141 (2011)
10. Cherif, Z., Danger, J.L., Guilley, S., Bossuet, L.: An easy-to-design PUF based on a single oscillator: the loop PUF. In: Proceedings of 15th Euromicro Conference on Digital System Design (DSD), pp. 156–162 (2012)
11. Delvaux, J., Verbauwhede, I.: Fault injection modeling attacks on 65 nm Arbiter and RO Sum PUFs via environmental changes. IACR Cryptol. ePrint Archive **2013**, 619 (2013)
12. Delvaux, J., Verbauwhede, I.: Side channel modeling attacks on 65 nm Arbiter PUFs exploiting CMOS device noise. In: IEEE 6th International Symposium on Hardware-Oriented Security and Trust (2013)
13. Ganji, F., Krämer, J., Seifert, J., Tajik, S.: Lattice basis reduction attack against physically unclonable functions. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1070–1080 (2015)

14. Holcomb, D.E., Burleson, W.P., Fu, K.: Power-up SRAM state as an identifying fingerprint and source of true random numbers. IEEE Trans. Comput. **58**(9), 1198–1210 (2009)
15. Karaklajic, D., Schmidt, J., Verbauwhede, I.: Hardware designer's guide to fault attacks. IEEE Trans. VLSI Syst. **21**(12), 2295–2306 (2013)
16. Koçabas, Ü., Peter, A., Katzenbeisser, S., Sadeghi, A.: Converse PUF-based authentication. In: Trust and Trustworthy Computing—5th International Conference, TRUST 2012, Vienna, Austria, 13–15 June 2012. Proceedings, pp. 142–158 (2012)
17. Lai, V., Diessel, O.: ICAP-I: A reusable interface for the internal reconfiguration of Xilinx FPGAs. In: Proceedings of International Conference on Field-Programmable Technology (FPT), pp. 357–360 (2009). https://doi.org/10.1109/FPT.2009.5377616
18. Lim, D.: Extracting Secret Keys from Integrated Circuits. Master's thesis, MIT, USA (2004)
19. Maes, R.: Physically Unclonable Functions—Constructions, Properties and Applications. Springer (2013)
20. Maes, R., van der Leest, V., van der Sluis, E., Willems, F.: Secure key generation from biased PUFs. In: Proceedings of 17th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), pp. 517–534 (2015)
21. Maiti, A., Nagesh, R., Reddy, A., Schaumont, P.: Physical unclonable function and true random number generator: a compact and scalable implementation. In: Proceedings of ACM Great Lakes Symposium on VLSI, pp. 425–428 (2009)
22. Maiti, A., Gunreddy, V., Schaumont, P.: A systematic method to evaluate and compare the performance of physical unclonable functions. IACR Cryptol. ePrint Archive **2011**, 657 (2011)
23. Majzoobi, M., Koushanfar, F., Devadas, S.: FPGA PUF using programmable delay lines. In: IEEE International Workshop on Information Forensics and Security (WIFS), pp. 1–6 (2010)
24. Majzoobi, M., Rostami, M., Koushanfar, F., Wallach, D.S., Devadas, S.: Slender PUF protocol: a lightweight, robust, and secure authentication by substring matching. In: Proceedings of IEEE Symposium on Security and Privacy Workshops, pp. 33–44 (2012)
25. Malkin, T., Standaert, F., Yung, M.: A comparative cost/security analysis of fault attack countermeasures. In: Proceedings of Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 159–172 (2006)
26. Merli, D., Schuster, D., Stumpf, F., Sigl, G.: Semi-invasive EM attack on FPGA RO PUFs and countermeasures. In: Proceedings of the 6th Workshop on Embedded Systems Security (WESS 2011) (2011)
27. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., Schmidhuber, J.: Modeling attacks on physical unclonable functions. In: Proceedings of 17th ACM Conference on Computer and Communications Security (CCS), pp. 237–249. ACM, New York, NY, USA (2010)
28. Rührmair, U., Sölter, J., Sehnke, F., Xu, X., Mahmoud, A., Stoyanova, V., Dror, G., Schmidhuber, J., Burleson, W., Devadas, S.: PUF modeling attacks on simulated and silicon data. IEEE Trans. Inf. Forensics Secur. **8**(11), 1876–1891 (2013)
29. Rührmair, U., Xu, X., Sölter, J., Mahmoud, A., Majzoobi, M., Koushanfar, F., Burleson, W.P.: Efficient power and timing side channels for physical unclonable functions. In: Proceedings of 16th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), pp. 476–492 (2014)
30. Sahoo, D.P.: Design and Analysis of Secure Physically Unclonable Function Compositions. Ph.D. thesis (2017)
31. Sahoo, D.P., Saha, S., Mukhopadhyay, D., Chakraborty, R.S., Kapoor, H.: Composite PUF: a new design paradigm for physically unclonable functions on FPGA. In: Proceedings of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 50–55 (2014)
32. Sahoo, D.P., Mukhopadhyay, D., Chakraborty, R.S., Nguyen, P.H.: A multiplexer-based Arbiter PUF composition with enhanced reliability and security. IEEE Trans. Comput. **67**(3), 403–417 (2018). https://doi.org/10.1109/TC.2017.2749226. http://doi.ieeecomputersociety.org/10.1109/TC.2017.2749226
33. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Proceedings of Design Automation Conference (DAC), pp. 9–14. ACM Press, New York, NY, USA (2007)

34. Tajik, S., Dietz, E., Frohmann, S., Seifert, J., Nedospasov, D., Helfmeier, C., Boit, C., Dittrich, H.: Physical characterization of Arbiter PUFs. In: Proceedings of 16th International Workshop Cryptographic Hardware and Embedded Systems (CHES), pp. 493–509 (2014)
35. Tajik, S., Lohrke, H., Ganji, F., Seifert, J.P., Boit, C.: Laser fault attack on physically unclonable functions. In: Proceedings of 12th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC) (2015)
36. Tobisch, J., Becker, G.T.: On the scaling of machine learning attacks on PUFs with application to noise bifurcation. In: Proceedings of 11th International Workshop on Radio Frequency Identification: Security and Privacy Issues (RFIDsec), pp. 17–31 (2015)

Fault Tolerance in 3D-ICs

Raviteja P. Reddy, Amit Acharyya and Saqib Khursheed

Abstract The systems with emerging technologies like Internet-of-Things and beyond Von-Neumann architectures can be produced in large scale only if they are resilient-aware, cost-effective and secure. The resilient and cost-effective solutions can be achieved by incorporating fault tolerance techniques at the architecture level of the system design is one of the plausible solutions. The choice of various fault tolerance techniques gives the designers a freedom to incorporate these in the early stage of the design and in turn leading to high yield and reliable architectures. Through-silicon-via (TSV) interconnects based three-dimensional integrated circuits are emerging technologies consisting of vertical communication between the stacked dies, leading to the decrease of wire length and thus enhances the system performance. However, yield and reliability are the major issues that hinder resilient and cost-effective solutions for 3D-IC design. These can be addressed by incorporation of fault tolerance techniques.

1 Introduction

CMOS technology is approaching its scaling limits and stacking of dies is a feasible solution to decrease delay and power of interconnects, while increasing transistor density. Dies can be stacked by existing technology by directly bonding of TSVs [1]. This enables heterogeneous technology integration of numerous dies. TSVs are basically metal pillars that pass through the silicon substrate to connect the metal pads and all dies in the stack. The third dimension significantly reduces the wire delay and

R. P. Reddy · A. Acharyya (✉)
Electrical Engineering, Indian Institute of Technology, Hyderabad, India
e-mail: amit_acharyya@iith.ac.in

R. P. Reddy
e-mail: ee15resch11006@iith.ac.in

S. Khursheed
Electrical Engineering and Electronics, University of Liverpool, Liverpool, UK
e-mail: S.Khursheed@liverpool.ac.uk

R. S. Chakraborty et al. (eds.), *Security and Fault Tolerance in Internet of Things*,
Internet of Things, https://doi.org/10.1007/978-3-030-02807-7_8

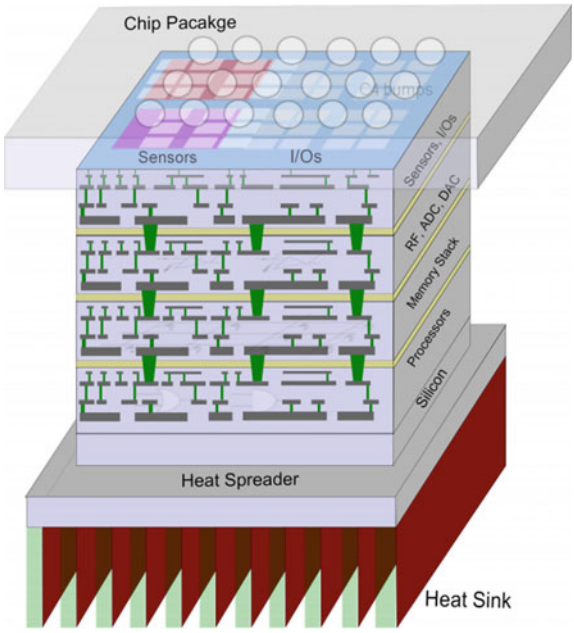


Fig. 1 Heterogeneous 3D-IC [32]

power, and also delivers high-density interconnect for data transfer between layers [2]. A conceptual view of a 3D IC is shown in Fig. 1. Heterogeneous integration of memories, CPUs, sensors, radio frequency (RF) and analog circuits are combined in the same package to avoid slow and long off-chip wires. A heat sink and spreader are on the other side of the chip. However, 3D integration introduces new technology challenges. Through silicon via consists of defects due to the manufacturing process and thermal mechanical stress, crack in dielectric around the TSVs, delamination between landing pad and TSVs, as well as short and open defects due to improper copper filling and electro migration forms voids and causes delay of signal and malfunctioning of 3D-IC. The yield and reliability in 3D-ICs is affected due defects in TSVs and accelerated by thermo mechanical stress and on-field operation. These challenges are addressed by incorporating fault tolerance techniques for enhancing reliability and yield of 3D-ICs.

2 Defects in TSVs

The manifestation of defects in TSVs occur during the fabrication process and bonding stage, when diverse dies are combined and one defective TSV can possibly fail other TSVs due to defect clustering effect, as well as the known-good dies. TSVs in 3D integration, can lead to new reliability and yield challenges. The deficient TSV materialization process can lead to a void in TSVs, and void growth is hastened

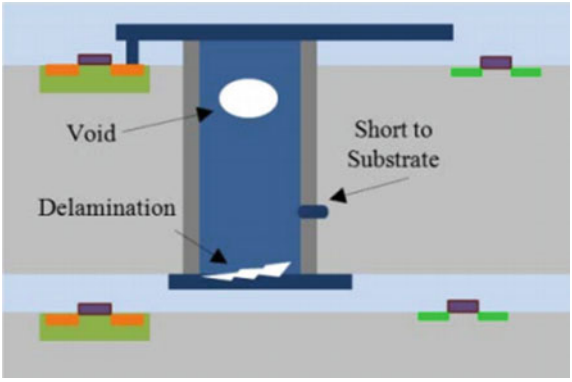


Fig. 2 Defects in TSVs [19]

by electro migration effect [3, 4], moreover, the EM effect is driven by temperature during on field operation. The discrepancy of coefficient of thermal expansion between silicon and TSV, the thermal mechanical stress can enterprise crack between silicon substrate and dielectric layer [5, 6]. This crack in interface is analyzed [7] by considering the impacts of keep-out-zone (KOZ) length, TSV pitch, thickness dielectric of layer, TSV dimension and TSV placement. The cracks can take place at the boundary TSVs and landing pad and causing delamination defect due to thermal-mechanical stress during fabrication or normal operation [8–10]. For a TSV interface that employs micro bumps, studies [11–13] show that TSV reliability is also susceptible due to both EM effect and TSV-induced stress, and cracks can transpire in the intersections [14, 15]. The enormous size discrepancy between TSVs and metal wires leads to the higher current density in the metal interconnects which is higher than TSVs, leading to EM problems [16, 17]. Electro migration is modeled between wires and TSVs (landing pad) in [18] for finding the effect of dimensions of a TSV and landing pad. Hence, from all defects, three defects types have been pondered as foremost TSV defects are void, delamination and short to substrate as shown in Fig. 2.

3 Fault Tolerance Techniques and Classification

The manifestation of various defects in TSVs leads to the need of fault tolerance techniques in 3D-ICs for a resilient system design. Without fault tolerance techniques, the cost of adding immoral dies with defective TSVs will be high, predominantly design with high functional TSV and high failure rate. There are two types of fault tolerance techniques that are incorporated in 3D-IC design: (1) Redundant TSV and (2) Time Division Multiplexing Access (TDMA without redundant TSVs). The redundant TSV fault tolerance techniques are further classified into four types of

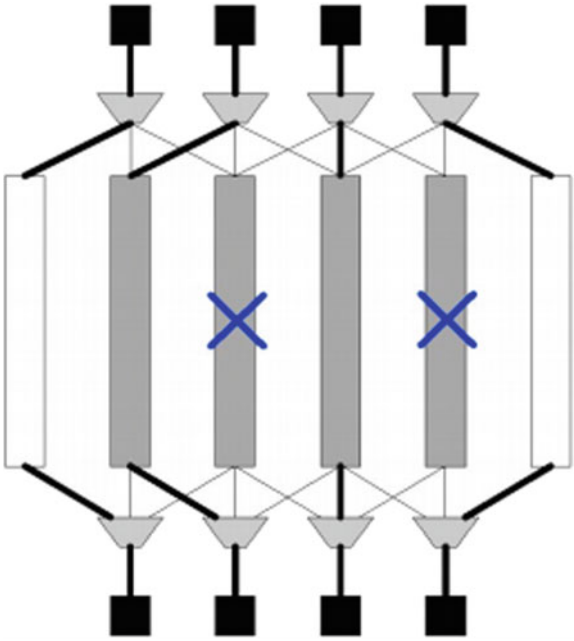


Fig. 3 Signal switching [21]

configuration strategies: (i) Signal switching, (ii) Signal shifting, (iii) Cross bar, (iv) Router based and (v) Online fault tolerance.

4 Signal Switching

A redundant TSV strategy is proposed by Samsung to enhance the yield of 3D memory products. The four signal TSVs (grey color) and two redundant TSVs are paired to form a group of six TSVs, as shown in Fig. 3. The redundancy ratio is 1:2 and it works even if any two TSV fails within the group. This signal switching improves the yield and lifetime reliability. In overall, a number of redundant TSVs are accompanying with a group of TSVs. If one of the TSVs fails, it is switched by one of the redundant TSVs. In the conservative structure, the redundant TSVs are steadfast and stationary in location. In this scheme there is no difference between regular and redundant TSVs. If a failure occurs at a TSV, the residual TSVs are all switched to the neighboring ones, so a failed TSV is always paired with adjacent TSV. This decline in the by-pass track leads to reduction in routing complication.

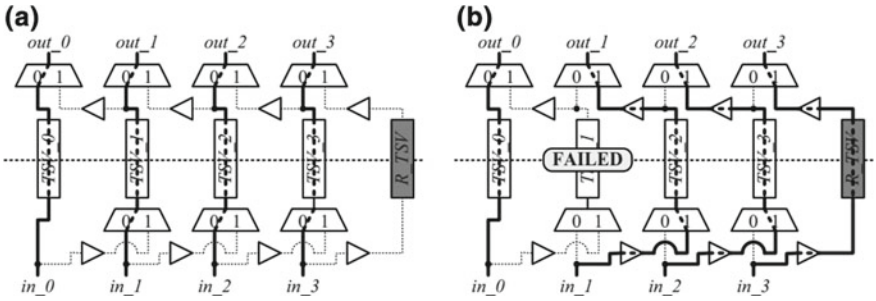


Fig. 4 Recovery mechanism of TSV **a** Normal operation. **b** TSV_1 failed [19]

5 Signal Shifting with TSV Chains

The Signal shifting is developed to connect functional TSVs in a group of TSVs with one redundant TSV to form a chain of TSVs as shown in Fig. 4. If there is one defective TSV in a group of TSVs, signal shifting is done to repair it with the redundant. If each TSV group contains TSVs, the redundancy ratio of this technique is 1:N, and it can accommodate one TSV failure in the group. The shifting of signal to neighboring TSV require two mux each TSV, when one TSV is failed. The delay increased due to the capacitance loading of the extra wires used for signal shifting is balanced by buffers pairs added to each TSV. The redundant TSVs are located at the last location and allied as chain. When there is no failed TSV, all signals are delivered by unique TSVs, as shown in Fig. 4a. When one of the TSV is failed, the signal of the failed TSV needs to be shifted. The shifting for one failed TSV effects all signals between the redundant TSV and the failed TSV. For example, let TSV 1 be failed. The shifting of signal paths are shown in Fig. 4b. In each TSV chain structure can recover only defective TSV and only one of them can be recovered if two or more defective TSVs in a chain. The e-fuse array is used in the architecture and programmed by scan-chain to provide selection lines to the multiplexers. All signals connected to MUXs are set to 0 by default. When the testing for TSV connectivity is done, signals are scanned into program the e-fuses, so that each MUX obtains an applicable control signal.

Due physical design challenges and manufacturing constraints, TSVs are not encouraged to be located randomly on a plane. From the built-up point of view, a regular assignment of TSVs enhances the yield, due to the accurate lithographic process. In real-world designs, TSVs are recommended to be placed in block uniformly, which are confirmed in floor plan stage. In each TSV block, TSVs are placed in a grid-based form to justify the pitch restraint of bond pads. Examples of TSV blocks are shown in Fig. 5. It is not suggested for a TSV-chain to contain TSVs of other TSV blocks due to lengthy wires for signal shifting. Hence, a TSV-chain in design is recommended to contain TSVs in the same TSV block. Furthermore, each TSV block contain only one redundant TSV. In case the number of TSVs in a TSV block

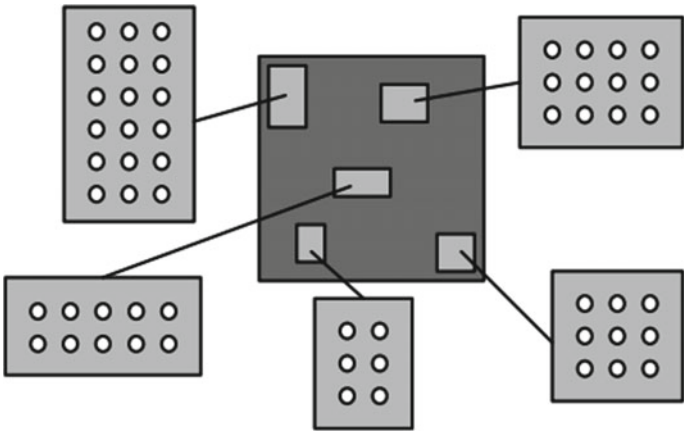


Fig. 5 TSV blocks [19]

is too large for TSV-chain, the TSV block needs to be subdivided into a smaller TSV blocks as shown in Fig. 5.

6 Signal Shifting with Optimal Grouping

Figure 6a demonstrates the idea of optimal grouping technique. A design may consists of regular TSVs and redundant TSVs and they are subdivided into groups with a grouping ratio of $gr = Ngr : Ngs$, with number of regular TSVs (Ngr) and redundant TSVs (Ngs) in a group. The routing block is shown in Fig. 6b, it performs re-routing of signals to overcome malfunctioning TSVs. A block of fuse-ROM provides control signals to routing blocks. In this optimal grouping technique, the routing block consists multiplexers, which are used to select defect free signal paths for re-routing the defective TSVs. In Fig. 6c each group has two redundant and two regular TSVs, if a defective TSV is found in the group, redundant TSVs are used to repair the group. The grouping ratio is 2:2, it will be able to repair the group if one or two defective TSVs are found through and re-routing of signals is performed using defect free TSVs.

