Confidential Remote Computing

Kubilay Ahmet Küçük

Kellogg College
The Department of Computer Science
University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2022

Leave to Supplicate: 29 March 2023
Final Corrections: 24 October 2022
Viva Result: 23 August 2021
Viva Voce, Thesis Defense: 16 July 2021
Thesis Submission: 12 March 2021

Confirmation of D.Phil. Status Notification: 20 February 2019
Confirmation Result: 28 January 2019
Confirmation Viva: 14 January 2019
Submission for Confirmation: 19 December 2018

Transfer Result: 21 March 2017
Transfer Examination: 17 March 2017
Submission for Transfer of Status: 25 February 2017
Start Date: 11 October 2015

Confidential Remote Computing

Kubilay Ahmet Küçük

Kellogg College
The Department of Computer Science
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Michaelmas 2022

Abstract

Since their market launch in late 2015, trusted hardware enclaves have revolutionised the computing world with data-in-use protections. Their security features of confidentiality, integrity and attestation attract many application developers to move their valuable assets, such as cryptographic keys, password managers, private data, secret algorithms and mission-critical operations, into them. The potential security issues have not been well explored yet, and the quick integration movement into these widely available hardware technologies has created emerging problems. Today system and application designers utilise enclave-based protections for critical assets; however, the gap within the area of hardware-software co-design causes these applications to fail to benefit from strong hardware features. This research presents hands-on experiences, techniques and models on the correct utilisation of hardware enclaves in real-world systems.

We begin with designing a generic template for scalable many-party applications processing private data with mutually agreed public code. Many-party applications can vary from smart-grid systems to electronic voting infrastructures and block-chain smart contracts to internet-of-things deployments. Next, our research extensively examines private algorithms executing inside trusted hardware enclaves. We present practical use cases for protecting intellectual property, valuable algorithms and business or game logic besides private data. Our mechanisms allow querying private algorithms on rental services, querying private data with privacy filters such as differential privacy budgets, and integrity-protected computing power as a service. These experiences lead us to consolidate the disparate research into a unified Confidential Remote Computing (CRC) model. CRC consists of three main areas: the trusted hardware, the software development and the attestation domains. It resolves the ambiguity of trust in relevant fields and provides a systematic view of the field from past to future. Lastly, we examine the questions and misconceptions about malicious software profiting from security features offered by the hardware.

The more popular idea of confidential computing focuses on servers managed by major technology vendors and cloud infrastructures. In contrast, CRC focuses on practices in a more decentralised setting for end-users, system designers and developers.

Acknowledgements

Our civilisation today offers us unlimited opportunities and possibilities. It is impossible to thank every individual contribution enabling today's technology stack, and in a world without them this thesis would not have been available for the readers. I was able to write this thesis and make the relevant contributions by the help of collective greater wisdom that the humankind has brought. Many of the inventions today we all use in our daily life, such as devices and infrastructures, civil system and others that I cannot name all, I thank you all for giving me the unique opportunity of writing a thesis titled "confidential remote computing". Although, it may not mean too much for the world, it has been a huge step for me.

Supervision

I would like to express my sincere gratitude to my supervisor Prof. Andrew Martin for the continuous support for my studies and related research, for his patience, motivation, and immense knowledge. He has profoundly shaped my intellectual journey, providing guidance whenever I strayed from my path. He has been always giving me the right amount of guidance at precisely the right times. He taught me a lot both technically and in thinking style, encouraged me to strengthen my weaknesses, prevented me wasting time. Our meetings were highly likely to be one of the best part of my Oxford experiences. These meetings widened my perspective, gave so much fruits for thoughts and they always felt like the weekly medicine that pushed me forward in my research. His predictions and prophecy on future risks were always very precise and helpful to adjust my research.

Dear Professor, you have given me so much of time, care and attention and it is all together a life lesson to live longer. I thank you wholeheartedly for your kindness and support at all times.

Examination

It has been a great pleasure to defend my thesis before Ivan Martinovic and Chris Mitchell. My *viva voce* was truly a unique experience, the constructive feedback I received helped to reach into this final shape. Having the rigorous assessment of

Prof Mitchell was truly difficult and at the same time enabling this thesis to reach to a much better version, I warmly thank Prof Mitchell for his generous feedback, time and efforts. Having Prof Martinovic for both my viva and confirmation of status was a long experience of his support, providing a continous feedback, and enabling a version of this thesis ready for the members of public. Further, I owe huge thanks to Jan Van Kleek his feedback in my confirmation, and huge thanks for David Wallom and Joss Wright for their feedbacks and assessments in my Transfer of Status. I am very thankful and grateful to all professors who examined and assessed this thesis.

Funding

No part of this thesis would be possible without huge financial funds made available. This research received financial support from Intel Corporation, which provided a grant covering hardware, travel expenses, a living stipend, and a significant portion of my tuition fees. Intel's financial support is definitely the key enabler of my doctoral research adventure. I thank our colleagues from Intel Corporation who provided insight and expertise that greatly assisted in this study. I thank especially Michael Steiner for being our main host at Intel's Hillsboro campus, in collaboration efforts and opportunities, research training, and giving us deep research insights. I thank Simon Johnson from Intel for his expertise, leading the projects and fruitful discussions that provided me a new perspective. Many thanks for all our colleagues at Intel who provided us lectures, trainings, technical discussions, monthly calls, novel ideas and research opportunities, we enjoyed our stays and experiences grately at Portland campus, namely I warmly thank Anand Rajan, David Ott, Brian Mcgillion, Richard Chow, Carlos Rozas, Somnath Chakrabarti, Mona Vij, Matthias Schunter, Mic Bowman, Huaiyu Liu, Claire Vishik, Vinay Phegade, Joseph Cihula, Ioannis Schoinas, Jason Martin, Sridhar Iyengar and other colleagues.

I thank our Computer Science Department for funding part of my studies, and the various teaching opportunities in Oxford that financially helped me. Further, I thank with gratitude for the available UKRI/InnovateUK funds that financially helped me during my DPhil.

Collaboration

Being a student of and then serving as a teaching assistant for David Grawrock for Trusted Computing Infrastructures around five years gave me unique opportunities and moments to receive very valuable feedback, insights, and have very delightful discussions. I thank David with my deepest sincerity for providing me lots of ideas, giving time, and being available in perfect times. His collaboration on a paper widened the perspective of the thesis and enriched the content greatly.

Further, I thank Robin Ankele, Andrew Simpson, N. Asokan, Andrew Paverd, Steve Moyle, Nick Allott, Alexandru Mereacre for our collaborations, publications, coordinations, very insightful discussions, improvements and all of their efforts that is challenging to list here.

Perspective

Throughout my DPhil journey, I was fortunate to encounter numerous esteemed professors and researchers whose wisdom greatly benefited my work. I learned and profited from their knowledge, experience, perspective, vision and publications. I thank Sean Smith (Dartmouth), June Andronick, Raoul Strackx, Dirk Pattinson, my college advisor Kasper Rasmussen, Shweta Shinde, Michael de Jong, Mike Bursell, Paul Kocher, Ahmad-Reza Sadeghi, Virgil Gligor, Adrian Perrig, Srdjan Capkun, Kari Kostiainen, Rajeev Gore, David Kohlbrenner, Michał Kowalczyk, Lok Yan.

Social

During my studies, research and Oxford experiences, there are number of societies, colleges, colleagues and friends contributed my well-being and welfare related matters. I thank my great office mates in Robert Hooke Building 112, Ahmad Atamli, Ranjbar Balisane, Robin Ankele, Yudhistira Nugraha, Olivia Sturrock, Anjuli Shere, Justin King-Lacroix, George Chalhoub.

Many thanks for my friends and colleagues at Cyber Security Centre and the Department of Computer Science; Siddhartha Datta, Frederick Barr-Smith, Jack Sturgess, Ulrik Lyngs, Jaclyn Smith, Matthew Katzman, Andrius Vaicenavičius, Alina Petrova, Stuart Golodetz, Temitope Ajileye, Richard Baker, Katriel Cohn-Gordon, Chad Heitzenrater, Martin Dehnel-Wild, Kevin Milner, Andikan Otung, Yashovardhan Sharma, Martin Strohmeier, Arianna Schuler Scott, Marc Roeschlin, Ivo Sluganovic, Klaudia Krawiecka, Sean Sirur, Hayyu Imanda, Ilias Giechaskiel, Eduardo dos Santos, Anirudh Ekambaranathan.

I extend my heartfelt thanks to all the staff and administrators in my department who contributed to my wonderful memories at Oxford. I thank Wadham College, Sarah Lawrence Programme, Queen's College, New College and its warden Miles Young, St Antony's College, Somerville College, Kellogg College, Christ Church College and Christ Church Boat Club, Computer Science Graduate Society (CoGS) and Joint Consultative Committee with Graduates (JCCG) for making my Oxford experiences unbelievable.

Lastly, I am profoundly grateful to my family and close friends for their unwavering support. Indeed, several more pages could be dedicated to the other invaluable individuals and organisations. Thank you all anonymous contacts that were not mentioned here.

Contents

List of Figures

List of Tables

List of Acronyms

AO Algorithm Owner. 12, 97, 98, 100, 101, 103, 105–107, 114, 121–131, 167, 168

CRC Confidential Remote Computing..... 2–6, 9, 11, 12, 14, 17–23, 38, 40, 41, 135–145, 147, 153–157, 159–162, 166, 169, 171–173, 175, 204, 206, 212

DO Data Owner..... 12, 98, 100, 101, 123, 126, 128–131

DRTM Dynamic Root of Trust for Measurement 20, 27, 29, 35, 66, 158, 187, 209

EDL Enclave Description Language..... 79, 164

EPM Early Private Mode..... 108, 120, 121, 124, 127, 128

HO Hardware Owner. 12, 97, 98, 101, 103, 105–107, 114, 122–124, 126–128, 167, 168

IEF Internal Enclave Functions..... 88, 105, 110, 121

PAL Piece of Application Logic..... 27, 29, 30

PC Loader Protected-Code Loader..... 41, 108, 122, 124, 164

PCR Platform Configuration Registers..... 20, 31, 34, 37, 48, 64, 209

PIEF Public Internal Enclave Functions..... 121–124

PTS Platform Trust Services..... 31

QE Quoting Enclave..... 37, 49, 72, 73, 76, 79–82, 165, 182, 189

RTM Root of Trust for Measurement.. 24, 27, 30, 34, 36, 43, 44, 65, 73, 79, 146, 209, 213

RTR Root of Trust for Reporting..... 27, 43, 146, 162

RTS Root of Trust for Storage..... 27, 43, 146, 158, 162

SGX Software Guard eXtensions 10–12, 15, 16, 20, 27, 29, 34, 35, 37, 38, 40–45, 48–50, 52–54, 56, 57, 60, 63–67, 69–73, 75–79, 81–92, 102–105, 107–110, 112, 113, 115–120, 122, 123, 126, 127, 157–164, 166, 171, 174, 175, 178, 181–184, 187–190, 192, 194–201, 204–206, 209, 210, 213

SIEF Secret Internal Enclave Functions 121, 123, 124

SRTM Static Root of Trust for Measurement 35, 66

SSIEF Serialised Secret Internal Enclave Functions 121–123, 126–131

TaCB Trustable Computing Base 30, 122

TCB Trusted Computing Base 13, 16, 17, 25, 30, 38, 42, 43, 45, 48–51, 53, 64, 69, 70, 72, 73, 75, 79, 83–86, 88, 90, 92, 94–96, 98, 102, 105, 107–109, 111, 113–115, 119–122, 124, 132, 137, 145, 160, 161, 163, 164, 170, 176, 199, 204–206, 209, 210

TCG Trusted Computing Group 26, 157

TEE Trusted Execution Environment ... 12, 17, 33, 34, 41, 42, 44, 45, 50, 63, 64, 73, 100, 102–104, 116, 119, 132, 141, 158, 163, 176, 178, 181, 182, 185, 187, 197–199, 201, 205, 207, 208, 210, 211

TPM Trusted Platform Module . 10, 12, 16, 20, 27, 28, 31, 33, 34, 37, 38, 41, 42, 44–46, 48–50, 52, 53, 57, 60, 63–66, 76, 77, 83–86, 88–92, 141, 157–162, 175, 204–207, 209, 213

TXT Trusted eXecution Technology .. 20, 27, 29, 35, 36, 42, 60, 64, 65, 187, 209

Part I

Introduction and Background

Chapter 1

Introduction

Contents

1.1 Thesis Introduction

The increasing market adoption and availability of dynamically-launched measured applications gained more popularity with the hardware enclaves. New hardware instructions to launch enclaves promised better confidentiality and strong integrity guarantees for building secure industrial applications in commodity systems. System designers and developers proposed many practical use cases to utilise the new instructions of Intel and AMD. These practice areas include applications with edge computing, key protection and virtual hardware security module operations, blockchain applications, data protection in use, and content protection related to digital rights management. The popularity of integrating the new instructions

with daily software applications brought the requirement of good practices of hardware-software co-design.

The research gap between hardware instructions and high-level applications is crucial in designing secure systems. Before any micro-architectural attacks take place, configuration or implementation mistakes can ruin the potential security guarantees inherited from hardware. Semantic errors in hardware-software co-design can be challenging to detect and cause financial damages in corporate settings. This thesis presents peer-reviewed practices, templates and models for building secure systems with the available hardware instructions. Unlike the confidential computing paradigm that moved entirely[1] into the server domain with cloud services, our study focuses on commodity hardware offering a wide range of applications. Our paradigm *Confidential Remote Computing (CRC)*, which differs from the more popular term confidential computing with its threat model.

In confidential computing, cloud vendors are the owners of hardware instructions, and they control it. These hardware owners also build hypervisors, library operating systems, containerised services and an extensive software stack that they fully control. They offer isolated execution on top of a large software stack that remains unknown or too large to analyse. In summary, enclaves operate in the environment of cloud vendors where clients (customers or users of cloud services) must blindly trust what cloud vendors implement and remain potentially vulnerable to the issues of classical cloud computing services. In the confidential computing paradigm, the hardware owners also become the enclave designers and owners, offering services with unmodified applications in large code bases.

Our model with *Confidential Remote Computing* considers hardware owners different from enclave designers and data providers. We offer design principles where enclaves remain small, live short or pre-defined period of time (in contrast to long-running jobs) and perform security-critical operations mainly. Our model brings modified solutions with partitioning and designs smaller enclaves without large

[1] Online. `https://edc.intel.com/content/www/us/en/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/12th-generation-intel-core-processors-datasheet-volume-1-of-2/001/deprecated-technologies/`

operating system libraries. We study the implications of federated computations between different stakeholders. Although they use the same hardware instructions, our model goes in a different direction from confidential computing, targeting daily computers instead of servers. With the release of new hardware instructions from other processor manufacturers, agile enclaves (or isolated realms) may become part of daily computers in the near future.

1.2 Introduction to Confidential Remote Computing

We may trust our mobile devices with a €100 transaction but we do not trust them to buy a house: why is that? There is simply no evidence presented by the underlying systems as to whether they will continue behaving as they promised when the amount in question changes up to millions. These underlying systems vary from the software stack of the mobile devices to the remote servers of the banks. Even the banks themselves cannot trust their own infrastructure to handle large operations in a fully automated way. There is always a limited guarantee on how much of the user assets can be protected. If not, a malicious software at the user's device or remote servers can cause irrecoverable financial damages. **(Q 1) Can system designers and developers enable end-users or other institutional participants to trust a computer to perform critical operations even in the presence of malware?**

Fortunately, a similar question has been addressed in fields other than the computer science discipline. Governments do not rely on other governments' officers to handle their critical operations. Simply, they open an embassy with their own staff operating there, following private tasks and processing private data. How much corruption the surrounding country has does not affect the embassy's operations. It would be ideal to have digital embassies on someone's computer, helping us to trust the remote operations we run there. In computing literature, these digital embassies are identified with a relatively similar term called enclaves. Today, enclaves are revolutionising how computer programmes are designed with security in mind. This

dissertation extensively presents how confidential, remote computations can be built on trusted hardware enclaves.

The problem becomes more challenging when underlying systems of users or companies need to trust each other in the presence of malware. Two mutually distrustful banks may need to perform a fraud analysis on each other's customer data. They may agree on a known, non-secret algorithm to perform a computation over their private data. The question is not only to detect integrity problems (*e.g.*, the computation is maliciously tampered with) but also to offer a secure environment as the computation must be completed correctly after all. There must be pieces of evidence presented to each participant about the execution. **(Q 2) Can an ideally transparent, verifiable, trusted infrastructure serve to tens of thousands of participants within a hour?**

A large number of participants holding private inputs may request the scalable trusted infrastructure to complete a publicly known task in a limited time. One of the next challenging questions is whether this infrastructure can also complete tasks with private algorithms. Besides protecting the private algorithms, the infrastructure must still satisfy the private data providers. **(Q 3) What security risks appear when hardware, algorithm and data owners are not separated in a single computation?**

Upon building an ideal system handling specified requirements, this system must be generalised to serve as a template. Ideally, such systems can be modelled based on their design trade-offs of Confidential Remote Computing, and the recent developments in trusted hardware, secure software development and remote attestation domains can help to realise them. Systematising the technologies and methods may help to extend such systems in future directions. This requires an overview of how the solutions and requirements co-evolved through decades. **(Q 4) How does Confidential Remote Computing evolve through its hardware, software and attestation domains?**

While deploying next-generation applications with CRC, the initial question of *can we operate in the presence of malware?* does continue to be challenging. If

CRC can help us to create secure applications, there is a long-lasting debate around whether malware can also profit from these advancements to become superior. It must be shown that attackers will not be able to abuse the security solutions to create next-generation malware with advantages. Ideally, we do not want to secure the malware with the new security techniques. **(Q 5) How, fundamentally, does Confidential Remote Computing makes benign applications stronger but malicious applications weaker?**

The dissertation of Confidential Remote Computing answers these questions in respective order in the following chapters outlined in Section 1.5.

1.3 Publications and Author's Notes

We list the related publications and explain how they relate to the thesis. Each publication begins with a brief personal note about its contributions to the thesis and its production phase.

1.3.1 In Part I

Some of the content in the background chapter was presented in the following poster.

[1] Olusola Akinrolabu, Robin Ankele, Ahmad Atamli, Ranjbar Balisane, Ravishankar Borgaonkar, Pardeep Kumar, Kubilay Ahmet Küçük, Yudhistira Nugraha, Piers O'Hanlon, Thomas Spoor, Tina Wu, and Andrew Martin. "Trustworthy Systems". In: Oxford Cyber Security Open Day, 2017.

1.3.2 In Part II

Chapter 4 is mostly based on the following publication, where SGX-based TRE was designed, implemented, and evaluated. I was the first author of the paper and took the main role in producing the manuscript.

[2] Kubilay Ahmet Küçük, Andrew Paverd, Andrew Martin, N Asokan, Andrew Simpson, and Robin Ankele. "Exploring the use of Intel SGX for Secure Many-Party Applications". In: *Proceedings of the 1st Workshop on System Software for Trusted*

Execution - SysTEX '16. SysTEX '16. Trento, Italy: ACM Press, 2016, 5:1–5:6. URL: http://doi.acm.org/10.1145/3007788.3007793

The following paper was led by my colleague, Robin. Our dissertation topics were aligned and some of our work intersects. I co-authored the paper whilst Robin made the major contribution and used it in his thesis [3]. I lightly use my part of the content with the use cases.

[4] Robin Ankele, Kubilay Ahmet Küçük, Andrew Martin, Andrew Simpson, and Andrew Paverd. "Applying the trustworthy remote entity to privacy-preserving multiparty computation: Requirements and criteria for large-scale applications". In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ ATC/ ScalCom/ CBDCom/ IoP/ SmartWorld)*. IEEE. 2016, pp. 414–422. URL: https://ieeexplore.ieee.org/abstract/document/7816873

Following paper was also led by Robin. Although I am not a co-author in the paper, I contributed with TRE implementation and measurements. I published the relevant TRE measurements in [2] and therefore mention them in my dissertation.

[5] Robin Ankele and Andrew Simpson. "On the performance of a trustworthy remote entity in comparison to secure multi-party computation". In: *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE. 2017, pp. 1115–1122. URL: https://ieeexplore.ieee.org/document/8029564

1.3.3 In Part III

I presented the following poster at Aalto University. Later, the extended work resulted in the following journal publication [6].

[7] Kubilay Ahmet Küçük and Andrew Martin. "Framework of secret differential privacy on private data". In: ORA, https://ora.ox.ac.uk/objects/uuid:afaa9c13-8630-431d-862a-bf04bf4f4663/download_file?safe_filename=ssg-oxfordv2.pdf. Secure Systems Annual Demo Day, Aalto University, Finland,

2017. URL: https://wiki.aalto.fi/pages/viewpage.action?pageId=117688774

I was the first author in the following paper published in a journal special issue. I made the main contributions and produced the manuscript. Chapter 5 is mainly based on this journal publication.

[6] Kubilay Ahmet Küçük, David Grawrock, and Andrew Martin. "Managing Confidentiality Leaks Through Private Algorithms on Software Guard eXtensions (SGX) Enclaves: Minimised TCB on Secret-Code Execution With Early Private Mode (EPM)". in: *EURASIP Journal on Information Security, Special Issue on Recent Advances in Software Security, Springer* 2019.14 (2019). URL: https://doi.org/10.1186/s13635-019-0091-5

I presented some of the content of [6] in OSEW 2019.

[8] Kubilay Ahmet Küçük and Andrew Martin. "Enabling the Use of Strongly-Private Algorithms". In: Open-Source Enclaves Workshop (OSEW) 2019, Wozniak Lounge - Soda Hall, UC BERKELEY, 2019. URL: https://keystone-enclave.org/open-source-enclaves-workshop/slides/OSEW19_AhmetKucuk_Oxford.pdf

I presented some other content of [6] in CordaCon 2019 with a focus on Conclaves.

[9] Kubilay Ahmet Küçük. "Managing Private Algorithms in SGX Enclaves". In: *CordaCon 2019, R3 Corda Conference DevDay, London UK*. R3. 2019. URL: https://www.r3.com/videos/managing-private-algorithms-in-sgx-enclaves-university-of-oxford/

1.3.4 In Part IV

The overall findings and insights from 2017 to 2020 resulted in production of the following article. Chapter 6 is mostly based on this article. It is released as a pre-print for open access.

[10] Kubilay Ahmet Küçük and Andrew Martin. "CRC: Fully General Model of Confidential Remote Computing". In: OpenAccess, arXiv:2104.03868 PrePrint, https://arxiv.org/abs/2104.03868. 2021.

I presented the following findings to a highly relevant research audience in Intel's community day. The work I presented [11] there later resulted in production of this manuscript [12].

[11] Kubilay Ahmet Küçük. "How infeasible the malware deployment in SGX is in real-life? Can Malware benefit from SGX?". In: Intel's 2nd SGX Community Day. Last Accessed 28 Feb 2021. Intel, Hillsboro, Portland OR, 2020. URL: https://www.intel.com/content/dam/www/public/us/en/documents/research/kubilay-kucuk-malware-infeasibility-sgx.pdf

Chapter 7 is mostly based on this article. It was accepted at HASP, and I presented it in Chicago.

[12] Kubilay Ahmet Küçük, Steve Moyle, Andrew Martin, Alexandru Meceacre, and Nicholas Allott. "SoK: How 'Not' to Architect Your Next-Generation TEE Malware". In: *Hardware and Architectural Support for Security and Privacy (HASP) 2022*. ACM, 2022.

1.4 Thesis Contribution & Research Question

We realise a generic *Confidential Remote Computing* model based on hardware, software and attestation technologies. Throughout the thesis, we answer the question of *how to implement confidential remote computations on trusted hardware enclaves*. We contribute to the field with the feasibility of CRC applications on their security features and limitations. CRC model enables strong data and code secrecy for benign applications. In contrast, we show that CRC does not strengthen potentially malicious applications. We explain our detailed contributions in the following sections. We encourage interested readers to refer to the preface for an informal introduction.

The main contributions of the thesis come with the following research problem:

How can we systematically promote partitioned architectural templates and techniques to offer integrity and confidentiality features for scalable many-party and remote computations by correctly utilising commodity hardware instructions without introducing new opportunities for a stronger malware?

In order to answer the above question, we split the problem into smaller portions.

Experts are able to build many-party applications with cryptographic techniques or hardware support. We explore the new potential opportunities with Software Guard eXtensions (SGX) by assessing its primitives with measurements and providing a novel architecture model to serve as a template for trusted third-party implementations. We present the insights gained in Chapter 4 where we implement a smart grid application on an SGX-enabled processor and provide a comparison with an instance implemented utilising a Trusted Platform Module (TPM). Our deployment addresses a stronger adversary model, and essentially enables a scalable service with tens of thousands of participants where each computation is independently measured for their identity through the enclave's lifecycle. Previously, with the TPM-based instance the attestation protocol (*i.e.*, Final State Attestation (FSA)) has been relying on the same measurement while serving tens of thousands of participants. This issue gave attackers a large attack window, identity mismatches due to time-of-use and time-of-check, and no flexibility to accommodate the code changes (neither a mechanism to reflect any changes to the measured identity). We offer a scalable deployment where an enclave identity is individually measured and attested at each lifecycle, giving us a flexible environment for updates. We implemented a smart-grid use case; the dynamic environments with flexible identities are further useful for block-chain smart contracts.

An enclave binary before the execution is open for inspection by the system admins; it can be reverse-engineered, or the object code can be visible. Once loaded, the application binary would have data segments and code segments of applications split by labels in the memory. Although a memory encryption engine can help to hide these code segments, object code is visible before the memory encryption begins. Whilst data can often be loaded into encrypted memory dynamically at

runtime, there must always be an initial public code visible by reverse engineering. Algorithms implemented in code can also be dynamically loaded, but a novel technique is necessary to prepare and extract dynamically loaded new content. Moreover, even after extracting the new code segments into an encrypted memory, there is one more problem. If the adversary controls the execution environment, she can run the code many times and learn how it processes the private data. The algorithm owner must be able to protect the application logic by enforcing their execution policies. We consider algorithms as semantic behaviour of black-box systems where an algorithm (*i.e.*, a way of describing how a task proceeds) contains code blocks. Our solution in Chapter 5 is not only interested in protecting the code segments, but also in limiting the semantic leakage of an algorithm by measuring the output. We, therefore, use the wording as protection for the *private algorithms* in this thesis rather than only encrypting the code segments. We acknowledge that not every algorithm is implemented in code or contains a piece of code, and a piece of computer code may not necessarily contain an algorithm to be protected. We make novel contributions with templates for algorithm partitioning in enclaves, and an early private mode to prepare the payloads and the host enclave. We present techniques and templates for secret code execution where dynamic loading leads to industrial services with multiple stakeholders. During our research on the theme, the problems were marked as future work by Intel. Today, similar techniques with similar acronyms to ours can be found in Intel's SGX SDK in a standardised manner. The author received a PhD studentship from Intel for the part of the work presented in this thesis.

After the collection of techniques and templates we present in earlier chapters of the thesis, in Chapter 6 we provide a novel overview of past techniques to systematise our work as a consolidated paradigm of Confidential Remote Computing. The published and presented work of system models, architecture models, adversary models, communication flows and templates throughout the thesis form the basis of the CRC model. Besides the practical implications presented throughout the thesis, CRC stands at the largest ring of the X-chart presented in Section 6.5 where the

extension of the domains may generate new paradigms in the future. By standing at the largest ring of Figure 6.4, CRC is the model using secure loaders, enabling frequent updates, considering multi-stakeholder scenarios. We provide an index of our modelling throughout the thesis in Section 6.1.

Under the systematisation part of this thesis, we continue with the answers to concerns (before they become widely believed myths) about whether hardware assistance can help malware developers abuse the security features to create a more advanced threat. We provide a systematisation of the knowledge in Chapter 7 to evaluate enclave-based malware and malware in the wild (non-enclave-based malware).

- Contributions of Part II (Chapter 4):

  Our principal contributions in Part II are: Empirical performance measurements of basic SGX operations that are used in many-party applications (Section 4.3), an architectural design and prototype implementation of a representative real-world application, which could serve as a template for other such many-party applications (Section 4.5), a systematic comparison of an SGX-based TRE[2] against a previous TPM-based system, in the context of many-party applications (Section 4.6).

- Contributions of Part III (Chapter 5):

  The novelty and contributions of our work are as follows: We consider mutually distrustful entities[3] *(HO, DO, AO)* with conflicting interests in the cloud, and we differentiate the private algorithms and the private data. We show the flawed practices while utilising Trusted Execution Environment (TEE) in the cloud, and we create a taxonomy for the secure execution of private algorithms in untrusted remote environments. We provide practical insights into enclave development and debugging and perform a security analysis on existing dynamic code loaders with interpreter enclaves. We evaluate our execution model in three adversarial settings in the cloud.

[2]A disclaimer and a review on TRE can be found in Sections 3.4 and 3.6.
[3]Defined in Section 5.1.