The optimal grouping and arrangement among redundant TSVs and regular ones has an effect on the cost in terms of hardware overhead (redundant TSVs and multiplexers) is presented Fig. 7. For illustration four redundant TSVs and eight regular TSVs are in two grouping ratios, as shown in Fig. 7 [4]. Organization 1: Grouping ratio is 2:1. Four groups are arranged, with one redundant TSV and two regular TSVs in each group. Organization 2: Grouping ratio is 4:2. Two groups are arranged, each with two redundant and four regular TSVs. Organization 1 can tolerate one defective TSV within a group and hence the group can be repaired (case 1.1). In one group

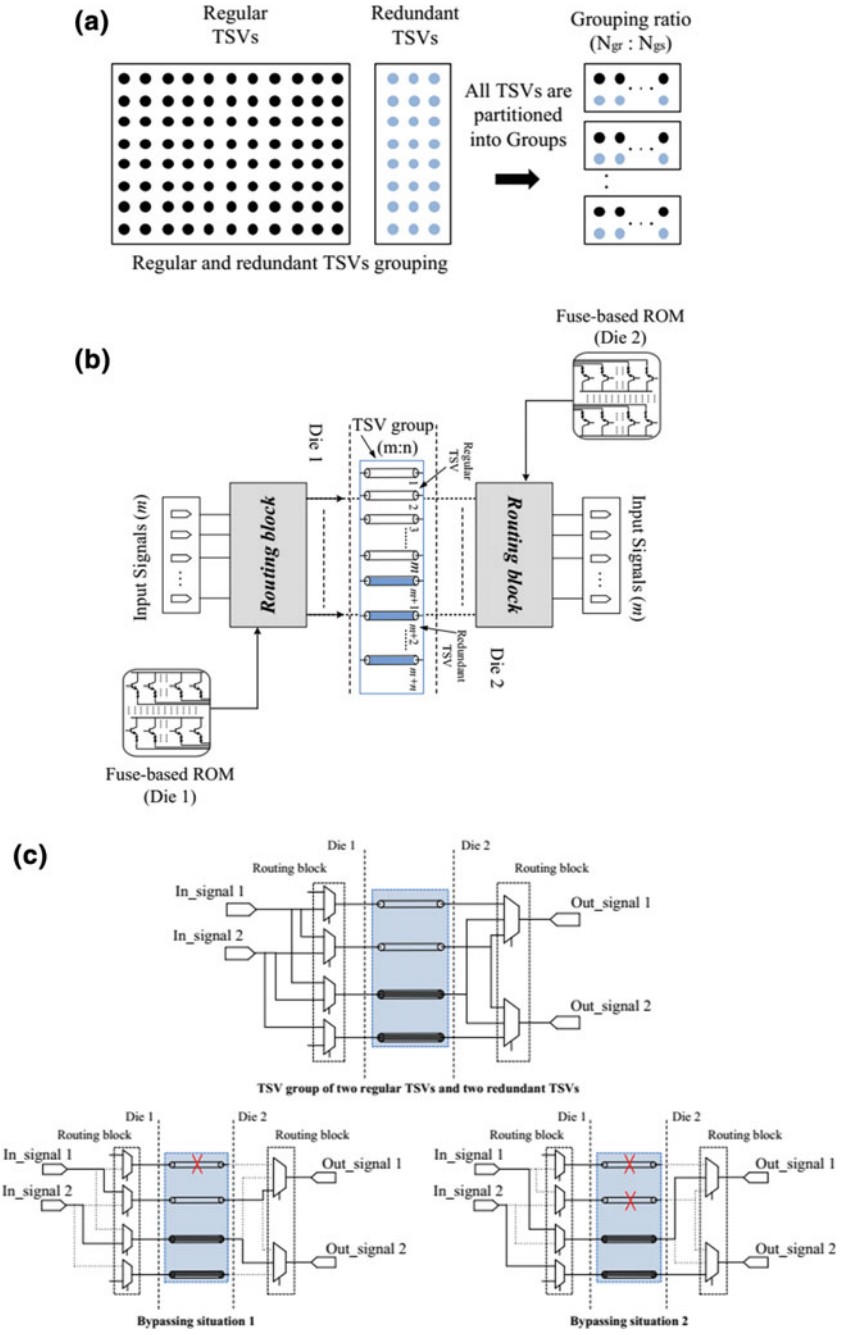


Fig. 6 **a** Grouping of TSVs. **b** TSV groups containing the routing block. **c** Grouping and bypassing to avoid defective TSVs [33]

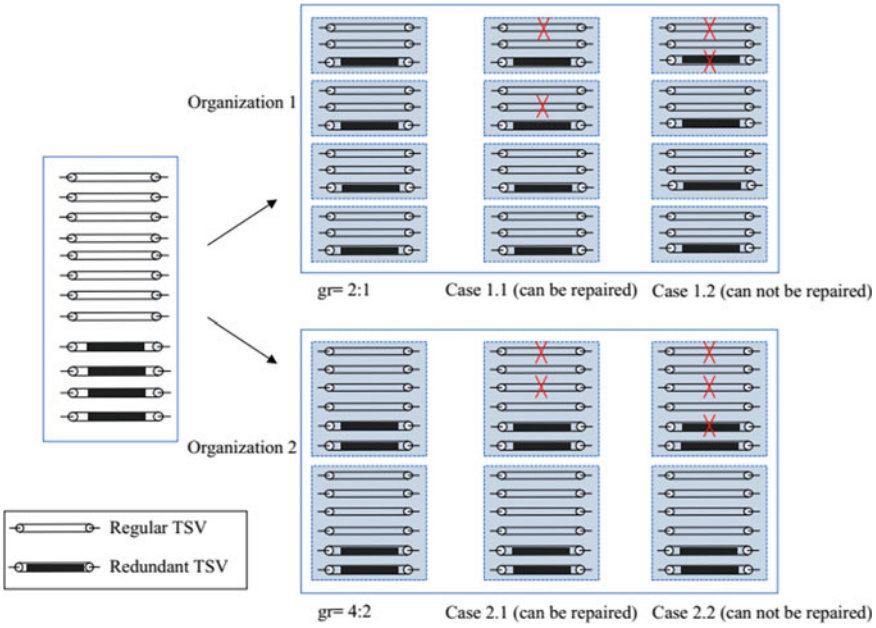
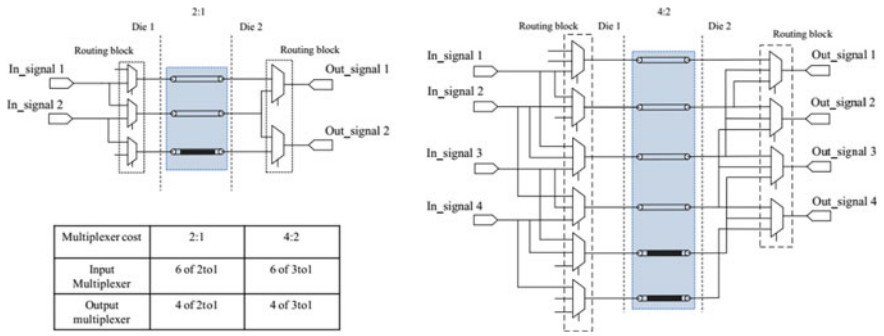


Fig. 7 Grouping of regular and redundant TSV scenarios [4]

consists of two defective TSVs, as in case 1.2, the group cannot be repaired. However, with the same percentage redundancy, organization 2 can repair two defective TSVs at maximum.

Similarly TSVs exist in a group of case 2.1, as shown in organization 1 can be repaired. The optimal grouping incurs extra cost in terms of redundant TSVs and multiplexers. Figure 8 [4] illustrates the various multiplexer used for both grouping ratios 4:2 and 2:1, and concludes the cost of multiplexer in the Table (Fig. 8). The higher yield is obtained in the grouping ratio of 4:2 (Fig. 8), it needs more number of multiplexer and leading to more area overhead. It models the ratio of redundant TSVs and regular TSVs to be grouped in order to accomplish paramount yield with minimal cost in terms of hardware overhead of redundant TSVs and multiplexers. To addresses the placement of timing critical signals, there is a possible way out proposed in Hsieh and Hwang [19], such that it exhibits a timing-aware TSV placement methodology and if signal rerouting is needed due to the defective TSV present in a group and the placement methodology provides the most timing critical signal with minimal delay affect.



Multiplexer cost	2:1	4:2
Input Multiplexer	6 of 2to1	6 of 3to1
Output multiplexer	4 of 2to1	4 of 3to1

Fig. 8 Multiplexer cost for grouping of TSVs [4]

7 Cross Bar

The short vertical interconnects of 3DICs are used in three-dimensional network on chips (3DNoCs). The vertical and horizontal links are supported by 3DNoCs. A cluster of TSVs can be implemented as vertical links. TSVs are compatible with standard CMOS process and allows fine pitch bonding for high density TSV integration. The cross bar technique [20], for defect-tolerant TSV-based multi-bit vertical link empowers noteworthy yield enhancement even in the presence of defects at very low cost. Similar to the conventional fault tolerance techniques, cross bar technique is based on redundancy and leading to robust, fault-tolerant and vertical link architecture (for TSVs) to solve problems of low yield.

There are various state-of-the-art techniques to enhance wafer yield designs, the cross bar technique uses hardware redundancy, incorporated at design time, with re-configuration option after post-manufacturing process. The redundancy made available in the form spare pads and in order to reduce the overall hardware complexity it provides re-configurable routing. The TSV interconnect map is provided with dynamic routing option to leverage the post manufacturing configurability. The cost-effectiveness and high yield is achieved due to reconfiguration with minimal number of pads and extra logic. The combination of resources like testing and re-configurability helps in obtaining high yield. This solution makes feasible testing of each interconnect, diagnose defects and identify failed TSV, and finally by re-routing the failed TSVs over to the spare pads the functionality is restored. Figure 9a [20] shows the dynamic routing technique, all pads are driven by a 2×1 crossbar, and each signal can be routed to two different TSVs. The crossbar is very cost-effective and to very low area overhead as low as possible for each additional rerouting degree of freedom. The defects are overcome by shifting signals to the adjacent pads, and additional shifting of the original connections over to other adjacent. To understand the retrieval scheme, we consider Fig. 9b. Assuming that pad 2 is affected by one of the defects e.g. in an open circuit defect, the routing of signal 3 normally goes through its associated pad 3, while signal 2 is re-routed through pad 1, and hence

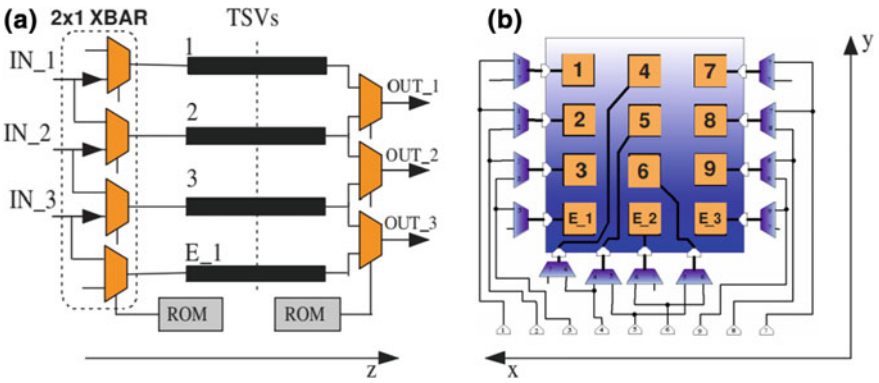


Fig. 9 **a** Crossbar scheme for dynamic routing. **b** Extra pads (E 1 E 2…) are spread nearby the TSV cluster, abridging fault avoiding by means of a multiplexers [20]

signal 1 is re-routed to pad E1. Signals outside this column are not shifted since the defect is present inside the first cluster; the recovery procedure is achieved locally. The appropriate routing information is expounded off-chip in order to minimize complexity of hardware during testing of chip and subsequently stored on-chip into a One Time Programmable (OTP) memory (e.g. a fuse ROM). The prominence of the testing stage is obvious, as it regulates all the required inputs correctly set up the crossbar up. To test the physical interconnect scan chains are reused, which are normally incorporated in the design, thus incurring no overhead.

8 Router Based

The state of the art fault tolerance techniques assume that TSV defects are uniformly distributed. In real time defect clustering of TSVs takes place due to the occurrence of defects depends on the bonding quality and dies that are being stacked. To provide the feasible solution to the defect clustering effect, the router based [21] solution provided TSV repair framework, with a appropriate hardware architecture, such that defect TSVs are re-routed through redundant TSVs which are not the neighboring TSVs. The router based technique enhances stacked 3D-IC yield and it is more effective in the presence of defect clustering effect. Initially a TSV grid is formed to link the TSV pads, switches and wires. Furthermore, redundant TSVs are also connected to two borders of the grid, as shown in Fig. 10. If one of the TSV is defective (the one with “X” mark in Fig. 10), the signal is re-routed through neighboring defect-free TSV (see the solid line in Fig. 10). The defect free TSV is used by the defective TSV for re-routing.

Hence, the original signal for defect free TSV needs to be re-routed. The re-routing option will be available till the redundant TSV available on the borders. TSVs are

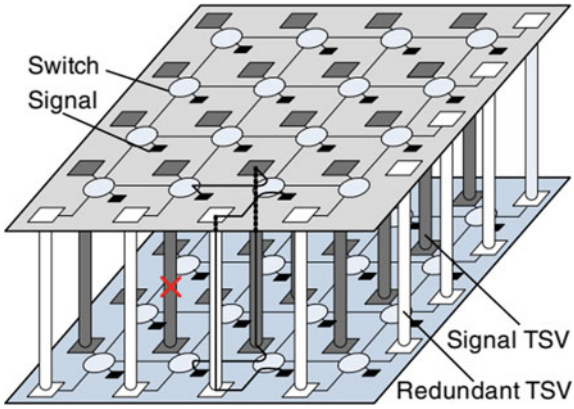


Fig. 10 Router based TSV redundancy architecture [21]

normally placed uniformly for better bonding of TSVs and grouped 3D-IC designs [22, 23], these regularly-placed TSVs are naturally grouped together to build the router based architecture. In case that TSVs are not regularly placed, we can also map them into a logical TSV grid and apply router based repair architecture. Compared to the previous techniques, more hardware area overhead occurs in this architecture (i.e., additional switches and wires) compared to the previous fault tolerance techniques.

8.1 Designing of Switch and Re-routing Paths

The number of redundant TSVs in the design determine the switch design. The redundant TSVs are positioned in the east and south edges of the grid as shown in Fig. 10. The placement of redundant TSVs in such way constrains directional re-routing from north to south or from west to east. Figure 11a shows the graphic of the switch design. The corresponding TSV and its original signal have two ports in the switch named as signal port and TSV port. In addition, to connect in four different directions, there are four ports to connect to other switches far apart. The design standard is that the signal port and two linking ports (North and West) have a mux capable of linking to the TSV port and the residual linking ports (East and south). In Fig. 11b [21] a 4×4 grid for is shown for better understanding of repair paths and re-routing capability.

Originally the mux of signal port connects to TSV port. The signal port re-routes through defect free TSV once the defective TSV is found. The connection between TSV and signal is represented as a repair channel as shown in Fig. 11b with solid arrow. The repair channels exists continuously starting from a defective TSV and terminate at redundant TSV. The virtual connection between defective TSV and redundant TSV is considered as repair path. For example, the clustered defects on the top and bottom is shown in Fig. 11b, and the respective repair paths with the help

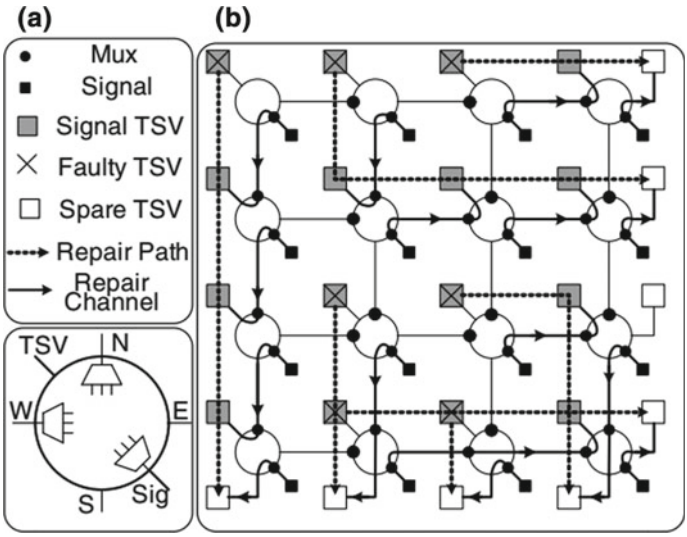


Fig. 11 **a** Switch design. **b** TSV defect cluster with repair paths [21]

of redundant TSVs find disjointed repair paths as shown in Fig. 11b with dashed arrow.

8.2 Router Based with Processor

The various kinds of redundant TSVs allocation techniques and their repair algorithms are discussed and the other alternate solution is built-in self-repair (BISR) with redundant TSVs. The earlier discussed fault tolerance solutions are effective for killer defects that occur at time $t=0$ and these solutions would be ineffective on-field repair [24]. The latent defects that are not sorted out results in killer defects $t>0$ and hence in-effective for on field repair. The deterministic repair algorithms for re-routing the defective does not meet the timing constraints and becoming less effective with circuit aging. The processor based on-field repair solution for TSV based stacked 3D-ICs.

The lifetime reliability of router based fault tolerance technique is enhanced with the judicious use of the redundant TSVs. The problem of latent defects is addressed by conducting on field test and diagnosis of TSVs. The testing and diagnosis requires a processor core and non-volatile memory as shown in shown in Fig. 12. The 3D logic-on-logic/memory are designs with multiprocessor system-on-a-chip (MPSoC) designs and hence heterogeneous integration is possible. The test data, diagnosis patterns for defective are stored in the non-volatile memory for in-field repair, online test and repair, activated intermittently or by events. Due to the clustering effects of

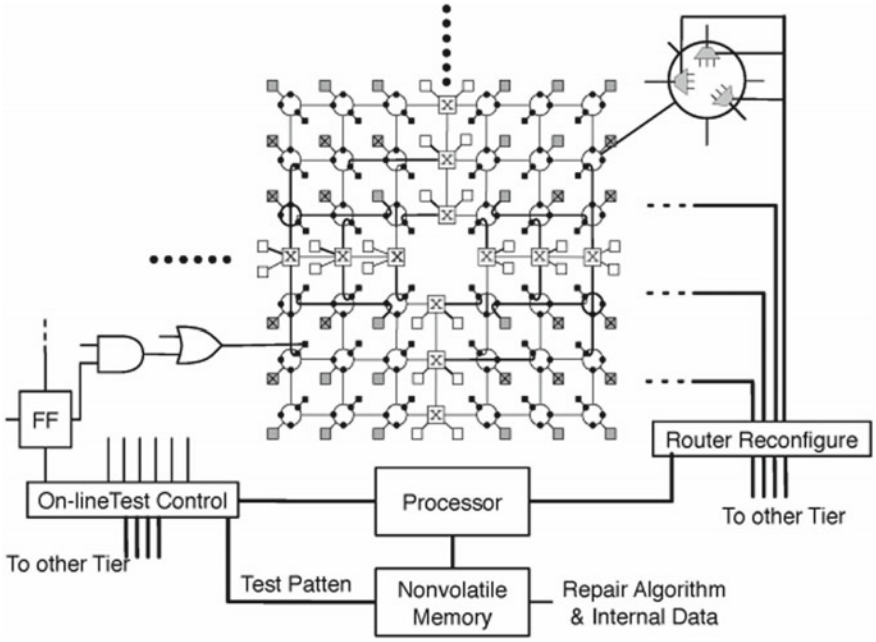


Fig. 12 Router frame work with processor [24]

latent faults, unless the redundancy ratio is quite high, the situation may lead to that some faulty TSV grids lack spare TSVs while the others have many redundant TSVs. This scenario can be resolved by letting spare TSV sharing between TSV grids, as shown in Fig. 12 [24].

9 Online Fault Tolerance Technique

The online fault tolerance technique for testing and repairing of TSV group is shown in Fig. 13 [25]. It consists of: routing block, recovery block and detection block and used for test and repairing of group of TSVs. If a group is denoted by m: n grouping ratio it implies that there are "m" number of regular TSVs and "n" number of redundant TSVs. Hence, each group can tolerate "n" number of defective TSVs. As well as the yield and repair capability of the group are reliant on number of redundant TSVs, but at the cost of area overhead. The optimal grouping ratio is done at the design time for reduction of area overhead. The testing of each TSV is done with the help of detection block. Delay test is employed in the detection block for differentiating the defective and defect-free TSV. Mainly tested for short-to-substrate, delamination and void kind of defects. The test result from the detection block is updated in the status register which are present on the both dies. The functionality of the recovery block is

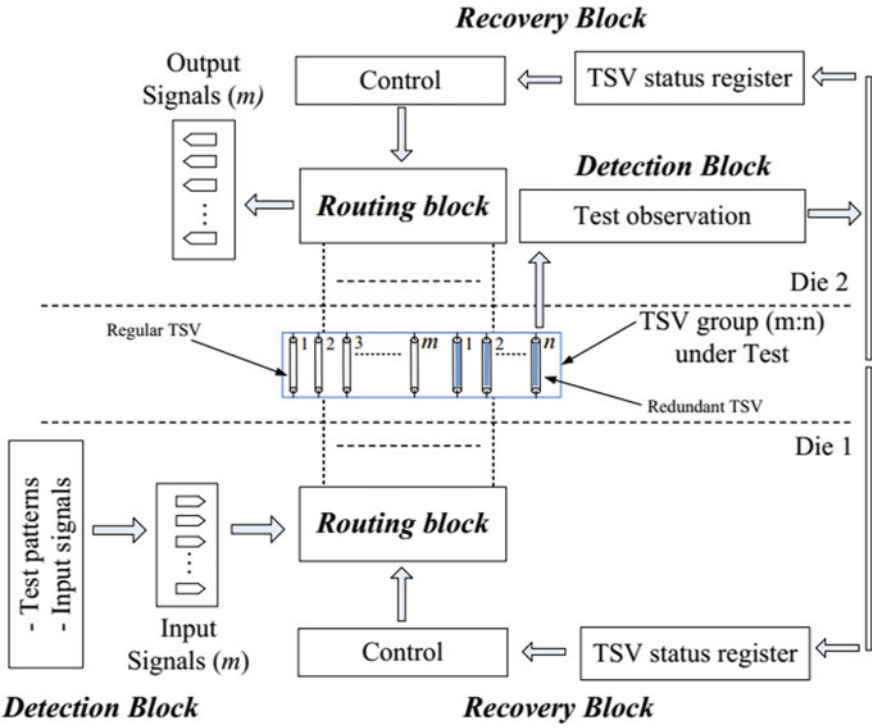


Fig. 13 Architecture of online fault tolerance technique [25]

re-route through defect free TSV in presence of defect free TSV and present on both dies and shown in Fig. 13. As well as it comprises of control unit and status register. The status of TSV is stored in the status register where '0' represents defect-free and '1' represents defective TSV. The required control signals for re-routing through defect free TSV and configuration of routing block are provided by the control block. The routing block comprises of de-multiplexers and multiplexers to connect TSV and its respective signal line. The required selection lines for de-multiplexers and multiplexers are delivered by the control unit of the recovery block.

Figure 14 shows the in detail insights of online fault tolerance technique and for explanation a grouping ratio of 4:2 is taken with four regular and two redundant TSVs and hence it can only work if it has two defective TSVs. The testing of each TSV is done with the help of test input and test observation block and TSV status registers stores test result on both dies. Once the defective TSV is found, the re-routing process is initiated by the routing block and re-configuration of signal path takes place. The connection for output (Die2) and input (Die1) signals are similar, the difference is output signals uses multiplexers and input signals uses de-multiplexers. For a grouping ratio of 4:2, each signal has three possible TSVs that is 1-to-3 de-multiplexer. The control is also used to report when the number of defective TSVs is

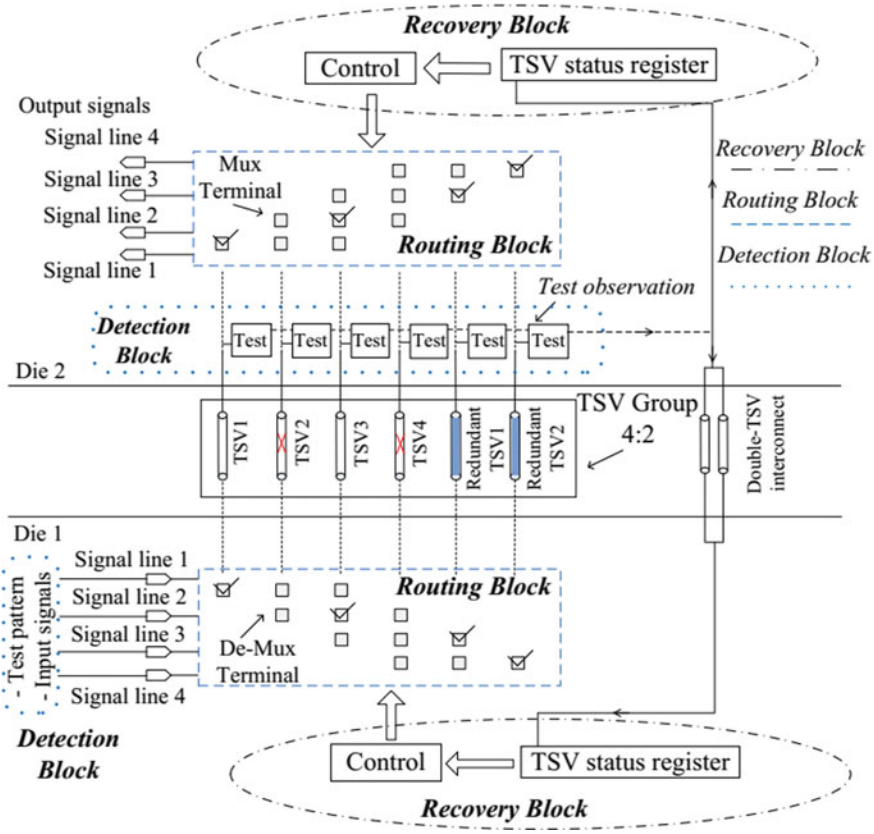


Fig. 14 Recovery and routing blocks with 4:2 grouping ratio [4]

higher than the maximum tolerance limit of a TSV group. For indicating the number of defective TSVs in a group the control unit provides the required information. To illustrate the working of the recovery block a 4:2 grouping ratio undertaken with two defective TSVs (TSV2 and TSV4) in the group (Fig. 14).

Die2 and Die1 consists of configuration circuit. The connection rules are followed by the configuration circuit. A signal line is assigned to the TSV as shown in Fig. 14 with a right tick in the connection box and no other signal will be allocated to that TSV. This is because one signal line can only be engaged to one TSV. If a TSV is defective, all connection boxes (Mux or de-Mux terminals as shown Fig. 14) that correspond to that TSV cannot be used. According to the test results and connection rules stored in the status register. Based on the linking rules and test results stored in the TSV status register, the accessibility of a TSV is determined. The first signal line is linked, than moves to the following signal line and so on until all input signals are linked to a TSV.

10 Time Division Multiplexing Access Based Fault Tolerance Technique

The fault tolerance technique based on TDMA is shown in Fig. 15 [26]. It consists of three modules—Routing, Testing, and TDMA. These modules primarily performs two tasks: (1) finding all the TSVs which are defective; (2) signal are re-routed through defect-free TSVs. If a design consists of "*m*" number of TSVs and based on the state of the art these can be alienated at the design time into groups. Routing, Testing, and TDMA modules are built based on the number of TSVs per group and shown in Fig. 15. For illustration, four TSVs are shown in Fig. 15. TDMA makes likely communication of numerous liberated messages without mutual meddling of each other [27]. For example, in Fig. 15, input signal 1 (TSV1) uses time slot 1 and input signal 2 (TSV2) uses time slot 2 until the last input signal. It recurrences in a cyclical modus, still all the input signals are completed and that slot becomes free or allocated to another input signals. For each TSV in the design TDMA segment assigns time slot and generates the control signals necessary for inter-die communication and tests the individual TSVs for defects. The TDMA module consists of demultiplexers, multiplexers, counter and oscillator. The inputs to the architecture are *Enable* (*En*), signal lines and *Testmode* for the respective TSVs. The oscillator generates clock signals on die1 when *En* is active high, and the clock signals are received by the counter for producing the required selection signal for demux6, demux1, mux2, mux1 on die1, and demux1 on die2. To select particular input signal lines mux1 on die1 is 4:1 is used and counter provides necessary control signals. The system works in normal mode or *Testmode* (Fig. 15) based on the 2:1 type Mux2 on die1. *Testmode* is the external signal and used as the selection line for mux2 (Fig. 15). The output of mux2 is one of the input signal lines out of all available if the *Testmode* = 0, else the output is *Testmodebar*. Input signal lines are passed through Demux1 of die1 and *Testmodebar* to the Routing module as per the selection lines. Likewise, demux6 on die1 and demux1 on die2 are 1:4 type which obtains *Testresult* (Fig. 15) as input from the comparator, selection lines from the TDMA module, and the output of these demultiplexers are selection lines for the demuxes in the Routing modules on die2 and die1, correspondingly. The algorithm for TDMA fault tolerance technique is shown in Fig. 16.

10.1 Finding Defective TSVs

The module for testing present in Fig. 15 tests each TSVs in the group. A pull-down network on die1 and pull-up network on die2 are used for testing. The test results is obtained at the output of flip-flop and updated in the status register of TSV on die1 as shown in Figs. 18 and 19. Test results are subsequently passed to the Routing module through double TSV interconnect on die1 and die2. The propagation of signals like *Testresult* and selection lines for demux1 of die2 are passed from die1

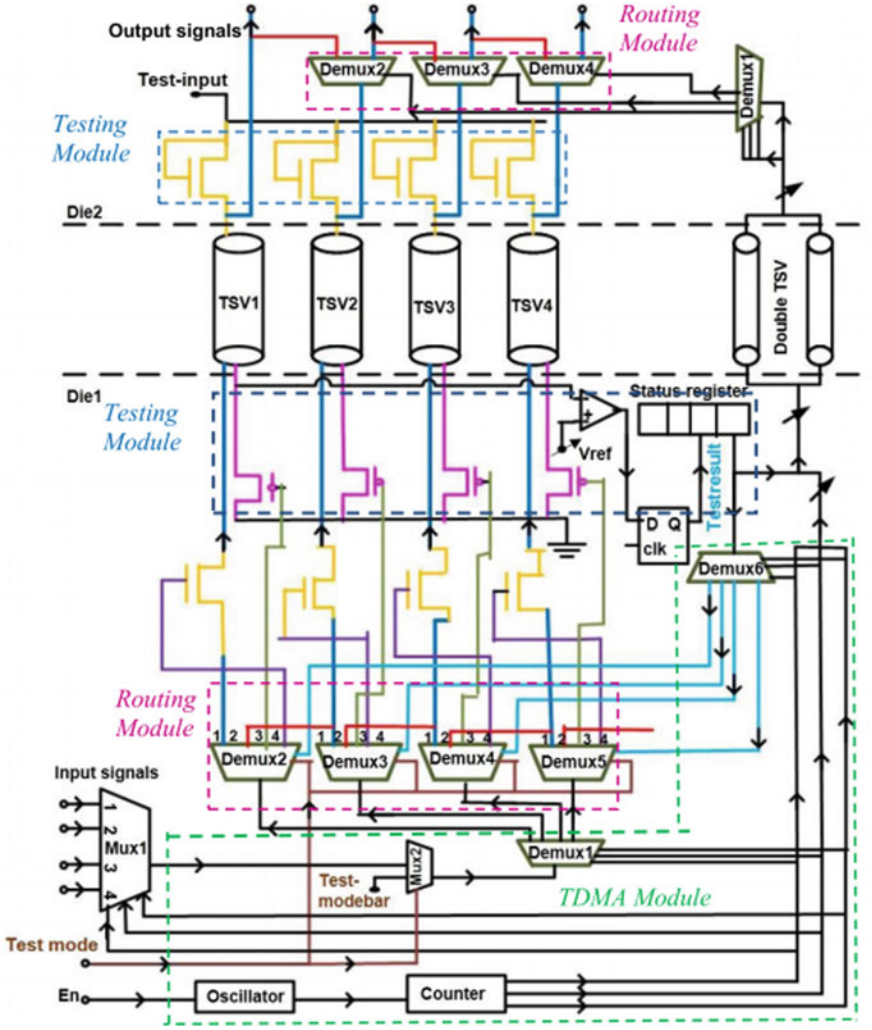


Fig. 15 Architecture of TDMA based fault tolerance technique [26]

through a double TSV interconnect. The delay test is used by the testing module to differentiate between defective and defect-free TSVs and the TSVs are tested for short-to-substrate and void/delamination defect types. If the TSV is non-defective, it passes through the normal path as shown in first row of Table 1 and Fig. 15 and if it defective, signal is bypassed through the defect-free TSV as shown in second row of Table 1 and Fig. 15.

A TSV is modeled as transmission line T-model [28], as shown in Fig. 17a [28], where *R*tsv and *C*tsv are TSV resistance and capacitance. The corresponding electrical model for delamination/void defect is shown in Fig. 17b [29], *R*open denotes

Algorithm Proposed Fault Tolerance Technique (PFTT)
Input: Number of Regular TSV 'm'
Output: PFTT with *TDMA*, *Testing* and *Routing* modules

1 Grouping of Regular TSV with 'n' TSV per group
2 Building of *TDMA*, *Testing* and *Routing* modules
% *Testing of TSVs for fault detection*
3 **While** *Testmode* = 1 **do**
4 **for** I = 1 to n **do**
5 Switch ON PMOS and test TSV
6 Compare V_{ref} with V_{TSV}
7 *Testresult* is passed to the Routing Module
% *Re-routing of signal through defective-free TSVs*
8 **If** (*Testresult* = 1)
9 Cut-off the signaling path of defective TSV (NMOS off)
10 Re-route the signal
11 **end if**
12 **end for**
13 **end while**
% *Normal mode of operation*
14 **If** (*Testmode* = 0 and *Testresult* = 0) // Normal mode
15 Pass input signals from die1 to output signals on die2
16 **end if**
17 Return Proposed fault tolerance technique

Fig. 16 Algorithm for TDMA FTT [26]

Table 1 Output lines of de-mux in routing module

Testmode	Testresult	Output functionality	Symbol
0	0	Normal path to TSV	▲
0	1	Re-routing Path	●
1	0	For switching On P-MOS	▰
1	1	For switching Off N-MOS	◆

the open circuit defect and upsurges delay of signal. The short-to substrate defect is a resistive pathway between TSV and substrate is shown in Fig. 17c [30]. The capacitance *C*tsv and resistance *R*tsv are taken as 200 fF and 200 milliohms respectively [28]. The TSV fabrication procedure disturb the signal transmission properties and stacking of dies leads to worsening of chip speed or envisioned functionality. Therefore, the at-speed features of the system are confirmed with the assistance of delay test. The delay across a defect free TSV is given by time constant (RC), which

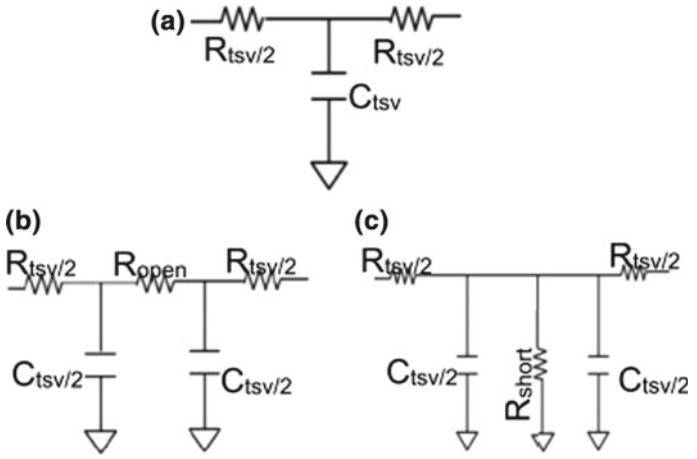


Fig. 17 Corresponding models of circuit for TSV. **a** T-model fot TSV. **b** Delamination/void defect. **c** Short to substrate defect [26]

is very low and the presence of defects increases the RC value, which in turn disrupts the timing constraints and leading to failure of the system.

The Testing module for single TSV is shown in Fig. 18. Here, pulldown network is acting as a resistive load by pMOS, which is ON during testing and pull-up network acting as a resistive circuit by nMOS. The change in resistance due to defects in TSVs is measured with voltage divider and Vtsv the output voltage is function of on-resistance of pMOS transistor (R *pull-down*), on-resistance of nMOS transistor (R *pull-up*), the capacitance (Ctsv) and the resistance of TSV (Rtsv). To differentiate delamination and void defect, is shown in Fig. 18 for each TSV is substituted by corresponding open circuit defect as shown in Fig. 17b. Likewise, to detect short-to-substrate defect, TSV in Fig. 18 is swapped with corresponding short circuit defect as shown in Fig. 17c. The testing of TSV is done by applying *Testmode* high to turn ON the die1 pMOS transistor (third row in Table 1 and Fig. 18) and as per the above discussion, all the TSVs can be tested by TDMA and testing module without surge in the peak current usage and does not aggravate the thermal stress and reliability degradation. The defective TSV is identified by comparing Vref with Vtsv of the defective TSV. To exemplify reference voltage, Vref is set to 50% of Vdd and it can also be carefully chosen based on the TSV delay (Rtsv$C$tsv) analogous to the alteration in TSV resistance (Rtsv) [31]. The delay in the TSV can be identified by changing the reference voltage and timing constraints of the design. The output of comparator is high if $V\text{tsv} < V\text{ref}$ implies TSV is defective and hence delay is larger than the expected delay and cut-off the signal path through defective TSV (fourth row in Table 1 and Fig. 15). The comparator output is low if $V\text{tsv} > V\text{ref}$ implies defect-free TSV.

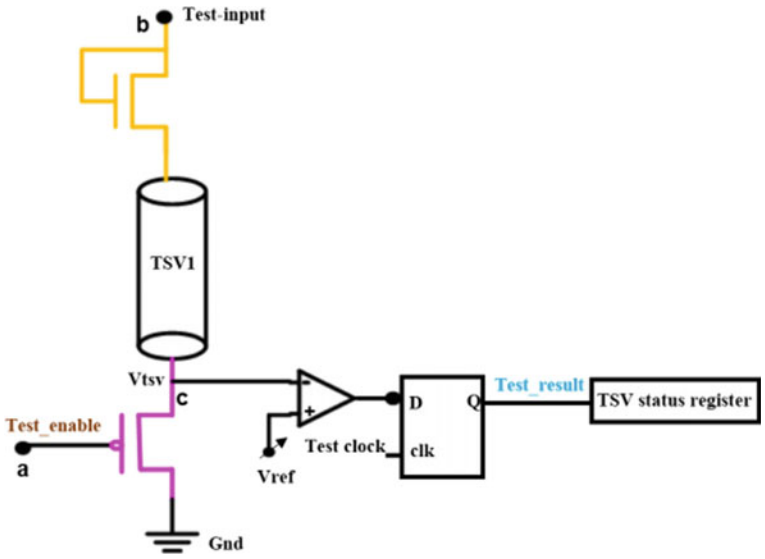


Fig. 18 Testing of TSV [26]

10.2 Re-route Through Defect-Free TSV

The module for routing shown in Fig. 15 shows the routing module present on die2 and die1, it consists of de-multiplexers to reroute the signals through defect-free TSVs. The Routing module is configured by signals from the modules of TDMA and Routing module. Die1 routing module contains of four 1:4 type demultiplexers that obtain the input signals from the demux1 of the TDMA module (Fig. 15), selection lines from the *Testmode* and *Testresult* (Fig. 15) of the comparator, and the functionality of output signals is explained in Table 1. *Testmode* is the input signal used for testing of TSVs, when *Testmode* is high it specifies die is under test, else die is under normal operation. If *Testresult* is high, implies TSV is defective and *Testresult* low, implies TSV is defect-free. The Routing module of die2 consists of 1:2 type demultiplexers that accept the input signals from the TSVs and selection lines are delivered by demux1 of die2.