– The Motivation for and Aims of Part III:

The motivation for Trusted Computing Base (TCB) minimisation comes from the secrecy guarantees that depend on a remote system's TCB components. There are a number of ways in which developers can cause security problems in a system. It is common for developers to include third-party software in their enclave TCB. Unfortunately, they often fail to perform security and compliance analysis between the third-party package and the underlying hardware. Additionally, developers may fail to understand hardware and software co-design whilst constructing secure systems. Careless construction of composite parts within a TCB may also cause the loss of initial security guarantees of the hardware. This may be the main, sometimes initial, source of security problems. In this chapter, we analyse two existing frameworks that address client-side secret code execution, but their TCB suffers from bad practices as explained below. To solve these architectural design problems, we present a solution with a smaller TCB, and secure TCB composition for secret-code execution in remote environments.

We aim to eliminate these bad practices, summarising the motivation for this chapter in three flaws:

* **Increase of TCB Size**: The TCB size must always be minimal in order to avoid security risks, and enable possibility of formal verification.
* **Weak software in the TCB**: The third-party software packages included in a TCB must pass the security requirements of all the assets at stake.
* **Non-Compositionality of Security**: Two secure components of a system may not necessarily comprise a single secure composition. Even secure software in the TCB may create additional security issues due to composition problems with the underlying hardware. This may also void the hardware security guarantees.

- Contributions of Part IV (Chapter 6 and Chapter 7):

  This chapter expresses the most comprehensive model of *Confidential Remote Computing*. The *Confidential Remote Computing* paradigm brings new opportunities to the computing world. Our model aims to provide a structured view of the technologies and methods behind the digitalisation trend of the concept of trust. Throughout the chapter, we make the following contributions with the novel *Confidential Remote Computing* model:

  CRC resolves the ambiguity of *trust* in relevant domains, and we connect the former understanding of *trust* in computations (local and certificate-based) to future computing models (with separated three entities; cloud providers, data providers and algorithm providers). We provide an extensive analysis on how *digital trust* can be derived from alternative methods such as micro-kernels, software-based attestation, and verification technologies, followed by the use of the hardware enclaves in the grid, edge and fog computing use cases. We consolidate the CRC model in the X-chart in Figure 6.4 in three domains; hardware, attestation and development, presented in Section 6.5. Each of these domains is surveyed and systematised with applied techniques and methods in the literature. We present the trade-offs in the CRC model, between the larger task size, more decentralisation cost and more transparency overhead. We conclude the chapter with a unified architecture utilising the analysed technologies, and present the lessons learned alongside the future research directions.

  - Position of Part IV Chapter 6 in the Thesis:

    This chapter provides the most generic form of the insights and the digital enclaves developed in Part II (with Public Code and Private Data) and Part III (with Private Code and Private Data). The importance of the systematisation presented is that it provides a novel model for the evolution of trust in history for *Confidential Remote Computing*. It shows how the perception of *trust* is adapted through the co-evolution

of the requirements and solutions. Each time a new stakeholder joins a computation, requirements and solutions evolve into a new stage. The final abstract model of the unified architecture is presented in the chapter as a basis for future research.

- Contributions of Part IV Chapter 7:

  This study presents the most comprehensive collection of misconceptions about malware in enclaves to date. Our novel contribution is that we systematically show why malware becomes weaker due to trusted hardware. We also clarify the ambiguity of *malware in an SGX-enclave* and *malware in the core SGX ecosystem*.

1.5 Brief Structure of the Thesis

We make three major contributions organised in three main parts containing one or more chapters. Parts of these chapters (from Chapters 4 to 7) correspond to published and pre-print papers.

After Part I with the introduction and background, the second part of the thesis presents a system processing private data with public code. Then the third part goes one step ahead and presents a system built with private code capabilities in addition. In the fourth part, we systematise and model our findings to a generic form. The last part contains our detailed conclusions and notes on future research directions.

- Details of Chapter 4:

  In this chapter, we explore the use of Intel SGX [13–15] to implement a TRE[4] for many-party applications. Firstly, we compare SGX with previous trusted hardware technologies in terms of specific characteristics required for many-party applications. In particular, we consider attestation performance in terms of the time required to perform a single attestation operation [16] and the overall rate at which a platform can perform attestations.

[4] A disclaimer and a review on TRE can be found in Sections 3.4 and 3.6.

Secondly, we design and implement the equivalent of TPM-based TRE using SGX for the smart grid use case. Due to the fundamental differences between the architectures of SGX and previous technologies, the implementation is far from a straight-forward porting task, and requires a complete redesign of the system. Thanks to the SGX architecture, our implementation requires significantly fewer lines of code, which both reduces the burden on the developer, decreases the likelihood of code defects, and minimises the amount of code that must be trusted by the verifier. However, the use of Intel's trusted libraries (*e.g.*, for cryptographic operations), which are only provided as closed-source binaries, makes it difficult to inspect and quantify the exact size of the software TCB. Although our implementation targets a specific application domain, we argue that its core features are common to all many-party applications, and thus could serve as an architectural template for such applications.

Finally, we perform a comparative evaluation of the performance of our SGX implementation against the previous TPM-based implementation. Using the smart grid as a case study, we benchmark the systems' end-to-end performance in a representative communication task (*i.e.*, obtaining the most recent consumption measurements from a smart meter). The results show that even an unoptimised SGX implementation exhibits comparable performance to the optimised TPM-based system, whilst addressing a stronger adversary model.

- Details of Chapter 5:

  Section 5.1 defines the taxonomy and ownership model, and Section 5.2 defines the problem of private algorithms. Section 5.3 provides a background on current binary execution mechanisms, as well as responsibilities of application developers for the chosen development model. Section 5.5 explains the trade-offs between design choices in TCB for private algorithms. We analyse the TCB components of existing frameworks with practical attacks against code

secrecy in Section 5.6, and Section 5.7 explains the bad practices in enclave design and development. Section 5.8 shows our design for secret code execution with reduced TCB size. Finally, Section 5.9 demonstrates our method in practice with three use cases of private algorithms for industrial use.

- Details of Chapter 6

  In this chapter, we attempt to consolidate the disparate research approaches into one *Confidential Remote Computing* model, from the past to present. We begin by explaining the participating entities in chronological order, divided into five subsequent stages in Section 6.2.3. The model then is structured through three main portions. First, we begin with the role of hardware technologies in *Confidential Remote Computing*. Then we introduce our model demonstrating the orthogonal research structured by each entity presented in five stages in three domains. Lastly, we demonstrate the key trade-offs, providing an overview of the deployed systems.

- Details of Chapter 7:

  We structured this chapter into four main categories. First, we describe the frequently seen characteristics of an ideal malware. Second, we demonstrate existing malware evasion techniques and high-scale, effortless malware distribution techniques. We show non-TEE malware evasion techniques and a delivery campaign for two reasons; (1) to define the assumptions and review the scope of malware detection, (2) to demonstrate a real-world scenario on scalable infection to give readers an understanding of malware in the wild. Third, we systematically evaluate twelve misconceptions about malware in TEE and present why these myths are far from the truth in practice. Finally, we compare malware in the wild with enclave-based malware and see if utilising enclaves provides any additional benefits to malware in practice.

1.6 Motivation and Closing Remarks

We briefly introduce the idea of CRC in this chapter. In the next chapter, we will present a literature review on *Trustable Computing* prior to CRC.

The notion presented throughout the thesis as *Confidential Remote Computing* will be presented in more detail in Chapter 6. To simply define to give readers an understanding, confidentiality refers to the secrecy of data and algorithms [5], remoteness refers to a physically remote location from the participants and implies the *integrity* requirement (integrity is enforced by the hardware instructions), and the computing refers to the turing-complete computational abilities[6]. Developers and system designers can utilise the available hardware technologies in various ways; we present peer-reviewed practices of using hardware instructions under the notion of CRC. Section 1.2 and Section 1.4 include answers to the question of why would the methods used under CRC notion be helpful. In contrast to the majority of other researchers (*i.e.*, instead of attacking these hardware instructions), we present novel practices and methodologies on how to utilise the hardware instructions correctly.

The pre-existing idea of confidential computing is different than CRC, as it targets server environments only. What we are adding to the overall literature is support for many party applications (Chapter 4), support for private algorithms (Chapter 5), and comprehensive systematisation of CRC (Chapter 6) and its malware arguments (Chapter 7). Confidential computing is mainly built and supported by major technology vendors. They use the hardware instructions to offer a better form of traditional cloud computing. It is mainly built with a vendor-controlled software stack managing all critical assets, which leaves us questioning how to trust these vendors to do the right thing. The original design idea of isolated enclaves was to remove the need to trust other companies, such as cloud providers. In CRC, we revert to original design principles and advocate for small enclaves

[5] Details will be explained in Chapter 5

[6] This is not to say turing-completeness is always a good capability in designing security systems. What we mean is to support complex operations in contrast to limited cryptographic protocols. The author is aware that turing-incomplete languages may be more beneficial for enclave programming. The final decision is to be made by the architects.

running in end-users computers. For example, the idea of confidential computing does not support enclaves running in end-users computers anymore; they focus on servers and cloud infrastructures only. Confidential computing considers big vendors to carry the computing in their *local* infrastructures without offering remoteness. It does not distinguish multiple stakeholder scenarios. CRC adds practices and templates for multiple stakeholders, such as different data owners, hardware owners and algorithm owners, presented in Chapter 5. Confidential computing does offer support and practices for servers but not for end-users with their own hardware. CRC extends the broad notion of confidential computing by adding significant additional valid practices and methodologies for individual use. We present more related work on CRC in Section 3.1. We strongly believe that our research will have more significance with the rise of RISC-V processors and ARM support for isolates/enclaves in the upcoming years. These processors will highly likely make the enclaves/isolates available to be programmed in end-user devices. Although the popular term is "Confidential Computing", the security concerns go wider than this.

Readers of this thesis may raise many other questions, as experienced colleagues[7] immediately did whilst reviewing. The author would like to try answering frequently asked questions[8] here in the introduction to help readers better understand the fundamentals and the motivation.

- **Question 1:** Where does Confidential Remote Computing come from, and why did we need this paradigm?

- **Question 2:** How is the content of the thesis structured, and why is it structured in its current form?

- **Answer 1:** Before the public availability of trusted hardware enclaves, the research community had been suggesting software security architectures; for

[7]This introduction may contain some terms or technologies unknown to the readers; these terms are defined throughout the thesis where appropriate. The readers may come from a broad spectrum of expertise, and this short introduction aims to answer the potential (feedback of draft reviewers) questions of field expert readers.

[8]These are not the research questions, see the next section for research questions.

example, Strackx *et al.* systematised them as *Protected Module Architectures* (PMA) [17]. Before they became more systematic as PMAs, early proof of concept studies and running prototypes came with a novel use seen in *Flicker* and *Trustvisor* [18, 19]. These studies utilised Trusted Platform Module's Platform Configuration Registers (PCR) for Dynamic Root of Trust for Measurement (DRTM) and Trusted eXecution Technology (TXT) to create isolated and attestable memory regions for critical parts of commodity software. The key idea was to perform critical operations in an isolated, secure environment and revert the system state. This enabled applications to perform security-critical operations even in the presence of malware in a system. Thanks to early studies in Flicker and Trustvisor, the results influenced the design of trusted hardware enclaves initially with Intel's Software Guard eXtensions. While SGX gained sudden popularity and market availability, too few of the enclave research community acknowledged the insights of these early studies. Further, the research community pushed enclaves to offer security hardening for their virtual machines, loading very large and unverified operating systems and libraries to them. They treated enclaves as if they would still hold the security guarantees while disregarding the initial threat model and assumptions. The announcement of Trusted Domain eXtensions from Intel might be the reason to split these use cases of secure virtual machines from secure enclaves. Pushing large operating systems (or most of the kernel functionalities) into the enclaves had another reason: marketing. It offered startups and cloud service providers a good business model, claiming that data will be secure in-use, as well as at-rest and in-transit. The direction of the community's consortium may recreate past problems, and make enclaves unusable for the commodity software. The community should not forget the initial reason why commodity software solutions needed the enclaves.

With these two reasons (and many other reasons to be seen throughout the thesis), the CRC reverts the original design principles.

- **Principle 1:** Keep enclaves as small as possible and show architectural design practices while offering data security and algorithm security (explaining why the authors chose the term *algorithm* over the *code* in the relevant section).
- **Principle 2:** Keep the control decentralised and show why cloud service provider should not be the enclave developer at the same time, as covert channels will nearly be impossible to avoid (diminishing or entirely destroying the security features expected from the hardware extensions).
- **Principle 3:** Remember to deploy capabilities to trust, measure and attest the kernel, omit the assumptions that the hardware will defend against attacks placed by the malicious operating system, and establish as many security mechanisms as possible in your kernel (highly likely to be a formally verified one).

The author would like to highlight by listing these principles that the CRC is a paradigm for building practical solutions available to end-users, system designers, developers and small-medium enterprises other than tech giants.

The CRC is neither the above principles only nor one of the chapters in this thesis. Besides the system, communication or threat models described, the CRC model itself is the understanding of the principles in designing a system architecture with trusted hardware enclaves. This thesis shares the author's observations, experiences and recommendations and promotes templates and techniques. Although the majority of the content is peer-reviewed, published, presented, or some to be published, some of the content may contain limited knowledge or unintended errors; please feel free to contact the author in such cases.

- **Answer 2:** Today's reality is that only a few industrial products utilise the available security features. Enclaves or their design are not fundamentally broken; the community often chooses the monetised path of developing their

systems under the umbrella of unmodified apps, towards the direction of centralisation. Practically, forcing unmodified apps to run in partitioning-encouraging development architecture as native is the root cause (problem) of never-ending projects. What a library operating system in an enclave gives us is an environment to run unmodified apps; it also enables accommodation of malware and therefore must provide a two-way sandbox as well. After all, it gives us more vulnerabilities and a larger unverified software. It may either end up becoming a malware distribution hub, or a vulnerability hub. Isn't this similar to where we have formed our research? What a large enclave would not give is a clean, stateless, transparent, verifiable and finally a software stack utilising the offered hardware security features correctly.

The thesis talks about malware in two different aspects.

- **Aspect 1:** Typical malware in the threat model that reads the secrets in the memory.
- **Aspect 2:** Superior malware that abuses the promising security solutions. We argue that if benign software products can utilise the enclaves, then why cannot malware hide itself through enclaves? What do they need to achieve their malicious goals?

Starting from these points, the thesis initially shares the author's experiences in protecting private data processed by known, publicly available algorithms. In the later part, the author explores what would happen if we process private data with another party's (a distrustful participant) private algorithm. The structure continues with the systematisation of CRC, aiming to bridge past experiences and give a template for future applications. Finally, the author collects the research community's arguments, some raised during community workshops, and some appeared in publications.

Our threat model excludes micro-architectural attacks. The use of hardware enclaves has brought more attention to lower-level attacks. Many researchers

recognise that hardware specifications are not matching the hardware implementations. The threat model has to have its limits, and it is never possible to address the strongest adversaries. Microcode updates happen at every boot cycle and the limit in CRC is at the ability to patch the trusted computing base once a vulnerability is known, published and patched.

ARM TrustZone has made its security architecture available for several decades. With the rise of the community's consortium, ARM may also offer practical enclaves besides their *secure world* memory regions. We anticipate that the implementation outcome and design choices do not cause ARM enclaves to suffer from large, unusable, malicious or vulnerable, not verifiable software stacks. They may integrate an underlying verified kernel with attestation capabilities (non-binary, behaviour and properties) together with the small, functional, read-only patterned, stateless enclaves handling mission-critical key operations with a well-defined life cycle. The behaviour of the enclaves must remain as attested and consistent, ideally long-running jobs avoiding identity mismatches.

Trust: An entity can be trusted if it always behaves in the expected manner for the intended purpose. ... and the entity can be unambiguously identified, operated unhindered and a user has experience about the consistent and good behaviour of the system.

— TCG, G. Proudler (C. Mitchell, 2005, ch 2)

Chapter 2

Literature Review on Trustable Computing

Contents

2.1 Digital Trust: Trusted, Trustable, and Trustworthy Computing

Trusted computing and relative terms have been iteratively defined in the literature over several decades. While the term *trust* is redefined with a better meaning, trusted or trustworthy computing meant different ways to derive trust, *e.g.*, trust based on hardware guarantees, or trust through formally verified software.

After initial efforts by a task force in 1967, the US Department of Defense introduced a set of requirements for trusted computer systems under four security levels—

interested readers are referred to the orange book [20] (1985). We use the following three of their definitions here to familiarise the readers with related concepts.

> **Trusted Computer System** - A system that employs sufficient hardware and software integrity measures to allow its use for processing simultaneously a range of sensitive or classified information.

In this definition a single computer would ideally satisfy the requirements and be trusted. In today's systems, we expect multiple trusted components (*i.e.*, isolated environments) running on an untrusted hardware; the recent perception creates a difficult threat model with physical attackers.

> **Trusted Computing Base** - The totality of protection mechanisms within a computer system – including hardware, firmware, and software – the combination of which is responsible for enforcing a security policy. A TCB consists of one or more components that together enforce a unified security policy over a product or system. The ability of a TCB to correctly enforce a security policy depends solely on the mechanisms within the TCB and on the correct input by system administrative personnel of parameters (*e.g.*, a user's clearance) related to the security policy.

The TCB definition mainly refers to the protection mechanisms. While this definition holds for most of the systems, TCB can refer to the piece of user-level software processing trusted computations. Any piece of software processing user assets may be considered within the TCB in today's commodity systems. A clear distinction in TCB can help to minimise the exposure. Otherwise, infected TCBs with the ability of handling critical operations can be used for deploying bigger attacks. For example, Winkler and Gomes *et al.* report over one million botnets in 2017 [21].

> **Trusted Software** - The software portion of a Trusted Computing Base.

We use the similar term in the latter chapters as the hardware TCB and the software TCB. A system may consist of the software TCB only, for example, with a formally verified kernel delivering the required security properties [22].

> An entity can be trusted if it always behaves in the expected manner for the intended purpose.

The behavioural definition of trust above is an earlier attempt by HP Labs (2002) [23] and Trusted Computing Alliance (TCPA 2003) [24]. TCPA was later succeeded by the Trusted Computing Group (TCG). The definition is often cited as (TCG 2004) [25], intending to refer to the original 2002 definition.

> **Trust** - Trust is the expectation that a device will behave in a particular manner for a specific purpose.

Although TCG's 2012-dated glossary excerpt above defines the *trust* similarly[1], we refer the readers to the definition updated by the same group of authors as follows in 2005:

> - it is safe to trust something when:
> (it can be unambiguously identified)
> (it operates unhindered)
> (the user has first-hand experience of consistent, good, behaviour) or
> (the user trusts someone who vouches for consistent, good, behaviour).

Proudler *et al.* extended the definition as in the above excerpt in 2005 [26, 27]. It requires an ambiguous identification of an entity, but the definition of the *experience* remains ambiguous due to multiple practical implications. In Section 2.2.3, we shall take a closer look into the practical expressions of the *experience*. There are several alternatives to define the experience. For example, the concept of *measurement* can refer to a piece of observation, an experience of a behaviour; however, it is often limited to static snapshot of a binary without revealing much about the behaviour of the measured software, we shall explain why in Section 2.2.3. A better way to express the behaviour can be to attach the log file of operations taken for a certain task to allow an external party to observe the steps. Another way is to focus on generic properties of a software where the precise behaviour does not matter. Similarly, a permission-based behaviour can also be an alternative where a sandbox enforces certain behavioural rules or policies. Section 2.2.3 shall discuss these different types of evidence.

[1] `https://trustedcomputinggroup.org/wp-content/uploads/TCG_Glossary_Board-Approved_12.13.2012.pdf` (It is worth highlighting that newer documents still do use the older definitions). Online, last accessed 15 Jan 2022.

2.1.1 Related Work in Trusted Computing

We intend to keep this introduction brief and refer the interested readers to the valuable resources. Smith provides a discussion about the devices *worthy of trust*, the ability to *choose to trust* and with the means to *communicate* their trustwortiness [28]. Martin presents a ten page introduction to trusted computing, presenting valid insights for today [25]. Although there are many other[1] or older resources with these definitions, we refer to the resource [25] for the definition of the concepts such as Root of Trust for Measurement (RTM), Root of Trust for Storage (RTS), Root of Trust for Reporting (RTR). A relatively recent book [29] by Bursell covers the past and present state of trust with hardware assistance, targeting a non-academic audience as well as industrial practitioners and also goes into the enclave-based computations. From 2008 to 2016, Springer's Trusted Computing conference series provided a decent collection of publications [30–37]. A few practical resources on TPM are present only, while there are many theoretical papers and specifications; Arthur, Challener, and Goldman fill this gap [38]. Parno, McCune, and Perrig's introduction to trusted computing focuses on understanding the platform state and presents the solutions on bootstrapping trust in commodity hardware with use of Dynamic Root of Trust for Measurement (DRTM) with Flicker [18] and Trustvisor [19]. More details of DRTM and Intel's Trusted eXecution Technology (TXT) (previously LaGrande) in depth are presented in Grawrock's book [40]. It is one of the earliest resources with the concept of enclave-based computations with DRTM, distinguishing the concept from network enclaves (similar to isolated intranets). Grawrock's Trusted Computing Infrastructures (TCI) course [41] is one of the rare university-level taught courses in the field. The design principles and the challenges in DRTM, TXT and the novel method [18, 19] on protecting Piece of Application Logic (PAL) later resulted in Intel's new instructions implemented in CPU's microcode (xucode, to be specific), thereafter the 6th generation Skylake architecture. These new instructions are presented with the technology called Software Guard eXtensions. SGX's new security-focused computation model brought new possibilities and challenges which we discuss in the rest of this thesis.

2.1.2 Emphasizing the Trustable Computing Paradigm

Previous discussions and attempts on the definition of *trust* is presented in Section 2.1. We further clarify the understanding of *trusted computing* and *trustworthy computing* used throughout the work we present. Through these, we emphasise the definition of *trustable computing* and build our work on top of this paradigm[2]. There have been arguments as to whether *trusted* was a wrong word to choose as it may imply something blindly trusted, rather than having a chance to make the trust evaluation. We argue that it is not a wrong statement; it has a valid and current research direction where systems/notions must be trusted and where trust would not be open for evaluation; those systems cannot continue the execution otherwise. Without the trusted computing primitives, it may not be possible to build systems to offer trustable services. Nevertheless, we emphasise *trustable computing* where it forks from *trusted computing*, and further technologies may be diversed to refer to the meaning with more precision. We shall give an example how this can be done in Section 2.1.2.1. The trust evaluation, auditing process, and derivation of the digital trust is further discussed in Section 2.1.3.

Trusted Computing refers to the systems that must be trusted for a valid execution. The trust evaluation may seem heavily backwards. The assessment or the auditing on the given evidence, reports, measurements, logs against some expectations or manifests help to reach to judgements about the "past" incidents, behaviours. If the evidence is not matching the expected values, the execution may halt. In trusted computing, the user wants trust their own device.

Trusted Computing in Practice: TPM is a trusted component in a system (often an external chip attached to the motherboard) which introduces notions trusted by default; operations relying on TPM cannot be valid otherwise. The question of why the TPM is trusted is explored elsewhere. The verified boot is done by checking the booting image with set of trusted signatures or expected hashes; the system cannot boot if there is a mismatch in configurations or hashed

[2]A note for readers would clarify the aim; our goal is not to redefine what has been defined, but rather to express the understanding of the author to align the readers with what to expect.

values. The measured boot follows a similar path, but the system can continue to operate without verifying the measurements. The previous evidence collected would tell about the past state of the system, without guaranteeing it to behave as identified in the future also. The guarantee that a trusted system would continue to behave correctly is explored in other disciplines. One such direction is *Trustworthy Computing*, which we shall explain next.

Trustworthy Computing refers to the systems relying on proofs generated in advance about a system to reason about future, probable behaviour or properties and to match the given specifications. The convention is often used in the area of formally verified pieces of code, systems, and protocols.

Trustworthy Computing in Practice: Trustworthy systems are more frequently desired in the embedded systems world. Formally verified kernel projects provide trustworthy environments for properties such as isolation, *e.g.*, seL4 [22] aims to implement a verifiably correct system in which there should not be any errors. Another project, *Singularity*, aims to construct a verifiably safe system where the kernel should fail safely in case of an error.

Trustable Computing refers to systems with a variable trust level, where a relative trust evaluation and decision takes place before proceeding with the execution. Trustable computing can differ from trusted computing by the participant types and machine ownership. In contrast to trusted computing, the user wants to trust an operation performed by another person's computer. The remoteness makes the problem relatively more difficult, or in other words it addresses a different and highly likely a threat model considering a more powerful adversary.

Trustable Computing in Practice:

The systems using isolated, dynamically allocated memory regions to perform trustable operations for remote participants, such as PAL supported by DRTM and TXT, or enclaves backed by SGX, offer trustable services. Trustable computing provides the architectural blocks to enable such services to be built on top of given hardware or software notions.

2.1.2.1 Trustable Computing Base

TCB can often be used to refer to the implementation of trusted computing technologies in the microcode or in a chip. What actually goes into the TCB in a system can be a long discussion to explore elsewhere. If the microcode is not up to date, certificates issued for an older/outdated version of the microcode would be revoked; the system would not be trusted by default. While certificates are revoked, there would be no way to trust such a system. Once the system has the valid trusted components (mostly referring to the static parts), further, a dynamic piece of application, *e.g.*, a PAL, is often considered as part of the Trusted Computing Base too. The attestation report of the PAL or an enclave, however, is not always expected to be trusted by every attesting/verifying entity. While one verifier chooses to trust a version of an enclave for a specific operation, another verifier with a more strict policy may not trust it, or there might be time-based differences of the decision due to policies. This is where the computation becomes *trustable*, not always necessarily trusted. The memory segments for a PAL or an enclave, therefore, can be better expressed as **Trustable Computing Base (TaCB)**. Smith expresses the typical definition of the TCB as *the minimal component that one must trust, because one has no choice* [42]; in contrast, the TaCB represents a relatively dynamic, agile computation environment without a necessarily binary-based identity. The late-load mechanism presented in Section 5.8.2 is an example for the TaCB. For example, an enclave is measured by the Root of Trust for Measurement at their creation enabling them to have an identity. But their identity changes as soon as the computation begins, while underlying system components such as Root of Trust for Measurement are trusted (*e.g.*, microcode version remains same throughout the computation; counting towards the trusted computing base). The enclave remains as the trustable software component through its lifecycle. Trustable computing does not replace trusted computing; without the trusted computing primitives and notions, we may not be able to build the trustable computing paradigm. Trustable computing notions may still be seen under the *trusted computing* paradigm, only with a less precise expression.

2.1.3 Derivation of Digital Trust

Digitalisation of trust is derived through various means. It is often observed between two abstract entities: a prover intending to convince others, and a verifier to check or evaluate. We provide a few scenarios to explicate the process.

Suppose that a TPM owner reports the final Platform Configuration Registers values (calculated through many one-way append operations), and in addition the verifier expects the stored measurement log (unshielded). This way the verifier can check the report against the log for integrity. An inconsistency would make the prover dishonest. The specific operations between the verifier and prover and more of the TPM-based attestation procedure is bound up in Platform Trust Services (PTS) [43–46], with features of manifest and report generations from logs and verification of reports[3].

In an electronic voting scenario, the mixer (of a mix net) is expected to shuffle/permute a secret input vote of x. Together with the result value, the prover provides a log file of how the permutations were done (where backwards computation is not possible). The input owner can check that the result value is correct by applying the same permutations. This would allow a voter to obtain the *casted as intended* property; full end-to-end e-voting schemes would also require *stored as casted* and *counted as stored* properties. Haines, Goré, and Sharma bring further formal proofs to verifiable mix nets [47] that the mixing was done correctly as promised, without intentional or unintentional errors. In both ways (and in many other use cases) with proofs or measurements, we observe a public (unprotected) log file of steps and the result value (shuffled or appended), and the verifier is expected to check if the remote system behaved correctly. This procedure helps to evaluate the integrity of a system or operations.

The more generic term of *audit-trail* described in 1988 by Merkle is used in one-time signature trees based on one-way functions [48] enabling today's blockchain systems. We refer interested readers to the tan book of the rainbow series from DoD [49] on auditing of trusted systems. The technical auditing with ethical

[3] `https://osdn.net/projects/openpts/wiki/FrontPage`

aspects, including functional auditing (intended behaviour, good or bad), process auditing and code auditing (validity, verification) is described by Mökander et al. on trustworthy *automated decision-making systems* [50–52]. These studies relate to our arguments on differentiating algorithms from a piece of code in Chapter 5.

2.2 Hardware Assisted Trust

The trust level of a system may vary depending on the underlying mechanisms used to build it. The security requirements based on a system's threat model would impact the architectural design choices; we introduce specific capabilities, scenarios and assumptions to describe the scope of our research. In short, our research targets trust and security in commodity systems (general-purpose computing platforms). Solutions and challenges for embedded systems and Internet of Things can be explored elsewhere [53]. Software-only root of trust and software-based attestation solutions offer limited guarantees while requiring strong assumptions for embedded devices. In practice, they address a weaker threat model only; interested readers are referred to facts and fiction on SWORT and SWATT [54]. Cryptographic protocols in multi-party computation and homomorphic encryption schemes offer strong security guarantees; addressing a stronger threat model. However, the current solutions offer either limited computational operations or a limited number of participants in many-party computations [5]. A Trusted Third Party (TTP) can also handle some critical operations between mutually distrustful parties. In many party applications (or, in multi-party computations) the number of participants may be hundreds of thousands, and these operations may include set-membership problems [55] or set-intersection problems. Solutions to these problems are helpful, for example, in fraud detection between mutually distrustful financial institutions. High number of private ledgers may need to investigate the intersecting value from their private transaction databases without disclosing the databases to each other, this is a set-intersection problem. In this thesis, we offer scalable many party applications in Chapter 4, and the template we present can include application logic of similar problems. These operations are often publicly known and agreed between participants, and the

processed data remains secret. Fully-homomorphic encryption schemes and secure multi party computation protocols solve the same problems [56, 57], but they allow less complicated operations or work with a smaller scale of participants.

Hardware assistance plays a crucial role in enabling scalable, trustable services in commodity systems. In our work we mainly focus on TPM and TEE. Aside from these we briefly refer to Physical Unclonable Functions (PUF), smart cards, and Hardware Security Modules (HSM) used to establish hardware based security in today's systems. PUF rely on the natural randomness of manufacturing of hardware devices, providing a unique, inexpensive source of identity and resilience to various physical attacks [58, 59]. Tens of billions of smart cards in use today are providing persistent storage to applications serving end-users; the data is protected by a small CPU with the capabilities to count and to require a PIN. Interested readers are referred to the survey on memories of smart cards [60] including various details of memory attacks and security requirements. Neve et al. call a smart card a *Personal Trusted Device*. The smart cards are dependent on terminal devices and can be potentially vulnerable to man-in-the-middle attacks and to other protocol or implementation based issues. It is not clear which entity (end-users, merchants/institutions or smart card issuer) would be responsible for damages in case of attacks or breaches. For example, a card brand mix-up attack can allow a MasterCard or Maestro card to be used in Visa transactions without requiring a PIN [61]. HSMs often serve in the financial cryptography domain to act as a secure co-processor, assisting systems with cryptographic operations and key management services in data centers. They are flexible to attach in enterprise settings. Smith's review provides insights including physical security aspects, attacks and resilience on HSMs [42]. We may expect TPMs and TEEs to replace HSMs in many business use cases, as for example there are attachable TEEs through PCI cards. Having solely a special hardware to implement a system with certain security guarantees is not enough. TPM, TEE or HSM can be secure by themselves, but using these hardware requires careful configuration and software integration. Hardware-software codesign for offering security products is not optional; it is a must. This requirement

is studied more in-depth with the notion of non-compositionality on inheriting or propagating the security features between protocols, software or hardware. Many researchers have previously studied Compositionality, especially in the domain of Security of Protocols [62, 63]. Cremers also addressed several issues about security protocol composition when a protocol employs many sub-protocols. We revisit and present our insights on the non-compositionality of security in Section 5.7.1. We show that composing SGX instructions with other existing software fails to offer security benefits. Section 6.3 presents a detailed overview of the role of different kernels and the hardware.

2.2.1 Secure Storage with Sealing, Binding, Quotes and Reports

Sealing and binding operations provide secure storage for data. A key difference between sealing and binding with TPM is that in sealing operation, the data is only available to a specific TPM, whereas in binding the data can be accessible by multiple TPMs as long as the binding key can be generated based on the system state (authorisation policies, PCR values). With TEEs, specifically SGX, keys generated based on an enclave's identity can be used to bind data to a specific software binary. This feature would be similar to TPM's binding capability. Two different software binaries loaded in enclaves can also verify that they are running on the same hardware CPU through local attestation with enclave *report*s. This can allow binding data for two enclaves.

2.2.2 Static and Dynamic Root of Trust for Measurement

Historically manufacturers would ship a platform from the factory and claim that it met the properties without a re-measurement. In static measurement, the platform would be measured at every reset; giving a clean state. From BIOS to an application, every component would be measured. This can result a very long chain, but there is another issue other than the length. With the complexity of the list of good software (due to the thousands of drivers or updates), the resulting chain hash might

become unusable; without an unambiguous identification (falling into contradiction with the initial definition of trust). Lyle's work explores the feasibility of attesting remote web services and applications considering long static trust chains [64–66]. In DRTM with TXT or SGX, it is possible to establish a short or shorter chain by eliminating intermediate components such as BIOS or the OS altogether. We emphasise that SRTM and DRTM are not necessarily replacing each other; a hybrid solution would be viable addressing a strong adversary model.

2.2.3 Attestation

Attestation, in the context of our work, refers to the activity of two abstract entities where one reports evidence of its status and another checks the validity of the reported information. The process can be remotely between networked machines or locally between two isolated applications behaving similarly to networked machines, although they are located within one hardware system. We intend to keep this section short and refer the interested readers to the detailed surveys on principles of remote attestation [67], on the hardware-based approaches [68], and further Banks, Kisiel, and Korsholm present a review on swarm attestation, control-flow attestation and formally verified remote attestation protocols [69].

The presented evidence can be very precise with binary measurements and exact information about the software and the system. But, this precise information may not be not be always helpful for two reasons; (1) precise information would be outdated fast as the system falls into a new state, and (2) precise information can be used to identify what vulnerabilities the reporting system contains. Alternatively, less precise information comparing to binary measurements but strongly representative properties of the software can be presented. Mulligan et al. from ARM outlines these problems as open research challenges. ARM's realm enclaves/isolates or TrustZone's *secure world*'s may offer native attestation support one day, including capabilities to report the software behaviour; this would enable a more trustworthy and decentralised world of platforms including IoT world aside from more generic-purpose computers.

While the attestation procedure covers the problems on identifying the running software in a system, provisioning is often used by manufacturers to enrol their devices by identifying the low-level components such as firmware, BIOS/UEFI or the microcode version. Apple's device activation or update process on IOS devices can be an example of the provisioning process. Intel similarly can perform a provisioning on their platforms as part of infrastructure services. During attestation process precision and the repeated identification information together can cause another issue— privacy disclosure, as users through the systems can be identified. To solve this challenge group signature schemes are used, such as Direct Anonymous Attestation (DAA) [71], and Enhanced Privacy ID (EPID) [72]. Section 6.5.4 presents more details on attestation mechanisms.

2.2.4 Remote Attestation with TXT

A longstanding challenge in distributed computing is that communicating entities have little or no guarantee of what software is running at the remote computer. This may be problematic for many reasons, but in particular it means that one party has no assurance that the other will implement a security policy as expected.

Remote attestation attempts to solve this problem by providing a precise account of the state of the remote system — and hence of the software which has been loaded and executed on that platform. Typically, this is achieved through the creation of a chain of values formed by taking a cryptographic hash of each binary before it is executed — the start of the chain being some known 'good' software, a 'Root of Trust for Measurement' [16] (related approaches [73, 74] have attempted to report the *semantics* of the software instead of the binary program code.)

Remote attestation requires providing some type of *attestation evidence* (*e.g.*, a cryptographic hash) of the binary software that has been loaded on the remote platform. The authenticity of this evidence must be guaranteed by some *root of trust* on the platform, which is usually provided by trusted hardware.

In the case of Intel TXT technology, that attestation is rooted in a special processor instruction (`SENTER`) which puts the processor in a special known good

state, facilitating the creation of a hash (measurement) of a software loader, and a particular binary to be executed thereafter. These hashes are stored in a *Platform Configuration Registers* of the TPM, and reported to a third party via a nonce challenge using a signature key bound to the TPM.

In general, the system being attested is called the *prover*, whilst the system evaluating the attestation is called the *verifier*. In all cases, the verifier is required to decide whether or not the attested system is trustworthy. This becomes exponentially more difficult as the size of the attested component (*e.g.*, the number of lines of code) increases. In order to use attestation successfully, it is therefore critical to minimize the size of the attested component.

Section 4.5.6 presents an attestation mechanism with SGX.

2.3 Focus TEE: Intel SGX

The Intel Software Guard eXtensions technology is a set of CPU extensions that allows applications to create isolated execution environments [75], called *enclaves*. An enclave protects the confidentiality and integrity of any data it contains from all other software on the platform, including a potentially untrusted OS. Since the platform's main memory may be under the adversary's control, SGX ensures that an enclave's data is encrypted and integrity protected before it leaves the boundary of the CPU. Enclaves also provide secure storage by allowing data to be *sealed* [76] such that it can only be decrypted by a specific enclave. SGX supports both local and remote attestation: enclaves can be attested by other enclaves on the same platform, or by remote verifiers. SGX remote attestation uses similar techniques to those described above, but also makes use of an architectural enclave from Intel, called a Quoting Enclave (QE). The prover (an enclave) first uses local attestation to attest itself to the QE, proving that it is a valid enclave running on same platform. The QE then signs an identifying hash of the prover enclave, which can ultimately be returned and verified by a Relying Party (RP).

SIGMA (SIGn and MAc) Protocol [77] is a key-exchange protocol that adds authentication to Diffie-Hellmann Key Exchange. The SIGMA protocol is used to

establish secure channels and bind these to the enclave's attestation. In contrast to discrete trusted hardware (*e.g.*, TPMs as a discrete IC), all SGX computation is performed by the main CPU. Having CPU for computations results in significant performance gain compared to discrete TPMs, which are typically inexpensive low performance devices attached to a relatively low speed bus. The typical use case for SGX is to enable an application to be partitioned into trusted and untrusted components, the former being protected by the enclave. This helps to isolate and protect security-sensitive components of an application. Minimising the size of the *trusted* components reduces the risk of security vulnerabilities, and reduces the burden on the verifier during remote attestation.

Throughout the thesis, we introduce more details of SGX in Section 3.3 with more background, Section 4.4 with benchmarking SGX instructions, Section 5.3 on how SGX enclaves work and how their content can be inspected, Section 5.7 on managing enclave TCB and the bad practices in enclave development, Section 5.8.1 on enclave partitioning for private algorithms, Section 6.4 on enclaves in practice in context of CRC, Section 7.2 on malware in an enclave.

2.4 Closing Remarks

Trusted computing and trustable computing has been interpreted differently by researchers. The term *Trusted Computing* refers to the root hardware that must be trusted— the trustworthiness cannot be established otherwise. If the trusted hardware is vulnerable or outdated, all past authorisations must be revoked and computations cannot continue. The term *Trustable Computing* refers to the dynamic portions of the system where a software is measured and there is a trust decision. The trust decision may change depending on external conditions: one may trust a piece of software today but not tomorrow, or while a server trusts that software for a certain operation, another server may not trust the same software. A practical example of this could be that SGX instructions are *trusted*; the execution cannot proceed if they are not patched and up-to-date, and the SGX enclaves are *trustable*; execution may continue for whoever agrees to trust the enclave report. In this

chapter, we presented our understanding to resolve past ambiguity. Although other researchers may not agree with all of the insights we present here, it helps readers to establish a better understanding for the rest of the thesis.

Chapter 3

Related Work

Contents

We aim to provide the essential background information to support the remainder of the thesis, and we present the work done by other researchers closely related to our research. This chapter has two main semantic portions;

- First, including a related work for three main chapters as follows: Section 3.1 provides a related work to Chapter 6. The related work in this section is related to the main theme of the thesis, CRC. Section 3.2 relates to Chapter 5 on hardware support for private algorithms, and answers the questions of solutions with obfuscation. Section 3.5 relates to the Chapter 4, on building a TRE with SGX.

- In the second semantic portion of this chapter, we continue to this chapter by providing a detailed review of the previous practical iteration of the TRE based on TPM hardware. The review of the TPM-based TRE helps us to continue with Chapter 4, where we build a prototype with SGX instructions.

3.1 Related Work on Systematisation of Confidential Remote Computing

Confidential Computing Consortium's July 2020 white paper [78] shows the benefits of hardware assistance in computations, *e.g.*, the encryption-in-use for data. It includes a good comparison of the cryptographic methods (*i.e.*, from homomorphic encryption standardisation) and the trusted hardware. For example, in homomorphic encryption schemes, publicly known code is mutually agreed on by the participants (data providers), therefore, they do not provide code secrecy to hide the private algorithms. The paper shows that trusted execution environments (TEEs) can provide attestability, code confidentiality, and better programmability on top of the existing guarantees of homomorphic encryption. The code confidentiality with TEEs is described as *requiring further work*, though until now, secret-code execution in TEEs has been studied in recent years [6, 79, 80], and a Protected-Code Loader (PC Loader) is also an integrated feature in the Intel SGX's SDK. Another short introduction [81] to confidential computing states how widely the hardware-assistance is adopted by major industry vendors. An abandoned patent [82] and a valid patent [83] state similar concepts. However, Naganuma's patent [83] proposes confidential computing through homomorphic encryption, with no use of hardware-assisted execution technologies. To the best of our knowledge, our work is the first attempt to model *Confidential Remote Computing* in academia.

While exploring the potential security enhancements with CRC, we consider the potential threats it can introduce. For this reason, we explore the related work on how the use of trusted hardware might empower the malware in Section 7.1.

3.2 On Protecting Private Algorithms with Hardware Support

In this section, we provide an overview of the studies on TCB minimisation and secure TCB design. Beyond the CPU and memory protections, Ports and Garfinkel demonstrate [84] the impact of malicious OS behaviour against the secure application design. Their arguments outline the attacks and mitigations via the trusted interface, and they explain the utilisation of partitioning methods to reduce the TCB size. Singaravelu *et al.* [85] also demonstrated three case studies in which they were able to use kernelised TCB at OS security level on a commodity computer.

Similar to the arguments found within Chapter 5, Piessens *et al.* [86] argues that developers have limited understanding of security guarantees offered by the execution infrastructure. Their work [86] focuses on language safety and full abstraction in programming-language translations. McCune *et al.* [87] utilises TPM, AMD SVM, and Intel TXT to provide secure hardware primitives for minimised software TCB. Strackx *et al.* [88] discuss the use of memory-safe languages on protected module architectures for code and data security, and they explain how secure enclaves might be written. In our work, we focus on confidentiality issues due to the TCB components for secret algorithms in a practical TEE called SGX.

Linn *et al.*'s work [89] in binary obfuscation aims to make reverse engineering difficult for application binaries. The work [90] from Wu *et al.* focuses on malware evasion, which hides the application code against the analysis. Similar techniques can partially protect the application code; however, they fail to maintain the algorithm secrecy in stronger adversarial assumptions. These cases include the decentralised settings where mutually distrustful parties are involved, and where the adversary has full physical access over the host platform. Even though binary obfuscation methods seem to have similar goals to our work, they remain in the scope of reverse engineering. Our work is in the domain of securing private algorithms executed in an untrusted cloud. Forensic tools can recover pieces of evidence about an operation from memory. This information retrieval applies mainly to the data. Some forensic

techniques can recover [91] the execution states of a known software. However, these techniques do not threaten the algorithm secrecy directly. We protect the private algorithms at runtime and provide protection before the execution.

A recent work called Golem Network[1] provides a decentralised marketplace for computational power. The golem enclaves are similar to the interpreter enclaves we discuss in Chapter 5. The third-party libraries included in the TCB of interpreter enclaves may break the security guarantees offered by the SGX. The difference to our approach is that a Golem enclave has a large TCB size due to the Library OS (Graphene-ng) and the unmodified applications (*e.g.*, Blender). With a larger TCB size, formal analysis becomes more difficult. Second, Golem Network provides data secrecy only, and inherited security guarantees might be limited, as we show in a case study on similar interpreter enclaves in Section 5.6. Similarly, in one of our industrial use cases, we present a template for computational power in Section 5.9. Our template does not use third-party libraries to process secrets, and it enables running algorithms strongly-private in remote computers.

3.3 A Brief Background On Intel SGX

In Chapter 4, we shall discuss and demonstrate building applications with SGX instruction. To familiarise the readers more with SGX technology, we provide a short background on SGX. More background and details on SGX can be found in Chapter 2, and more practical aspects will be introduced in Chapter 4.

Intel's SGX is a trusted hardware solution [15, 75, 76], which provides a novel development model for enclave binaries, as well as hardware-maintained (ring -3) integrity guarantees for computations at user-level (ring 3). It also provides trusted computing primitives such as *Root of Trust for Measurement* for its own execution, *Root of Trust for Reporting (RTR)*, *Root of Trust for Storage (RTS)*, and access control for multiple isolated memory regions.

[1]Security in Golem Network `https://docs.golem.network/#/About/Security` visited on 04/Jun/2019.

The RTM, at the lowest privilege level, generates trustworthy evidence about the state of a system [76]. In the case of SGX, RTM provides evidence about the enclave's memory layout, not about the system outside of the enclave memory. SGX's enclave development model helps by securing sensitive parts of user-level applications [75]. Enclaves can prove their identity to verifier entities who require evidence before proceeding to execution. Therefore, SGX is a decent TEE solution for application developers.

Other than the SGX instruction set, the SGX SDK includes SGX drivers, architectural enclaves and SGX trusted libraries. Intel actively improves the SDK and plans to add more instructions in SGX Version 2. SGX, together with SDK, is an ever-evolving software technology that provides a pratical understanding of a TEE.

3.4 Remarks on TRE: the Abstract Concept and the Practice

Trustworthy Remote Entity refers to a system (an entity), running in a remote location to its participants and offers some way of trustworthiness; in short, it is called TRE. TRE is a concept of a verifiable TTP, first-time introduced and specified by Andrew Paverd [92]. Prior to our work, there was only one practical prototype of TRE based on TPM. We implement a more advanced, practical prototype of TRE with SGX instructions, in respect to its initial design principles. In this thesis, we present a very detailed review of Paverd's TRE, renaming it to TPM-based TRE. As we call our prototype SGX-based TRE, (in other words, a verifiable TTP with SGX), TRE becomes a more abstract concept. The abstract TRE concept that Robin Ankele [3] uses refers to a TRE nearly equal to an enclave with TRE capabilities, based on our SGX-based TRE prototype. Together with our SGX-based implementation, the concept of TRE became more abstract. Up until our work, TRE was associated with the TPM-based implementation only. The work on SGX-based TRE presented in this thesis was also used by Robin Ankele, while we have done some collaboration. We considered using multiple TREs in a larger setting. Their work takes our SGX-based TRE, and extends it with more

privacy-preserving capabilities. Their thesis is orthogonal to our work and goes in a different direction with their novel contributions on privacy-preserving data analysis and data release. The first protype of the SGX-based TRE [2], implementation and the measurements are done by the author in this thesis (and related publications), and contributions are claimed within the thesis.

3.5 On Building TRE with SGX

Once a TRE is built on a trusted hardware, any data silo [93, 94] or application-specific code and privacy-preserving algorithms can be placed into the related component, and perform the computation for large number of participants.

In addition to Paverd's TRE [92], similar approaches using trusted hardware have been suggested. Koeberl *et al.* [95] propose a conceptual approach in which a generic compute provider, supported by a TEE, provides trust brokerage between different data providers. In particular, the TEE can apply privacy-preserving operations and filters on the intermediate data. Another study was proposed by Kim *et al.*, deploying network applications on Trusted Execution Environments [96], implemented a prototype on OpenSGX [97] demonstrating the use of TEEs as a *middlebox*. Their work includes several case studies in general aspects and secure execution of shared code.

Moat[98, 99], a tool to formally verify the confidentiality properties of SGX applications, may be used to verify properties of the TRE with reduced TCB.

Atamli-Reineh and Martin [100] explored different schemes for the software partitioning for SGX enclaves, partitioning a trusted application into multiple TEEs, and showed that the choice of partitioning scheme has an impact on security and performance.

The next section gives readers an understanding of the TPM-based TRE and discusses a potential SGX-based implementation. The background presented here helps to understand Chapter 4 presenting a TRE built on SGX.

3.6 Review of TPM based Trustworthy Remote Entity

Trustworthy Remote Entity (TRE), introduced by Paverd [92], is a remotely verifiable system with computational and communication abilities; it uses TPM to implement its attestation capabilities. The TRE is defined and described in his PhD thesis, including several publications. TRE is mainly placed between two mutually untrusted parties and solving the problem of blindly relying on a Trusted Third Party. Paverd's TRE comes with an attestation protocol (Final State Attestation as he defines) that allows parties to verify the software running in a remote entity. His definition of an ideal TRE is trustworthy, even if it is operated by someone adversarial. However, he assumes that adversaries are rational in respect to gain/cost trade-off in performing the attack. Trustworthy means that once a system (implemented to serve as a *Trustworthy Remote Entity*) is attested for a certain behaviour, it will continue to act as it promises. Paverd explains this assumption further as an attestation-based trust to create the Final State Attestation protocol, we revisit in Section 3.6.1.2. The attestation-based trust is presented as offering better guarantees than identity-based trust. The adversary definition and adversary model of Paverd is relatively weak. Under his assumptions, it provides better security guarantees. We address a stronger adversary in this thesis.

One important challenge is; TPM does allow one quoting at a time. In a case where 'many' users must verify the trustworthiness of the TRE, the attestation subsystem has to handle the quoting in a special way that TRE can support large-scale application use cases [92]. This topic is discussed in the remote attestation section with Final State Attestation protocol, where the TRE is quoting once and proving that it did not change its own state since the last quoting.

Nearly each trusted execution technology offers all the following features in hardware or via software-based solutions; platform integrity, secure storage, isolated execution, device identification, and device authentication. These points are essential

to build a fully functional TRE, and how they relate to the architecture of TRE is revisited in the future sections in detail.

3.6.1 Design Challenges

The concept of TRE starts with the question, *why should participants trust TRE?*. The key point is that TRE is not a Trusted Third Party that the parties blindly trust. TRE provides solid technical guarantees that it is trustworthy.

Usually, in a computing system, we do rely on the fact that the instructions of processors are implemented correctly, and we trust the manufacturer of the processor. The root of trust is not an unknown entity. It comes from processor vendors, such as Intel or ARM, which we have trusted for decades.

3.6.1.1 Identity-based Trust

Most current systems use identity-based trust, *e.g.*, by using digital certificates that identify the participants. Identity-based trust does not provide any guarantee about operating unhindered and does not give any information about the good behaviour of the entity. In identity-based trust, an honest but curious (HBC) adversary can perform some actions that other parties cannot detect. Entities using identity-based trust include provisioning (question of which one) and operator (question of who operates it) information. TTPs rely on identity-based trust.

3.6.1.2 Attestation-based Trust

In attestation-based trust, the entity is trusted as the state of the system (question of what it actually is) is known. The relying parties have the ability to describe the prover entity fully. TRE is designed to rely on attestation-based trust. TRE overcomes the scalability challenges of attestation-based trust with Final State Attestation protocol.

3.6.2 Different Architectures

Paverd built a TPM-based TRE on a x86 machine with TPM as bare-metal, single function, event driven. It has no internal isolation between components, and TRE provides integrity-protected measurement of all execution using the TPM 1.2.

In a client PC, trust decision about all the executed software is not feasible. Therefore, reducing the TCB is a challenge for the TRE architecture. TRE solves this by removing all unrelated components and running the trustworthy software as the only software and as TCB.

For memory protection against Direct Memory Access, TRE uses Protected Memory Regions. However, the TRE solution with TPM is still vulnerable to several types of hardware attacks, such as reading or modifying data on the memory bus.

TRE has two backup mechanisms as, online and offline backup [92]. Offline backup is done for the same platform, and the online backup is made for the different TREs. A backup can only be used by a TRE that is in the exact same state as the TRE when the backup was created. This method preserves privacy and secrets. The secret data silo can only be processed by a TRE which is in a certain state. These methods are implemented by relying on PCR values.

There are four different potential TRE architectures that are also proposed [92]; Linux-TPM, VM-vTPM, ARM TrustZone and Intel SGX. In detail, the TPM-based solution provides similar functionalities as the x86-TPM solution. ARM TrustZone does not give any remote attestation module by design; therefore, TPM is assumed to be included in *secure world* of Trust Zone. Intel SGX solution is potentially the best architecture solution in terms of the additional security benefits, reduced TCB size, CPU performance and attestation capabilities.

x86-TPM TRE prototype implementation has the same principle as exokernel [101], has no isolation between the components and all functionalities are mandatory to build up the TRE. The Linux-TPM is running the TRE software as an application on the OS. Running a TRE application on an OS causes a potential issue; full OS kernel is included in TCB, where most of the functionality of OS is not necessary for the TRE.

3.6.3 TPM-based TRE Benchmarking

Benchmarking is performed [92] by considering TCB size in terms of lines of code and measuring the computational performance in terms of signature and verification, and also for GET operation in DLMS-COSEM (Device Language Message Specification). By definition, ISO OSI model DLMS is Data link layer and COSEM is the presentation layer.

Binary attestation is used in all previously proposed TRE architectures [102]. It aims to minimize the TCB size and comes with the requirement of establishing an attestation-based relationship. Therefore, reducing TCB size is an important evaluation criterion in TRE design. On the other hand, computational performance is the second criterion for scalability, and it does not affect trustworthiness.

3.6.3.1 TRE TCB Size

In overall x86-TPM TRE prototype has 24,719 lines of code; as the core TRE functionality is 2.9 % of the TCB, and the attestation subsystem contributes to TCB by 0.9 % of code. The communication subsystem (24.1%) and the cryptographic libraries including the protocols and functions (58.3 %) make up the majority of the TCB [92]. Communication drivers and crypto libraries do not include non-essential features, and they are widely used & known libraries. These libraries are formally analyzed in the past against known vulnerabilities. Since the rest of the custom TCB code is relatively small, the full formal verification becomes easier.

In the potential x86-SGX architecture, SGX is used to be the root for trust. The communication subsystem is not included in TCB and is provided by an untrusted OS. In addition, the TPM driver and attestation subsystem can be removed from TCB; because Intel's mechanism provides the attestation functionality by using the Quoting Enclave. Also, the SGX driver does not affect the trustworthiness; the SGX driver is out of the TCB by design. The malicious driver of SGX in an untrusted OS does not provide any benefit to the adversary. Therefore, the advantages of the SGX architecture may reduce the TCB by at least 30 %. In

addition, SGX would provide much stronger security guarantees by having the encrypted memory execution for enclaves.

3.6.3.2 TRE Computational Performance

TLS handshaking is a computationally expensive cryptographic operation. Therefore, the first benchmark was done by measuring the time taken for the creation and verification of signatures. The second evaluation is the amount of time spent for performing a single DLMS-COSEM request and response. This GET operation is performed in all communication in the custom TRE application.

Paverd made the computational performance comparison between x86-TPM bare implementation and partial Linux-TPM prototype, also including different level of optimisation flags as -02 and -03. The performance of Linux-TPM based TRE implementation was better than x86-TPM based TRE implementation (bare metal). The memory operations for x86-TPM TRE was included as in assembly language. Linux-TPM uses memory management functionalities provided by host OS, therefore the TPM-based TRE is around 8 % slower than Linux prototype. Paverd also mentioned that if the Linux prototype is restricted to use the memory operations, then the performance of both implementations would be virtually identical [92].

Optimised memory operation functions (*e.g.*, memcpy and memset) cost approximately 2.300 lines of code more on TCB size, therefore trade-offs must be evaluated depending on scale and use case. These trade-offs exist with other TEE-based solutions such as x86-SGX TRE and VM-VTPM architectures.

For the DLMS-COSEM GET operation, the benchmark is performed to send and receive a request-response pair. Each pair is required to set up a TLS connection. This benchmark is showing that how many application-specific operation TRE can support per second. The processor type was same in the experiment, however different SSL/TLS libraries are used and compared the x86 and Linux prototypes of TRE. Due to memory operation, parallel processes, optimised network driver by OS and similar benefits of Linux OS, the TPM prototype was 37 % slower.

Using MatrixSSL instead of mbed TLS library allows three times more number of operations per second, however, MatrixSSL adds 9,000 lines of code to the TCB. Thus, the experiment clearly indicates the trade-off between TCB size and the performance [92].

In the overall performance evaluation of the smart grid application with x86-TPM based implementation, TRE requests the latest measurements from the clients and requests a bid, including the price information for the next period. TRE evaluates and notifies the client/consumer about the DSM's decision about a given bid. The slowest implementation was the x86-TPM prototype, and it is able to perform a minimum of 40,000 DLMS-COSEM operations in 30 minutes. In terms of request-response pair, it can support at least 20,000 individual consumers. There are several potential optimisations and other ways of implementation to increase the number of supported consumers.

3.6.4 Concepts of Trustworthy Remote Entity

If one asks the question of Yao's Millionaires' Problem [103], one of the first ideal solutions everyone claims is using an authorised third party for decision making. However, relying on a different entity is not acceptable as it is open to leaks of private information. If there was a way to make an independent entity only do certain tasks and prove its own state while it protects secrets and does not know about the processed data itself, the trustworthy remote entity comes into discussion. Recent studies [92, 104] showed that it is possible to deploy a TTP with trustworthiness properties on trusted hardware using trusted computing concepts. Trustworthiness comes with the given ability to decide on the level of trust and verify the state of such a system.