10.3 Illustration of TDMA Based Fault Tolerance Technique

Figure 19 illustrates re-routing through single defect-free TSV. In Fig. 19 TSV1 is taken as defective. TSV1 is to pass input signal1 from die1 to die2 as per the *Testmode, Testresult*, and control signals obtained from the module of TDMA. If control line and *Testmode* are low, input signal1 from mux1 is passed to demux1 via mux2 as

shown in Fig. 19. From demux1, it passes to demux2 and as per the selection lines it sends to the output signal1 of die2 via TSV1 or TSV2. If *Testresult* and *Testmode* are low, output1 of demux2 passes to TSV1 and obtained on die2 output signal1. If *Testmode* is low and *Testresult* is high, output2 of demux2 is passed to TSV2. On die2, the signal of TSV2 is passed to demux and further demux on die2 passes the output signal1 to its original TSV1 path via TSV2 as per the selection line acquired from double TSV. To shift over from defective to defect-free of a single TSV necessitates two demultiplexers each on die1 and die2, control lines produced from the TDMA module and *Testresult*.

Redundant and regular through-silicon via (TSV) are used in fault tolerance techniques of 3-D IC. However, the fabrication method of TSVs results in defects that decrease the reliability and yield of TSVs. In contrast, each TSV is accompanying with a substantial amount of on-chip area overhead. Hence, unlike the preceding discussed fault tolerance architectures, the time division multiplexing access (TDMA)-based fault tolerance technique does not use any redundant TSVs, decreases the area overhead and improves the yield.

11 Conclusion

Resilience aware system design is essential for CMOS and beyond CMOS technologies to perform intended functionality despite the complexities and disturbances in the underlying hardware. Reliability issues to specific to 3D-ICs are closely associated to TSVs. Thermal and stress issues in 3D-IC are the major source of cause for failure of TSVs. The various killer and latent defects in TSVs developed during the fabrication and on filed operation results in degradation of lifetime reliability, reduce the yield and failure of 3D-IC. The underlying latent defects in TSVs can result in killer defects due to aging of TSVs can result in failure of TSVs. Design-time approach to address these issues in 3D-IC involves inclusion of redundant TSVs along with regular functional TSV to enable online re-configuration or re-mapping to overcome the defective TSVs, as a result sophisticated redundant TSV architectures are introduced at design time. The number of TSVs available for a design is limited due its constraints of Keep-out-zone from neighboring TSVs, transistors and size of TSVs (in micrometer range). Hence, TDMA based fault tolerance technique is the alternative feasible solution based on the application requirement. Therefore, we have thrown light on various fault tolerance techniques to address the defective TSVs, leading to synergy between reliability, yield and cost-effectiveness of resilient 3D-IC designs.

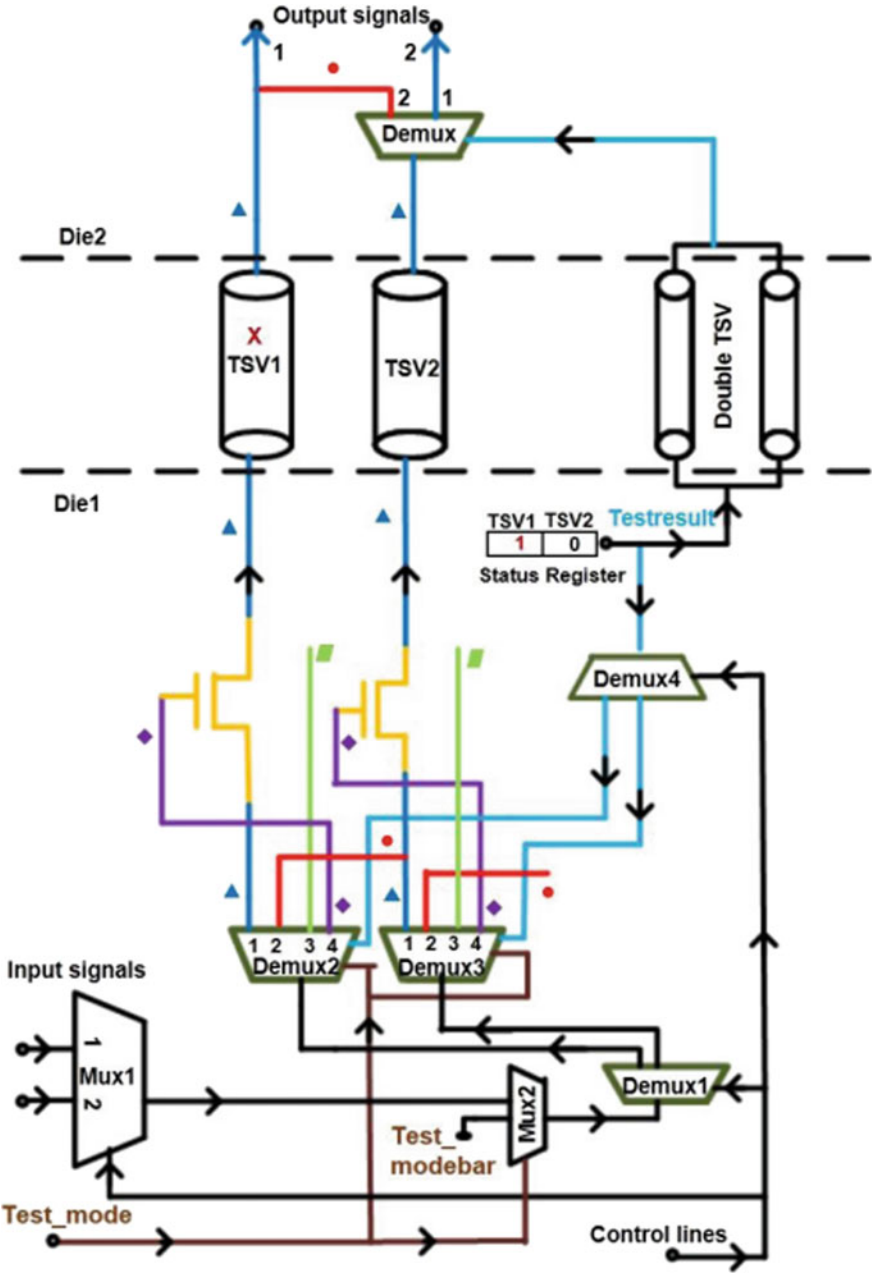


Fig. 19 Re-routing of signal for single defective TSV [26]

References

1. Banerjee, K., Souri, S.J., Kapur, P., Saraswat, K.C.: 3-D ICs: a novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration. Proc. IEEE **89**(5), 602–633 (2001)
2. Liu, C.C., Ganusov, I., Burtscher, M., Tiwari, S.: Bridging the processor-memory performance gap with 3D IC technology. IEEE Des. Test Comput. **22**(6), 556–564 (2005)
3. Frank, T., Chappaz, C., Leduc, P., Arnaud, L., Moreau, S., Thuaire, A., El Farhane, R., Anghel, L.: Reliability approach of high density through silicon via (TSV). In: 2010 12th Electronics Packaging Technology Conference (EPTC), December 2010, pp. 321–324. IEEE (2010)
4. Zhao, Y.: Investigation into Yield and Reliability Enhancement of TSV-Based Three-Dimensional Integration Circuits. Doctoral Dissertation, University of Southampton (2014)
5. Athikulwongse, K., Yang, J.S., Pan, D.Z., Lim, S.K.: Impact of mechanical stress on the full chip timing for through-silicon-via-based 3-D ICs. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(6), 905–917 (2013)
6. Liu, X., Chen, Q., Dixit, P., Chatterjee, R., Tummala, R.R., Sitaraman, S.K.: Failure mechanisms and optimum design for electroplated copper through-silicon vias (TSV). In: 59th Electronic Components and Technology Conference, 2009. ECTC 2009, pp. 624–629. IEEE (2009)
7. Jung, M., Mitra, J., Pan, D.Z., Lim, S.K.: TSV stress-aware full-chip mechanical reliability analysis and optimization for 3D IC. Commun. ACM **57**(1), 107–115 (2014)
8. Lu, K.H., Zhang, X., Ryu, S.K., Im, J., Huang, R. Ho, P.S.: Thermo-mechanical reliability of 3-D ICs containing through silicon vias. In: 59th Proceedings of Electronic Components and Technology Conference. ECTC 2009, pp. 630–634. IEEE (2009)
9. Lu, K.H., Ryu, S.K., Zhao, Q., Zhang, X., Im, J., Huang, R., Ho, P.S.: Thermal stress induced delamination of through silicon vias in 3-D interconnects. In: 2010 Proceedings of 60th Electronic Components and Technology Conference (ECTC), pp. 40–45. IEEE (2010)
10. Ryu, S.K., Lu, K.H., Zhang, X., Im, J.H., Ho, P.S., Huang, R.: Impact of near-surface thermal stresses on interfacial reliability of through-silicon vias for 3-D interconnects. IEEE Trans. Device Mater. Reliab. **11**(1), 35–43 (2011)
11. Kao, C.R., Wu, A.T., Tu, K.N., Lai, Y.S.: Reliability of micro-interconnects in 3D IC packages. Microelectron. Reliab. **53**(1), 1 (2013)
12. Ko, C.T., Chen, K.N.: Reliability of key technologies in 3D integration. Microelectron. Reliab. **53**(1), 7–16 (2013)
13. Tu, K.N.: Reliability challenges in 3D IC packaging technology. Microelectron. Reliab. **51**(3), 517–523 (2011)
14. Chakrabarty, K., Deutsch, S., Thapliyal, H., Ye, F.: TSV defects and TSV-induced circuit failures: The third dimension in test and design-for-test. In: 2012 IEEE International Conference on Reliability Physics Symposium (IRPS), pp. 5F–1. IEEE (2012)
15. Lin, Y.M., Zhan, C.J., Juang, J.Y., Lau, J.H., Chen, T.H., Lo, R., Kao, M., Tian, T., Tu, K.N.: Electromigration in Ni/Sn intermetallic micro bump joint for 3D IC chip stacking. In: 2011 IEEE 61st Electronic Components and Technology Conference (ECTC), pp. 351–357. IEEE (2011)
16. Frank, T., Moreau, S., Chappaz, C., Leduc, P., Arnaud, L., Thuaire, A., Chery, E., Lorut, F., Anghel, L., Poupon, G.: Reliability of TSV interconnects: electromigration, thermal cycling, and impact on above metal level dielectric. Microelectron. Reliab. **53**(1), 17–29 (2013)
17. Tan, Y.C., Tan, C.M., Zhang, X.W., Chai, T.C., Yu, D.Q.: Electromigration performance of through silicon Via (TSV)—a modeling approach. Microelectron. Reliab. **50**(9), 1336–1340 (2010)
18. Pak, J., Pathak, M., Lim, S.K., Pan, D.Z.: Modeling of electromigration in through-silicon-via based 3D IC. In: 2011 IEEE 61st Proceedings of Electronic Components and Technology Conference (ECTC), pp. 1420–1427. IEEE (2011)
19. Hsieh, A.C., Hwang, T.: TSV redundancy: architecture and design issues in 3-D IC. IEEE Trans. Very Large Scale Integr. VLSI Syst. **20**(4), 711–722 (2012)

20. Loi, I., Mitra, S., Lee, T.H., Fujita, S. Benini, L.: A low-overhead fault tolerance scheme for TSV-based 3D network on chip links. In: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, November 2008, pp. 598–602. IEEE Press (2008)
21. Jiang, L., Xu, Q., Eklow, B.: On effective TSV repair for 3D-stacked ICs. In: 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), March 2012, pp. 793–798. IEEE (2012)
22. Kawano, M., Uchiyama, S., Egawa, Y., Takahashi, N., Kurita, Y., Soejima, K., Komuro, M., Matsui, S., Shibata, K., Yamada, J., Ishino, M.: A 3D packaging technology for 4 Gbit stacked DRAM with 3 Gbps data transfer. In: IEDM'06 International on Electron Devices Meeting, 2006, pp. 1–4. IEEE (2006)
23. Zhang, T., Wang, K., Feng, Y., Song, X., Duan, L., Xie, Y., Cheng, X., Lin, Y.L.: A customized design of DRAM controller for on-chip 3D DRAM stacking. In: 2010 Custom Integrated Circuits Conference (CICC), pp. 1–4. IEEE (2010)
24. Jiang, L., Ye, F., Xu, Q., Chakrabarty, K., Eklow, B.: May. On effective and efficient in-field TSV repair for stacked 3D ICs. In: 2013 50th ACM/EDAC/IEEE on Design Automation Conference (DAC), pp. 1–6. IEEE (2013)
25. Zhao, Y., Khursheed, S., Al-Hashimi, B.M.: Online fault tolerance technique for TSV-based 3-D-IC. IEEE Trans. Very Large Scale Integr. VLSI Syst. **23**(8), 1567–1571 (2015)
26. Reddy, R.P., Acharyya, A., Khursheed, S.: A cost-effective fault tolerance technique for functional TSV in 3-D ICs. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. (2017)
27. Haykin, S.: Communication Systems. Wiley, New York, NY, USA (2008)
28. Katti, G., Stucchi, M., De Meyer, K., Dehaene, W.: Electrical modeling and characterization of through silicon via for three-dimensional ICs. IEEE Trans. Electron Devices **57**(1), 256–262 (2010)
29. Ye, F., Chakrabarty, K.: TSV open defects in 3D integrated circuits: characterization, test, and optimal spare allocation. In: 2012 Proceedings of the 49th Annual Design Automation Conference, pp. 1024–1030. ACM (2012)
30. Cho, M., Liu, C., Kim, D.H., Lim, S.K., Mukhopadhyay, S.: Design method and test structure to characterize and repair TSV defect induced signal degradation in 3D system. In: Proceedings of the International Conference on Computer-Aided Design, November 2011, pp. 694–697. IEEE Press (2010)
31. Sung, H., Cho, K., Yoon, K., Kang, S.: A delay test architecture for TSV with resistive open defects in 3-D stacked memories. IEEE Trans. Very Large Scale Integr. VLSI Syst. **22**(11), 2380–2387 (2014)
32. Lu, T., Serafy, C., Yang, Z., Samal, S., Lim, S.K., Srivastava, A.: TSV-based 3D ICs: design methods and tools. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (2017)
33. Zhao, Y., Khursheed, S., Al-Hashimi, B.M.: Cost-effective TSV grouping for yield improvement of 3D-ICs. In: 2011 20th Asian Test Symposium (ATS), pp. 201–206. IEEE (2011)

Formal Verification for Security in IoT Devices

K. Keerthi, Indrani Roy, Aritra Hazra and Chester Rebeiro

Abstract Online detection of cyber-attacks on IoT devices is extremely difficult due to the limited battery and computational power available in these devices. An alternate approach is to shrink the attack surface in order to reduce the threat of attack. This would require that the device undergo more stringent security tests before deployment. Formal verification is a promising tool that can be used to not only detect potential vulnerabilities but also provide guarantees of security. This chapter reviews several security issues that plague IoT devices such as functional correctness of implementations, programming bugs, side-channel analysis, and hardware Trojans. In each of these cases, we discuss state-of-the-art mechanisms that use formal verification tools to detect the vulnerability much before the device is deployed.

1 Introduction

The number of connected IoT devices has crossed 20 billion and expected to increase at a rate of 15% per year. IoT devices are typically battery operated, have low computing power, and less memory. Many of the devices have either no or tiny operating systems that have limited functionalities. Operating system functionalities are typically restricted to resource management for efficient energy utilization. Security

K. Keerthi · I. Roy · C. Rebeiro
Indian Institute of Technology Madras, Chennai, India
e-mail: keerthi@cse.iitm.ac.in

I. Roy
e-mail: indrroy@cse.iitm.ac.in

C. Rebeiro
e-mail: chester@cse.iitm.ac.in

A. Hazra (✉)
Indian Institute of Technology Kharagpur, Kharagpur, India
e-mail: aritrah@cse.iitkgp.ernet.in

R. S. Chakraborty et al. (eds.), *Security and Fault Tolerance in Internet of Things*,
Internet of Things, https://doi.org/10.1007/978-3-030-02807-7_9

features such as secure boot, trusted execution, and even memory protection are generally absent. These limiting features in IoT devices makes them vulnerable to a variety of cyber-attacks.

The attack surface for IoT devices is considerably large. An attacker may use one or more of the following attack vectors to compromise a device.

- First, there can be a weakness in the functionality of a device. For example, an operation that was intended to be present but is either absent in the implementation or not fully complete. For example, there may be an absence of strong authentication methods or an absence of meta-level encryption procedures.
- Even if the intended security operations are implemented in the device, there may be flaws in the implementation. This is especially a problem with cryptographic algorithms. The security guarantees of cryptographic algorithms are well studied and investigated. The weakest link is generally not due to the mathematical underpinnings of the algorithms but rather their implementations. The huge state space present in cryptographic implementations, makes detecting these flaws difficult.
- Programming bugs such as buffer overflows, arithmetic overflow and underflows, and format-string vulnerabilities present in the implementations, can be used to craft malicious payloads that can subvert execution leading to security breaches.
- Trojans present in the hardware or software can provide unauthorized access to the system or lead to information leakage. These Trojans are introduced due to multiple third parties involved in the design and manufacture of IoT devices. They are very difficult to detect but provide an easy vector for an attacker.
- IoT devices are also vulnerable to multiple physical attacks such as fault injection attacks, differential power analysis, and timing attacks. These attacks would require the attacker to have physical access to the device, disturb the device operations by injection of a fault, or passively monitoring the device's side-channels such as its power and energy consumption, electro-magnetic radiation, and execution time to glean secret information.

This huge attack surface is a serious concern especially because many of the IoT devices are used in cyber-physical systems such as process-control, smart-grid, and medicals systems. A compromised device in any of these critical infrastructures could lead to considerable losses.

Due to the constrained resources in an IoT device, detecting cyber-attacks is only possible from outside the device. Two potential directions to detect malware outside the device is by monitoring the network for malware signatures or by using side-channels such as the device's power consumption, execution time, and electromagnetic radiation to identify patterns that would indicate an attack. Both techniques face considerable challenges. For example, current side-channel analysis techniques requires a measurement setup and therefore only possible in a laboratory environment. Network monitoring techniques on the other hand will not be effective for attacks that do not transmit much over the network or those that camouflage their network activities. Furthermore both techniques will not be able to detect zero-day exploits.

An alternate approach to achieving security in IoT devices is to prevent rather than detect. This would require that the various attack vectors described above are eliminated at design time there by reducing the attack surface. Traditionally this is done by good design practices, use of secure coding techniques, static analysis, and extensive testing. However, these methodologies cannot provide guarantees of security; stronger mechanisms would therefore be required. One promising direction is the use of formal verification to provide security guarantees. Formal verification is a technique to mathematically ascertain the correctness of designs using a diverse set of mathematical and logical methods. Model checking [17] and theorem proving [23] procedures are often used to ensure the accuracy of implementations. In model checking, a model of a system is exhaustively and automatically verified with respect to a given specification; whereas in theorem proving, the system characteristics are derived mathematically and solved using automated reasoning techniques to infer the correctness of the system.

The main drawback of using a theorem prover is that the user has to explicitly provide the design and specification characteristics as algebraic constraints or theorems. Model checkers, on the other hand, can act automatically over the implementation with the given specifications and formulate SAT clauses from the design behavior. It is, therefore, more effective and easier to use a model checker as compared to a theorem prover for validating large implementations, such as crypto-designs.

In this chapter, we shall discuss various applications of formal verification in order to improve an IoT device's security. We would use formal verification to (a) prove the correctness of implementations with respect to its formal specification. As a case study, we would consider the correctness verification of a multi-precision library used for public-key cryptographic algorithms such as ECC and RSA. (b) We would then prove the absence of programming vulnerabilities, such as buffer overflows, in the multi-precision library. Multi-precision library implementations are especially interesting for formal verification due to their critical usage with security sensitive aspects in the device, and their extremely complex design space where comprehensive testing becomes practically impossible.