One of the main concepts [92] a trustworthy remote entity relies on is remote attestation. This concept is used to consider one entity as *prover* and possibly many others as *verifier*, where the prover entity would report its own state to any verifier on demand, so it only fulfils the intended behaviour.

Secondly, for integrity checks, the trustworthy remote entity's behavior [92] is isolated from any potentially malicious injections. Therefore, it is possible to rely on a state of a trustworthy remote entity once it is attested, as it will not change its behaviour into an unknown state.

Third, confidentiality is also provided within a trustworthy remote entity, in that no data is visible to any entity. In trusted execution environments, some technologies may support encrypted execution in certain parts of memory. This means certain operations running secure memory pages would not be visible to any entity. This provides the same confidentiality level of encrypting exchanged data because it is not possible to break the memory encryption mechanism of trusted hardware in a rational way.

All these concepts are assumed to be provided by an ideal *Trusted Hardware*, where the trustworthy remote entity was previously deployed using an external chip *TPM* by Paverd, [92] and also deployed on Intel's SGX technology recently reported to support this current research [2].

The concept of a trustworthy remote entity was also named as a compute provider and middle-box in some other studies. However, as trustworthiness is the crux, we keep using this name as it describes what the entity is in a clear way. Overall, the Trustworthy Remote Entity(TRE) still needs maturity in the context of secure multi-party computation problems.

3.6.5 Architectural Blocks on Trusted Hardware for TRE

In order to build a trustworthy remote entity, there are several functional blocks that have clear borders from one to other. Recent research showed that certain architectural blocks are compulsory, and some others could be used on demand.

The concepts of trusted computing and the properties of an ideal chip have been reported in the literature for many years. The literature review in Chapter 2 explains this further in a detailed background.

The trustworthy remote entity may act as a centralised communication point for several distrustful parties. Therefore, communication infrastructure is a key

element in architecture. All communication channels must be encrypted, and for cryptographic functionalities, libraries need to handle the operations, such as key exchange, protocols, signature and verification. Reporting and attestation are other core blocks that need to be explored. The trustworthy remote entity reports its own state to parties relying on them. This brings the functionality for *measurement* of the current software running in an isolated environment. These are the blocks that have to be included in every trustworthy remote entity.

The optional blocks, depending on the use case, are the computational and privacy-preserving blocks. The computational blocks may perform max, count or compare operations. The privacy-preserving blocks are aggregation, anatomisation, permutation, generalisation, suppression, perturbation, replacement, differential privacy and privacy budget blocks.

Although more in a lower level of the process of building a trustworthy remote entity, it is essential to use the system resources efficiently. In the TPM case, one core limitation [105] was the quoting, which takes approximately 700 ms, therefore, it was important to optimise the quoting operation for large-scale use. In the SGX case, there is a limitation on the amount of memory that can have encrypted pages. A certain encrypted and isolated memory area refers to *enclave*, and the number of enclaves used is a different issue too. While running many enclaves concurrently, a busy enclave may affect another one and assigning certain functionalities to an enclave is another important point. It is crucial to classify through additional experiments and validity studies - the functions, such as how frequently they would be called when included in the enclaves. Also, how big a function and how much memory it requires for a certain operation is important for this trade-off. In the worst case, very frequently calling a large function with consumption of a large memory area should not be included in the same enclave as less memory consumption less frequently called functions because each call of that tiny function enclave's performance is slowed down by the enclave's properties altogether. Also, in terms of a Trusted Computing Base size discussion, splitting the functionalities into different isolated execution environments is a better method

as, if one enclave included a vulnerability, it would not affect another one directly. This topic involves the internal structure of a single trustworthy remote entity.

In a higher level of abstraction, there might be multiple trustworthy remote entities in a distributed or pipelined setting where each entity is assigned to a certain task. This could be because of handling larger scale applications, in terms of the number of participants or amount of processed data. A countrywide election system may require a mesh network of entities, or a genomics case under a set intersection primitive could be solved by pipelined entities. Each type of trustworthy remote entity would act as a different block in this approach. Therefore, it is important to know how the external structure would form.

3.7 Closing Remarks

We presented the related work for the thesis and the remaining chapters. The review of the previous TRE implementation in Section 3.6 is relevant to the next chapter as we will build a similar system with SGX instructions.

Part II

Public Code and Private Data

Chapter 4

Implementation of Scalable Many-Party Computations on Intel's Software Guard eXtensions Enclaves

Contents

The theoretical construct of a Trusted Third Party has the potential to solve many security and privacy challenges. In particular, a TTP is an ideal way to achieve secure multiparty computation — a privacy-enhancing technique in which mutually distrusting participants jointly compute a function over their private inputs without revealing these inputs. Although cryptographic protocols exist to achieve this, their performance often limits them to the two-party case, or to a small number of participants. However, many real-world applications involve thousands or tens of thousands of participants. Examples of this type of *many-party* application include privacy-preserving energy metering, location-based services, and mobile network roaming.

Challenging the notion that a *trustworthy* TTP does not exist, recent research has shown how trusted hardware and remote attestation can be used to establish a sufficient level of assurance in a real system such that it can serve as a trustworthy remote entity (TRE). We explore the use of Intel SGX, the most recent and arguably most promising trusted hardware technology, as the basis for a TRE for many-party applications. We first compare SGX to previous technologies and point out new subtleties that influence the design of SGX-based trustworthy systems.

Using privacy-preserving energy metering as a case study, we design and implement a prototype TRE using SGX, and compare its performance to a previous system based on the Trusted Platform Module. Our results show that even without specialised optimisations, SGX provides comparable performance to the optimised TPM system, and therefore has significant potential for large-scale many-party applications.

In this chapter, we analyse the TPM-based TRE, create the system and adversary models for SGX-based TRE, and we define the security and performance requirements. Part of the content presented in this chapter is published in [2, 4].

4.1 Ideal Verifiable Trusted Third Party

An ideal Trusted Third Party is a theoretical construct used to describe the ideal solution to many types of security and privacy challenges. It is assumed to be an entity that is universally trusted by all participants, without having to provide any evidence of its trustworthiness. As the definition of TTP can vary depending on the application domain, some may not be universally trusted, and some TTP may also be required to provide evidence of its trustworthiness before use. Secure multiparty computation is a well-known example of a class of problems that could be solved using a TTP. As introduced by Yao [103], a multiparty computation problem consists of two or more participants, each holding some individual input, who wish to compute some function over their inputs jointly. However, since the participants are mutually distrusting, they do not want to reveal their private inputs to one another. Suppose some third party exists who is trusted by all participants. In that case, the ideal solution to this problem is for each participant to send their individual inputs to the third party, which can compute the function. Although the use of TTPs is inevitable in some real-world applications (*e.g.*, certification authority), conventional wisdom calls for solutions that avoid them, since they are assumed to be difficult to realize. Consequently, researchers have proposed numerous protocols for achieving different types of secure multiparty computation-based cryptographic primitives such as oblivious transfer [106] and homomorphic encryption [56]. However, with current techniques, these protocols have significantly worse performance (due to computational and network overheads) than the ideal case using a TTP. In particular, these performance constraints often limit these protocols to the two-party case, or to some small number of participants. However, a large class of real-world applications involve thousands or tens of thousands of participants. We refer to these as *many-party* applications.

Many-party applications. Many-party applications can often be modelled as communication systems, in which *many* mutually distrusting participants [4] wish to communicate with each other without diminishing their privacy. For example,

users of a location-based service (LBS) may send their locations to a service to receive location-relevant information, but this diminishes their privacy. Similarly, in the smart energy grid, the fine-grained energy consumption measurements from smart meters could be used to infer personal information about consumers and profile their behaviour. Communicating this information to the service provider is necessary for the smart grid to function, but it also diminishes users' privacy [107, 108]. In both cases, privacy-preserving operations could be performed on the communicated information to hide individuals within a group: In the smart grid, measurements could be aggregated over multiple consumers [107, 109], and in LBS, the user's location could instead be reported as an area containing multiple other users (*i.e.*, location cloaking [110]). In both of these cases, these groups could contain many participants (*e.g.*, thousands or tens of thousands). The fundamental question is, therefore, how to perform these privacy-preserving operations *securely at the required scale*. Current cryptographic approaches do not provide sufficient performance for these many-party applications.

Trustworthy Remote Entities. Challenging the conventional wisdom regarding TTP, various research efforts have shown how trusted hardware and remote attestation can be used to establish the trustworthiness of a remote system, to the extent that it could serve as a TTP. Paverd *et al.* [92, 109] proposed the concept of a *trustworthy remote entity (TRE)*. The TRE is essentially a TTP that provides a very high level of assurance of its state and behaviour, thus making it *trustworthy* rather than simply *trusted*. In a many-party application, the TRE is situated as an intermediary in the communication paths between participants and performs privacy-enhancing operations on the communicated information. For example, in the energy metering scenario described above, the smart meters communicate directly with the TRE, which performs privacy-preserving operations (*e.g.*, aggregation [111, 112]) on the sensitive energy measurements. Any participant who individually trusts the TRE can take part in the communication without loss of privacy. The principal requirement is consequently to provide all participants with sufficient evidence of the TRE's state and behaviour for them to make informed trust decisions. This is

achieved using remote attestation. In their implementation of the TRE, Paverd *et al.* [92, 109] used Intel Trusted eXecution Technology and the Trusted Platform Module as the basis for this remote attestation.

4.2 System Model and Requirements

This section presents a basic system model of a many-party application. We then define our assumed adversary model and list the security and performance requirements for a TRE in this context. As we previously clarified in Section 3.4, we conceptualise the term TRE and use TPM-based TRE and SGX-based TRE to refer to practical implementations. We use the following system model to realise the implementation of TRE on SGX in Section 4.5.

4.2.1 System Model

Fundamentally, a many-party application consists of a set of participants $\mathcal{P} = \{p_1, p_2, ... p_n\}$ each of whom holds some sensitive information s_p. The overall objective is to compute some function, $f(s_{p1},\ s_{p2},\ ..\ s_{pn})$, over all participants' private inputs. In the most challenging case, these participants may mutually distrust one another, and thus do not wish to reveal their sensitive information to one another.

All participants use remote attestation to ascertain a common TRE's current state and behaviour. The exact mechanisms used by the participants to verify the attestation are beyond the scope of this work. However, they could include security audits or formal analyses of the attested software, either undertaken by the participants themselves or by an entity they trust. The attested and analysed code here is assumed to be public in this part and chapter of the thesis. In contrast, Chapter 5 considers case studies with private portions of the system. Through this process, the participants ascertain the current state and behaviour of the TRE. Having received this strong assurance that the TRE will not divulge or misuse their information, participants send their information directly to the TRE. The TRE can then compute the required function securely and efficiently.

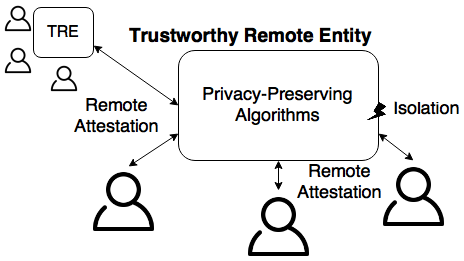

Figure 4.1: Concept of Trustworthy Remote Entity

4.2.2 Adversary Model

In the above system model, the aim of the adversary $\mathcal{A}$ is to learn the secret information s_p of any of the participants. We assume $\mathcal{A}$ has full control over the TRE's platform. This means that $\mathcal{A}$ is capable of loading and executing arbitrary software, modifying the OS and other system software, and/or modifying the pre-OS components (*e.g.*, bootloader) of this platform. Since $\mathcal{A}$ also has physical access, he may add or remove hardware components, reset the platform, and exploit any software or hardware side-channel attacks. Adding or removing hardware components means that the communications between the CPU and RAM can be visible. We consider reading the main memory. However, we do not consider where a powerful-adversary targeting the CPU caches, registers, or attacks involving with interfering with the CPU package. We assume that the CPU is physically secure and the CPU instructions can operate as specified. We do not consider the micro-architectural attacks against the CPU package. Side-channel attacks can contain a wide spectrum of attacks; we do not consider the side-channel attacks based on power nor targeting the CPU in this threat model. We consider the controlled side-channel attacks based on reading the memory pages. We assume that $\mathcal{A}$ has full control of all communication channels, and thus also has all the capabilities of the classical Dolev-Yao network adversary. In other words, we intend the TRE to be trustworthy even if the TRE's operator is malicious.

In many-party protocols, the correctness of the TRE's output naturally depends on the correctness of the inputs provided by the participants (*i.e.*, members of $\mathcal{P}$) or whether the TRE has mechanisms for detecting invalid inputs. Although

this is an important consideration, it mainly depends on the specific type of applications and protocols in which the TRE is used, and is thus beyond the scope of our present discussion.

4.2.3 Security Requirements

Although the overall many-party application may have numerous security requirements, we consider the following requirements, which are applicable to the TRE itself:

- **Computation Confidentiality:** The TRE's internal computation must not be observable by any other party. No external entity should be able to observe or interfere with the computation performed by the TRE. Since it is assumed that the adversary has physical access to the platform, any sensitive information written to memory must be encrypted, the confidentiality and integrity must be protected from untrusted software on the same platform.
- **Computation Integrity:** Similarly, the TRE's computation must be protected from interference or manipulation by external sources. If the system includes any untrusted components, these first two requirements imply that TRE's execution must be isolated from the untrusted components.
- **Communication Confidentiality:** TRE requires a secure channel to communicate with other entities. The content of these messages must not be revealed to any other party.
- **Communication Integrity:** Similarly, the secure channel must have integrity protection. Communication between the TRE and other parties must be confidential and integrity-protected.
- **Strong Attestation:** The root of trust used for attestation must be trusted by all parties. The attestation must be unambiguously linked to the communicating entity (authenticity) to avoid masquerading attacks, and must convey the current state of the system (freshness) to prevent replay attacks.

Since the TRE's role is to perform privacy-preserving operations, various privacy requirements apply. However, these have been discussed previously [4], and are

beyond the scope of this chapter, since they apply to the specific operations performed by the TRE.

4.2.4 Performance Requirements

In almost all cases, the application domain in which the TRE is used gives rise to performance requirements. As an intermediary in the communication between participants, the following performance metrics are applicable to the TRE:

- **Scalability:** The overall rate at which the TRE performs privacy-preserving operations must be sufficient for the application.
- **Latency:** Time taken for each individual operation must be feasible for the application.

The TRE must fulfil these requirements. The application domain sometimes places hard constraints on these performance parameters. For example, the smart energy grid application places hard requirements on the rate at which privacy-preserving operations must be performed (*e.g.*, each smart meter must produce a measurement every 30 minutes [109]). The overall rate at which the TRE can communicate with smart meters and process their private information determines the number of smart meters that can be supported by a single TRE (*i.e.*, the TRE's *scalability*). Similarly, each individual interaction between the TRE and a smart meter must meet the latency requirements of the communication protocol to avoid timeouts.

* * *

In order to implement a TRE on trusted hardware, we begin with a detailed analysis of available hardware products in different families. We find that SGX is the ideal TEE for our goals, and we provide the first and the most comprehensive performance metrics of SGX features required for applications with a high number of participants. We map the TRE functionalities into the SGX-based enclave programming paradigm with the partitioning approach. As opposed to TPM-based

TRE, we split the new architecture into the trusted and untrusted parts, which helps us to reduce the TCB size. We implement the TRE with privacy-preserving operations for the smart grid use case. We demonstrate the generic architecture template and the network communication flow for remote attestation. Finally, we present our findings in comparison with TPM-based TRE.

4.3 Comparing Trusted Hardware

In this section, we compare current trusted hardware technologies with respect to the requirements defined in the previous section. Specifically, we consider Intel TXT (with TPM), ARM TrustZone, and Intel SGX, shown in Table 4.1. Although these hardware technologies are completely different, we attempt to systematise their features in respect to our requirements. The following Sections 4.3.2, 4.3.2, 4.3.3 are in line with the requirements listed in Section 4.2.3. The Section 4.3.4 is based on the requirements of Section 4.2.4. We conclude the section on choosing a TEE in Section 4.3.5.

4.3.1 Secure Computation

All three technologies can be used to protect the confidentiality and integrity of the TRE's computation against untrusted software running on the platform. In TXT, the platform transitions into a secure state through a partial CPU reset, which protects against any previously executed software. A subset of the TPM's PCR is also reset in order to record the state of the platform after the reset. In TrustZone [113], secure computation can be performed in the *secure world*, which is isolated from the *normal world* in which the platform's main OS and applications run. Normal World software can communicate with the secure world through well-defined interfaces, but cannot observe or influence the behaviour of secure world software. Similarly, an SGX enclave can be used to protect the confidentiality and integrity of data, and the integrity of the computations on this data, as explained in the previous section. Against a stronger adversary who has physical access to the platform and can read the platform's memory, TXT cannot achieve secure computation, since

this adversary can read and modify confidential data in memory. Depending on the platform configuration and the trusted OS, a TrustZone system may be able to withstand this type of adversary by mapping the secure world's memory to physical memory within the CPU package (*i.e.*, on-chip), which is assumed to be significantly more difficult for the adversary to access. SGX provides the strongest guarantees by encrypting all enclave memory before it leaves the CPU boundary.

4.3.2 Secure Communication

By default, neither TXT nor TrustZone provide a mechanism for establishing a secure communication channel (*i.e.*, confidentiality and integrity protected) with the trusted components[1]. In these cases, the trusted component must itself include secure communication functionality (*e.g.*, by including a cryptography library any required protocols). In contrast, the Intel trusted libraries included with SGX support key exchange and thus channel establishment using the SIGMA protocol.

4.3.3 Strong Attestation

Utilising the TPM, a TXT-based system can provide strong attestation of its state and behaviour to a remote verifier. The platform firmware itself provides the Root of Trust for Measurement. In contrast, TrustZone does not provide a default attestation mechanism. However, some trusted OS vendors have recently introduced key provisioning techniques for trusted applications[2]. SGX offers full remote attestation functionality for all enclaves.

4.3.4 Performance

In general, a discrete TPM is a relatively low performance co-processor. For example, micro-benchmarks have shown that a TPM requires on average 700 ms to perform a single quote operation, which is required for remote attestation. In typical TPM applications, the prover only performs a single attestation or a very small number of attestations. For example, in network access control, a system wishing to join

[1] Although in TrustZone systems, this may be provided by the trusted OS
[2] `https://www.trustonic.com/products/kinibi`

the network attests its state to the network controller only when it attempts to join the network. However, the TRE inverts this relationship in that a single prover must be attested by a (potentially large) number of verifiers. By comparison, in TrustZone and SGX these type of operations is performed on the main CPU, and thus do not incur the same performance penalty.

4.3.5 Choosing Trusted Execution Environment

TPM-based TRE deployment does not have memory protection. If an adversary is capable to simply read the memory, private information will be leaked. Ideal TRE on TPM runs as bare metal, single function. While this kind of deployment has advantages, such as verifying the overall system state or using the Final State Attestation protocol, there is also a disadvantage as having no isolation between the components. Therefore, a vulnerability in one component would compromise the entire system. Another possibility is to deploy a TRE based on ARM TrustZone, however this requires a software-based attestation mechanism in the secure world. Also, TrustZone supports only one secure world by hardware; it provides a separate memory region but no memory encryption for the user-level applications as listed in Table 4.1. Intel's SGX is capable of performing strong attestation with CPU-based performance and encrypted execution in memory pages. Therefore, we mapped the necessary functionalities to the instructions provided by SGX.

4.4 Benchmarking SGX

As SGX appears to be the most suitable of the current trusted hardware technologies, we performed a series of micro-benchmarks to establish the baseline performance of the basic SGX operations required for the TRE. We explain the measurements in this section.

The results of these benchmarks are shown in Table 4.2. For each operation, we varied the combined enclave stack and heap size, since our preliminary experiments indicated that it is this overall size that has an impact on performance.

[3]TPM itself has no RTM, solutions need SRTM or DRTM/Late Launch for measurement.

	TPM	SGX	TrustZone
Type	External	in CPU	in CPU
Performance	Slow	Fast	Fast
Price	Cheap	Fair	Fair
Environment	Co-Processor	Enclaves	Secure World
Development	Fair	Fair	Fair
Secure DRAM	N/A	Yes	N/A
Private Memory	No	No	Yes
Isolation	Limited	Yes	Yes
Architecture	Single	Multiple	Single
Trigger with	Drivers	Drivers/APIs	Autonomous
Initialisation	Ownership	BIOS	First
Attestation	Yes	Yes	No
is RTM	No[3]	Yes	No
is RTR	Yes	Yes	No
is RTS	Yes	Yes	Yes

Table 4.1: Features and functionalities of hardware TEE alternatives for building TRE. Fast quoting for remote attestation, memory encryption capability, isolation guarantees and measurement capabilities are crucial.

All benchmarks were performed on an SGX-enabled Dell Latitude E5570 laptop, with an Intel Skylake Core i7-6600U 2.60 GHz CPU, running Ubuntu 14.04 with the 2016-06 public SGX SDK. The platform's Enclave Page Cache (EPC) was set to 128 MB.

The first measurement shows the cost of creating an enclave using either 20 kB or 5 MB of memory. This operation would only be performed very infrequently, and will not have a significant impact on the TRE's performance. As expected, enclave creation time increases as the size of the enclave's memory increases, due to the fact that the enclave's pages are *measured* during initialization. The second, third, and fourth measurements show the cost of various steps in establishing a secure connection between the TRE and a relying party, and attesting the TRE. These must be performed at least once per relying party, but, depending on the application protocol, may be performed more frequently (*e.g.*, in cases where it is infeasible for the TRE to maintain concurrent connections with all relying parties, or where relying parties may be periodically reset). The second measurement shows the cost of initializing a remote attestation session with a single relying party. The third

	Operation	Lines of Code	Binary Size (kB)	Stack	Heap	Mean (ms)	Std. Dev. (ms)
1	Enclave Create	240	132.7	16 kB	4 kB	3.066	0.488
				4 MB	1 MB	46.722	7.988
2	Enclave Create	240	132.7	4 kB	16 kB	2.667	0.306
				1 MB	4 MB	22.701	6.886
3	Enclave Create	1370	827.7	4 kB	16 kB	9.986	0.488
				1 MB	4 MB	24.558	2.154
4	Init. Remote Attestation	1370	827.7	4 kB	16 kB	0.04	0.004
				1 MB	4 MB	0.055	0.012
5	Init. Secure Channel	1370	827.7	4 kB	16 kB	0.511	0.056
				1 MB	4 MB	0.611	0.083
6	Quote/SIGMA Protocol	1370	827.7	4 kB	16 kB	33.059	1.968
				1 MB	4 MB	31.764	1.25
7	Enclave Destroy	240	132.7	16 kB	4 kB	0.116	0.060
				4 MB	1 MB	1.158	0.103

Table 4.2: The full list of measurements of basic SGX operations (average and variance over 100 runs).

measurement shows the cost of generating the first message of the Diffie-Hellmann key exchange. The fourth measurement shows the longest operation, which includes both quoting and key exchange operations for the SIGMA protocol.

For completeness, the final measurement shows the cost of destroying an enclave, although this is not a frequent operation.

We identified the following operations as necessary to build a TRE:

- creating and destroying an enclave; in Tables 4.3, 4.4, 4.5, 4.6,
- initialising remote attestation; in Table 4.7,
- generating key exchange messages to establish a secure channel; in Table 4.8,
- generating the quote and performing the associated SIGMA protocol operations; in Table 4.9.

In order to determine which independent variables have an impact on these operations, we developed a test suite to perform the following benchmarks:

- **Memory size:** We configured two enclaves to use either a maximum of 20 kB or 5 MB of memory. These maximum sizes are defined in a configuration file and used during the compilation of the enclave.

	20 Kb	5MB
ECREATE Mean :	3.06611 ms	46.72217 ms
Standard Deviation	0.48865 ms	7.98859 ms
EDESTROY Mean :	0.11633 ms	1.15821 ms
Standard Deviation	0.06037 ms	0.10399 ms

Table 4.3: Costs of Creating and Destroying Enclaves with different Stack+Heap Sizes

- **Stack and heap size:** We split above mentioned 20 kB and 5 MB memory into various combinations of stack and heap (*e.g.*, 4 kB heap and 16 kB stack, 16 kB Heap and 4 kB stack, 1 MB heap and 4 MB stack, or 4 MB heap and 1 MB stack).
- **TCB Size:** We varied the amount of code included in the enclave. In one example, the enclave included 240 lines of code (together with trusted interfaces generated by the SGX SDK) and core trusted SGX libraries. The binary size of this enclave is 132.7 kB as in a shared object (.so file). The second enclave has 1370 lines of code, including the trusted interface, and its binary size is 827 kB. This second enclave includes cryptographic and key exchange libraries, which result in a larger binary size.

A small test application performs the following instructions:

ECREATE: Creating an enclave, initialisation of memory pages.

ECALL: Function calls from the untrusted part to the trusted part. Optionally it may return some data.

EEXIT: Destroying the enclave at the end.

We did not consider calls made by the enclave to the untrusted environment (*i.e.*, OCALLs) since these are not used in our TRE design.

4.4.1 Evaluation of Benchmark Results

Table 4.2 shows the measurements of seven different enclave configurations, for different values of the above variables. We used the same Ubuntu 14.04 running on Intel Core i7-6600U CPU for these benchmarks. The mean values are computed over 100 runs for each measurement and show the standard deviation of these runs.

Stack + Heap	16 KB + 4KB	4MB + 1MB
ECREATE Mean	3.06611 ms	46.72217 ms
Standard Deviation	0.48865 ms	7.98859 ms
EDESTROY Mean	0.11633 ms	1.15821 ms
Standard Deviation	0.06037 ms	0.10399 ms

Table 4.4: Costs of Creating and Destroying Enclave, 132.7 Kb Binary, 240 Lines of TCB (without Intel SDK libs). Stack larger than Heap.

Stack + Heap	4 KB + 16KB	1MB + 4MB
ECREATE Mean	2.66708 ms	22.70152 ms
Standard Deviation	0.30603 ms	6.88681 ms

Table 4.5: Costs of Creating Enclave, 132.7 Kb Binary, 240 Lines of TCB (without Intel SDK libs). Heap larger than Stack.

Among other factors, this variation is due to the fact that the system under test is also running the full Ubuntu OS in parallel with the enclaves. However, it is interesting to note that even in the presence of this OS, the enclave's performance remains relatively constant between different runs.

Measurement 1 shows the cost of creating an enclave using either 20 kB or 5 MB of memory. This gives an idea of the initial cost of setting up the enclave. The first observation is that increases in memory size increase the time taken for this operation. This is to be expected given that the enclave is *measured* when it is created.

In the second measurement, we also used 20 kB or 5 MB of memory, but we changed memory allocation for stack and heap sizes. This row shows that usage of stack costs more than using the heap.

The third measurement shows the impact of TCB size on performance. We kept the stack and heap size as in the second row, but increased the TCB size (*i.e.*, added more functionality into the enclave and imported trusted libraries). This row simply shows that larger TCB has a cost on SGX instructions. For 20 kB memory usage,

Stack + Heap	4 KB + 16KB	1MB + 4MB
ECREATE Mean	9.98613 ms	24.55805 ms
Standard Deviation	1.65747 ms	2.15418 ms

Table 4.6: Costs of Creating Enclave, 827.7 Kb Binary, 1370 Lines of TCB (without Intel SDK libs). Heap larger than Stack.

Stack + Heap	16 KB + 4KB	4MB + 1MB
RA Init Mean	0.04051 ms	0.05546 ms
Standard Deviation	0.00428 ms	0.01291 ms

Table 4.7: Remote Attestation Init; passing Service Provider's Public Key over Platform Services session.

Stack + Heap	16 KB + 4KB	4MB + 1MB
SC Init Mean	0.51133 ms	0.61101 ms
Standard Deviation	0.05686 ms	0.08305 ms

Table 4.8: Secure Channel Init; generating key exchange message with enclave's context.

the execution took 9.9 ms as more than using more stack. However, for the 5 MB, the cost was increased only 2 ms from 22.7 ms to 24.5 ms which is less than the cost of using more stack memory. This point should be explored in future work.

The fourth measurement shows the cost of initialization of remote attestation, where the service provider's public key is passed into enclave and platform services are enabled. This measurement is important to show the percentage contribution of each operation.

The fifth measurement shows the cost of generation of first message for the Diffie-Hellmann key exchange. This message is generated over the enclave's context and includes a generated public EC DH key.

The sixth measurement shows the longest operation which includes both quoting and key exchange operations for the SIGMA protocol. The quoting and secure channel is linked to avoid masquerading attacks. This functionality is provided by Intel's SGX SDK's trusted library, for which the source code is not provided at the time. Therefore, we are not able to split this to see the individual cost of each function. However, we already need to use the quoting and SIGMA protocol together in TRE, so there is no need to evaluate them separately in this chapter.

Stack + Heap	4 KB + 16KB
QUOTE + SIGMA	33.05933 ms
Standard Deviation	1.96837 ms

Table 4.9: Performing Quote and Generating Message for SIGMA Protocol Key Exchange.

The final measurement shows the cost of destroying an enclave, using the same memory configurations as in first measurement.

Overall, the quoting operation means that the prover enclave's context is verified with a hardware key (available for all enclaves in the same hardware) in the Quoting Enclave. This first verification is also called Local Attestation. The prover enclave first proves its identity to the Quoting Enclave and then the signature of Quoting Enclave proves the identity of enclave to the relying party. The SIGMA protocol includes a Diffie-Hellmann key exchange whilst mitigating the possibility of a man-in-the-middle attack.

This basic experiment showed that TCB Size and memory allocation has an exponential impact on the time taken for the above-described operations. Smaller TCB size also allows formal verification to check against runtime vulnerabilities.

The second conclusion is, having higher number of enclaves does not have any impact on the performance. Unless there is a busy enclave, one enclave does not affect another one. Since the execution is done in the CPU, any busy thread has an impact on all system. This evaluation shows that TRE on SGX can be designed with the partitioning of the application. Having a better enclave structure, and isolating the functionalities will give a better opportunity for formal verification.

4.5 Implementation of TRE on SGX

In this section, we describe our design and prototype implementation of the TRE using Intel SGX. Although our system is designed for a particular application (*i.e.*, privacy-preserving energy metering), the majority of the design would be typical to many other application domains. This architecture could therefore serve as a template for using Intel SGX to implement TREs or similar types of trustworthy systems for many-party applications.

The TRE's main functionality can be divided into three main categories: (1) establishing a secure channel to the enclave; (2) attesting the enclave and linking the attestation to the secure channel; and (3) performing the application-specific protocol (*i.e.*, aggregating energy measurements over multiple consumers in the smart grid).

4.5.1 Mapping TRE Functionality

The secure channel requires crypto libraries which can be included in the enclaves. The communication protocol used to establish a secure channel is a software-based task and would be the same in any other deployment. Therefore, this is independent from a TEE. Attestation requires Root of Trust for Measurement and Reporting, which is provided by hardware in a SGX based solution. SGX-enabled processors include a Quoting Enclave which is used to perform attestation. Communication subsystem and network operations can be placed out of TCB with and SGX-based solution. Because the private data never leaves the enclave without the encryption. Therefore, the messages can be transmitted over the untrusted communication subsystem. Enclaves are isolated from each other and also from the rest of the system. Memory pages assigned to an enclave are only accessible and readable by that specific enclave.

4.5.2 Smart Grid Use Case

The smart grid scenario consists of a service provider (SP) *e.g.*, the energy supplier, and multiple smart metering devices (SMDs). Each SMD performs frequent measurements of energy consumption (*e.g.*, every 30 minutes) and sends these to the SP. These fine-grained consumption measurements allow the SP to monitor the energy distribution network and enable *time-of-use* (ToU) billing. However, it has been shown that such detailed energy consumption traces can be used to infer private information about the consumer, leading to privacy concerns [107, 108]. In this context, each SMD is a participant $p \in \mathcal{P}$, and the energy consumption measurements are the participant's private input s_p. The objective of the TRE is to protect the privacy of individual consumers whilst still enabling the overall functionality of the smart grid.

Figure 4.2 shows the core components of the TRE and interactions between the TRE and other participants in the smart grid. In this scenario, the SP and SMDs are assumed to be mutually distrusting of one another. Following the design of Paverd *et al.* [109], the TRE is therefore placed as an intermediary in the

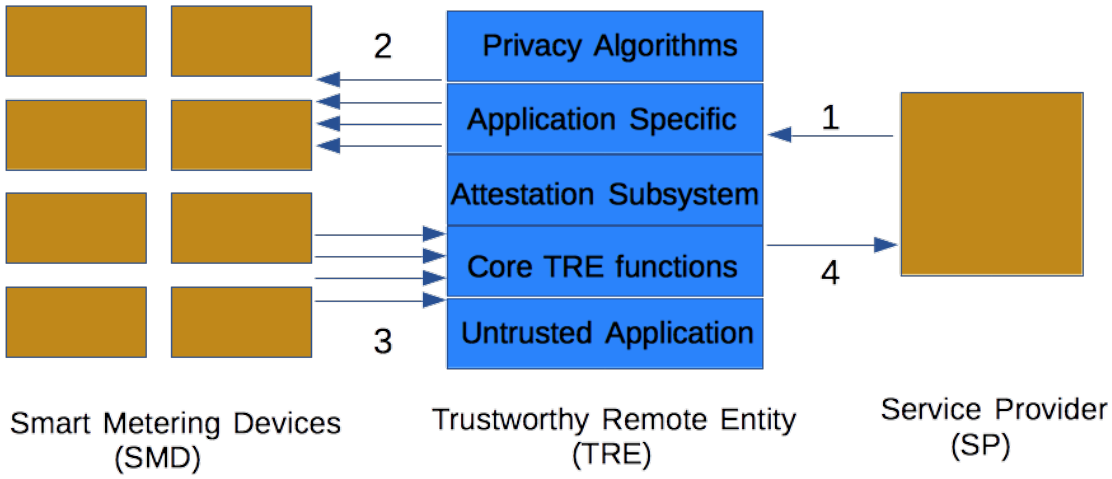

Figure 4.2: System overview of a TRE in the smart grid

communication path between these entities. The arrows in the figure show the sequence of communication between the SP and SMDs. The SP send requests to the SMDs via the TRE, and the SMDs respond via the TRE.

The TRE can thus perform various types of privacy-preserving operations on the SMDs' consumption measurements, including spatial and temporal aggregation, as explained below.

4.5.2.1 Spatial Aggregation

To monitor overall consumption in an area, it is sufficient for the SP to receive the sum (S) of the private consumption values from all SMDs in that area. This protects privacy by hiding each individual's contribution within the total. Although S could be computed using cryptographic techniques, the performance overhead would be infeasibly high due to the large number of participants. In contrast, the TRE can compute S very efficiently, and can scale linearly as the number of participants increases. However, the SP must trust the TRE to perform the aggregation correctly, and the SMDs must trust the TRE not to disclose their individual measurements. Although not included in our prototype, this spatial aggregation could also make use of differential privacy [114].

4.5.2.2 Temporal aggregation

In ToU billing, each participant's bill is computed by multiplying the participant's consumption values for each time period by the prevailing cost per unit of energy for that time period. In this case, spatial aggregation cannot be used because each consumer must receive an individual bill. Instead, the TRE performs temporal aggregation, by keeping a running total for each consumer within the TRE, and forwarding these totals to the SP at the end of the month. This facilitates ToU billing whilst protecting privacy because SP is unable to infer fine-grained consumption details from the total bill, but again requires the SP and SMDs to trust the TRE.

4.5.3 TRE Operation

As shown above, the TRE can be used to enhance privacy in the smart grid if it is trusted by the SP and the SMDs. The SP and SMDs are therefore the *relying parties*. By implementing the TRE using trusted hardware (*e.g.*, Intel SGX), the relying parties can use remote attestation to establish the trustworthiness of the TRE.

Before any relying party communicates with the TRE, the relying party runs a remote attestation protocol with the TRE to ascertain the precise software running in the TRE enclave. In theory, it is only necessary for each relying party to attest the TRE enclave once. However, this results in long-lived connections, which may be infeasible if the TRE has to maintain too many connections or if the relying parties are reset during the connection. In our implementation, we take the most conservative approach and assume that relying parties perform a new attestation and establish a secure channel for each message sent to the TRE (*e.g.*, once per 30 minutes). In reality, the performance of the TRE would improve if longer-lived connections were possible.

However, remote attestation is only useful if the relying parties can decide whether to trust the software in the TRE enclave. This means that the design of the TRE should aim to facilitate remote attestation and minimize the effort required to verify this attestation. Concretely, this calls for minimizing the software Trusted Computing Base of the TRE, since this minimizes the likelihood of software

defects, and makes the software more amenable to security audits or even formal analysis. The TRE's software must also be available to all relying parties, and must support reproducible builds in order to verify that the source matches the attestation. Intel SGX is, therefore an ideal trusted hardware technology for the TRE due to its hardware-enforced trusted execution environment and remote attestation capabilities.

4.5.4 TRE Components

This subsection describes the specific components of our prototype implementation of the TRE, as shown in Figure 4.2.

4.5.4.1 Untrusted Components

The TRE consists of an SGX enclave running on a (potentially untrusted) host platform. The host is responsible for establishing network connections and marshalling encrypted data between the relying parties and the trusted components of the TRE. The untrusted components, therefore, include the platform's hardware drivers, network communication library, and SGX interface driver. None of these components handles sensitive information in the clear.

4.5.4.2 Remote Attestation & Secure Channel

One of the main trusted components of the TRE is the subsystem responsible for establishing secure channels and attesting the TRE to relying parties. Our implementation follows the recommended SGX design patterns for remote attestation and secure channel establishment. Specifically, the TRE enclave performs a location attestation to the Quoting Enclave, which verifies that the TRE enclave is a legitimate SGX enclave running on the same platform, and produces a quote. Using the functionality provided by the SGX SDK, the TRE enclave uses the SIGMA protocol to establish a secure channel with a relying party and bind this channel to the quote. This component, thus consists of the trusted cryptography and key exchange libraries. Compared to Paverd's TPM-based implementation [92,

109], this component was significantly easier to implement due to the inclusion of this functionality in the SGX SDK.

4.5.4.3 Application-Specific Components

Every TRE includes trusted components that are specific to the application domain in which the TRE is used. In this use case, the application-specific components consist of the application logic for parsing the messages from the SP and SMDs. In our implementation, the TRE communicates with SMDs using the IEC 62056 (*DLMS-COSEM*) message specification, which has been mandated as the communication format for all SMDs in the UK. DLMS-COSEM supports a request-response model, in which messages are encoded as binary data, with typical message lengths in the range of 20 to 30 bytes. In addition to the DLMS-COSEM parser, the TRE also includes the specific privacy-preserving algorithms used in this scenario (*e.g.*, spatial and temporal aggregation).

4.5.4.4 Privacy-Enhancing Algorithms

In addition to the application-specific functionality, the TRE can include a library of standard privacy-enhancing algorithms, which can be re-used as necessary for different applications. For example, the functionality required to enforce differential privacy [114] could be included in this component. Although not used in our prototype, differential privacy guarantees could be applied to both spatial and temporal aggregation algorithms.

4.5.4.5 TRE Core

The role of the TRE core is to bring together the above functional building blocks, and to provide the trusted side of the interface between the TRE enclave and the host platform. This component, therefore, includes the SGX trusted interface libraries, as well as the trusted runtime libraries, such as the C/C++ standard library, and the trusted memory management subsystem. Again, compared to the TPM-based TRE, the majority of these libraries are already available through the SGX SDK.

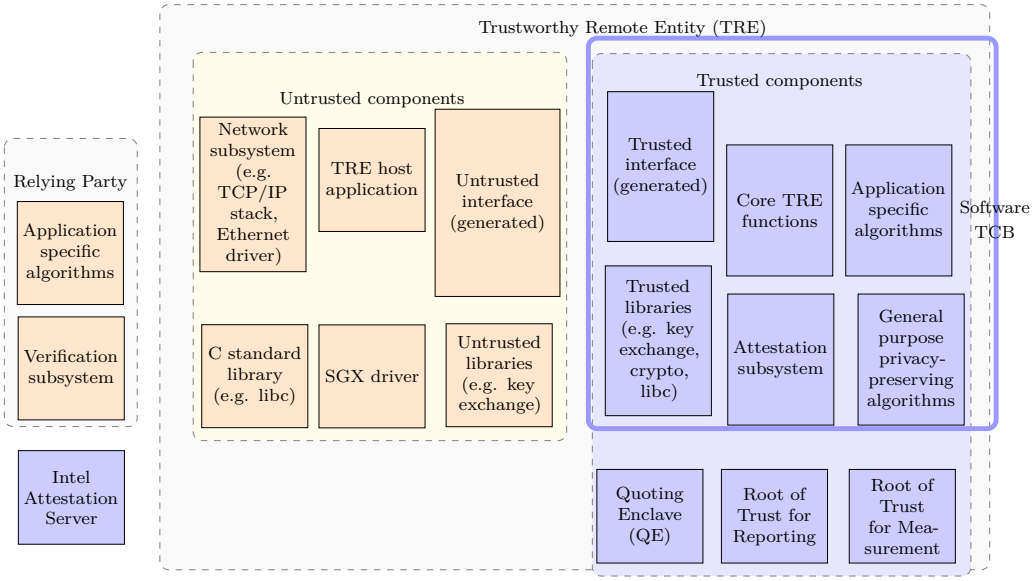

Figure 4.3: Component diagram for the Trustworthy Remote Entity implemented on Intel SGX.

4.5.5 Architecture Model

Figure 4.3 shows the main components of our SGX-based design of the Trustworthy Remote Entity. In order to minimize the amount of code that must be attested and trusted (*i.e.*, the software trusted computing base), the overall design philosophy was to minimize the components that are included within the enclave. Each TRE component is therefore classified as either trusted or untrusted. Other components in the architecture, such as the Intel Attestation Server (IAS) are trusted, but are not on the same platform as the TRE. These are assumed to be secure. We describe each of these components in detail below.

Relying Party: Each relying party includes some application-specific components, as well as some common components, such as functions to establish the secure channel and verify the TRE's attestation. Figure 1 shows only one relying party; however, there would be thousands or hundreds of thousands of relying parties in a many-party application.

Untrusted Components: The untrusted components include network communication, the host application, the platform's libraries (*e.g.*, the C standard library) and the SGX interface driver. These components are shown in orange colour and

do not need to be trusted because they do not handle sensitive information in the clear. Also, there are untrusted libraries and untrusted interfaces used to make enclave calls from untrusted code to the enclave through the trusted interface.

Trusted Components: Blue coloured boxes represent the trusted components, some of which are provided by Intel. Some of these components, such as the trusted libraries, are closed-source and provided by Intel. However, it is assumed that these components are trustworthy. Components that are implemented by the TRE developer would usually be open-source, so that verifiers could inspect these components and make informed trust decisions.

The trusted interface is used to marshal enclave calls from the untrusted part. Interfaces are automatically generated[4] and represent the enclave functions interacting with the untrusted part. We included the trusted interface in TCB, however the developer usually would not change the content of this interface directly, but would instead regenerate it using the supplied tool. Also, Intel provided Trusted SGX Libraries, Key Exchange Library, Crypto Library are included in TCB. However, the Intel provided libraries are closed source, therefore we are not able to show the size of these libraries. Also, one trusted key exchange Enclave Description Language (EDL) file is imported in custom EDL file. Other than the software TCB components, the Intel-provided Quoting Enclave, Root of Trust for Measurement, and RTR are shown in the trusted part.

The enclave itself consists of four components; core functionality to initialize the TRE, attestation subsystem including any method for attestation or potential future optimizations for remote attestation, general-purpose privacy-preserving algorithms, and application-specific algorithms.

Core Deployment: In addition to the TRE's critical functionality, there are several methods which are mostly SGX-related calls and cannot be changed from one design to another. Functions such as starting platform services, initialization of remote attestation, and generating keys for the secure channel are handled by this core component. Passing relying parties the Elliptic Curve Digital Signature

[4]At compile time, the Intel-supplied `edger8r` tool reads the Enclave Description Language file and generates the untrusted and trusted interfaces.

$$msg1 = ga||GID$$
$$S = gb||SPID||TYPE||SigSP(gb, ga)$$
$$msg2 = S||CMACSMK(S)||SigRL$$
$$T = ga||PS\_SECURITY\_PROPERTY||QUOTE$$
$$msg3 = CMAC(SMKCMAC, T)||T$$

Figure 4.4: Intel-recommended attestation messages, with binding to a secure channel.

Algorithm (ECDSA) public key and generating the enclave's Elliptic curve Diffie-Hellman (ECDH) key (*i.e.*, msg1 in Figure 3) operations are also handled by this component. Additionally, verification of attestation result is handled in this component

Attestation: The remote attestation also includes a local attestation to the Quoting Enclave, through which the QE verifies that the prover is a legitimate enclave running on the same hardware. The attestation subsystem follows Intel's recommended approach for attesting an enclave.

Application-Specific Algorithms: For the Smart Grid application, we implemented the basic commands of the DLMS-COSEM protocol. The DLMS-COSEM messages are 20-byte long hexadecimal data representing simply the query and/or amount of consumption. In this prototype, the DLMS-COSEM protocol is used for message exchange between the energy provider and TRE, and also between TRE and smart metering devices. Initially, an energy provider company performs remote attestation of the TRE and sends the first query asking for the consumption data of certain areas in which smart metering devices are deployed. The responsible TRE uses a DLMS-COSEM message to ask to all smart metering devices to report their measurements. Before responding, each smart metering device performs a remote attestation of the TRE. In this use case, both the energy provider company and energy consumers (individual clients) are considered to be relying parties. In general, each entity with which the TRE communicates should first perform a remote attestation of the TRE to establish whether or not the TRE is in a trustworthy state. After receiving the consumption data, the TRE performs the necessary

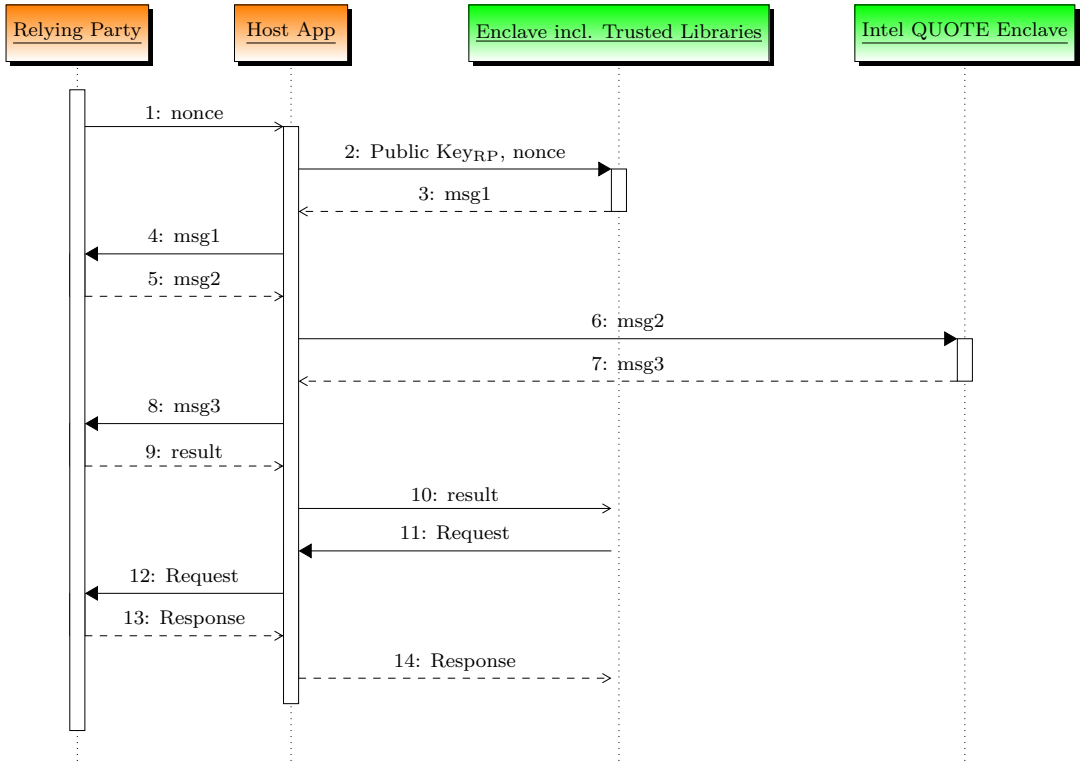

Figure 4.5: Flow of remote attestation and secure channel establishment SGX, and flow of DLMS COSEM GET Request Response Pair

privacy-preserving operations and delivers necessary information to the relevant entities. In this smart grid example, the aggregated amount of energy consumption is delivered to the energy supplier for purposes of balancing the energy grid.

General Purpose Privacy Preserving Algorithms: This component represents an independent feature, that is used to perform privacy-preserving algorithms on confidential data in the isolated execution environment. Although not included in our prototype, this library of functions could include operations used to achieve *e.g.*, differential privacy [114]. In this application, the TRE performs aggregation over the measurements such that the energy supplier cannot identify any individual consumer's usage patterns.

4.5.6 Interaction of Entities

Figure 4.5 shows message exchange flow of Relying Party, Host Application on the untrusted part, and trusted part on SGX hardware includes the Quoting Enclave and TRE's Enclave including the Key Exchange Library, cryptography Library and

the Trusted Interface. The flow starts with a nonce from the Relying Party, however also the Host Application could be the first entity to start the flow.

In the Smart Grid use case, the first flow starts from the energy provider as the first relying party, asking the recent consumption data to TRE. Second type of flow is triggered Host Application of TRE, asking all Smart Metering devices to attest itself and deliver the last energy measurement.

We numbered the exchanged messages in the flow from 1 to 14. The steps from 1 to 10 are representing the Remote Attestation with use of Quoting Enclave and Secure Channel Establishment including the SIGMA protocol.

Once the Enclave receives the Attestation Result shown in 10th step, Enclave calls the EGETKEY instruction and verifies the CMAC of attestation result and checks if the attestation is passed. If the result is passed, then Relying Party and Enclave can securely talk over the established channel.

For all experiments and for measurements of TRE, we used the same network setup. One SGX enabled machine is assigned as TRE including the trusted and untrusted application and another machine over the network is configured to be Relying Party. Since the Intel Attestation Services are not yet available to use[5], we deployed a simulation of IAS talking to our Relying Parties. Communication with IAS is also not shown in the flow.

Generation of messages are shown in Figure 4.4. First message (msg1) includes the context of Enclave not yet signed by Quoting Enclave. Without a use of SIGMA protocol, msg1 would not be delivered to Relying Party, instead the context of Enclave would be directly passed to Quoting Enclave by the Host Application. Since TRE's remote attestation is linked to secure channel, context and keys are exchanged to the relying party to be signed, over that message Quoting Enclave and the Trusted SGX Library generates the msg3 which is a 1460-byte message including the full attestation report.

Relying party contacts to IAS after receiving msg3 and depending on own security policy, deciding whether or not trust to TRE's identity, verification result

[5]Only a test purpose sandbox is available to use for Remote Attestation by Intel, requires registration. Last Accessed: 8th June 2016 https://software.intel.com/formfill/sgx-onboarding

message is including the state if the attestation is passed or not. Once the enclave receives the result at 10th step, session has been established and keys are exchanged. If relying party is convinced that the session is secure, they exchange the secrets.

Step 11th to 14th show an example Request - Response pair between TRE and relying party under secure channel. For Smart Grid case, it could be either from Relying Party to TRE as an overall consumption request, or from TRE to Smart Metering devices to collect recent measurements.

4.6 Evaluation

We benchmarked the performance of the Smart Grid application built on SGX technology. Also measured the TCB size of our prototype and described the differences from TPM-based TRE.

Operation	**Line of Code**	**Binary Size (kB)**	**Stack**	**Heap**	**Mean (ms)**	**Standard Deviation (ms)**
DLMS-COSEM GET Request & Response	1370	827.7	16 kB	4 kB	56.05	2.653

Table 4.10: Measurements on TRE's Operations (in one life-cycle, creation to destroy) on SGX Hardware.

4.6.1 Software TCB Size

Table 4.11 shows the sizes of the software TCB in the TPM-based TRE [92, 109], compared to our SGX-based TRE. All code was formatted using the same coding conventions and counted using SLOCCount.[6]

In the SGX-based TRE, the TCB measurements include both the third-party code, and the trusted library code (which was counted from the published library sources), since both are part of the TCB. This is a conservative worst-case TCB size, since it could be argued that the trusted libraries are common to all enclaves, and so will be audited/verified to a greater extent than third-party code.

[6] `http://www.dwheeler.com/sloccount/`
[7] SGX Drivers are not in TCB (2,483 LoC)

Table 4.11: TCB Sizes of different TREs

	TPM-TRE	**SGX-TRE**
Crypto Libraries	14,408	2,529
Communication	5,969	858
Memory Management	1,035	774
C/C++ Library	854	7,528
Core TRE	720	229
Application Specific	507	507
Attestation	221	364
Drivers	1,005	- [7]
SGX Trusted	-	2,968
Total	24,719	15,757

The total TCB size of the TPM-based TRE was 24,719 Lines of Code, reviewed in Section 3.6.3.1.

The most notable differences are the cryptographic library and communication subsystems. The TPM-based TRE requires a full TCP/IP stack, network interface driver, and TLS library. In contrast, the SGX-based TRE can outsource network connectivity to the untrusted host and only requires a small set of interface functions and cryptographic primitives provided by the `tCrypto` and `tKeyExchange` trusted libraries (*i.e.*, to support the SIGMA protocol and send and receive encrypted messages). Furthermore, the SGX `tcrypto` library makes use of *AES-NI*. The TPM-based TRE also includes the TPM hardware driver and library, but, in SGX, the drivers used to set-up and call the enclave are part of the untrusted host platform.

On the other hand, the SGX-based TRE includes a significantly larger set of C/C++ standard library functions, since the TPM-based TRE contains only a small customized subset of standard C library functions. The SGX-based TRE also includes other SGX trusted libraries, such as `tservice` (incl. `selib`, `tseal` and `tae_service`) and `trts`, which are not present in the TPM-based TRE.

The relative TCB sizes definitively show that even with our conservative posture of counting the SGX trusted libraries as part of the TCB, the SGX-based TRE has a smaller TCB and is thus easier to attest than the TPM-based TRE. The

absolute size of the TCB is within the same order of magnitude as fully-verified systems such as seL4 [22].

Crypto Libraries: Crypto Libraries including verification, signature operations, encryption/decryption functionalities are the major part of TCB in both TRE prototypes. TPM-based TRE was including 14,408 LoC as Cryptographic functions and protocols. In our prototype, we imported Intel provided Trusted Libraries, therefore we are not able make a clear comparison of Crypto Libraries, however we expect to have similar numbers as the required functions are not different. Only key exchange functionality has a slight difference as our prototype uses the SIGMA Protocol in the Intel provided library.

Communication: TPM-based TRE includes 5,969 line of code for TCP/ IP stack, Ethernet hardware driver and network subsystem. In SGX-based implementation we placed all these network subsystems to out of TCB, therefore we had at least %25 less TCB size than TPM-based implementation. However, for the communication of secure execution we needed to include a trusted interface in the TCB, and we considered the interface as the communication component of the TCB. Size of trusted interface is 858 lines, which adds again approximately as %5 percent of TPM-based TRE to our TCB. The overall TCB reduction is 5,111 lines of code in the Communication component.

Memory Management: This component is also provided by Trusted Libraries in SGX SDK. The difference of these libraries is that a Trusted Library is a safe[8] version of related standard library. We don't expect any major TCB size difference on this component.

Roots of Trust: TPM prototype uses a TPM driver sized 1,005 LoC, while SGX driver is placed out of TCB. We use the Intel-provided SGX driver however it could be a third-party driver as well, nevertheless driver does not have any impact on security guarantees of SGX. Within this component SGX prototype had 1,005 lines of code less TCB than TPM prototype. The Root of Trust for SGX Hardware

[8]In trusted libraries, illegal operations are removed from the source. SGX technology does not allow some instructions to be called within the secure context

is within the CPU, implemented in microcode. We rely on the fact that there is no possibility of accessing to microcode in a rational sense.

C Library: C Libraries are also imported from the Trusted Libraries. It is also assumed to be very similar amount of code, therefore we don't expect any saving on TCB size.

Core TRE: TPM prototype was including the core functionalities as we did and also a functionality for offline and online backups, using the sealed storage. We haven't implemented any sealed storage functionality on SGX-based TRE. Therefore, yet our component is relatively small, however this component would be nearly same size once all functionalities are implemented.

Application Specific: Application-specific code is independent from the hardware or technology we use. Therefore, we always assume to have ported code for application-specific code.

Attestation: To address the quoting performance of TPM chip, TPM-based TRE is using an optimized attestation protocol called Final State Attestation. This optimisation added 221 lines of code including the size of normal attestation subsystem into TCB of TPM prototype. In our prototype, the attestation performance on SGX was already sufficient to have a scalable result in many party applications. Therefore, we didn't have any extra optimisation on the attestation subsystem. On the other we use the attestation functionality from Trusted Library, therefore we are not able to provide the size of attestation subsystem. Potentially we expect a larger attestation component in our prototype once an optimisation protocol is applied to SGX's attestation mechanism.