The chapter would also discuss other state-of-the-art research in the use of formal verification for device security. In particular, (c) the chapter discusses the use of formal verification to validate physical attack countermeasures. While applying countermeasures for physical attacks such as side-channel analysis is easy, proving their effectiveness is considerably more challenging. Formal verification would help considerably to achieve these security proofs. (d) Another application of formal verification is to detect the presence of hardware Trojans in designs. We discuss a recent work which demonstrates the use of model checking tools to identify a Trojan that leaks sensitive information from the device. (e) Finally we discuss the use of formal verification to ensure completeness of security goals such as meta-level authentication and encryption.

The organization of this chapter is as follows: Sect. 2 provides the background about symbolic model checking covering SAT and BDD based model checking. The section also describes a model checking tool called CBMC for C based bounded model checking [18]. Section 3 describes the use of model checking to verify that

an implementation is correct with respect to its formal specification. A case study of a multi-precision library used for ECC and RSA is verified. Section 4 describes program vulnerabilities and presents the use of CBMC as a tool to detect such vulnerabilities. Section 5 presents the use of formal verification for side-channel countermeasures, while Sect. 6 describes the use of formal verification for hardware Trojan detection. Section 7 shows how formal verification can be used to identify meta-level authentication issues, while the final section has the conclusion.

2 Background: Symbolic Model Checking

The term *symbolic model checking* is popularly interpreted as BDD-based model checking, however any model checking technique that works on a symbolic representation of the implementation can be called *symbolic* model checking. There are mainly two kinds of symbolic methodology found in the literature; namely *BDD-based model checking* and *SAT-based model checking*.

2.1 BDD-Based Model Checking

Binary Decision Diagrams (BDDs) [8] are compact canonical representations of Boolean functions. BDDs utilize self-similarity in the decision trees based on Shanon's expansion to give a more compact representation. Ken McMillan first proposed model checking algorithms using BDD in his famous doctoral thesis [24]. The use of BDDs in model checking was instrumental in bringing the technology into practice. Experimental results showed that the BDD-based approaches were able to handle 10^{20} states and beyond [9]—which was unthinkable with algorithms that work on explicit representations of the state space.

To perform BDD-based model checking [14], a BDD representation, $\mathcal{Z}$, for the temporal property $\neg\varphi$ is created (when, φ be the temporal formula of interest). Then, the product of $\mathcal{Z}$ with the BDD for the transition relation of the design-under-test (DUT) is computed. Let the BDD for the product be $\mathcal{P}$. In the final step of the model checking, the strategy is to check whether $\mathcal{P}$ is empty, that is, whether the product has any fair path [10, 15].

2.2 SAT-Based Model Checking

SAT is the traditional short form for the Boolean satisfiability problem. Given a Boolean formula f, the problem is to determine whether f is satisfiable, that is, whether there exists any valuation of the variables in f, under which f evaluates to $True$. Verification methods based on the SAT problem have recently emerged as

a promising solution. Dramatic improvements in SAT solver technology over the past decade have led to the development of several powerful SAT solver tools [21, 28, 29, 31]. Verification methods based on these tools have been shown to push the boundaries of functional verification in terms of both capacity and efficiency, as reported in several academic and industrial case studies [2, 3, 7]. This has fueled further interest and intense research activity in the area of SAT-based FPV.

Bounded Model Checking (BMC) [16] based on SAT methods was introduced by Biere et al. in [5, 6, 16] and is rapidly gaining popularity today as a complementary technique to the existing BDD-based model checking. Given a temporal logic property, φ, to be verified on a finite state transition system $\mathcal{M}$, the essential idea is to search for counter-examples to φ in the space of all executions of $\mathcal{M}$ whose length is bounded by some finite integer k.

The problem is formulated by constructing the following propositional formula:

$$f^k = I \wedge \bigwedge_{i=0}^{k-1} R(S_i, S_{i+1}) \wedge (\neg\varphi^k)$$

where I is the characteristic function for the set of initial states of $\mathcal{M}$, $R(S_i, S_{i+1})$ is the characteristic function of the transition relation, relating the variables in S_i with those in S_{i+1}, of $\mathcal{M}$ for time step i. Thus, the formula $\left(I \wedge \bigwedge_{i=0}^{k-1} R(S_i, S_{i+1})\right)$ precisely represents the set of all executions of $\mathcal{M}$ of length k or less, starting with an initial state. $\neg\varphi^k$ is a formula representing the condition that φ is violated by a bounded execution of $\mathcal{M}$ of length k or less. Hence, f^k is satisfiable iff there exists an execution of $\mathcal{M}$ of length k or less that violates φ. f^k is typically converted to conjunctive normal form (CNF) and is solved by a conventional SAT solver.

Due to the success of SAT solvers in bounded model checking, there has been growing interest in their use for unbounded model checking. Few of the recent works in this direction can be found in [22, 25].

2.3 CBMC: The Formal Verification Tool

There are various academic as well as industrial tools available for formal verification of the system. In this work, we will evaluate the security attributes by symbolic model checking approach with the help of tool named **CBMC** which is a C based model checker [18]. CBMC takes two inputs: (i) the program to be verified (written in C or C++), and (ii) the formal specification, as shown in Fig. 1. The loops in the program ($\mathtt{p}$) are first unwound and then a Boolean model $\mathtt{Q}(\mathtt{p})$ is obtained, which is checked for satisfiability with the negation of the specification ($\neg\mathtt{q}$) using an in built SAT solver. If the model is satisfied, which means that the negation of the specification is satisfied, then the verification fails. CBMC reports this failure with a counter-example. If the model is not satisfiable, then for all the possible combinations of the

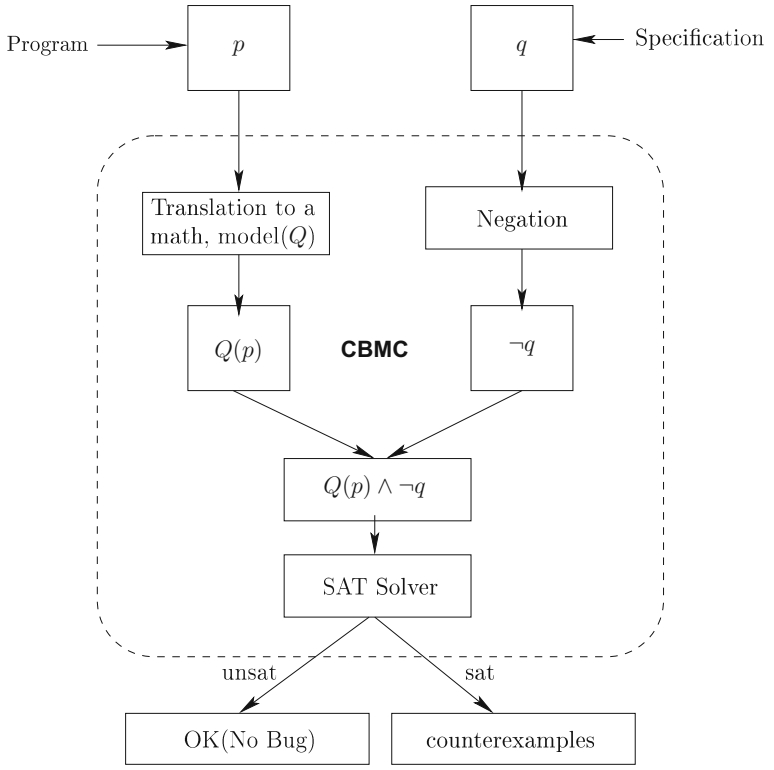


Fig. 1 Flow of a C program verification using CBMC, showing a SAT solver that checks the correctness of program p with respect to the specification q. The counter example generated gives the program inputs for which specification q fails [13]

input, the specification conditions are correct, therefore the program is verified to be correct based on the given specifications.

3 Correctness of Crypto Implementations

An important security requirement for any program is to ensure its correctness with respect to a specification. This becomes even more critical when the program in consideration is an implementation of a cryptographic algorithm. An error in a crypto-implementation, could be exploited to leak secrets such as the cryptographic key. Extensive testing is the time-honored way of checking the correctness of a crypto implementation. This however cannot provide guarantees because of the huge state space of crypto-implementations. For example, the state space of a typical implementation of an elliptic curve cryptographic scheme is in the order of 2^{256}. This is too

huge a space to exhaustively test. A more complete technique is to use formal verification, which generates stronger guarantees of correctness based on a *functional specification* of the algorithm.

There have been several efforts to prove the functional correctness of crypto-implementations using formal verification. For example, ciphers like AES, MARS, Twofish, RC6, Serpent, IDEA, and TEA were considered in [19]; several hash functions and block ciphers in [4, 32]. For public-key ciphers, functional correctness has been discussed in [1, 12].

In this section we discuss the formal verification of a multi-precision library. This library forms the base over which public-key algorithm like ECC and RSA are built.

3.1 Correctness of a Multi-precision Library

A multi-precision library for public-key cryptography includes implementations of finite-field operations such as addition, subtraction, multiplication etc. These implementations could have many flaws, most of which can be detected by formal verification.

The multi-precision library we consider is written in 'C'. We use the model checking tool CBMC (ANSI-C Bounded Model Checking) [18] as a tool for verification. The approach we follow is to use a hierarchical verification technique to handle the scalability issues and the huge state space of the library.

Implementation Aspects and Notations. A central data-structure used to define multi-precision numbers is the `bignum_t` structure, which comprises of an array called `digits` used to store the multi-precision number and `sign` to store its sign.

```
typedef struct{
    word digits[MAXDIGITS];
    int  sign;
}bignum_t;
```

The multi-precision elements is represented using the macro `MAXDIGITS`—based on the number of digits in the multi-precision number. It depends on the `word` length of the processor executing the multi-precision library. For instance to represent a field element in $\mathbb{F}_{256}$ on a micro-controller platform with a 16 bit word size, `MAXDIGITS` would be defined as $\lceil 256/16 \rceil = 16$.

The finite field elements are represented using the above structure as follows: $A = (a_{15}, a_{11}, a_{10}, \ldots, a_2, a_1, a_0)$, where a_i $(15 \leq i \leq 0)$ are word sized numbers stored in `digits` in the `bignum_t` structure, a_{15} is the most significant digit, while a_0 is the least significant digit. Multi-precision operations are denoted in capitals with a '$*$' on top. For example $A \overset{*}{+} B$ represents multi-precision addition, while the operations over the digits in the multi-precision number are in the standard representation, such

as $a_i + b_i$ for digit addition. In the remainder of this section we demonstrate several cases of formal verification of multi-precision operations.

Case 1 (Multi-precision Addition): Let A and B be two multi-precision numbers with n digits. To perform multi-precision addition ($S = A \overset{*}{+} B$), we add the digits of A and B as shown in Eq. 1.

$$\begin{aligned} \mathrm{c}_{-1} &= 0 \\ (\mathrm{c}_i, s_i) &= a_i + b_i + \mathrm{c}_{i-1} \qquad (0 \leq i \leq n-1) \ , \end{aligned} \tag{1}$$

where s_i holds the sum of digits and c_i the carry of each digit addition. The output is stored in $(\mathrm{c}_{n-1} || \, S)$.

Verification of Multi-precision Addition. In Eq. 1, we first verify that the carry and sum of each digit (i.e. c_i, s_i) is correct. Each digit is of 8, 16, or 32 bits depending on the execution platform. To perform the verification, we provide conditions to CBMC (in the form of assertion statements) that will evaluate to *true* if the addition is correct and *false* otherwise. The condition to verify the addition of the digits of A and B considering the carry that occurs from one digit to the next, i.e. verifying $(\mathrm{c}_i, s_i) = (a_i + b_i + \mathrm{c}_{i-1})$ is

$$((\mathrm{c}_i, s_i) - (b_i + \mathrm{c}_{i-1}) = a_i) \qquad 0 \leq i \leq n-1; \qquad \mathrm{c}_{-1} = 0 \ . \tag{2}$$

CBMC will perform the multi-precision addition $A \overset{*}{+} B$ and verify digit-by-digit checking exhaustively over the possible valuations of A and B, for the above condition to be satisfied. If a failure is obtained, it means that the specification failed for some values of A and B. CBMC will return the values which caused the failure. This counter-example is the proof that CBMC provides of a verification failure.

Listing 1.1 Implementation of multi-precision addition

```
void BN_uadd (bignum_t *S, bignum_t A, bignum_t B)
      {
         int i, j;
         word c = 0;
         S->sign = 1;      /* sign of result forced to positive */
         for( i = 0; i < MAXDIGITS; i++){
            S->digits[i] = A.digits[i] + c;
            c = ( S->digits[i] <  c );
            S->digits[i] = S->digits[i] + B.digits[i];
            c = c + ( S->digits[i] < B.digits[i] );
         }
         while( c!=0 ){
            S->digits[i] = S->digits[i] + c;
            c = (S->digits[i] < c);
            i = i + 1;
         }
      }
```

Listing 1.2 Specification for multi-precision addition

```
int BN_uadd_specification(bignum_t S, bignum_t A, bignum_t B)
    {
       int j;
       word z, c = 0;
       for(j = 0; j < MAXDIGITS; j++) {
          z = (S.digits[j] < c);
          S.digits[j] = S.digits[j] - c;
          c =  (S.digits[j] < B.digits[j]) + z;
          S.digits[j] = S.digits[j] - B.digits[j];
          if (S.digits[j] != A.digits[j]) return (false);
       }
       return (true);
    }
```

Listing 1.3 Verification of multi-precision addition

```
void BN_uadd_verify ()
     {
        bignum_t S,A,B;
        int nondet_int();
        A.sign = nondet_int();
        B.sign = nondet_int();
        __CPROVER_assume(A.sign==1 && B.sign==1);

        BN_uadd(&S,A,B); /* Compute S = A + B */
        assert(BN_uadd_specification(S,A,B));
     }
```

Listing 1.1 gives the implementation of multi-precision addition shown in Eq. 1. Functional correctness of the implementation is done by digit addition of the result with operand B as given in Eq. 2. The implementation aspects of the addition specification is shown in Listing 1.2. Implementation `BN_uadd_verify` in Listing 1.3 performs the formal verification, which invokes `BN_uadd` and `BN_uadd_specification`. CBMC performs the following steps for verification:

1. CBMC unwinds all the loops in the program, based on the number of iterations specified using command '`--unwind N`'. Each copy of the loop is replaced by an `if` statement to check the terminating conditions and at the end of `N` copies, an *unwinding assertion* is added by CBMC to avoid further iterations. The unwinding of `BN_uadd_specification` in Listing 1.2 is shown in Step 1 of Fig. 2. We have used an additional variable P to represent the return value of the function, which is initialized to 1 and made 0 if *false* is returned (corresponding to Line 10 of `BN_uadd_specification`). The assertion in the last line of Step 1 corresponds to the assertion in Line 9 of the function `BN_uadd_verify`.
2. The next step is to rename the program variables as shown in Fig. 2 Step 2, which is transformed to SSA (static single assignment). For example $\mathtt{j} = \mathtt{j} + 1$ is converted to $\mathtt{j}_2 = \mathtt{j}_1 + 1$. In Step 2, each `if` block renames the variable P to $P_1, P_2, \ldots, P_n$. Therefore, the assertion statement should check all the n values of P instead of one variable.

Step 1

```
P = 1;
j = 0;
if(j < MAXDIGITS) {
 //body
 z = (S.digits[j] < c);
 S.digits[j] = S.digits[j] - c;
 c = (S.digits[j] < B.digits[j]) + z;
 S.digits[j] = S.digits[j] - B.digits[j];
 if (S.digits[j] != A.digits[j]) P = 0;
 j = j + 1;
 if(j < MAXDIGITS){
    //body
     j = j + 1;
     ...
     if(j < MAXDIGITS){
        //body
        j = j + 1;
        assert(!(j < MAXDIGITS));
      }
     ...
    }
  } assert(P==1);
```

Step 2

```
P1 = 1;
j1 = 0;
if(j1 < MAXDIGITS){
  //body
  z1 = (S1.digits[j1] < c1);
  S2.digits[j1] = S1.digits[j1] - c1;
  c2 = (S2.digits[j1] < B1.digits[j1]) + z1;
  S3.digits[j1] = S2.digits[j1] - B1.digits[j1];
  if (S3.digits[j1] != A1.digits[j1]) P2 = 0;
  j2 = j1 + 1;
  if(j2 < MAXDIGITS){
     //body
     j3 = j2 + 1;
     ...
     if(jn−1 < MAXDIGITS){
        //body
        jn = jn−1 + 1;
        assert(!(jn < MAXDIGITS));
      }
     ...
  }
} assert(P2==1 && P3==1 && ... && Pn==1);
```

Step 3

$\mathcal{C}$:= $(P_1 = 1)$
$\wedge\ (j_1 = 0)$
$\wedge\ z_1 = ((j_1 < \text{MAXDIGITS}) \wedge (S_1.\text{digits}[j_1] < c_1))\ ?\ 1 : 0$
$\wedge\ S_2.\text{digits}[j_1] = (j_1 < \text{MAXDIGITS})\ ?\ S_1.\text{digits}[j_1] - c_1 : S_1.\text{digits}[j_1]$
$\wedge\ c_2 = ((j_1 < \text{MAXDIGITS}) \wedge (S_2.\text{digits}[j_1] < B_1.\text{digits}[j_1]))\ ?\ (1+z_1) : z_1$
$\wedge\ S_3.\text{digits}[j_1] = (j_1 < \text{MAXDIGITS})?(S_2.\text{digits}[j_1] - B_1.\text{digits}[j_1]) : S_1.\text{digits}[j_1]$
$\wedge\ P_2 = ((j_1 < \text{MAXDIGITS}) \wedge (S_3.\text{digits}[j_1] \neq A_1.\text{digits}[j_1]))\ ?\ 0 : P_1$
$\wedge\ j_2 = (j_1 < \text{MAXDIGITS})\ ?\ (j_1 + 1) : j_1$
$\wedge\ \ldots$
$\mathcal{P}$:= $(P_2 == 1 \ \wedge\ P_3 == 1 \ \wedge\ \ldots\ P_n == 1)$

Fig. 2 Given the function `BN_uadd_specification`, Step 1 shows how CBMC unwinds the loop and the Step 2 shows the renaming of the variables in function body. Step 2 is converted to Boolean formula, with set of constraints $\mathcal{C}$ and properties $\mathcal{P}$ is given in Step 3

3. In the third step, CBMC uses set of rules to convert SSA statements are converted to Boolean formula [18]. The rules define a set of constraints ($\mathcal{C}$) and set of properties ($\mathcal{P}$). These are shown in Step 3 of Fig. 2.
4. The SAT solver in CBMC, try to solve $\mathcal{C} \wedge \neg\mathcal{P}$ and returns counter-example if solution found, which means that verification is failed; otherwise it returns verification successful

Case 2 (Multi-precision Subtraction): Let A and B be two multi-precision numbers of n digits each. To perform multi-precision subtraction ($D = A \overset{*}{-} B$), we subtract the digits of A and B as shown in Eq. 3.

$$\begin{aligned} \mathrm{b}_{r_{-1}} &= 0 \\ d_i &= (\mathrm{b}_{r_i}, (a_i - \mathrm{b}_{r_{i-1}})) - b_i \qquad (0 \leq i \leq n-1)\ , \end{aligned} \tag{3}$$

where d_i holds the difference between digits and b_{r_i} holds the borrow of each individual digits. The output of the result is stored in **d**.

Verifying Multi-precision Subtraction. To verify subtraction, we assume that multi-precision addition has already been verified and proven to be correct. Thus, verifying $D = A \overset{*}{-} B$ is simply done using $A = D \overset{*}{+} B$. In the specification function for multi-precision subtraction, we invoke multi-precision addition (Eq. 1) with operands D and B. The result is verified to be equal to the original value of A using an `assert` statement.

Case 3 (Multi-precision Left-Rotation): The multi-precision left-rotate takes a multi-precision number A having b bits and an integer m. It shifts A left by m bits and the bits that fall off at the most significant end are inserted in the least significant end. This is represented as $A_r = A \overset{*}{\lll} m$.