4.6.2 End-to-End Performance Evaluation

To measure performance, we deployed our prototype SGX-based TRE on the same hardware used in our initial benchmarks. We simulated the SP and SMDs using a second platform connected via a local ethernet network, including a single network switch. Interaction with the Intel Attestation Server is necessary to verify an

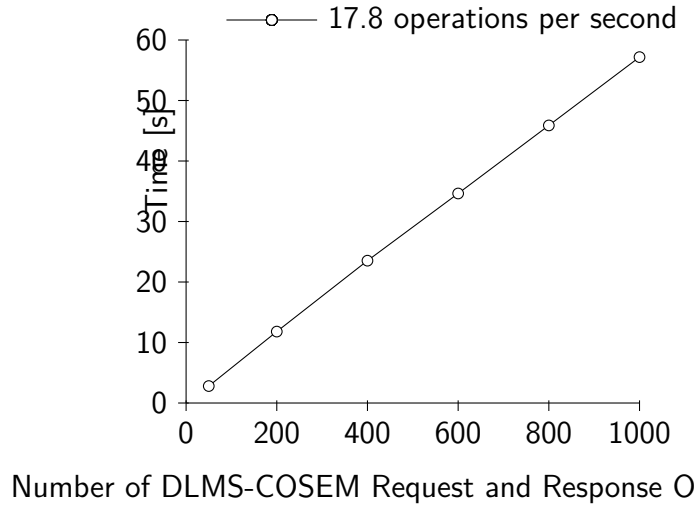

Figure 4.6: Overall Performance of TRE on SGX

attestation; however, since the relying party performs this, it does not affect the scalability of the TRE, and is thus excluded from our performance measurements.

The overall objective of this experiment was to determine how many clients a single TRE can support in a many-party application. This is an important metric because it determines the total number of TREs required for the application (*e.g.*, to handle communication with all smart meters in a country such as the UK). Our test suite measured the time taken to perform the following sequence of operations for a single relying party: (1) create the TRE enclave, (2) establish a secure channel to an SMD over the network, (3) perform remote attestation, (4) encrypt, send, decrypt, and process messages, and (5) destroy the enclave. In Figure 4.6, the average time required for this sequence was 56.05 ms with a standard deviation of 2.653 ms (measured over 100 samples), shown in Table 4.10. Repeating this test for a batch of participants confirmed that this time scales linearly in the number of relying parties, as expected. In practice, the TRE enclave would not be created and destroyed between every interaction. This would reduce the time per interaction by approximately 10 ms (based on Table 4.2), bringing the average latency to approximately 46 ms. In the smart grid scenario, the TRE must perform two interactions with each SMD in every 30 minute period [109], and thus a single SGX-based TRE can support approximately 20,000 relying parties. This is the

same number of participants as the TPM-based TRE, which enjoys the performance benefits of various optimizations, including a highly scalable attestation protocol [92]. We plan to investigate similar optimizations for the SGX-based TRE.

In this case study, the processing performed by the TRE is not computationally intensive. As a result of this, a significant percentage of the time per interaction is spent establishing the secure channel and performing remote attestation. Although this may not always be the case, we argue that there is a large class of TRE applications for which this will apply, and therefore that it is beneficial to optimize these processes.

4.6.3 Security Evaluation

The security of the implemented system relies on cryptographic libraries and underlying secure hardware primitives. Furthermore, in order to benefit from the available SGX instructions, we created a template (Section 4.5.5) where the CPU instructions can offer meaningful security guarantees. We do not provide any formal verification or security specification. We implemented our SGX-based prototype by utilising the available host and in-enclave instructions evaluated in Table 4.2. Compared with the TPM-based TRE, our solution provides better integrity (only Internal Enclave Functions can execute) and confidentiality guarantees (only Internal Enclave Functions can aggregate the data read from smart-metering devices). We are not aware of any attacks that can force our enclave to behave differently at runtime, while in TPM-based TRE case the application code can be modified after the attestation with DMA attacks. Our threat model keeps the attacks that compromise the CPU completely out of scope, we assume that CPU instructions operate as specified. Suppose there is an attack to compromise the CPU instructions. In that case, we rely on Intel's capability to push a TCB update for the microcode and revoke the attestation reports of all enclaves running on unpatched hardware. We claim security guarantees in respect to our assumptions and the threat model. The challenge we solve is to offer meaningful integrity and confidentiality guarantees by utilising the available SGX instructions we previously investigated. Our goal is to

enable a better security level which was not available previously with a TPM-based implementation. We keep the micro-architectural attacks that compromise the CPU completely out of scope. Our contribution is that we show first time how SGX instructions can be used to offer security for many-party applications. SGX does not protect against side-channel attacks and does not provide oblivious memory. If the control flow or data memory access pattern of the enclave depend on secret data, this secret information could potentially be leaked to the adversary through side-channel attacks [115] (since the adversary may control the host platform). However, by design, neither the control flow nor the memory access pattern of our SGX-based TRE depend on any secret data. The number of participants communicating with the TRE is not secret information, since this can be inferred from network traffic analysis. Furthermore, our TRE implementation does not perform any OCALLs, and thus is not vulnerable to Iago attacks [116]. Since the TRE is event-driven [92], the SGX enclave is invoked in response to commands or messages originating from the relying parties. The enclave uses the return values of the ECALLs to issue commands to the host platform (*e.g.*, to initiate new connections and send messages). While the TPM-based TRE does not protect against hardware attacks on the platform's memory, the memory-encryption functionality of SGX ensures that the SGX-based TRE's secrets are protected even against an adversary with the capability to read the platform's memory.

4.6.4 Computational Performance

We performed measurements to see how many clients can be supported with one TRE in a many-party application. Therefore, we measured the time taken for end-to-end operations. The setup of the measurement environment is same as we performed the benchmarks.

Meaning of one full operation includes creating an enclave, performing remote attestation, establishing secure channel, exchanging encrypted messages and decrypting on receiving entities, and destroying the enclave. This operation took 56.05 ms as a mean value and 2.653 ms standard deviation. The measured enclave's

binary size is 827.7 kB and TCB size is 1370 lines of code excluding the trusted libraries since they are closed source. The enclave is configured to use 4 kB Stack and 16 kB Heap as a memory. These values are the minimum possible memory configuration for this application. We used the SGX_EMMT (Enclave Memory Measurement Tool) bound to GDB in Linux SGX SDK. This tool shows the peak usages of the memory at runtime. We measured the peak usages as 1976 bytes for Stack and 16384 bytes for Heap. The configuration must be 4 K as the SGX_SIGN tool require so, therefore we configured it as 0X1000 and 0X4000, for Stack and Heap repeatedly, anything below these values cause crash at runtime.

The performance result corresponds to 17.8 operations per second and we benchmarked the SGX-based TRE 50 to 1000 participants shown in Figure 4. The linear graph shows that TRE can support 32,040 clients in 30 minutes as a trust brokerage between energy provider authority and energy consumers.

Computational performance of slowest TPM-based TRE prototype was supporting 20,000 clients in 30 minutes of time period. This result is including FSA optimized attestation mechanism.

4.6.5 Architectural Evaluation

TPM-based TRE relies on a requirement that platform must have no untrusted component, therefore all unnecessary components are removed from the system. TPM prototype is running bare metal without an OS and all system is included in TCB.

In the SGX-based prototype, by hardware it is possible to have isolated execution in the enclaves. Therefore, the requirement of removing all untrusted components is not required. Only the components or libraries imported into the isolated execution must be trusted.

Since the TPM-based TRE has no untrusted component and once the state of the system is known, it cannot change it at run time. We assume that the source code is formally verified or have no runtime vulnerability that will cause a change at runtime. Therefore, TPM-based prototype uses FSA which proves TRE did not

change it is state and relies on the Quote operation that was made in first request. So once a quoting is done, there is no need to perform slow quoting for each participant.

Enclave's running on SGX hardware have strong isolation, therefore FSA optimisation is also possible in terms of architecture on SGX-based TRE.

4.7 Closing Remarks

In this chapter, we built a system serving to high number of participants to process their private data with a publicly agreed code base. In the next chapter, we will make the threat model more difficult by enabling the use of private algorithms. Processing private data with private algorithms in someone else's computer makes the problem difficult and we explore the feasible industrial scenarios in Section 5.9.

As future work, we plan to optimize the design and implementation of our SGX-based TRE and explore the use of multiple enclaves and different enclave structures for the TRE.

* * *

Contributions

Our principal contributions in Part II are:

(1) Empirical performance measurements of basic SGX operations that are used in many-party applications (Section 4.3).

(2) An architectural design and prototype implementation of a representative real-world application, which could serve as a template for other such many-party applications (Section 4.5).

(3) A systematic comparison of an SGX-based TRE against a previous TPM-based system, in the context of many-party applications (Section 4.6).

Chapter Recap

In this chapter, we explore the use of Intel SGX[13–15] to implement a TRE for many-party applications. Firstly, we compare SGX with previous trusted hardware technologies in terms of specific characteristics required for many-party applications. In particular, we consider attestation performance in terms of the time required to perform a single attestation operation[16] and the overall rate at which a platform can perform attestations.

Secondly, we designed and implemented the equivalent of TPM-based TRE using SGX for the smart grid use case. Due to the fundamental differences between the architectures of SGX and previous technologies, the implementation was far from a straightforward porting task, and required a complete redesign of the system. Thanks to the SGX architecture, our implementation requires significantly fewer lines of code, which both reduces the burden on the developer, decreases the likelihood of code defects, and minimises the amount of code that must be trusted by the verifier. However, the use of Intel's trusted libraries (*e.g.*, for cryptographic operations), which are only provided as closed-source binaries, makes it difficult to inspect and quantify the exact size of the software TCB. Although our implementation targets a specific application domain, we argue that its core features are common to all many-party applications, and thus could serve as an architectural template for such applications.

Finally, we performed a comparative evaluation of the performance of our SGX implementation against the previous TPM-based implementation. Using the smart grid as a case study, we benchmarked the systems' end-to-end performance in a representative communication task (*i.e.*, obtaining the most recent consumption measurements from a smart meter). The results show that even an unoptimised SGX implementation exhibits comparable performance to the optimised TPM-based system, whilst providing stronger security guarantees.

Part III

Private Code and Private Data

Chapter 5

Taxonomy, Modeling, Development, Attacks, Solutions on Protected-Code Loaders

Contents

Many applications are built upon private algorithms, and executing them in untrusted, remote environments poses confidentiality issues. To some extent, these problems can be addressed by ensuring the use of secure hardware in the execution environment; however, an insecure software-stack can only provide limited algorithm secrecy.

This chapter aim to address the above problem, by exploring the components of the Trusted Computing Base in hardware-supported enclaves. First, we provide a taxonomy and give an extensive understanding of trade-offs during secure enclave development. Next, we present a case study on existing secret-code execution frameworks; which have bad TCB design due to processing secrets with commodity software in enclaves. This increased attack surface introduces additional footprints on memory that breaks the confidentiality guarantees; as a result, the private algorithms are leaked. Finally, we propose an alternative approach for remote secret-code execution of private algorithms. Our solution removes the potentially untrusted commodity software from the TCB, and provides a minimal loader for secret-code execution. Based on our new enclave development paradigm, we demonstrate three industrial templates for cloud applications: ① computational power as a service, ② algorithm querying as a service, and ③ data querying as a service.

Parts of this work have been published in peer-reviewed papers and included in conference presentations [6–9].

In the previous chapter, we run enclaves with a code that is agreed mutually by all participants of the computation. Our aim in this chapter is to offer a native solution for running private algorithms with enclaves. We show why previous

solutions with interpreter enclaves fail (Section 5.6), and we present practical templates for industrial use (Section 5.9). In Chapter 4, enclaves were running with a publicly known code. Now, in this chapter enclaves will have an additional capability (Section 5.8.1) and address an extended threat model.

* * *

Chapter Motivation

The motivation for TCB minimisation comes from the secrecy guarantees that depend on a remote system's TCB components. There are a number of ways in which developers can cause security problems in a system. It is common for developers to include third-party software in their enclave TCB. Unfortunately, they often fail to perform security and compliance analysis between the third-party package and the underlying hardware. Additionally, developers may fail to understand hardware and software co-design while constructing secure systems. Careless construction of composite parts within a TCB may also cause the loss of initial security guarantees of the hardware. This may be the main, sometimes initial, source of security problems. In this chapter, we analyse two existing frameworks that address client-side secret code execution, but their TCB suffers from bad practices explained below. To solve these architectural design problems, we present a solution with a smaller TCB, and secure TCB composition for secret-code execution in remote environments.

We aim to eliminate these bad practices, summarising the motivation for this chapter in three points:

- **Increase of TCB Size**: The TCB size must always be minimal in order to avoid security risks, and enable possibility of formal verification.
- **Weak software in the TCB**: The third-party software packages included in a TCB must pass the security requirements of the all assets.
- **Non-Compositionality of Security**: Two secure components may not necessarily comprise a single secure composition. Even secure software in the TCB may create additional security issues due to composition problems

with the underlying hardware. This may also void the hardware security guarantees.

5.1 Strongly Private Algorithms

Managing trust in remote execution environments is an enduring challenge. This is due to the fact that privacy-sensitive data and private algorithms remain unprotected in remote computers. There are a number of reasons why owners of private algorithms may need to run their algorithms in an untrusted environment. This may occur when the Algorithm Owner (AO) requires the capabilities of a Hardware Owner (HO), for example, to gain larger computing power in the cloud. In this case, the cloud infrastructure provider would be compensated and should take responsibility for an algorithm and system security. An alternative reason to run a private algorithm in a remote environment would be for the benefit of the *HO*, for example, the distribution of an industrial software product to end-users. In this case, the *AO* would be compensated, and thus responsible for security. It may also be the case that the consumer, whether that be the *AO*, or the *HO*, is responsible for security.

5.1.1 Ownership Taxonomy:

We use the following taxonomy throughout the rest of this chapter. Our classification based on asset owners' behaviour is taxonomic, therefore we call it the ownership taxonomy, some readers may call this section a system model/assumptions only about certain role holders. A ***Hardware Owner*** refers to any entity managing its own computational system. An ***Algorithm Owner*** refers to any entity who owns the intellectual property of the secret code. Cloud infrastructure providers offer computational power, and they maintain the hardware and the software stack. In a cloud environment, the *HO* is a threat against the secrecy of private algorithms of the customers. Similarly in DRM[1], a *HO* is considered to be the end-user who may threaten the private algorithms with a commercial interest. From the *AO*'s

[1]Digital Rights Management in Trusted Computing. `https://en.wikipedia.org/wiki/Trusted_Computing#Digital_rights_management` visited on 04/Jun/2019.

position, the *HO* is considered to be both the cloud-provider (executing server-side code) and the end-user (executing client-side code). In short, the *HO* may be one or both of the following entities:

- an entity who generates revenue by selling the computational power
- and/or, an entity that needs the private algorithm to compute a secret value in her environment.

For both reasons, the *HO* must offer a trustworthy confidentiality service — either procedurally or through technical means — in order to convince the *AO* to use its services. Otherwise, the *AO* must take countermeasures before releasing its private algorithm.

The *AO* can use obfuscation techniques to protect the code before sending it to an execution environment. However, obfuscation methods are open to reverse engineering if they are software-based only. The goal of the obfuscation method is another aspect, for example, it might aim to obfuscate the syntax only or the semantics. Authenticated obfuscation techniques for protecting hardware IP [117, 118] are orthogonal to our research but outside of the scope of our work. On the other hand, the *HO* must make an effort within confidentiality management to gain the trust of the *AO*. For example, the *HO* develops the software stack to offer isolated execution environments and it aims to provide a remotely verifiable Trusted Computing Base.

A ***Data Owner*** *(DO)* refers to an entity who has privacy-sensitive inputs. In the Section 5.9, we consider two cases: ① with communication of the *AO* and the *HO* where their assets and adversarial settings are considered, ② similarly, between the *AO* and the *DO* where the *DO* as an entity controls the *HO* (with other words data owner has full control of the hardware owner's environment). The case where the *HO* may collude with the *AO* against the *DO* is out of the scope of this chapter.

5.1.2 Taxonomy of the Private and Public Assets

The *AO* and the *DO* have a choice to keep their assets as private or public. An asset, either an algorithm or a data-set, might be weakly or strongly private/public.

Called as	Refers to	Properties	Example
Private Data or Private Algorithm	**Strongly Private**	Secret and Protected	A Secrecy-Critical Application Logic or Data-set
Not Private	**Weakly Private**	Special but not Protected	A Code or Data uploaded to today's cloud servers
Not Public	**Weakly Public**	Open but not Trusted	A Code or Data; too complex or too big for inspection
Public Data or Public Algorithm	**Strongly Public**	Inspected and Attestable	A Code or Data with an evidence of trustworthy measurement

Table 5.1: Definition of the Private and Public modifiers.

We summarised the meanings of these new concepts in Table 5.1. In the rest of this chapter, we use *private* or *public* keywords for the assets which are desired to be *strongly* private or public. We consider the *weakly* private or public assets to be *not private* and *not public*. These concepts are defined as follows.

Weak-Private Assets: An algorithm or a data-set may not be publicly accessible, and be private against any third-party entities. Suppose that the asset owner sends this piece of information to a cloud platform for a remote computation. We classify this case **Weakly Private**, as the cloud operator may access the assets.

Strong-Private Assets: A security mechanism in the cloud may protect the confidentiality of these assets. Suppose that the threat model of this mechanism considers a malicious cloud owner. We call the assets **Strongly Private**, as they are protected against the insider threats in the cloud.

A private asset, therefore, can be a *strongly private asset* if the execution model satisfies the confidentiality requirement. We describe our secret-code execution model in the Section 5.8 which allows *Private Algorithms* and *Private Data* to be strongly private.

Weak-Public Assets: An asset might be accessible, but it may be too complex to analyse or too big to process. If an asset does not give much evidence about its trustworthiness, we call it **Weakly Public**.

Strong-Public Assets: We often use the keyword *public* to refer to the inspectable and attestable assets. For example; a public algorithm refers to a piece

of code that is ① open for inspection by members of public, and ② attestable through a trustworthy evidence. These are the **Strongly Public** assets.

5.1.3 Distinguishing the Private Data and the Private Algorithm

Much of the current research [2, 119–125] is to keep *data* confidential in a Trusted Execution Environment. The novelty of our work is that, in addition to the input values and constant values used in an algorithm, we are adding algorithm secrecy. Our aim is to keep additional information secret *how* and *where* these values are used.

A function in a computer program may include multiple assets such as sub-methods, operators, and constant values. The *AO* defines the *order* and *types* of these assets in a function.

We summarise these assets in the following expression (in Figure 5.1) to distinguish the data and the algorithm. For the given function (f), the *AO* may not necessarily provide the data (*input (x) and constant (c)*). To point out the difference, the *AO* nevertheless defines the mathematical operations and the application logic. In a decentralised setting, one or more *DO* can provide confidential values, such as the inputs and the constants. The *AO* would provide the secret behaviour of how the algorithm uses the given values.

The code segments and data segments are both held in the memory. Encrypted memory pages can protect these two. In this chapter, we consider protecting the algorithmic behaviour of an application. For this reason, we use the term *private algorithms* where black-box querying a piece of code to learn its behaviour is limited.

$$function_f(input_x) = method_m(input_x)operator_o(constant_c) \tag{5.1}$$

Figure 5.1: Splitting the Algorithm and the Data in a function.

5.2 The Problem of Secret-Code Execution (SCE) with Private Algorithms

The success of a commercial algorithm may be measured on how widely it is distributed. However, the integrity and confidentiality of this algorithm must also be ensured. The problem arises when either party cannot trust the other, whether that be the execution environment, or the product.

The problem is that the *AO* needs to run code on the *HO* and the *AO* does not want the *HO* to discover any details regarding the code.

In untrusted remote environments, a computation including commercially valuable algorithms (for example, feature engineering in machine learning, business logic, or financial applications) may pose confidentiality concerns. Ideally, an *AO* would keep their private algorithm in their own physical location. A decentralised setting, however, requires the *AO* to send their private algorithm to an execution environment that may be owned by another entity (*e.g.*, the *DO* or the *HO*). A hospital holding secret data may require all computations to be performed in their environment (considered as the *DO* or *HO*), while the hospital environment is a threat against secrecy of the algorithm. If the *AO* uses a cloud service for more computational power, uploading the secret binary to cloud service may leak the application code due to reverse engineering. In another case, an authority may need to run a private algorithm on computers of end-users (considered as the *HO*), requiring security guarantees to be managed by the *AO* or the *HO*.

5.2.1 The Security Problem on Hardware-Software Composition

In order to solve the problem of private algorithms, both distrustful parties could utilise secure hardware enclaves. A fundamental question arises as to whether the *AO* should develop its enclave, or trust the enclave developed by the *HO*.

In Section 5.3, we explain the *enclave* concept and its development model in detail. In short, the term *enclave* refers to the area[2] of one entity that is surrounded by another entity. With other words, an enclave is an application loaded into an isolated memory area, surrounded by a host environment (a potentially malicious environment or another abstract entity). Intel introduced the enclave concept into the security world with their Trusted Execution Environment on a new instruction set called SGX [15]. Enclaves are developed by an entity called ***Enclave Developer*** *ED*, explained in Section 5.3.

Developers may use the existing application binaries inside secure hardware enclaves with small or no porting effort. However, third-party packages programmed with no security in mind will surely bring security risks. TEEs cannot convert a non-secure application to a secure application. In the building-block approach to application design, even if each block is secure, the overall system may not be secure due to the non-compositionality of security [62].

Practical TEEs [126] are currently the subject of widespread research, as their correct use and capabilities are not yet fully known. The aim of this chapter is to answer the following research questions:

- How can we correctly utilise TEEs' security guarantees to protect private algorithms?
- What are the responsibilities of TEE application developers?
- What is the impact of the software TCB components to code confidentiality?

In this chapter, we examine these questions in recent studies and in our solution. We show the challenges and responsibilities on the TCB design and its requirements for secret-code execution in a remote environment. We analyse the existing frameworks for confidentiality management, and we evaluate the attacks against code secrecy in the software stack. Finally, we present a new solution with a smaller TCB size and address a stronger adversary model.

[2]memory area or network area within the context of computer science, or physical area in terms of lands. In the context of this chapter, an area for an enclave is a memory area.

5.2.2 Research Direction of the Chapter

The Horizon2020 funded research projects under SERECA[3] focus on a number of goals to build secure enclaves. These goals include: application partitioning [127]; trusted architectures for web services [128, 129]; container architecture [130] and library support for unmodified applications (SGX-LKL[4]); better integrity [131] and isolation [132]; and enclave memory safety [133] in the cloud.

European Commission funded several projects on TEE research under SecureCloud (TRUSTEE)[5]. The main direction of TRUSTEE projects is dependability in the cloud. In the context of SecureCloud, the *dependability*[6] notion includes confidentiality, but this applies to data only. The SecureCloud solutions [121–125] focus on processing confidential data via public algorithms in an untrusted cloud. There is no concern about keeping the algorithms secret in SecureCloud projects. The advantage of our work is that we keep the algorithms secret in the untrusted cloud. Both SERECA and SecureCloud projects consider the confidentiality and the privacy of the *data* only.

The nature of enclaves requires enclave code to be publicly known. The existing work focuses heavily on private data processing through those enclaves. The difference with this chapter is that we use the publicly known enclaves to enable private algorithms in the cloud. We show two different approaches in the Section 5.4: ① the *HO* develops an enclave that maintains the code-secrecy after its release, ② the *AO* develops an enclave that ensures the code-secrecy before its release.

There are other projects worth mentioning listed under Intel SGX Academic Research[7] page. GrapheneSGX [134] provides library support for unmodified binaries. Projects [2, 119, 120, 135, 136] on privacy-preserving data-analysis provide data confidentiality. Ryoan [137] provides a two-way sandbox for enclaves.

[3] `https://www.serecaproject.eu/index.php/publications/papers` visited on 04/Jun/2019.

[4] Linux Kernel Library. `https://github.com/lsds/sgx-lkl` visited on 04/Jun/2019.

[5] `https://www.securecloudproject.eu/papers/` visited on 04/Jun/2019.

[6] `https://www.securecloudproject.eu/project-overview-securecloud/` visited on 04/Jun/2019.

[7] `https://software.intel.com/en-us/sgx/academic-research` visited on 04/Jun/2019.

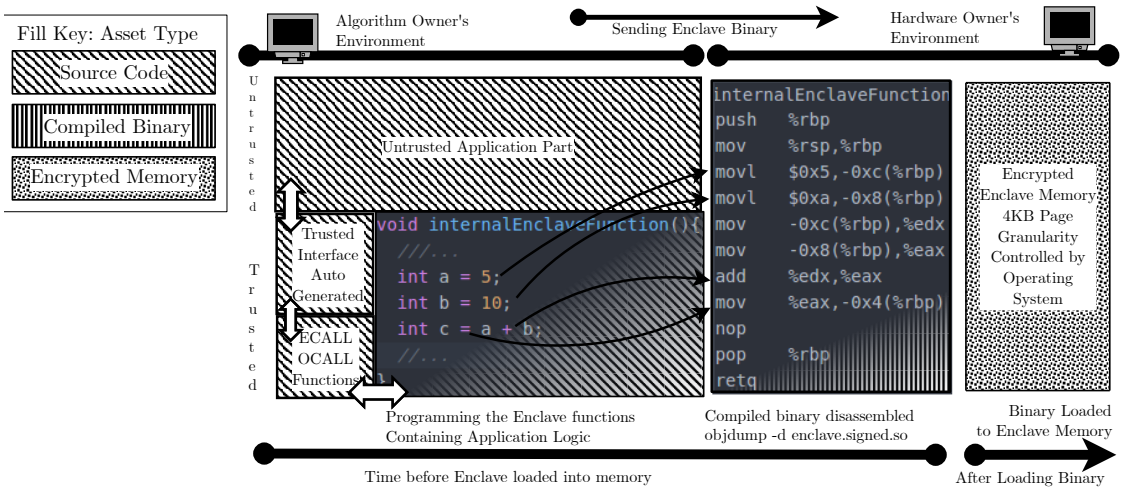

Figure 5.2: Default Enclave Execution Mechanism of SGX between the Algorithm Owner (*AO*) and Hardware Owner (*HO*). The operator of the machine, the *HO*, can inspect the enclave code by reverse engineering the enclave binary. Enclaves must not include any secrets embedded in binary. Enclaves can, however, receive secrets through a secure channel or recover sealed data.

Moat [138] helps to formally verify the enclave code. AsyncShock [139] exploits the time-of-check-to-time-of-use (TOCTTOU) bugs. VC3 [140] from Microsoft provides both data and code confidentiality, but this applies to MapReduce functions only. Controlled-channel attacks [115] can leak the secret data. We analysed these attacks against the other frameworks [141, 142] executing private algorithms client-side in Section 5.6.

5.2.3 How Universal is the Enclave Research?

This chapter is in the domain of the Enclaves and TEEs. We currently use SGX-enabled hardware and we make our contributions with Intel SGX enclaves. However, enclave research goes beyond Intel's SGX hardware. ARM and AMD also provide TEE solutions. Microsoft recently announced[8] the OpenEnclave SDK which provides an abstraction to the underlying hardware.

An enclave programming model may be considered as independent from the SGX hardware. For example, a trusted hardware [143] may adopt Intel's SGX enclave

[8] `https://github.com/Microsoft/openenclave` visited on 04/Jun/2019.

programming model. Ideally, developers may replace the underlying hardware with an open hardware solution, providing stronger security guarantees.

5.3 How do SGX Enclaves work?

The enclave programming model requires an Enclave Developer *(ED)* to program, compile and trigger the binary into allocated enclave memory before execution [144]. In a decentralised setting, there could be three separate entities who program, access, and call the enclave binary. An issue may occur, however, if these entities are from different parties with conflicting interests.

A key feature of SGX hardware is that the CPU can measure the enclave memory [75]. This measurement represents the identity of the enclave. An enclave can contain any C functions except a few illegal instructions [145]. Enclaves can directly or indirectly communicate with the system, other applications, other enclaves, and other entities in the network. We give further information ① on enclave development with different SDKs in Section 5.5, and ② explain some good and bad practices on enclave TCB design in Section 5.7.

Other than splitting an application into **Trusted** and **Untrusted** components (partitioning), the trusted part (for example, enclave binary as a shared object; .so file in Linux) contains [146] two parts. These are the **Internal Enclave Functions (IEF)**, which contain application logic, and the **Interface Functions**. Along with other parts of the enclave, the *IEF* are open for inspection before the initialisation process (Fig. 5.2). As is standard, if the *AO* (who places the application logic into enclave binary) and the *HO* (who calls the binary and executes the binary) are mutually distrustful, then the *HO* may gain information about the *IEF* and the algorithms compiled into the enclave binary.

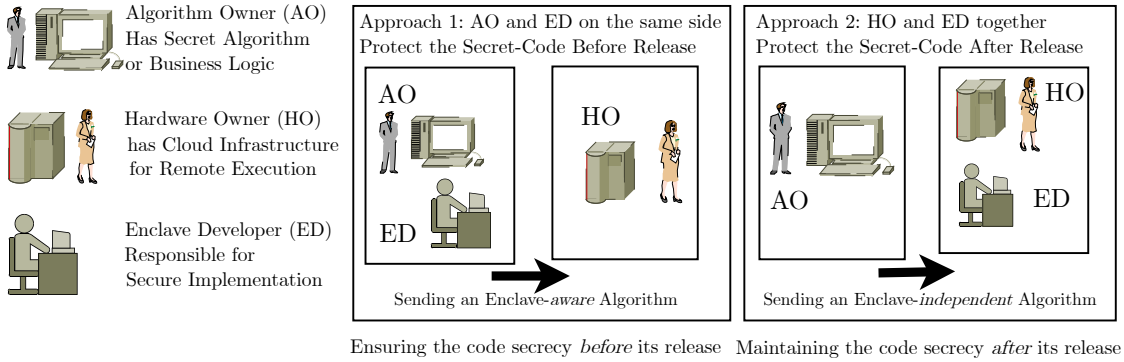

Figure 5.3: This figure shows the three separate stakeholders in running private algorithms inside enclaves; and two approaches to secret code protection. **Approach 1**: Enclave Developer (*ED*) works with the Algorithm Owner (*AO*) to protect the code within a secret part of the enclave. This requires the *AO* to perform early operations on the private algorithm before release. **Approach 2**: The *ED* works with the Hardware Owner (*HO*) to develop a publicly known interpreter enclave. This does not require the *AO* to perform early operations on the private algorithm.

5.4 What Differs on Protecting the Private Algorithms Before Releasing it or After Receiving it?

There are two ways to secure private algorithms between parties with conflicting interests. The *AO* may secure the private algorithm via early operations on the algorithm prior to delivery. Alternatively, the *HO* may preserve the secrecy of the private algorithm after the delivery.

Both Approach 1 (protection before) and Approach 2 (protection after) (Fig. 5.3) have advantages and disadvantages in terms of usability and security. In Approach

	Approach 1	**Approach 2**
Responsible for Secrecy	Algorithm Owner	Hardware Owner
Secrecy Ensured	Before Sending the Code	After Receiving the Code
Development Type	Enclave-Aware coding	Enclave-Independent code
TCB/Threat Analysis	In Section 5.8	In Section 5.6
Enclave Developer	The Private Algorithm must be developed for Enclave	The Enclave must be made for Private Algorithm

Table 5.2: Two approaches on *Secret-Code Execution* through SGX enclaves.

2 (Fig. 5.3), the *HO* provides an enclave containing the dynamic code loader and execution method for the private algorithm. To provide this, the *ED* works with the *HO* to create a publicly known enclave for common use. The publicness of this enclave code is required, as the *HO* must convince the users to trust her enclave implementation. If this enclave code is not open-sourced, the *HO* can implement hidden functionalities. She will be free to signal the secret-code and the secret-data to herself through covert channels. In this case, the *AO* does not need to perform any extra operations on the private algorithm. This approach gives better usability to the *AO*, but it requires her to trust to the TCB designed by the *ED* and the *HO*. Section 5.6 evaluates the confidentiality management of secret-code handled by the *HO*, as is displayed in Approach 2 (Table 5.2). We classify the development types as follows: **Enclave-aware coding**: The developers create the algorithms to operate as the enclave code with no intermediary layer. **Enclave-independent coding**: The developers create the algorithms for an intermediary layer (such as interpreters). This layer may or may not run in an enclave.

SGX hardware can help to preserve the confidentiality of the enclave applications against direct memory peeping and memory snooping attacks. Also, the *ED* can optionally implement the enclave with resilience to a set of side-channel attacks in order to address a stronger adversary model. To use an enclave for secret computation, developers may create publicly known enclaves with code-loading ability combined with interpreters, and load a secret-code on top of the interpreter to execute it. A weakly developed enclave may provide neither confidential data processing nor confidential execution of the secret-code.

* * *

Development, Attacks, Solutions on Protected-Code Loaders

After the previous introduction (problem definition on private algorithms and brief background), taxonomy and modeling, now we move onto the next half of the chapter where we make our novel contributions.

We begin by evaluating candidate SDKs for private code execution and analyse their TCB size. For secret JavaScript code execution on Graphene SDK, we perform a security analysis on previously deployed systems and we report novel code leakage through interpreter-based side-channel attacks. We highlight the enclave developer's responsibilities on secure enclave development.

Our novel internal enclave partitioning paradigm and late-load mechanism similar to shell-code injection present Early Private Mode and Protected-Code Loader respectively. We demonstrate three industrial and practical use cases where private data and private algorithms from different providers can execute in a remote environment owned by a third resource provider. Finally, our methods help to build systems serving Algorithms as a service, Data as a service and Computational Power as a service.

5.5 SDK and TCB for the Interpreter Enclaves

Interpreter Enclaves for private algorithms may consist of two parts: the dynamic secret-code loader mechanism and the script interpreter mechanism. The interpreter mechanism with rich functionalities may have system dependencies.

At the time of writing this chapter, Intel SGX SDK only supported the use of C programming language to create an SGX enclave. A recent survey by Data Science [147] shows that researchers prefer Python and other languages over C and C++ for data analytics. In addition to this, web applications and computations on the client side widely use the Javascript language. To execute Python or Javascript code inside an enclave, the language interpreter must be embedded or ported to the enclave binary. A common way is that *ED* may prefer to port or adopt the open-sourced interpreters. To develop such an enclave, the *ED* can use the partitioning method using Intel SDK (recommended if no system library dependency is required). This can be achived by removing or replacing the illegal instructions and providing a trusted interface to the ported binary. In this method, the TCB size stays smaller in comparison to unmodified binaries (*e.g.*, an enclave comprising an unmodified application code) supported with a Library Operating System (LibOS). If developers

prefer running unmodified binaries that require system support inside an enclave, they must either use a shim layer [148] to forward the calls, or embed the unmodified binary interpreter within a LibOS, (*e.g.*, Graphene LibOS with GrapheneSGX SDK). The Linux Kernel Library (LKL) is an alternative LibOS for this goal. Both of these LibOSs can support unmodified interpreters, but the act of using a LibOS increases the TCB size dramatically. The interpreters are able to handle complicated tasks and provide rich functionality; however, they require a LibOS support.

5.5.1 Understanding the TCB of Enclaves

The SGX development model helps developers to create applications that, ideally, have a minimal TCB size. Because the larger TCB on a commodity computer may contain potential vulnerabilities, minimising the TCB reduces the risk of having vulnerable code. Developers are responsible for deciding what to include in the TCB of an enclave. If a developer includes an arbitrary code from third-party resources in an enclave, this may weaken or destroy the integrity and confidentiality guarantees of the SGX hardware. The design and implementation of enclave (*i.e.*, TCB) components are crucial for the security of the application. In other words, the underlying secure hardware may not protect the assets processed by the bad software stack. The co-design of hardware and software ensures proper management of the integrity and confidentiality guarantees.

5.5.2 Comparison of SDKs for Enclaves

An application may require system calls to operate, or it may run entirely without any system dependencies. Depending on the requirements of an application, there are three different ways of developing enclaves:

- Partitioning an application and using trusted interfaces within the enclave. We describe application partitioning with Intel SDK in 5.5.3.
- Using LibOS inside an enclave to support the dependencies. We describe development with LibOS with Graphene SDK [134] in 5.5.4.
- Using shim layers to filter system calls from enclave to outside world [148].

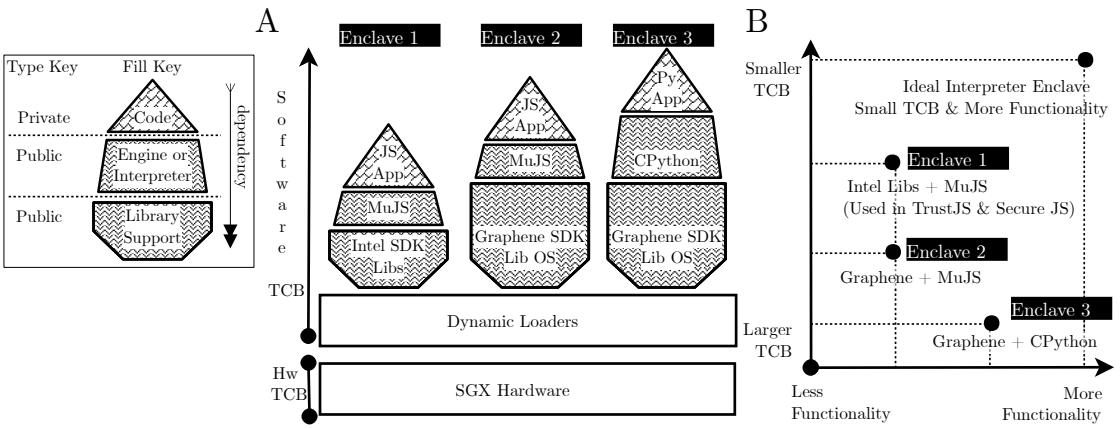

Figure 5.4: Three different TCB to deploy Dynamic Code Loaders and Interpreter Enclaves for Private Algorithms. **A)** TCB Size may vary depending on SDK and interpreter of the language. Graphene SDK TCB > Intel SDK TCB. CPython TCB > MuJS TCB. **B)** TCB Size vs Functional Capabilities in design of Interpreter Enclaves. Ideally, the Interpreter Enclave will provide rich functionalities and will have a small TCB size.

5.5.3 Developing Enclaves with Intel SDK

Intel's SGX SDK requires developers to split an application into two parts: trusted and untrusted. Developers compile both parts into executable binaries. The untrusted application calls the trusted binary and maps it into a memory area allocated for the enclave. Now, the untrusted application can interact with the enclave via the trusted interface. The trusted interface then passes the data or requests to the Internal Enclave Functions via **E**nclave **CALL**s (ECALLs). If any internal enclave function needs a system call, this request goes through the **O**utside **CALL**s (OCALLs) to the untrusted application in the outside world. If the internal enclave function does not require the use of system calls, it may perform all of its computations without OCALLs, further reducing the risk of a system call-based attack [116].

5.5.4 Developing Enclaves on Graphene SDK

The Graphene SDK places the Graphene LibOS [134] (similar to the LKL[9]), into an enclave. In this setting, the enclave does not need to make any OCALLs for system dependencies to the untrusted operating system, as the Graphene LibOS

[9]Linux Kernel Library. `https://github.com/lsds/sgx-lkl` visited on 04/Jun/2019.

can handle all necessary system calls. Graphene-supported enclaves can contain unmodified binaries. The amount of code inside an enclave, however, can extend to tens of thousands of line of code, making the process of formal verification very difficult. Both the larger enclave code and the larger TCB size can increase the risk of security vulnerabilities.

5.5.5 Private Algorithms on SGX Enclaves

Once an interpreter is either ported or embedded into an enclave, a dynamic loader is required to fetch, decrypt and load the code for execution. Fig. 5.4 shows the feasibility of three approaches to deploy interpreters in enclaves based on their TCB components. The interpreter enclave would ideally have a small TCB size and support rich functionalities.

Loader + MuJS[10] JavaScript Interpreter + Intel SDK: Developers can use a dynamic code loader at runtime to fetch Javascript code and decrypt the blob inside the enclave. MuJS interpreter ported for Intel SDK can interpret the loaded secret-code in the execution phase. This method provides a small TCB size compared to that which is used with LibOSes. *Enclave 1* within Fig. 5.4 shows the TCB components of this approach. It provides comparably minimal TCB size to that shown in Fig. 5.4 for an interpreter enclave. The *Enclave 1* is used [141, 142] by Goltzsche *et al.* and Fernandez *et al.*, as analysed and evaluated in Section 5.6.

Loader + MuJS JavaScript Interpreter + Graphene SDK: The second method (*Enclave 2* in Fig 5.4) to create an interpreter enclave is to deploy MuJS on Graphene SDK. This method reduces the development effort because Graphene Library OS can support an unmodified MuJS interpreter, though it increases the TCB size. MuJS interpreter is smaller in comparison to CPython interpreter (while there are advanced JS engines, such as V8, MuJS is a lightweight option). As such, deploying a lightweight interpreter on a LibOS would not be an ideal model, because the majority of the TCB would not be in use.

[10] `http://git.ghostscript.com/?p=mujs.git;a=summary` Online Repository. Visited on 04/Jun/2019.

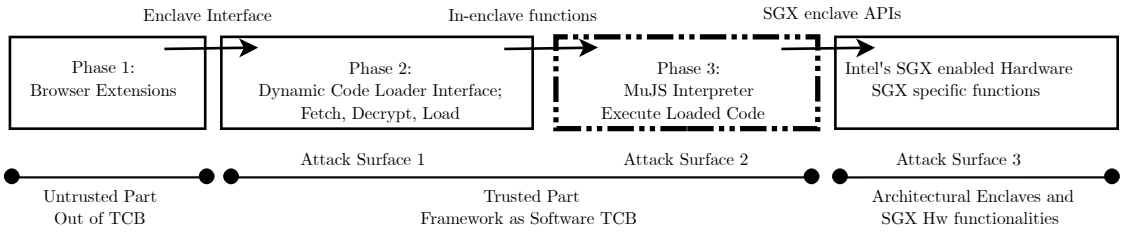

Figure 5.5: Attack surfaces on TrustJS and SecureJS in two phases (2 and 3). **Phase 1:** Browser extension receives the code. **Phase 2:** Dynamic code loaders prepare the code blob. **Phase 3:** Interpreter executes the code. The confidential code may leak due to attacks placed on surface 1 and 2, targeting the weakly developed enclave.

Loader + CPython Python Interpreter + Graphene SDK: The third method to deploy an interpreter enclave is to use Python, running over the Graphene LibOS, and to combine a dynamic code loader to fetch encrypted Python code at runtime and load it into the interpreter. This method can provide richer functionality for data analytics applications. The practical example of CPython Interpreter and Graphene SDK, without a dynamic secret-code loader, is available in the GrapheneSGX SDK repository[11].

5.6 Case Study: Leaks on Frameworks enabling Confidential Code Execution

Both TrustJS [141] and SecureJS [142] frameworks enable a dynamic load of JS code at runtime. These frameworks use the first method described in Section 5.5.5. This method involves porting the MuJS for the Intel SDK, and creating a JavaScript *Interpreter Enclave*. Their performance results and a detailed comparison is available in this [142] study. However, direct porting of a third-party interpreter may not help to hide a private algorithm. According to the threat model of Intel SGX described in Section 5.7, the *ED* is responsible for the security of interpreter enclaves. We evaluate the attack surfaces and the software attack vectors in the following sections.

[11] `https://github.com/oscarlab/graphene/tree/master/LibOS/shim/test/apps/python` visited on 04/Jun/2019.

5.6.1 Attack Surface on Interpreter Enclaves

The load and execution flow is the same in both frameworks of TrustJS and SecureJS. First, the Interpreter Enclave fetches the secret Javascript code in an encrypted blob. Inside the enclave memory area, MuJS needs to read the Javascript code as plaintext. The enclave dynamic loading mechanism prepares the received blob for the interpreter. Secondly, the interpreter parses each Javascript operation and calls the corresponding C functions. Frameworks use a custom application-specific code for the first phase of the preparation of the encrypted blob. Then, the MuJS interpreter (open-source) performs the second phase of execution.

TrustJS is not[12] an open-sourced enclave, however, SecureJS was open-sourced[13] in June 2018. To analyse the code execution in SecureJS's TCB, we focus on debugging MuJS on Intel SDK displayed in Fig. 5.5, phase 3.

Attack Vectors: A straight-forward composition of ported third-party software packages on secure hardware enclaves introduces new attack vectors. Previously performed attacks [115, 149–152] have shown how commodity software leaks secrets, while defence mechanisms explain how to mitigate these attacks on SGX enclaves. For this chapter, we chose a non-trivial attack [115], performed with `data-dependent data access` and `data-dependent control flow` weaknesses in interpreter enclaves. If the enclave code has input-dependent control-transfer; *e.g.*, calling different methods based on an input, the adversary can observe the called page externally and learn the private input. Similarly, if the data access pattern is based on an input parameter, the adversary can learn the private input.

Evaluation of TrustJS and SecureJS: MuJS was not designed to be an oblivious interpreter, and its use in an enclave for secret-execution requires secure composition and further isolation techniques. The direct use of the unmodified binary of MuJS in an enclave in a security domain brings potential security threats. We have identified two examples of weaknesses in MuJS used in TrustJS and SecureJS frameworks. These frameworks rely on the implementation of the MuJS interpreter.

[12]TrustJS published on 23rd April 2017, the source-code was not published as of writing this chapter, on 28th of May 2018.

[13]SecureJS. `https://github.com/AsierRF/SecureJS` visited on 04/Jun/2019.

TCB Component	TCB Size (LoC)
Main ECALL	110
Internal Enclave Functions	387
Crypto Functions	254
Attestation Mechanism	386
MuJS Interpreter	13.022
Trusted Intel Libraries	Not Included

Table 5.3: TCB Components and their size in line of C code. Based on Software TCB Size of SecureJS

Along with the interpreter, the dynamic code loader can be another target for secret code leaks. By design, the dynamic code loader must be public, and therefore open for inspection. This is because, as explained in Section 5.4, the *HO* (the platform owner, the cloud owner, the infrastructure provider) has to convince the public that her enclave does not perform any dishonest operations. Open-sourcing the interpreter enclave is in the interest of the *HO*. The *HO* can increase her profit in this type of cloud scenario, if she can convince more users to upload their secrets.

In this case study, the *AO* needs to attest the public code loaders. The attestation guarantee, however, would only prove that a known component is in execution. In building-block enclave design, developers must carry out security analyses on binaries and utilised third-party packages.

Analysing the TCB Size: The Interpreter Enclaves of this analysis have used the trust relationship shown in Fig. 5.3 (Approach 2). The Interpreter Enclave must, therefore, be public to gain the trust of the *AO*. Unfortunately, TrustJS framework is closed-sourced; *i.e.*, the interpreter enclave is not open for inspection. Thanks to similar open-source framework SecureJS, we were able to analyse its TCB. Table 5.3 shows the size[14] of the Software TCB components in the SecureJS framework. The MuJS Interpreter ca 13 KLoC takes %92 of the full TCB (ca 14 KLoC). The rest of the interpreter enclave (1137 LoC) includes a small loader, a decrypter code base (excluding the crypto implementations) and one *ECALL* that handles all enclave operations in a single flow. For the components other than the interpreter,

[14]Using SLOCCount. `https://www.dwheeler.com/sloccount/` visited on 04/Jun/2019.

a formal verification might be feasible. The MuJS executes the secret Javascript code, and processes the secret parameters of given secret Javascript functions. By porting the MuJS interpreter to SGX, the compiled binary can provide the required functionality, however, the SGX hardware cannot turn an insecure design into a secure one. In fact, an incorrect composition weakens the security guarantees of the hardware. The Interpreter Enclave must satisfy the secrecy requirement for confidential execution. These frameworks rely on implementation of MuJS for confidential execution. As a consequence, using the commodity software package for secret processing becomes a weakness due to the confidentiality requirement.

A

```
//...
opcode =   *pc++;
switch  (opcode) {
case OP_POP:  js_pop(J, 1);  break;
case OP_DUP:  js_dup(J);  break;
case OP_DUP2:  js_dup2(J);  break;
case OP_ROT2:  js_rot2(J);  break;
case OP_ROT3:  js_rot3(J);  break;
case OP_ROT4:  js_rot4(J);  break;

case OP_NUMBER_0: js_pushnumber(J, 0);
break;
case OP_NUMBER_1: js_pushnumber(J, 1);
break;
case OP_NUMBER_POS: js_pushnumber(J,
*pc++); break;
case OP_NUMBER_NEG: js_pushnumber(J,
-(*pc++)); break;
case OP_NUMBER: js_pushnumber(J,
NT(*pc++)); break;
case OP_NUMBER: js_pushliteral(J,
ST(*pc++)); break;
//...
```

B

```
//…
static Rune*
bsearch(Rune c, Rune *t, int n, int ne)
{
  Rune *p;
  int m;

  while(n > 1) {
    m = n/2;
    p = t + m*ne;
    if(c >= p[0]) {
      t = p;
      n = n-m;
    } else
      n = m;
  }
  if(n && c >= t[0])
    return t;
  return 0;
}
//…
```

```
p = bsearch(c, __tolower2, nelem(__tolower2)/3, 3);
```

Figure 5.6: Disadvantage of Third-party Packages for Confidentiality Management in Enclaves. Direct port of commodity software may leave additional side-channel traces which ruins the confidentiality guarantees of the hardware. **A**) Excerpt showing the Input Dependent Control Flow in `js_run` method of `jsrun.c` in MuJS interpreter. **B**) Excerpt showing the Input Dependent Data Access in `bsearch` method of `utftype.c` in MuJS interpreter. (Both Accessed on May 2018 Revision.)

5.6.2 Weaknesses in MuJS Interpreter

MuJS is a lightweight Javascript interpreter, not designed in the security domain. The *ED* must, therefore, be careful about including third-party source code in the TCB. For this chapter, we debugged the MuJS interpreter for the input dependent methods including the data access and the control flow. These particular weaknesses may leak the confidential source code and the confidential parameters. Therefore,

our aim is to observe two parts of confidential information on source code; first leaking the Javascript functions, and secondly, leaking the secret string parameters.

5.6.2.1 Leaking the called Javascript functions via Data-Dependent Control Flow Weakness:

MuJS parses Javascript code to extract the necessary operation code (`opcode`). The `opcode` for the JS function is used to call the relevant C method that corresponds with the Javascript method. The corresponding C methods are placed in different memory pages. We report that the control flow in the `jsrun.c` file containing the `jsR_run` method shown in Fig. 5.6 is dependent on the input parameter of `opcode` containing the Javascript method. The `jsR_run` method is used to parse and call C methods. This may leak the function names of confidential Javascript methods. The control-flow is therefore, non-oblivious.

5.6.2.2 Leaking the String Parameters via Data-Dependent Data Access Weakness in MuJS:

MuJS contains `utf8` and `rune` string operations for performing string transformation operations. The string operations are performed over look-up tables. The number of accesses to a look-up table tells the position of the character searched in given table. An often called function for table look-up `bsearch` in `utftype.c` exposed in Fig. 5.6 may leak the position of the character in the table. This means the execution has non-oblivious data-access.

5.7 Managing the Software-TCB on Enclaves

Both of the fundamental security notions, namely integrity and confidentiality management, require hardware and software co-design and implementation. A secure hardware (a TEE or SGX v1.0 in practice) provides limited security guarantees if its software contains weaknesses. For example, a buffer overflow vulnerability allows an attacker to break the integrity of the target software. If an enclave contains a run-time vulnerability, attackers may take the control of the execution,

compromising its integrity. The *ED* should avoid any run-time vulnerability to ensure the integrity guarantees of underlying instructions.

For confidentiality management, SGX can provide page-level secrecy in its enclave memory. The content of a memory page is encrypted by the hardware. An enclave that contains commodity software (*i.e.*, unmodified binary programmed without security in mind) may allow attackers to learn the secrets processed by that software. The *ED* should follow secure programming practices.

SGX hardware threat model states the *ED*'s responsibilities in *SGX Blogs* and *SGX Documentations* since the time of launching SGX in 2015 (SGX Programming Reference, SGX Developers Guide, SGX Developer Reference, SGX Enclave Writers Guide) [144–146, 153, 154]. The *ED* must keep their development environment malware-free. During programming or compiling an enclave, a malware may infect the enclave and perform malicious operations. The *ED* must also keep their enclave code vulnerability-free. An important post by Intel [153] mentioned that[15] the type of side-channel attack identified on the RSA implementation was well-known. Developers must avoid any weak or vulnerable code in the enclave that plays a role in an attack.

Secure programming techniques can help to keep enclaves free against runtime bugs. The *ED* has the responsibility to program their enclaves with resistance to software-based side-channel attacks. SGX hardware cannot bring automatic security guarantees for an enclave that contains vulnerabilities.

SGX hardware can provide integrity control for memory accesses; so that an enclave can only access to its own address space. The untrusted operating system, nevertheless, controls the allocation of the enclave memory pages. By design, SGX cannot provide any protection against any denial of service attacks; if the operating system refuses to give the enclave resources, they cannot operate. If the enclave binary programmed by the *ED* is weak (*i.e.*,, performing any data access or control flow based on input data), the operating system can observe the pages requested

[15]A recent blog post by Intel [153] stated following: *In general, these research papers do not demonstrate anything new or unexpected about the Intel SGX architecture. Preventing side channel attacks is a matter for the enclave developer.*

Notion	Hardware Feature	May Break
Integrity	Memory Access Checks	Runtime Vulnerability in TCB
Confidentiality	Page-level Secrecy	Non-oblivious Software Stack
Fault Tolerance	Sealed Storage	Bad Software Implementation
Enclave Availability	Refuses to Operate Open	Vulnerabilities in Microcode

Table 5.4: Hardware-enhanced security guarantees in enclaves, and the cases when bad software stack may break these guarantees. Enclave Developers (*ED*) are responsible for the secure development.

or used during execution. It has been shown by Hahnel *et al.* [149] and Tsai *et al.* [115] how the *ED* can mitigate some potential the side-channel attacks.

If the *ED* places vulnerable code inside the SGX enclave, code flaws may cause integrity problems which can be no longer be controlled by SGX (or similar trusted hardware solution). The enclave implementation must contain no code that intentionally causes information leakage through side-channel attacks or covert-channel attacks. These kinds of attacks do not show weakness in SGX technology [153]. The *ED* must analyse its implementation to prevent the side-channel attacks [150–152] against the software running in SGX enclaves.

Previous studies [155–158] have shown solutions for page-level and cache-level side-channel attacks. Seo *et al.* [155] provided a solution for the controlled page-level attacks with a compiler-level scheme, and this [156] study solved the same attack with verifiable page faults. Another study [157], solved the issue of data leakage in side channels via randomisation. Additionally, recent work [158] has enabled Address Space Layout Randomisation (ASLR) for SGX enclaves. The *ED*s may benefit from these solutions. Utilising an oblivious RAM solution [159, 160] may also reduce the attack surface by hiding memory content, and the memory access patterns.

5.7.1 Bad Practices in Enclave Development

Based on the issues listed in Table 5.4, we briefly explain the bad practices in enclave development in this section. We see that in Section 5.6, while SGX instructions are not insecure and while the Javascript interpreter has no vulnerabilities by default, using this default and unmodified interpreter in an enclave (utilising

SGX instructions) causes important security issues due to non-compositionality of security [62, 63]. Even if SGX instructions are secure and implemented as they are specified, and even if the component to be integrated with an enclave is designed and implemented securely, the composition (resulting system) may not be able to offer the security guarantees out of the box, explained earlier in Section 2.2.

5.7.1.1 Integrity and Confidentiality:

SGX technology is a practical TEE solution for securing application secrets. The enclave TCB design is crucial for providing integrity and confidentiality. For confidentiality management, third-party software included in the TCB may fail to provide secrecy for private algorithms. As such, TCB components that perform sensitive operations require special attention from developers as it is possible for arbitrary software, running on trusted hardware, to leak application secrets. Due to their weak TCB components, TrustJS and SecureJS frameworks may fail to preserve the confidentiality of private algorithms. Secrets must not be processed by arbitrary software unless required security mechanisms are in place.

Impact of TCB Size on Integrity & Confidentiality Management: Increasing the TCB size reduces the chance of formal verification of the source code. SGX hardware provides strong integrity guarantees for execution and memory isolation. However, a weak software stack included in the TCB may leak secrets that are stored or processed in the enclave. Similarly, a run-time exploit may give control of the enclave to an attacker.

The New Attack Vectors against the Software running on Trusted Hardware: Commodity software products and unmodified binaries running on trusted hardware need additional memory protections for confidentiality management. Table 5.4 shows the *ED*'s responsibilities to preserve security guarantees provided by the trusted hardware. The non-oblivious software stack shown in Section 5.6 may cause confidentiality leaks, otherwise. Without deploying the security mechanisms that are necessary in a remote environment, the performance metrics becomes obsolete. We list the bad practices in SGX development as following:

- **Building a software stack with commodity software:** Developers lose security guarantees of the underlying SGX hardware due to weak TCB construction caused by commodity software. Porting the commodity software to SGX does not directly compose it with SGX.
- **Runtime vulnerabilities are still a threat:** Developers cause serious integrity flaws due to runtime vulnerabilities. Enabling the features of the SGX technology within applications does not mitigate the runtime vulnerabilities. Consequently, a developer who places a runtime vulnerability in the software TCB of a SGX enclave does not show security flaw in the SGX technology.

5.7.1.2 Two Aspects of the Availability:

Availability of an SGX enclave may refer to two sub-notions. First, the SGX enclave binary would refuse to operate if there is no legitimate Intel SGX hardware. This notion is independent of the *ED*. The security issues may arise if SGX instructions have a design flaw. Nonetheless, this would not threaten the validity of enclave research. Because the microcode can receive patches, and the secure hardware solutions in future can avoid the known security issues.