Verifying Multi-precision Left-Rotation. We verify multi-precision left-rotate, by checking the bit position before and after rotation. I.e., the specification given to CBMC checks that m most significant bits of A is shifted to corresponding least significant positions in A_r. The remaining $(b - m)$ bits of A are left shifted by m bits in A_r.

For example: let $A = (a_{15}, a_{14}, a_{13}, \ldots, a_2, a_1, a_0)$, be a multi-precision number on a 16-bit platform and a_i $(11 \leq i \leq 0)$ be 16 bit words in A. $(A \lll m)$ where $m = 16$, will result in $A_r = (a_{14}, a_{13}, a_{12}, \ldots, a_1, a_0, a_{15})$. Verification is done by checking whether all the bits are shifted exactly m bits to lower position and also the most significant m bits are shifted to least significant m bits.

3.2 Verifying an ECC Implementation

Elliptic curve cryptography implementations have a pyramid-like structure as shown in Fig. 3. The operations in the base of the pyramid are multi-precision algorithms like addition, subtraction, multiplication, and inversion; all done in the underlying finite field of the Elliptic curve. The verification of the point addition and doubling functions is thus carried out to ensure that the correct invocation of all the lower level finite field functions.

The verification of ECC scalar multiplication is done in a similar manner, since the algorithm invokes the point addition and point doubling functions. Given that point addition and point doubling have already been verified to be correct, the scalar multiplication involves the number of times point addition and point doubling is computed based on the scalar.

Results. This section contains the results of the verification of our Elliptic curve crypto-library. Implementation of four out of the five NIST specified Elliptic curves over $\mathbb{F}_{192}$, $\mathbb{F}_{224}$, $\mathbb{F}_{256}$, and $\mathbb{F}_{384}$ [34] were verified. CBMC Version 5.7 [18] on an Ubuntu 14.04 Linux machine on a quad core Intel i5-3340 CPU @ 3.10 GHz was

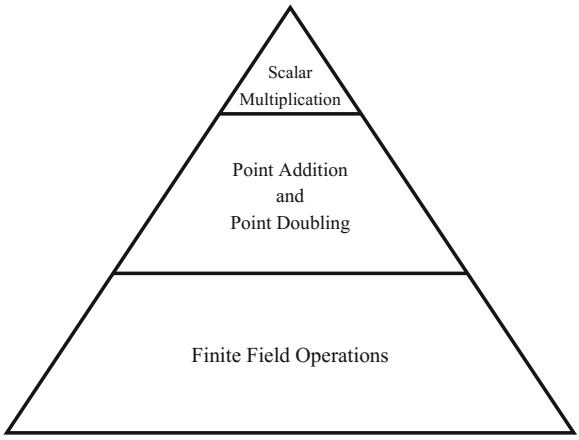


Fig. 3 Pyramid-like structure of an ECC implementation, where the bottom layer finite field operations include multi-precision addition, multiplication etc.

Table 1 Time taken (in hours) by CBMC to verify scalar multiplication operation of 4 different curves, assuming a 16 bit word size

Function		Description	Time (h)
Scalar multiplication	192-bit	$\mathbb{F}_{192}$	23
	224-bit	$\mathbb{F}_{224}$	34
	256-bit	$\mathbb{F}_{256}$	64
	354-bit	$\mathbb{F}_{384}$	183

used as the platform for the study. In all cases, we assume a 16 bit word size that forms a digit used to partially represent a multi-precision number. Table 1 shows the verification time for the four different NIST specified curves. Figure 4 shows the verification time and execution time for 4 NIST specified Elliptic curves $\mathbb{F}_{192}$, $\mathbb{F}_{224}$, $\mathbb{F}_{256}$ and $\mathbb{F}_{384}$ [34] and also the increase in verification time of Karatsuba multiplication as field size increases.

3.3 Verifying an RSA Implementation

RSA also has a similar pyramidal structure like ECC but with just two levels. The base comprises of multi-precision operations just as in ECC, albeit with much larger numbers. The upper level comprises of modular exponentiation, which is used for encryption and decryption. Hence it is clear that the verification is done using the assume-guaranteed hierarchical verification technique, where all the finite field operations are verified and found to be correct.

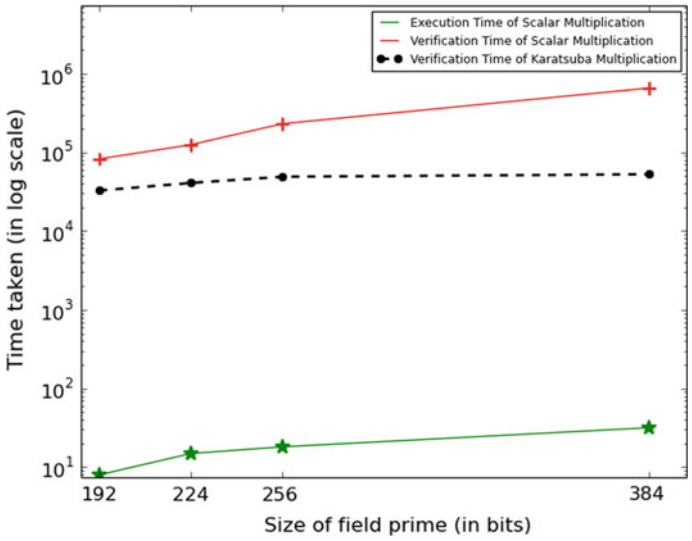


Fig. 4 Time taken in log scale for execution and verification of scalar multiplication for the 4 NIST Elliptic curves $\mathbb{F}_{192}$, $\mathbb{F}_{224}$, $\mathbb{F}_{256}$, and $\mathbb{F}_{384}$

4 Program Vulnerability Detection

Even though programs may be verified to be correct, minor bugs in the code can be exploited to subvert the implementation and execute malicious payloads. To detect such bugs, which we call programming vulnerabilities, we need to analyze the implementation. Formal verification tools such as CBMC [18] can be used for this purpose.

The first step is to classify bugs according to vulnerability numbers present in the CVE database [26]. This database lists publicly known vulnerabilities and exposures. The CWE database classifies lists according to various known weaknesses [27]. The sub-category CWE-310, View-658 lists the various weaknesses of software written in the C programming language. Another sub-category View-702, lists potential weaknesses introduced during implementation. To detect program vulnerabilities, we assume that these databases would help identify a majority of the vulnerabilities in the implementation. We begin this section by highlighting the various programming vulnerabilities, before discussing the detection details. The CVE database [26] reports 8690 buffer overflow (including arithmetic overflow) vulnerabilities. This can be mapped to 82 CWE weaknesses [27], of which we found that 42 are applicable to a multi-precision library. The model-checking tool CBMC [18], can verify the array bounds, pointer safety, and integer arithmetic etc. This covers 83% of the classified bugs in the databases.

Buffer Overflow. Buffer overflow is a serious security concern, which has been studied for several years. A buffer is a continuous chunk of memory, such as an array, and is associated with a pointer. Buffer overflows occur when the pointer is accessed beyond the bounds. An attacker may use buffer overflows to write into an illegal memory that could then be used to subvert execution. Programming languages such as *C* and *C++* are prone to buffer overflows, where there are no built-in bound check conditions. These overflows, such as in the example given below, are typically not detected at compile time.

```
int buffer[10];
int i = 10;
buffer [i+1] = 20;
```

Buffer overflows have different variants such as—(1) Heap overflows: where the overflow occur in dynamically allocated memory; (2) Stack overflows: exploits overflow the stack based buffer, which can change the local variables in the program, return address, and even function pointers.

Consider the buffer overflow example above, CBMC will detect the buffer overflow in the program by inserting proper bound check conditions for each array access. The output from CBMC tool is as follows:

Result from CBMC

```
State <S> file <name.c> line <L> function main
------------------------------------------------
  i=10 (00000000000000000000000000001010)

Violated property:
  file <name.c> line <L> function main
  array 'buffer' upper bound
  (signed long int)(1 + i) < 10l

VERIFICATION FAILED
```

CBMC performs buffer overflow checks by inserting conditions in the source code that would validate a pointer access is legal. It would then check whether the conditions are reachable or not. As an example, conditions such as

$$(\texttt{i} >= 0) \text{ and } \neg(\texttt{i} >= \texttt{MAXDIGITS} - 1)$$

These conditions automatically determine if an array of size `MAXDIGITS`, accessed with index `i`, violates the lower or upper bound properties (i.e., whenever `a[i]` occurs in the program).

Integer Overflow. An integer overflow could occur in programs that deal with arithmetic operations, where a computation result exceeds the range of the representation. Integer overflows can also occur due to improper type conversion, which can affect the security of the program and lead to unintended behavior of the program.

```
int i = -2147483648;
i = i - 1;
```

The example given above is an integer overflow, where the value of `i` has the lowest value in its representation and the subtraction changes its value to `2147483647`.

To detect potential integer overflows, CBMC automatically inserts specific properties for each variable as given below.

$$(\texttt{i} < \texttt{INT\_MIN}) \,||\, (\texttt{i} > \texttt{INT\_MAX})$$

This determines whether the defined variable `i` with data-type (in our example `int`) violates the underflow or overflow property. CBMC determine whether the property is reachable for each access of these variables. For the integer overflow program given above, the verification result returned by CBMC is as follows:

Result from CBMC

```
State <S> file <name.c> line <L> function main
------------------------------------------------
i=-2147483648 (10000000000000000000000000000000)

Violated property:
  file <name.c> line <L> function main
  arithmetic overflow on signed -
  !overflow("-", signed int, i, 1)

VERIFICATION FAILED
```

4.1 Program Vulnerability Detection in a Multi-precision Library

Direct verification for buffer overflow and integer overflow need not be effective for a crypto implementation due to its large mathematical model. For detecting all possible vulnerabilities, we have used a hierarchical verification. The method verify all the lower level implementation for vulnerabilities. These verified implementation can be used for verification of higher level implementations. Hierarchical verification will help to determine the flow of these vulnerabilities in the program and also help to speed up the verification.

Table 2 shows the verification time for vulnerability detection in a multi-precision library.

Table 2 Program vulnerability detection in a multi-precision library

Function	Description	Time (s)	Number of vulnerabilities detected
Copy	$x \leftarrow y$	0.002	1
Addition	$r = x + y$	0.333	3
Subtraction	$r = x - y$	0.398	3
Compare	$x == y$	0.628	0
Karatsuba multiplication	$r = x \times y$	53.46	16
Left-shift	$r = x \gg 1$	0.004	0
Right-shift	$r = x \ll n$	0.006	2

5 Formal Verification of Side Channel Countermeasures

IoT devices that are physically accessible to an attacker are vulnerable to side-channel attacks. These attacks detect sensitive information flowing through unintended covert channels such as the device's power consumption. For example, if the device performs an operation, E, such as $y \leftarrow E(x, k)$, with a secret key k and plaintext bits x, then information about k is leaked through the device's power consumption.

Masking is a popular countermeasure used to prevent this leakage [11]. With this countermeasure, sensitive variables like k are masked so that there is no leakage through the power traces. The mask is then removed at the end of the operation. For example, if E is a linear function, then for a randomly chosen value of mask, r, the operation $E(x, k \oplus r)$ is done instead. No leakage about k is present in the power consumption if r is secret. Moreover, the correct result y is obtained by computing $E(x, k \oplus r) \oplus E(x, r)$. This works because E is a linear function and $E(x, k \oplus r) = E(x, k) \oplus E(x, r)$.

Masking non-linear operations is not so trivial. It is time consuming and error-prone as information can leak through intermediate operations. For example, consider the non-linear operation $(o = x \wedge k \wedge r)$. A value of $o = 1$, which can be distinguished in the power consumption, will leak the value of k, which would also be 1 in this case. Formal verification has been used to verify perfect masking. The property used in formal verification is that every intermediate operation I that is used in the computation of the non-linear function E, should be perfectly masked [20]. We assume that every intermediate operation I is Boolean and has parameters x bits of the plaintext, k secret key, and random mask r. The property fed to the model checker to verify perfect masking is the following:

$$\exists x \, \exists k \, \exists k' \, (\sum_{r} I(x, k, r) \neq \sum_{r} E(x, k', r)) \ .$$

The property verifies that for any input x and a pair of keys k and k' ($k \neq k'$), the probability distribution of $I(x, k, r)$ differs from that of $I(x, k', r)$. If this holds then some information about the secret key k is leaked through the side-channel. In this case, the intermediate operation I is not perfectly masked. If the model checker finds that the above property is not satisfied, it means that no information is leaked. Hence the intermediate operation I is perfectly masked.

Besides power consumption based channels, execution time can also lead to information leakage. Developing constant time implementations is a difficult task in modern processor environments because the execution time not only depends on the implementation but also depends on micro-architectural components such as cache memories, branch prediction, multi-threading, etc. We could have secret independent branching to avoid the timing attack based on the secret data. The secret dependency, the implementation should avoid (a) conditional branching should not depend on secret data (b) indirect load using the secret data [33].

6 Detecting Hardware Trojans Using Formal Verification

To minimize development costs and time-to-market, most system developers operate in a fabless mode, integrating multiple third party Intellectual Property cores into System on Chips (SoC) that are used in IoT devices. This design flow creates multiple opportunities for malicious code or circuits to be introduced into the device that could act as Trojans. For example, Trojans in connected IoT devices could act as backdoors and permit unauthorized users to enter into the device and access privileged information. This can lead to denial of service attacks, manipulation of data, or interception of sensitive data.

A typical Trojan is designed to be passive most of the time and active only when triggered. In a normal setting, the trigger is an extremely rare event, for example, a specific time event or particular inputs. Due to this stealthy nature, detecting the presence of Trojans is a considerable challenge especially when they are introduced in the hardware. Even 100% test coverage, will not guarantee the absence of Trojans in a hardware design. An attacker, however, with knowledge of the trigger conditions, could easily activate the Trojan to compromise the device.

In the literature several algorithms have been suggested to detect hardware Trojans. One of the most popular is FANCI, which identifies stealthy signals in a design [35]. Another tool called VeriTrust, uses the fact that gates triggered by Trojans, will not be driven by functional inputs. VeriTrust marks these gates as suspicious [36]. While these approaches can detect a Trojan with certain level of accuracy, they will not be able to guarantee the absence of a Trojan in the design. Moreover, they can only evaluate combinational parts of the design and would require considerable manual analysis.

In [30], Rajendran et al. proposed to use formal verification to detect Trojans in hardware designs. If a Trojan is present, formal verification can guarantee finding its trigger condition. Rajendran et al. use model checking for the purpose of detecting

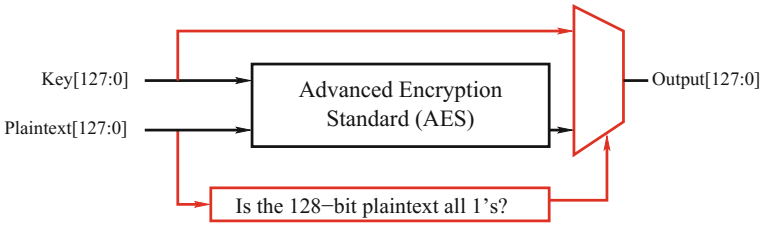


Fig. 5 The Trojan is triggered when the plaintext (128-bit) is all ones, which triggers the multiplexer and returns the secret key as Ciphertext [30]

hardware Trojans that leak sensitive information from the device. The input to the model checker is the target property to be checked along with a formal description of the design in temporal logic. The property to be checked is: "does the design leak sensitive information?". If a Trojan is present in the design, the output of the model checker will be a set of states denoting the trigger condition. As a case study, [30] considered a Trojan that could leak an encryption key as shown in Fig. 5. In normal working conditions, the design would encrypt data with the stored key. However, when a trigger is fed via the plaintext input, the output of the circuit would be the AES encryption key rather than the ciphertext, thereby leaking the key. In the simplest case, a specific value of plaintext, for example all 1s, could act as trigger resulting in leakage of the entire secret key or a subset of the secret key. This can be modeled formally using the property:

$$\exists i \in I \ni D \models (s == o) \ .$$

It means that there exists some trigger i from the set of possible input patterns I such that in the design D, the secret s is mapped to the output o. The model checker would search for an input assignment over the entire input space I that would satisfy the key leakage property, which is $(s == o)$. If such an assignment can be found then the corresponding input becomes the trigger for the Trojan.

The limitation of this approach is that the model checking property should comprehensively capture all trigger and leakage conditions if the Trojan is to be detected. Rajendran et al. [30] discusses multiple other options for creating the trigger and leaking the secret key. For example, the trigger could arrive over multiple clock cycles. The secret key leakage could be only a few bits of the entire key. Alternatively, the leakage could be a function of the key rather than the actual key bits.

Another limitation is in the scalability of formal verification to detect well concealed hardware Trojans in larger circuits. The results of Rajendran et al. were limited to preforming bounded model checking on 12 clock cycles of the designs. It would take considerably longer to verify if the Trojan's trigger occurred much later clock cycles. The scalability of formal verification to detect better concealed Trojans in larger designs is still an open problem.

7 Leveraging Formal Verification to Identify Meta-level Authentication Loopholes

Designers are often more keen to adopt security counter-measures over the cryptographic implementations and apply formal certification procedures to eliminate possible security flaws in the system design. However, there can be functional gaps in the meta-level of the implementation which may lead to weaknesses. Such gaps are often manifested, as designers (while implementing) are unaware of the IoT environment where these devices will be deployed in future. Such meta-level gaps can be categorized as follows:

`[Level-1]` *Absence of Authentication.* The access points of a secure implementation may require the use of authentication, may be in the form of passwords, which is completely oblivious to the designer. Hence, (s)he has not performed/devised any password or authentication checks in the high-level invocation of this implementation which may led to exploitation in the implementation and extract out secure data.

`[Level-2]` *Inability to Provide Strong Authentication.* Though the designer has implemented an authentication mechanism to safeguard the access of a secure implementation, however there may be shortcomings in the formulations where the authentication mechanism may be simple enough (for example, in case of simple passwords or reduced set of variations in the key space) that it may get regenerated by exhaustive enumeration of the possible variations of the encryption within computational limits.

`[Level-3]` *Missing Checks over Authentication Process.* Even if a strong authentication wrapper is present around the invocation of a secure implementation, there may be loopholes in the usage of the authentication process which may lead to serious flaws in the design. A strong authentication may get destroyed when multiple attackers can log through the same authentication strategy and distribute the search space to derive the encryption strategy/key. There is a possibility that this approach may lead to break-through in the authentication barrier which may have been computationally infeasible by single or two simultaneous user. Therefore, placing a limit on the number of accesses (for example, there may be checks to prohibit more than two login simultaneously) may resolve these issues.

The above discussion points to the fact that it may not be sufficient to formally certify the secure design/implementation, but we need to formally model the environment where this implementation is being deployed/invoked from. Then, another round of formal certification is mandatory to ensure the flaws in the meta-level authentication over the design. To adopt the formal certification in this level, we perform the following strategy:

- First, we abstract the functionality of the secure implementation in the form of *assumptions* (assume properties).
- Then, we formally model the authentication wrapper which invokes the secure design implementation in the form of assume properties.

- Next, the mentioned three-level attributes are captured in terms of a set of formal specifications.
- Finally, formal verification is performed over the authentication model (which instantiates the design in the form of assume properties) with respect to the formal specifications formed.

8 Conclusions

Recent developments in formal verification have significantly extended the capabilities of these tools. Theorem proving and model checking can potentially be applied to solve several hard problems in security, especially in the IoT domain. This chapter provided an overview of formal verification applied to solve five critical security issues related to an IoT device. Formal verification scales very well for some of the problems considered, such as detecting programming bugs in software and proving side-channel security.

Innovative usage of the formal verification tools is required to solve certain problems involving huge state space. This chapter demonstrated the use of a hierarchical verification methodology for verifying the correctness of cryptographic implementations, which has considerably huge state space making a naïve invocation of the formal verification tool fail. Identifying certain security problems, such as detection of hardware Trojans, though feasible with formal verification, is very restricted. The state-of-the-art can for instance, only detect Trojans that are triggered in the first few clock cycles of the device operation. If the Trojan is well concealed, for example, gets triggered much later in the device operation, then identifying them would be considerably more difficult.

References

1. Affeldt, R.: On construction of a library of formally verified low-level arithmetic functions. In: Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, 26–30 March 2012. pp. 1326–1331 (2012)
2. Amla, N., Kurshan, R.P., McMillan, K.L., Medel, R.: Experimental analysis of different techniques for bounded model checking. In: Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pp. 34–48 (2003)
3. Amla, N., Du, X., Kuehlmann, A., Kurshan, R.P., McMillan, K.L.: An analysis of SAT-based model checking techniques in an industrial environment. In: Proceedings of International Conference on Correct Hardware Design and Verification Methods (CHARME), pp. 254–268 (2005)
4. Appel, A.W.: Verification of a cryptographic primitive: SHA-256. ACM Trans. Program. Lang. Syst. **37**(2), 7:1–7:31 (2015). http://doi.acm.org/10.1145/2701415
5. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of BDDs. In: Proceedings of 36th Annual Design Automation Conference, pp. 317–320 (1999)

6. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. Lect. Notes Comput. Sci. **1579**, 193–207 (1999)
7. Biere, A., Clarke, E.M., Raimi, R., Zhu, Y.: Verifying safety properties of a PowerPC microprocessor using symbolic model checking without BDDs. In: Proceedings of International Conference on Computer-Aided Verification (CAV), pp. 61–71 (1999)
8. Bryant, R.: Graph-based algorithms for Boolean-function manipulation. IEEE Trans. Comput. **35**(8), 677–691 (1986)
9. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. **98**(2), 142–170 (1986)
10. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L.: Sequential circuit verification using symbolic model checking. In: Proceedings of 28th Annual Design Automation Conference, pp. 46–51 (1991)
11. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) Advances in Cryptology—CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, 15–19 August 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer (1999). https://doi.org/10.1007/3-540-48405-1
12. Chen, Y., Hsu, C., Lin, H., Schwabe, P., Tsai, M., Wang, B., Yang, B., Yang, S.: Verifying Curve25519 software. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014, pp. 299–309 (2014)
13. Clarke, E., Kroening, D.: The CPROVER User Manual (2006)
14. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another look at LTL model checking. In: Proceedings of International Conference on Computer-Aided Verification (CAV), pp. 47–71 (1994)
15. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2000)
16. Clake, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. J. Form. Methods Syst. Des. **19**(1), 7–34 (2001)
17. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2001)
18. Clarke, E.M., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, 29 March–2 April 2004, Proceedings, pp. 168–176 (2004)
19. Duan, J., Hurd, J., Li, G., Owens, S., Slind, K., Zhang, J.: Functional correctness proofs of encryption algorithms. In: 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2005, Montego Bay, Jamaica, 2–6 December 2005, Proceedings, pp. 519–533 (2005)
20. Eldib, H., Wang, C., Schaumont, P.: Formal verification of software countermeasures against side-channel attacks. ACM Trans. Softw. Eng. Methodol. **24**(2), 11:1–11:24 (2014). http://doi.acm.org/10.1145/2685616
21. Goldberg, E., Novikov, Y.: BerkMin: A fast and robust SAT-solver. In: Proceedings of Design Automation and Test Conference in Europe Conference (DATE). pp. 142–149 (2002)
22. Kang, H.J., Park, I.C.: SAT-based unbounded model checking. In: Proceedings of 40th Annual Design Automation Conference, pp. 840–843 (2003)
23. Kroening, D., Strichman, O.: Decision Procedures—An Algorithmic Point of View. Texts in Theoretical Computer Science. An EATCS Series. Springer (2008). https://doi.org/10.1007/978-3-540-74105-3
24. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
25. McMillan, K.L.: Applying SAT methods in unbounded symbolic model checking. In: Proceedings of International Conference on Computer-Aided Verification (CAV), pp. 250–264 (2002)
26. The MITRE Corporation: Common Vulnerabilities and Exposures. https://cwe.mitre.org/
27. The MITRE Corporation: Common Weakness and Enumerations. https://cwe.mitre.org/
28. Moskewicz, M., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of 38th Annual Design Automation Conference, pp. 530–535 (2001)

29. Nguyen, D.M., Stoffel, D., Welder, M., Kunz, W.: Conflict driven learning in a quantified Boolean satisfiability solver. In: Proceedings of International Conference on Computer-Aided Design (ICCAD), pp. 442–449 (2002)
30. Rajendran, J., Dhandayuthapany, A.M., Vedula, V., Karri, R.: Formal security verification of third party intellectual property cores for information leakage. In: 29th International Conference on VLSI Design and 15th International Conference on Embedded Systems, VLSID 2016, Kolkata, India, 4–8 January 2016, pp. 547–552. IEEE Computer Society (2016). https://doi.org/10.1109/VLSID.2016.143
31. Silva, M., Sakallah, K.A.: GRASP: a search algorithm for propositional satisfiability. IEEE Trans. Comput. **48**(5), 506–521 (1999)
32. Smith, E.W., Dill, D.L.: Automatic formal verification of block cipher implementations. In: Formal Methods in Computer-Aided Design, FMCAD 2008, Portland, Oregon, USA, 17–20 November 2008, pp. 1–7 (2008)
33. Tsai, M., Wang, B., Yang, B.: Certified verification of algebraic properties on low-level mathematical constructs in cryptographic programs. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, 30 October–03 November 2017, pp. 1973–1987 (2017)
34. U.S. Department of Commerce, National Institute of Standards and Technology: Digital Signature Standard (DSS) (2000)
35. Waksman, A., Suozzo, M., Sethumadhavan, S.: FANCI: identification of stealthy malicious logic using Boolean functional analysis. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, 4–8 November 2013, pp. 697–708. ACM (2013). http://doi.acm.org/10.1145/2508859.2516654
36. Zhang, J., Yuan, F., Wei, L., Liu, Y., Xu, Q.: VeriTrust: verification for hardware trust. IEEE Trans. CAD Integr. Circuits Syst. **34**(7), 1148–1161 (2015). https://doi.org/10.1109/TCAD.2015.2422836

SENSE: Sketching Framework for Big Data Acceleration on Low Power Embedded Cores

Amey Kulkarni and Tinoosh Mohsenin

Abstract Ever-growing IoT demands big data processing and cognitive computing on mobile and battery operated devices. However, big data processing on low power embedded cores is challenging due to their limited communication bandwidth and on-chip storage. Additionally, IoT and cloud-based computing demand low overhead security kernel to avoid data breaches. In this chapter, we present, "SENSE", **S**ketching and **En**cryption on **S**calable heterogeneous **E**ngine for data reduction and encryption. SENSE is a heterogeneous framework which consists of three important kernels: 1. sketching module for data reduction, 2. an accelerator for efficient sketch recovery using scalable and parallel reconstruction architecture and 3. a host processor to perform post processing. SENSE framework can reduce data up to 67% with 3.81 dB signal-to-reconstruction error rate (SRER). One of the critical challenges in big data processing on embedded hardware platforms is to reconstruct the sketched data in real-time with stringent constraints on error bounds and hardware resources. We explore Orthogonal Matching Pursuit (OMP) algorithm for sketch data recovery. OMP is a greedy algorithm with high computational complexity which has emerged as an important tool for signal recovery, dictionary learning and sparse data classification. We use a domain specific many-core hardware named Power Efficient Nano Cluster (PENC) designed by EEHPC lab at University of Maryland, Baltimore County. To demonstrate efficiency of SENSE framework, we integrate it with Hadoop MapReduce platform for face detection application. The full hardware integration consists of tiny ARM cores which perform task scheduling and application

This work is an extended version of the paper *LESS: Big data sketching and Encryption on low power platform* [1] and *Low Overhead CS-Based Heterogeneous Framework for Big Data Acceleration* [2].

A. Kulkarni (✉) · T. Mohsenin
University of Maryland, Baltimore County, USA
e-mail: ameyk1@umbc.edu
URL: http://www.csee.umbc.edu/

T. Mohsenin
e-mail: tinoosh@umbc.edu

R. S. Chakraborty et al. (eds.), *Security and Fault Tolerance in Internet of Things*,
Internet of Things, https://doi.org/10.1007/978-3-030-02807-7_10

processing, while PENC acts as an accelerator for sketch reconstruction. We show performance of SENSE framework on face identification application.

Keywords Big data acceleration · Many-core platform · Sketching algorithms · Internet of things · Security

1 Introduction

In recent years, Artificial Intelligence (AI) is tremendously growing in IoT devices, right from home assistants, bio-medical trackers, surveillance, to semi-autonomous cars. These Internet of Intelligent Things (IoIT) devices generate about Quintilian of data everyday. In most crowd-sourced with heavy data processing applications, intelligence is performed in cloud. Current real-time IoT devices takes advantage of both real-time embedded devices and cloud computing. It typically follows following steps on embedded devices: 1. Gather data from different sensors on real-time embedded devices, 2. Perform on-device intelligence based real-time data 3. Add intelligent inferences based on crowd-sourced cloud data, 4. Finally update cloud with newly collected real-time data and inferences. Thus embedded devices perform not only on-device intelligence, but also need to send huge data, while constraints on power and latency. To solve AI on embedded devices, the trend has started to adopt low-power host with accelerated hardware for machine learning, and signal processing applications. FPGAs, embedded GPUs such as nvidia TX2, Qualcomm's snapdragon, and domain specific many-cores such as movidius myriad2 are being widely used as hardware accelerators. However, these embedded platforms can suffer from potential communication bandwidth bottleneck for large data transfers [3, 4]. Therefore, data transfer reduction with encryption is of utmost importance.

The compression algorithms should be sensor-agnostic, and should allow energy-efficient, and low memory transmissions. Different algorithms have been proposed, however these solutions encounter various challenges such as: 1. withstanding continuously changing sensory information, 2. Hardware overhead of decompression algorithm, including chip area, processing power and latency of operations, and 3. High decompression error rate. Adaptive sketching techniques have demonstrated exceptional decompression (reconstruction) error rate [4, 5], however the reconstruction of sketched signal is computationally intensive [2, 6]. We propose sketching based framework called "SENSE", **S**ketching and **En**cryption on **S**calable heterogeneous **E**ngine, consisting of two kernels: sketching original signal and reconstruction of sketched signals; first data "sketching" is performed on gathered real-time sensor data to acquire compressed measurements at embedded device/in cloud, and then "reconstruction" kernel recovers sparse data at cloud servers/embedded devices using very less number of linearly transformed measurements. The "reconstruction" can be performed using different algorithms such as Compressive Sampling (CoSAMP), Iterative Hard Thresholding (IHT), L1-minimization, and Orthogonal Matching Pursuit (OMP). Among all reconstruction algorithms OMP achieves high reconstruction accuracy with lower computational resources. In this work, we adapt

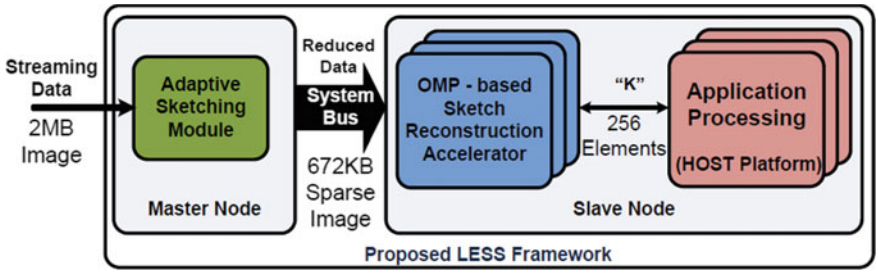


Fig. 1 Proposed hardware efficient sketch-based heterogeneous framework for big data acceleration, adaptive sketching is implemented using our previous work [7], reconstruction of sketched data is achieved using OMP algorithm on hardware accelerator platform, whereas application processing is performed on host platform on reconstructed data

OMP as reconstruction tool. OMP consists of computationally intensive vector sort, matrix multiplications and least square kernel. Therefore, to achieve real-time and low power constraints, SENSE framework is implemented on heterogeneous architecture, where OMP is implemented on domain specific many-core accelerator.

Figure 1 shows heterogeneous SENSE framework that consists of an accelerator for efficient sketch recovery using a reconfigurable and parallel OMP reconstruction architecture and a general purpose HOST platform for application processing. The heterogeneous SENSE framework can achieve about 48% data reduction with satisfactory reconstruction accuracy. Analysis of the proposed SENSE framework is performed in terms of execution time and power consumption on different platforms including Quad-Core ARM A15 CPU, embedded GPU available on nvidia TK1 platform, and domain specific many-core platform named Power Efficient Nano Cluster—PENC[1] to chose best accelerator platform for sketch reconstruction. To demonstrate real-time performance of face detection application using SENSE framework, we implemented Hadoop MapReduce platform. The performance of SENSE framework is shown in terms of reconstruction quality, hardware overhead cost for end-to-end framework, and overall reduction in data transfers.

2 Background

2.1 Orthogonal Matching Pursuit (OMP) Algorithm

OMP algorithm has emerged as an important tool for signal recovery, dictionary learning and sparse data classification, and has potential in various emerging applications ranging from wearable and mobile computing to real-time analytics processing on servers.

[1]PENC many-core platform is developed by EEHPC lab at University of Maryland Baltimore County, USA, Web: http://eehpc.csee.umbc.edu/.

2.1.1 Signal Recovery Problem

Let us assume D to be a k-sparse signal of length n. Let ϕ be the measurement matrix projected onto the original signal, D. Measurement matrix (ϕ) must be incoherent with the basis of the sparse signal, D. If D is not sparse in its original bases, it can be transformed to another domain in which the signal is sparse. Then the measurement matrix has to be uncorrelated with the signal in the transformed domain [8]. The size of ϕ is $m \times n$, where $m \ll n$ and represents the number of measurements. $\hat{D}$ is a m-length vector containing the measurements obtained by the projection of ϕ onto D. Therefore, signal need to be converted to a transformed basis, ψ to induce sparsity and $\hat{D}$ is obtained as:

$$\hat{D} = \phi\psi D = \phi D \tag{1}$$

where ϕ needs to be chosen such that the restricted isometric property (RIP) of order $2k$ is satisfied, where k represents the sparsity of the signal, D.

2.1.2 OMP Algorithm and Analysis

Matching pursuit algorithms have least computational complexity among all reconstruction algorithms. OMP is an iterative greedy matching pursuit algorithm with lowest complexity and signal-to-reconstruction error [9, 10]. OMP unlike Principle Component Analysis (PCA) and Singular Value Decomposition (SVD) algorithms, can adapt to continuously changing big data sets due to its complexity [4, 11] (Fig. 2).

Notations Used in OMP Algorithm

- D = Original data, S = Sampled reduced data (measured data), $\hat{D}$ = Reconstructed original data
- n = Length of the original data
- m = Number of measurements
- k = Sparsity
- R = Residual matrix

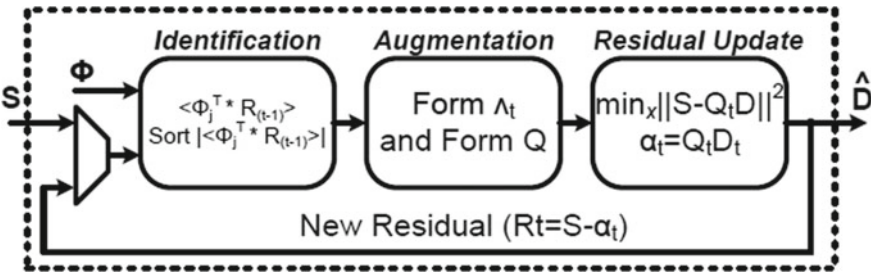


Fig. 2 Basic block diagram of OMP algorithm

- ϕ = Measurement Matrix ($size: m \times n$)
- λ = Maximum index after dot product
- t = No. of iterations (usually equal to k).

Algorithm 1 OMP Reconstruction Algorithm

1:**Initialization**

– $R_0 = S$, $\Lambda_0 = \emptyset, _0 = \emptyset$ and $t = 0$

2:**Identification**

– Find Index $\lambda_t = max_{j=1\ldots n}$ subject to $| < \phi_j R_{t-1} > |$

3:**Augmentation**

– Update $\Lambda_t = \Lambda_{t-1} \bigcup \lambda_t$
– Update $Q_t = [Q_{t-1}\ \phi_{\Lambda_t}]$

4:**Residual Update**

– Solve the Least Squares Problem
$D_t = \min_D \|S - Q_S D\|^2$
– Calculate new approximation: $\alpha_t = Q_t\ D_t$
– Calculate new residual: $R_t = S - \alpha_t$

5: Increment t, and repeat from step 2 if $t < k$
After all the iterations, we can find correct sparse signals.

OMP is a greedy algorithm, it finds the sparsest solution iteratively by computing support of D and subtracting it from measurement vector S at every iteration. OMP has three different phases, *Identification, Augmentation* and *Residual Update*. In identification phase index i of highest magnitude of $\phi * R$ is chosen as potential vector to find closest approximation to D. At each iteration, index i is added to the list of estimated support vectors in augmentation phase. The residual update phase generates residual for next iteration. In residual update phase, formed augmented matrix Q is used in least square regression model to find linear relationship between augmented matrix Q and measured vector S. Finally, the amount of contribution that column S provides is subtracted to obtain a residue. The OMP algorithm takes k iterations to determine correct set of columns [12, 13].
The computational complexity for each step is explained below:

- Identification phase $< \phi R >$ performs inner product operation of $\phi(M \times N)$ matrix with residue vector $R(1 \times M)$. Thus computational complexity of $\mathcal{O}(MN)$. Maximum of $< \phi R >$, which gives a $N \times 1$ vector. Hence, it has a computational complexity of $\mathcal{O}(N)$.
- Residual update phase consist of the least squares problem. In every iteration, i, Φ matrix consists of i columns of size M. Thus, the new matrix, Q, is of size $i \times M$.

Doing a $(Q^T Q)$ gives a square matrix $(i \times i)$. Thus, a cost of $\mathcal{O}(iM)$. The cost of inverting this $i \times i$ matrix by LU Decomposition is $\mathcal{O}(i^3)$. The cost of $Q^T y$ is $\mathcal{O}(i^2)$.

- To calculate the new approximation: $\alpha_t = Q_t\, x_t$
  Q is of size $i \times M$ and x is of size $1 \times i$. Thus computational complexity of $\mathcal{O}(iM)$.
- Compute new residual: $R_t = y - \alpha_t$, y and α_t are $M \times 1$ matrices. The subtraction at each iteration will take M computations, hence, $\mathcal{O}(M)$.

Therefore, the total cost per iteration will be $\mathcal{O}(MN)$. If the signal is k-sparse, OMP algorithm will be iterated k times, giving a total computation complexity of $\mathcal{O}(kMN)$.

OMP Acceleration Analysis on Various Platform

For all platforms, the measurement matrix ϕ is stored on-chip to reduce external memory overhead. Furthermore, Monte-Carlo simulations are performed since measurement matrix and sparse image is based on random variables. Figures 3 and 4 show the comparison of execution time and energy consumption between the ARM CPU, K1 GPU, and PENC many-core architecture [14–16]. Overall comparison between ARM CPU and K1 GPU shows that, ARM CPU performs best for smaller image sizes and K1 GPU performs better for large image sizes i.e. for higher computational complexity. Compared to ARM CPU and K1 GPU implementation PENC many-core platform performs 8× and 177× faster and saves 15× and 200× energy consumptions, respectively. Additionally considering chip area for TK1 platform in 28 nm and PENC platform in 65 nm, PENC many-core platform is the most efficient choice for OMP kernels. In this work, we adapt PENC many-core as an accelerator which provides programmability and parallelism capability with higher energy efficiency as compared to ARM A15 CPU and embedded GPU.

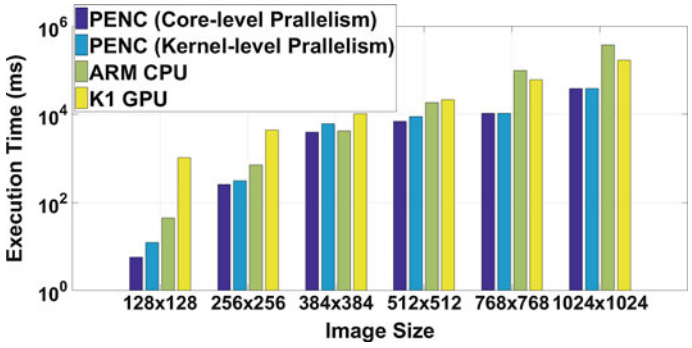


Fig. 3 Comparison of OMP execution time analysis on quad-core ARM CPU, K1 GPU at maximum clock rate of 2320.5 MHz and 852 MHz respectively, with PENC many-core (both core-level and kernel-level parallelism) implementations at 1 GHz

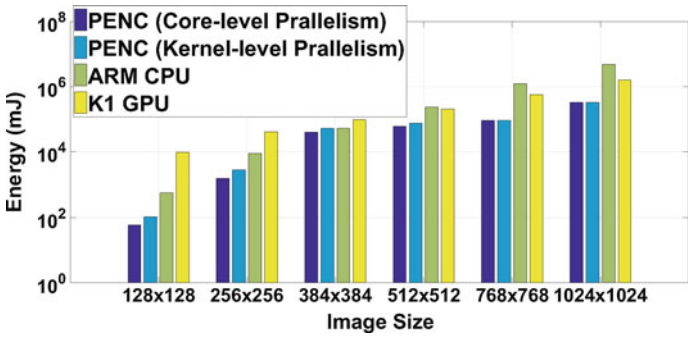


Fig. 4 Comparison of OMP energy consumption analysis on quad-core ARM CPU, K1 GPU at maximum clock rate of 2320.5 MHz and 852 MHz respectively, with PENC many-core (both core-level and kernel-level parallelism) implementations at 1 GHz

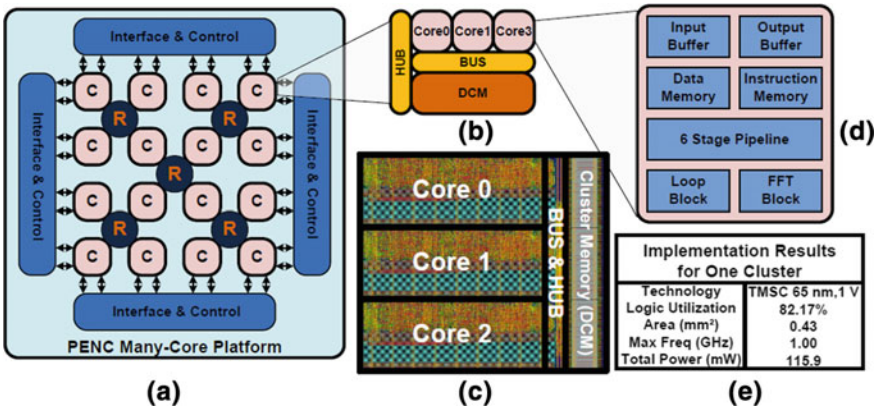


Fig. 5 **a** Power efficient nano clusters (PENC), many-core architecture. **b** Bus-based cluster architecture. **c** Post-layout view of bus-based cluster implemented in 65 nm, 1V TSMC CMOS technology. **d** Block diagram of core architecture. **e** Post layout implementation results of bus-based cluster (consisting 3 cores + bus + cluster memory)

2.2 PENC Many-Core Architecture

Power Efficient Nano Clusters (PENC), many-core architecture is composed of 64 processing clusters (192 Cores) connected through routers in a three-level GALS hierarchical tree. The lowest level consists of four clusters connected by a router with five ports: one for each cluster and one for communication to the next level. GALS hierarchical tree structure of PENC many-core allows us distributed computing and scalability, thus efficient embedding of an extra processing core or cluster to the chip. The lightweight cores also help to ensure that all used cores are fully utilized. While the lightweight cores are ideal for sketching and reconstruction kernels, they often require large amounts of memory for their model data. This is addressed with the

cluster-level shared memory that is interfaced to the bus. The shared memory can be accessed within the cluster on the bus and from other clusters through the router. Figure 5 shows the block diagram of PENC many-core with the details of bus-based cluster and processor block diagram. It also gives brief idea of implementation results on 65 nm, 1 V TSMC CMOS technology.

2.3 Security Definitions for SENSE Framework

Figure 1 shows the SENSE framework for secured big data processing, in which encryption is achieved by using One-Time Random Linear Projections (OTRLP) based sketching technique and decryption is performed by OMP sketch-reconstruction algorithm. In big data system scenario, master node receives streaming data $\mathbb{D}_{n\times n}$, where n is size of streaming data matrix.[2] Sketching algorithm performs learned random projections on streaming data to obtain $\mathbb{R}_{m\times n}$. The reduced data $\mathbb{R}_{m\times n}$ and keys $\mathbb{K}$ are transferred over the system bus to the slave node. The slave node consists of OMP sketch-reconstruction kernels to recover sketched signals using keys $\mathbb{K}$ and big data processing platform.

Considering streaming data signal (plaintext) $\mathbb{D}_{n\times n}$ is sparse and seed keys $\mathbb{K}$ which generate one-time measurement matrix $\phi_{m\times n}$ be a measurement matrix such that $m < n$, where m is the number of measurements to be taken and n is length of the original signal. Then sketching problem can be stated as: Reconstruct $\mathbb{D}_{n\times n}$ from the knowledge of

$$\mathbb{R}_{m\times n} = \phi_{m\times n}\mathbb{D}_{n\times n} \tag{2}$$

A private key has two functions $E_k : \mathbb{D}_{n\times n} \rightarrow \mathbb{R}_{m\times n}$ and $D_k : \mathbb{R}_{m\times n} \rightarrow \mathbb{D}_{n\times n}$. Thus, $D_k(E_k(\mathbb{D}_{n\times n})) = \mathbb{D}_{n\times n}$ is unfeasible without knowing key $\mathbb{K}$ to determine $E_k(\mathbb{D}_{n\times n}) = \mathbb{R}_{m\times n}$. In case of known n linearly independent messages an attacker can deduce content of messages, thus data transfers are not secure under KPA if the same matrix is used multiple times. We use one-time random linear projections (OTRLP) scenario [17] in which each measurement matrix is only used one time and all measurement matrices are statistically independent.

3 Application Case Study and Analysis

The implementation results shows that, the domain specific PENC many-core platform is energy efficient and requires less chip area as compared to existing off-the-shelf platforms. Therefore, we choose PENC many-core platform for hardware efficient OMP reconstruction in the SENSE framework. To demonstrate efficiency

[2]For convenience to explain overview of the framework, we selected row and column size to be same. In real-time streaming data can be of different column and row sizes.

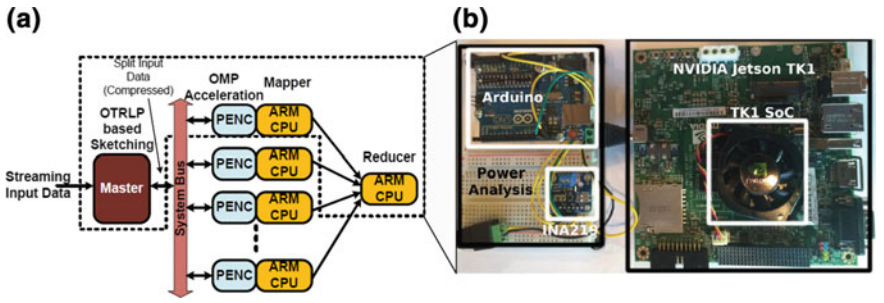


Fig. 6 **a** Integration of SENSE hardware framework with hadoop mapreduce, sketch reconstruction is achieved using OMP algorithm on PENC many-core platform, map and reduce is performed on ARM CPU. **b** MapReduce setup on Nvidia TK1 platform with current measurement setup using a TI INA219 and an Arduino Uno

of the SENSE framework "face detection" application implemented targeting two different big data benchmarks for face detection [18, 19] as shown in Fig. 6.

3.1 Hadoop Implementation on Embedded Platform

The MapReduce platform is a essential module in Hadoop framework, it performs parallel data processing over several distributed nodes. The MapReduce is designed to perform Big Data applications efficiently, however each node has limited storage capability with constraints on bandwidth. In Hadoop framework, first the data is distributed off-line, i.e. before application processing, to each parallel processing node also known as "map" node. Thus, application data size, data bandwidth and storage capacity of each "map" node is critical to performance of hadoop cluster.

The MapReduce framework typically consists of three important kernels: 1. Master Node 2. Map Stage 3. Reduce Stage as shown in Fig. 6. The master node accepts job requests from user, schedules them to different mappers and monitors the job (program running on Map and Reduce stages) status. The map stage accepts the $< key, value >$ from the master and processes scheduled tasks, whereas reduce stage performs final operations. The size of big data set necessitates compression on underlying platform.

In this work, we implemented MapReduce on Nvidia TK1 platform, where PENC many-core is adapted as a hardware accelerator for data reconstruction at each map node, and general purpose ARM A15 CPU performs application processing. We built native Hadoop 2.6.3 [20] libraries from source for TK1 platform. Figure 6 shows integration of Hadoop MapReduce architecture with the proposed sketch-based SENSE framework, to evaluate different applications. The experiment is performed in four different stages: 1. The images to be analyzed are sketched using OTRLP kernel with 33% measurements i.e only 33% of the image is transferred. The resulting

transformed image are stored as binary files for distribution. 2. *SequenceFile* is used to create a persistent data structure for binary key-value pairs. The key is generated from the name of the file and a value is the binary data from the compressed file. It ensures that the binary data of each image is not segmented before facial recognition occurs. 3. At the consumer (mapper) end, reconstruction of the sketched image is performed using PENC many-core platform. While the reconstructed image is placed into a queue for the consumer thread, the producer thread reads in the next key-value pair. 4. Finally, the consumer thread passes reconstructed data application processing.

3.2 Face Detection Application

Face detection is important for application such as expression identification, face recognition, and lip reading. In past decade, the researchers have explored different techniques for face detection including image processing, Principal Component Analysis (PCA), machine learning, and in recent years deep learning. Face detection can be performed at hand-held embedded devices, general purpose compute platform, and cloud servers, depending on the applications. Therefore, we believe face detection application is a right fit to demonstrate SENSE framework. The face detection is implemented using Haar feature based cascade classifier in OpenCV [21, 22]. Cascade classifier is a machine learning tool, consisting of series of simple classifiers stages including linear regression, support vector machines. Each classifier updates the votes based on training weights on region of interest until the candidate is passed or rejected [23]. The stages of classifier are selected based on data size, and variance in data, the trade-off exist between classification accuracy and number of stages. In this work, we use 20-stage cascade classifier trained on 5000 images of size 512×512 consisting of 60% positive sample [24, 25]. The face detection is evaluated on different number of images ranging from 25 to 1000 with up to 6 faces in each image. The SENSE framework can be efficiently adopted for different machine learning algorithms. SENSE framework has been evaluated for different machine learning algorithms including support vector machine, nave bayes, logistic regression, and k-nearest neighbors [7]. For seizure detection application, computations are reduced by 16× while energy consumption of processing is reduced up to 68%.

In order to analyse decompression error rate, reconstruction performance analysis is performed in terms of mis-classification analysis. Figure 7 shows the example of original image transition to reconstructed image and face detection on hadoop MapReduce platform with SENSE framework. Table 1 shows execution analysis of the proposed SENSE framework integrated with hadoop MapReduce for face detection application. The proposed SENSE framework reduces data transfers by 26% with 67% reduction in data. To demonstrate efficiency of the PENC sketched data reconstruction acceleration, we implemented MapReduce platform in two different cases, 1. ARM CPU is used for CS reconstruction and processing i.e for master,

Fig. 7 Visual representation of image before and after each stage of processing. **a** Original image. **b** Reconstructed image from the stored compressed image. **c** Successful facial identification of the reconstructed image

Table 1 Execution time analysis of SENSE framework integrated with MapReduce for face detection application. In PENC + ARM CPU, sketch data reconstruction is performed on PENC and application processing on ARM CPU whereas in ARM CPU only implementation the sketch data reconstruction and processing performed on ARM CPU. The SENSE platform achieves 26.15% reduction in original data

Size of data	Application execution time			Execution
	PENC + ARM (s)	ARM only (s)	Improvement (%)	Time overhead (%)
25 images (11 MB)	14	17	22.26	8.76
50 images (22 MB)	27	34	26.22	6.75
250 images (108 MB)	144	183	26.97	4.49
500 images (217 MB)	281	359	27.90	4.36
1000 images (434 MB)	583	741	27.01	4.08
2500 images (1.08 GB)	1,473	1,884	27.90	3.99
5000 images (2.17 GB)	3,015	3,906	29.55	3.78

CS reconstruction and mapper, reducer. 2. Combination of PENC and ARM CPU, in which PENC is used for CS reconstruction and ARM CPU is used for master, mapper and reducer. Compare to ARM CPU implementation, PENC + ARM implementation reduces application processing time by 22–29% and saves 32–34% energy consumption as shown in Table 2. Additionally we also perform hardware overhead analysis of CS reconstruction on hadoop MapReduce platform. SENSE framework has very low execution time overhead of 3.7% and negligible energy consumption overhead of 0.002% when tested for 5000 images. Tables 1 and 2 shows that, the increase in number of data sizes SENSE framework will have insignificant execution and energy consumption overhead.

Table 2 Energy consumption analysis of SENSE framework integrated with MapReduce for face detection application. In PENC + ARM CPU, sketch data reconstruction is performed on PENC and application processing on ARM CPU whereas in ARM CPU only implementation the sketch data reconstruction and processing performed on ARM CPU

Size of data	Application energy consumption			Energy
	PENC + ARM (J)	ARM only (J)	Improvement (%)	Consumption (%)
25 images (11 MB)	166	223	34.31	0.007
50 images (22 MB)	322	436	35.52	0.005
250 images (108 MB)	1,736	2,309	32.94	0.003
500 images (217 MB)	3,391	4,536	33.74	0.003
1000 images (434 MB)	7,060	9,349	32.41	0.003
2500 images (1.08 GB)	17,044	22,769	33.58	0.003
5000 images (2.17 GB)	35,210	46,946	33.33	0.002

4 Summary

In this chapter, we propose SENSE framework consisting of adaptive big data sketching with encryption, and sketched data reconstruction for efficient big data processing on hardware. We chose OMP algorithm, which has highest reconstruction accuracy and lowest computational complexity among all matching pursuit reconstruction algorithms. The OMP reconstruction algorithm is implemented on different off-the-shelf embedded devices including ARM A15 CPU, Nvidia TK1, and domain specific PENC many-core platform. The PENC many-core platform performs 16-bit data computation, thus consume 15× and 18× less energy and 16× and 8× latency reduction in reconstruction time as compared to low power ARM CPU, and K1 GPU respectively, while achieving satisfactory range of signal quality. The experimental analysis insists to adapt PENCS many-core platform as a hardware accelerator for sketched data reconstruction. Therefore, SENSE heterogeneous framework is implemented using PENC many-core for sketched data reconstruction and ARM A15 CPU for application processing. To demonstrate the efficiency of SENSE framework, we implement Hadoop MapReduce architecture on nvidia TK1 platform for face detection application. The SENSE framework achieves 26.17% data transfer reduction with negligible energy overhead of 0.002% and very low execution overhead of 3.7% when tested for 5000 images with each image consisting of 1–6 faces.

References

1. Kulkarni, A., Shea, C., Homayoun, H., Mohsenin, T.: LESS: big data sketching and encryption on low power platform. In: 2017 Design, Automation Test in Europe Conference Exhibition (DATE) (2017)
2. Kulkarni, A., et al.: Low overhead CS-based heterogeneous framework for big data acceleration. In: Proceedings of the ACM Transactions Embedded Computing System 17, 1, Article 25 (2017), 25 pages, GLSVLSI '16, pp. 57–62. ACM (2017)
3. Kulkarni, A., et al.: CS-based secured big data processing on FPGA. In: 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 201–201 (2016)
4. Rouhani, B., et al.: Sketch: An automated framework for streaming sketch-based analysis of big data on fpga. In: 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 187–194 (2015)
5. Mirhoseini, A., et al.: Perform-ml: performance optimized machine learning by platform and content aware customization. In: Proceedings of the 53rd Annual Design Automation Conference, DAC '16, pp. 20:1–20:6. ACM, New York, NY, USA(2016)
6. Kulkarni, A.M., Homayoun, H., Mohsenin, T.: A parallel and reconfigurable architecture for efficient OMP compressive sensing reconstruction. In: Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI, GLSVLSI '14, 299–304. ACM, New York, NY, USA (2014)
7. Kulkarni, A., Jafari, A., Sagedy, C., Mohsenin, T.: Sketching-based high-performance biomedical big data processing accelerator. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1138–1141 (2016)
8. Candès, E., Wakin, M.: An introduction to compressive sampling. Signal Process. Mag. IEEE **25**(2), 21–30 (2010)
9. Tropp, J., Gilbert, A.: Signal recovery from random measurements via orthogonal matching pursuit. IEEE Trans. Inf. Theory **53**(12), 4655–4666 (2007)
10. Needell, D., Vershynin, R.: Signal recovery from incomplete and inaccurate measurements via regularized orthogonal matching pursuit. IEEE J. Select. Topics Signal Process. **4**(2), 310–316 (2010)
11. Korde, A., Bradley, D., Mohsenin, T.: Detection performance of radar compressive sensing in noisy environments. In: International SPIE Conference on Defense, Security, and Sensing (2013)
12. Tropp, J., Gilbert, A.: Signal recovery from random measurements via orthogonal matching pursuit. IEEE Trans. Inf. Theory **53**(12), 4655–4666 (2007)
13. Kulkarni, A., Mohsenin, T.: Accelerating compressive sensing reconstruction OMP algorithm with CPU, GPU, FPGA and domain specific many-core. In: 2015 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 970–973 (2015). https://doi.org/10.1109/ISCAS.2015.7168797
14. Page, A., Attaran, N., Shea, C., Homayoun, H., Mohsenin, T.: Low-power manycore accelerator for personalized biomedical applications. In: Proceedings of the 26th Edition on Great Lakes Symposium on VLSI, (GLSVLSI '16), pp. 63–68. ACM, New York, NY, USA. 978-1-4503-4274-2 (2016). https://doi.org/10.1145/2902961.2902986
15. Kulkarni, A., Pino, Y., Mohsenin, T.: Adaptive real-time Trojan detection framework through machine learning. In: 2016 IEEE International Symposium on Hardware Oriented Security and Trust, (HOST '16), May 2016, pp. 120–123 (2016)
16. Kulkarni, A., et al.: Low energy sketching engines on many-core platform for big data acceleration. In: Proceedings of the 26th Edition on Great Lakes Symposium on VLSI, GLSVLSI '16, pp. 57–62. ACM (2016)
17. Bianchi, T.: Analysis of one-time random projections for privacy preserving compressed sensing. IEEE Trans. Inf. Forens. Sec. **11**(2), 313–327 (2016)
18. Martinez, A., Benavente, R.: The AR face database. In: CVC Technical Report #24 (1998)
19. Jain, V., Learned-miller, E.: FDDB: a benchmark for face detection in unconstrained settings. Technical Report (2010)

20. Apache kernel description (2016). http://www.apache.org/
21. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001. CVPR 2001, vol. 1, pp. I–511–I–518 (2001)
22. Girshick, R.: Fast r-cnn. In: The IEEE International Conference on Computer Vision (ICCV) (2015)
23. Haar Feature-based Cascade Classifier for Object Detection. http://docs.opencv.org/. Accessed 2 Feb 2016
24. Lienhart, R., Kuranov, A., Pisarevsky, V.: Pattern Recognition: 25th DAGM Symposium, Magdeburg, Germany, September 10–12, 2003. Proceedings. Chapter Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection, pp. 297–304. Springer, Berlin, Heidelberg (2003)
25. Sinha, P., Balas, B., Ostrovsky, Y., Russell, R.: Face recognition by humans: nineteen results all computer vision researchers should know about. Proc. IEEE **94**(11), 1948–1962 (2006)