The second point of availability is related to fault tolerance. An enclave-based cloud service must be available. As the operating system controls the resources, it can force an enclave to die. In case of a failure, the enclave can recover from a sealed state. The recovery process would, however, be dependent on how this fault tolerance mechanism is implemented. A bad software implementation may cause enclave to fail on execution, or at recovery. The *ED*, therefore, are responsible for the secure development of an enclave that is resilient to the failures.

5.8 Secret-Code Execution in Reduced TCB

We extended the enclave development model with a new mode called **Early Private Mode (EPM)**, shown as **State 1** (S1) in Fig. 5.8. The EPM helps to SCE in enclaves preserving strong security guarantees, and avoids the issues explained in Section 5.7.1. The overall architecture in Fig. 5.13 in the Appendix, ① provides

confidential execution of private algorithms, ② keeping a minimal TCB size, ③ without the inclusion of any commodity software. In short, the EPM runs when an *AO* executes the enclave in their own execution environment by setting the EPM flag as ′true′. The enclave then outputs the given function reference by reading from runtime memory.

The conventional enclave lifecycle contains three states. The developer firstly compiles the enclave binary, which is then delivered and executed, before attesting the identity of the loaded enclave binary. In our design, we added an **Early Private Mode** (EPM; or State 1; S1) to serialize the secret code before compiling the standard enclave binary. The S1 serialises the given secret internal enclave function and outputs it to an encrypted blob. After verifying the enclave identity via remote attestation, we load the serialized secret code for execution (Fig. 5.9, **State 5**; S5).

5.8.1 Further Enclave Partitioning: Public and Private Internal Enclave Functions

By default, in enclave development [161], binaries are split into trusted and untrusted parts. Trusted enclave part includes a Trusted Interface (TI), ECALLs and OCALLs, along with **Internal Enclave Functions**. We extend this paradigm with **Public IEF** (PIEF), and **Secret IEF** (SIEF) as shown in Fig. 5.7. In S1 (Fig. 5.8), the *AO* produces **Serialised Secret Internal Enclave Functions (SSIEF)** called Asset 1 (A1) within EPM. At this early stage, a copy of the SIEF is created, and sealed to enclave state, to be loaded later in S5 (After Remote Attestation).

The second state (S2) is not different from the ordinary release mode[16], but includes one additional configuration parameter. By setting the EPM flag as ′false′, the build process excludes the Private Enclave Part, and outputs the standard enclave binary. The compiled binary includes the TI, ECALL and OCALL functions, the PIEF and the Private Code (PC) Loader.

[16]Release Mode: `https://software.intel.com/en-us/documentation/intel-sgx-web-based-training/debugging-enclaves` visited on 04/Jun/2019.

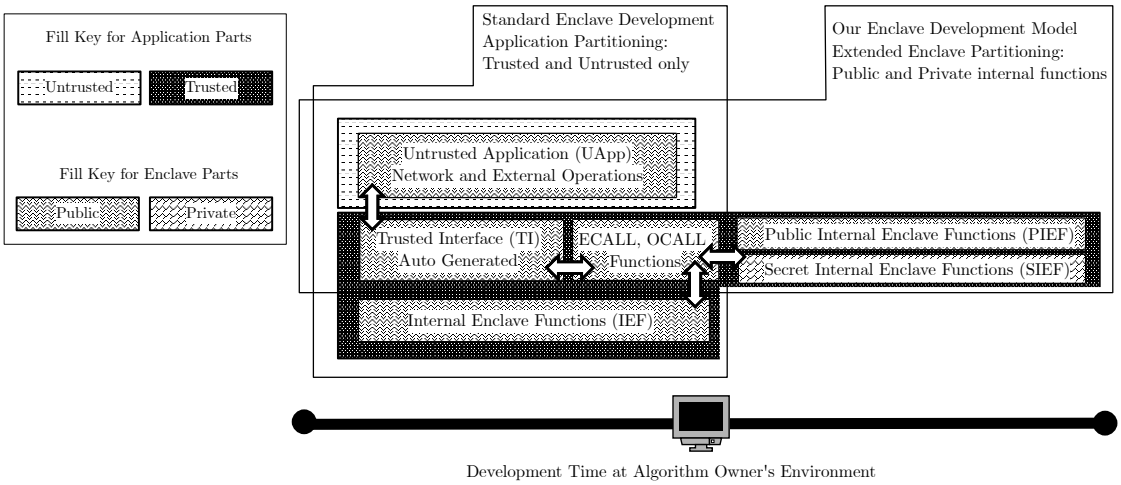

Figure 5.7: Extending the Enclave development model with Private and Public parts. In addition to the application partitioning into trusted and untrusted, we introduce the *enclave partitioning* for internal functions. These functions never communicate directly with the outside world.

5.8.2 Late-Load of Secret Code at the Fifth State

Before execution of the enclave at **State 3** (S3), the *HO* has a chance to investigate the enclave code known as Asset 2 (A2). A2 will contain no secrets that have been embedded in advance. First, the A2 is loaded into the main memory at **State 4** (S4) via the SGX APIs triggered by the **Untrusted Application** (UApp) binary. Afterwards, the *HO* cannot see the content of the A2 in the enclave memory (other than the side-channel footprints explained in Section 5.7). To reduce information leaks via side-channels, we keep the TCB clean from third-party commodity software, not processing any secrets in the PIEF. After Remote Attestation (either TLS-based [162, 163] or Diffie-Hellman & SIGMA based [154]) between the AO and A2 (*i.e.*, the AO attests the A2 enclave binary), the PC Loader extracts A1, including the SSIEF, to the pre-allocated executable memory. To execute the secret code, A2 calls the address of A1, passing the execution flow. This operation provides runtime recovery of the secret code invisible to the *HO*.

The late-load method for a dynamic portion of enclave enables the TaCB introduced in Section 2.1.2.1.

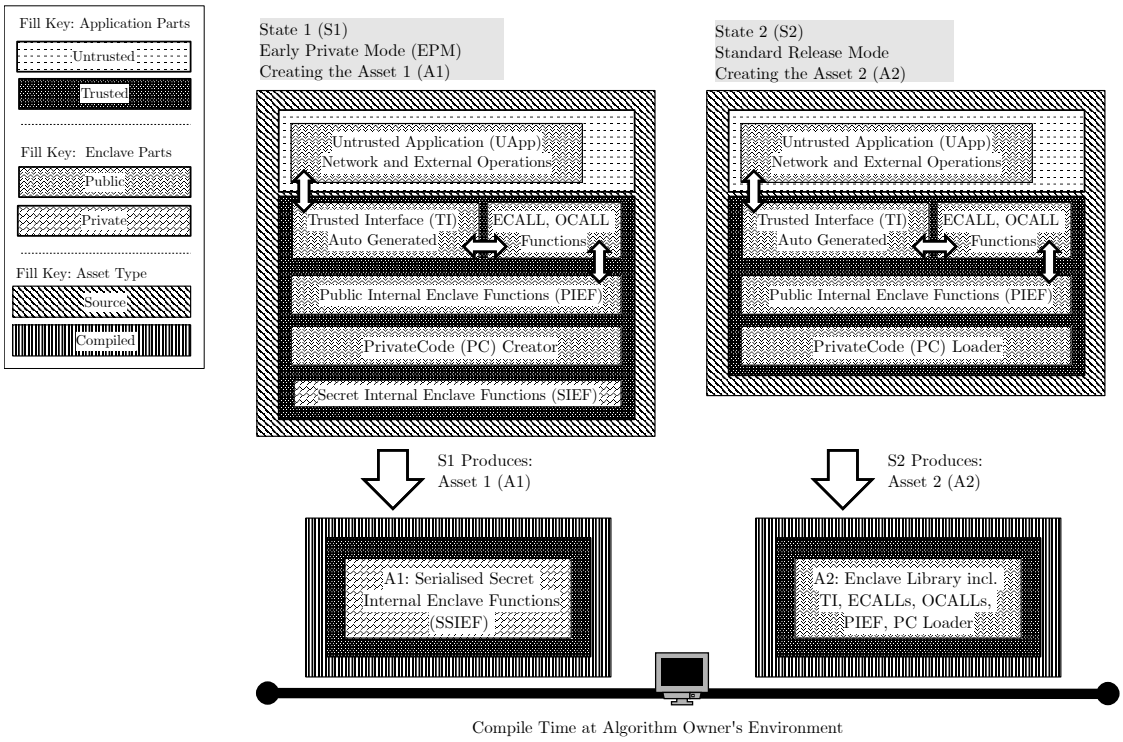

Figure 5.8: The Early Private Mode (EPM) runs before releasing the enclave binary. It creates the Asset 1 (A1) that includes the Serialised Secret Internal Enclave Functions (SSIEF). The standard release mode in State 2 (S2) creates the enclave binary including the Private Code Loader.

5.8.3 Managing Security: Adversarial AO vs Adversarial HO

In addition to the classic threat model of SGX applications presented in Section 4.2.2, where malicious OS and software stack threaten the enclave, we now consider a two-way adversarial case. The *HO* aims to learn the secret code sent by the *AO*. To do so, the *HO* can query the SIEF offline with all possible input set, depending on the application. In order to prevent this, the *AO* has to bind the SIEF to the *HO*'s input parameter, while attesting A2. The parameterisation of the SIEF is kept out of the scope in this chapter. Nevertheless, *AO* can mitigate the offline-querying attacks by locking the SIEF into a specific input. In contrast, the *AO* may take control of the enclave via a malicious SSIEF in order to signal back any *HO*-specific private data used in the PIEF or SIEF. This possibility threatens the privacy of the *HO* if it represents multiple end-users (Data Owners, *DO*). As a countermeasure,

the *HO* can physically limit any information leakage by isolating their environment after loading the SIEF. We do not, however, consider PIEF to be malicious against the *HO*, as it must be open for inspection, and it must be trusted by both parties. This requirement of PIEF being trusted, comes from standard enclave development where the all enclave code is being public. In our model, we keep a part of the enclave private, while rest of it is still known and being open for inspection. In short, the *AO* can lock the SIEF against offline attacks targeting code secrecy, and the *HO* can control the physical execution environment including the network infrastructure to mitigate the covert-channels.

5.8.4 Comparing the Approach 1 and Approach 2

The secret code execution in reduced TCB (Approach 1) uses direct export and import of the C code. The interpreter enclave method (Approach 2) enables the use of high-level languages.

Usability: Approach 1 requires the *AO* to be aware of enclave development, and it requires them to take an additional step of EPM before releasing the code. In Approach 2, the *AO* does not need to have information about the enclave development, as the interpreter provides a layer of abstraction.

Security: Using an interpreter leaves more memory traces, potentially leaking information via side-channels. It requires oblivious interpreters and oblivious memory layout in order to hide the control flow. Loading native C functions with the size of memory pages do not leave further traces (at page-level granularity), other than the number of calls.

TCB Size: Even though using a very small interpreter, Approach 2 dramatically increases the TCB while weakening the threat model. The native C execution method in Approach 1, the TCB contains approx. 1500 LoC for PC Loader, Attestation and other Enclave functions (excluding any application-specific functionalities of the enclave). In comparison to interpreter enclaves of 14 KloC, our method provides a TCB that is ten times smaller. Also, Approach 1 addresses a stronger threat model towards side-channel attacks by leaving less footprints.

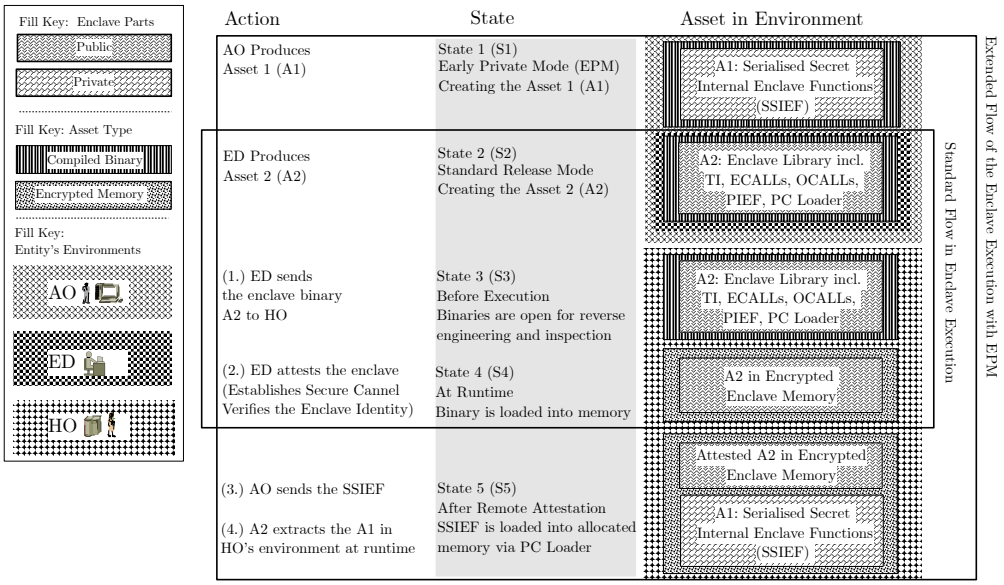

Figure 5.9: The *Standard Execution* of an enclave, and the *Extended Execution* flow with EPM for private algorithms. The difference is that the *AO* takes also a role in development of the enclave for private parts. The Asset 1 containing the private algorithms is loaded late in the *HO*'s environment for secret-code execution after verifying the attestation report.

#	Use Case For	Alg. Status	Data Status	Result	Execution
1	**Computing Power (CP)**	Private Alg. by AO	Public/No Data by DO	A.O. gets	only at H.O.'s environment
2	**Algorithm Querying (AQ)**	Private Alg. by AO	Private Data by DO	D.O. gets	D.O. controls H.O.'s env.
3	**Data Querying (DQ)**	Private Alg. by AO	Private Data by DO	A.O. gets	D.O. controls H.O.'s env.

Table 5.5: Three Use Cases on *Secret-Code Execution* through SGX enclaves.

5.9 Industrial and Practical Use Cases

The *AO* needs to run private algorithms on a remote, untrusted computer. We show (Table 5.5) three examples where our execution model can achieve this goal. These scenarios differ from each other by the execution environments and the participant who receives the result.

5.9.1 SCE-CP: Secret-Code Execution on Computational Power

In the first scheme, the *AO* has a set of private algorithms, such as ① a private compression algorithm, ② a private sorting algorithm. These algorithms need high computational power. The *AO*, therefore, rents a remote server from the *HO*. This computation includes embedded constant values and the algorithm provided by the *AO* only. The *DO* does not involve in this scenario. The *AO* wants to get the secret output of the computation.

5.9.1.1 SCE-CP in Practice:

The *AO* creates the SSIEF containing the algorithm and the enclave binary. ① The *AO* sends the enclave binary to the *HO*. The *HO* executes the enclave on a SGX-enabled machine. ② The *AO* attests the enclave, and establishes a secure channel. ③ After verifying the enclave identity, the *AO* sends the encrypted SSIEF. The *HO* loads the encrypted content and execution begins. The result of the computation is encrypted with the *AO*'s key. ④ The *AO* receives the result and decrypts it.

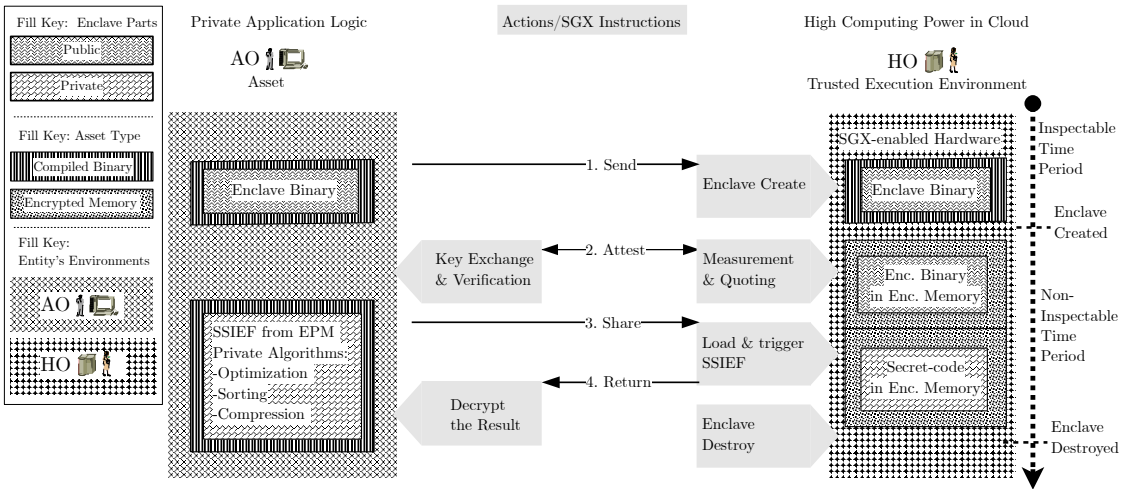

Figure 5.10: Secret-Code Execution (SCE) Computational Power. The *AO* creates a session with the enclave binary, and shares the SSIEF. The SSIEF is extracted into the enclave memory at runtime.

5.9.1.2 SCE-CP Establishing the Mutual Trust:

The *AO* designs and implements the enclave binary. Sharing the SSIEF after the attestation process gives the *AO* an inherited trust. The SSIEF is decrypted and executed only inside the *Non-Inspectable Time Period* (in Fig. 5.10). This scheme removes the chance of the *HO* to disassemble any content of the enclave code.

5.9.1.3 SCE-CP Malware in SGX argument:

There is a long-lasting argument [164] that an SGX enclave may include malicious code. We evaluate this argument for the SCE-CP use case. Our execution model allows the *AO* to include arbitrary software in the enclave. Without the use of EPM and SSIEF, the *AO* may send the enclave code in plain text, or the *AO* may entirely avoid utilising enclaves. This, however, does not stop the *AO* from sending malware to the cloud environment. In fact, the *AO* is free to run an experiment with malicious or benign code in the cloud. At the bottom, the *HO* controls the hardware resources, and observes the usages. The *HO* charges more to the *AO*, if the resources are used more. At all times, the *HO* can observe all I/O traffic of the enclave. The *HO* can refuse to give resource at any time. The enclave, resource-wise, is one of the most visible parts in the system. Through the enclave or not, the malicious *AO* has full access to the cloud machine. We conclude that use of an enclave does not increase the existing attack surface in the SCE-CP scenario. We will further provide an in-depth systematisation of malware arguments in enclaves in Chapter 7.

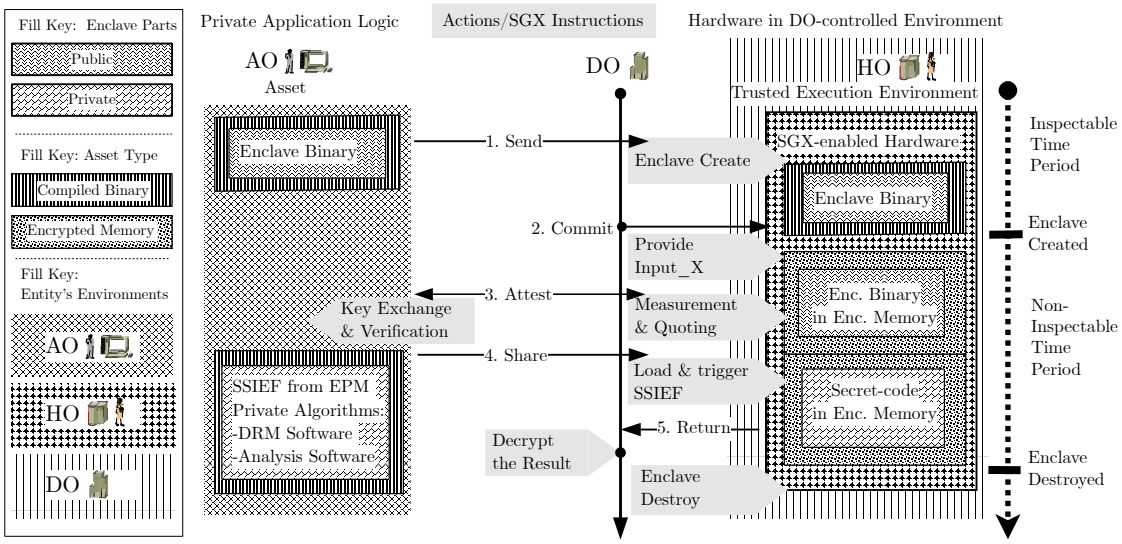

Figure 5.11: Secret-Code Execution (SCE) Querying the Algorithm. The *DO* provides an input to the secret algorithm and receives the output. Input commitment prevents querying the secret-code with a different parameter in an offline repeated execution.

5.9.2 SCE-AQ: Secret-Code Execution on Algorithm Querying

In the second scheme, the *DO* has private input data, such as ① an image containing health data, ② a data-set collected from sensors. As an untrusted entity, the *AO* has a private algorithm that can process this private input. The SCE-AQ scheme (Fig. 5.11) outputs a private value that the *DO* must receive only. We assume that both entities are mutually distrustful. They do not want to share their private assets with each other. A collusion between the *AO* and the *HO* against the *DO* can leak the secret data. We, therefore, consider the *HO* to be controlled by the *DO*.

5.9.2.1 SCE-AQ in Practice:

The *AO* uses the EPM and creates the SSIEF containing the algorithm, and the enclave binary. ① The *AO* sends the enclave binary to the *HO* that is controlled by the *DO*. ② Different from the SCE-CP, the *DO* interacts first with the enclave and commits the input. ③ The *AO* attests the enclave and verifies the identity. ④ If convinced, the *AO* shares the SSIEF with the enclave. The key point is that

#	**Example Input_x**	Example Method	**Example Result_y**
001/256	**00-1111-00**	SSIEF(function_f())	11
002/256	**11-0101-11**	SSIEF(function_f())	10
..	**.. ....**	SSIEF(function_f())	..
256/256	**00-0110-01**	SSIEF(function_f())	00

Table 5.6: Brute-Force Querying the Algorithm by Re-using the Enclave

neither the *DO* nor the enclave can change the execution after this stage. ⑤ The result of the computation is returned to the *DO*.

5.9.2.2 SCE-AQ Brute-Forcing the Algorithm Secrecy:

This scheme returns the computation result to the *DO*. This allows the *DO* to gradually learn about the algorithm. A malicious the *DO* may want to find the all possible input and output streams of the algorithm. The Table 5.6 shows an example to learn the function behaviour by brute forcing the input streams. The *DO* can freeze and clone the memory state of the machine to perform an attack against the algorithm. In our mechanism, however, the independent input from the *DO* comes before the SSIEF. Once the *DO* commits to an input parameter (② step, Fig. 5.11), the enclave does not accept any other interaction other than accepting the SSIEF (④ step, Fig. 5.11). If the *DO* clones the enclave at any stage, the result does not change. The *AO* can thus measure how much secrecy of the algorithm has been already leaked.

5.9.2.3 SCE-AQ Preventing the Data Leaks:

In contrast, the *AO* may want to leak the secret inputs. Even though the *AO* can execute any code, the SSIEF cannot communicate with any other entity. The *DO* physically controls the environment, and does not allow any I/O operation.

5.9.3 SCE-DQ: Secret-Code Execution on Data Querying

In the third scenario, the *AO* and the *DO* goes into another joint computation. The difference is that the result must be returned to the *AO* only. The *DO* has a special private data-set. The *AO* wants to run a secret query on this data-set.

Suppose that, the *DO* is an authority who collects road-data and driving experiences of citizens through sensors. In another example, the *DO* can be a car manufacturer company who collects data from their cars. The *AO* is an insurance company, or a government authority, who wants to run a private algorithm on that data-set.

5.9.3.1 SCE-DQ in Practice:

Similar to SCE-AQ, ① the *AO* sends the binary, ② the *DO* commits the input, ③ the *AO* attests the enclave, ④ the *AO* shares the SSIEF. Differently, at the step ⑤, the *DO* reduces the bandwidth of the result. ⑥, the *AO* receives the result and decrypts it.

5.9.3.2 SCE-DQ Limiting the Data Leakage:

The difference in the SCE-DQ from the SCE-AQ is that the *AO* receives the secret result. At the end of the computation, the *DO* does not let SSIEF to return an arbitrary value directly. In this example, we allow returning only 1-bit of data (boolean) as a result. This operation means, the *DO* does not know what software has processed her input, and does not know the result. The *DO*, however, knows

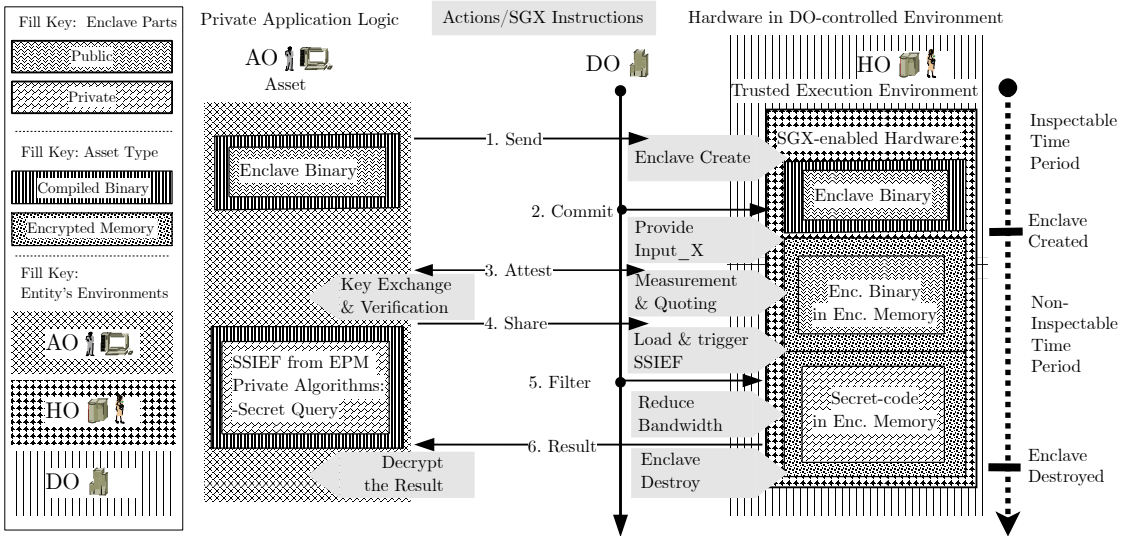

Figure 5.12: Secret-Code Execution (SCE) Querying the Data. The *AO* runs a secret query on a secret data set. After the secret-code execution, the *DO* has control of the bandwidth for filtering the return value.

Description	**SCE CP**	**SCE AQ**	**SCE DQ**
Development Time Public/Private Partitioning: Asynchronous			
Compile Time EPM and Enclave: Asynchronous			
***t* Time Cost, *m* Memory Size**	1. Step: Execute the Enclave		
Worst Case 2*t* total time cost	3. Step: Exec SSIEF	4. Step: Exec SSIEF	
Other Operations: Existing costs			

Table 5.7: Overhead of Three Use Cases on *Secret-Code Execution*.

the type of the value, and permits SSIEF to return this secret value to the *AO*. By doing so, the *DO* can control how much secrecy of data is disclosed.

5.9.4 The Overhead in SCE Components

We observed two types of computational costs in our use cases: ① The asynchronous operations that are independent of the main computation; ② The synchronous time cost that our model adds on top of the existing cost. Table 5.7 summarises the memory space (m) and computing time (t) overheads of the components. In the worst case, the SSIEF recovery costs an equal amount of time to the cost of the enclave creation (t). This overhead sums up to Worst($2t$) of total time for the SSIEF execution. The computation itself, however, can take time and memory as required by the *AO*. The SSIEF is executed in the preallocated memory area during enclave creation. This operation, therefore, does not require additional memory. We keep the remote attestation costs and operations out of the scope in this study.

5.10 Closing Remarks

In conclusion, our late-load method for secret-code execution provides stronger security and native execution performance, requiring only a small additional development effort from the Algorithm Owner. Future work may explore automated ways of deploying our design, however, Approach 2 in private algorithms will continue to have better usability.

* * *

Contributions

The novelty and contributions of our work are as follows:

① we consider mutually distrustful entities *(HO, DO, AO)* with conflicting interests in the cloud,

② we differentiate the private algorithms and the private data,

③ we show the bad practices on use of TEE in the cloud,

④ we create a taxonomy for secure execution of private algorithms in untrusted remote environments,

⑤ we provide practical insights to enclave development,

⑥ we perform a security analysis on existing dynamic code loaders with interpreter enclaves,

⑦ we evaluate our execution model in three adversarial settings in the cloud.

Chapter Recap

Section 5.3 provides a background on current binary execution mechanisms, as well as responsibilities of application developers for the chosen development model. Section 5.5, explains the trade-offs between design choices in TCB for private algorithms. We analyse the TCB components of existing frameworks with practical attacks against code secrecy in Section 5.6, and Section 5.7 explains the bad practices in enclave design and development. Section 5.8 shows our design for secret code execution with reduced TCB that is providing better security. Finally, Section 5.9 demonstrates our method in practice with three use cases of private algorithms for industrial use.

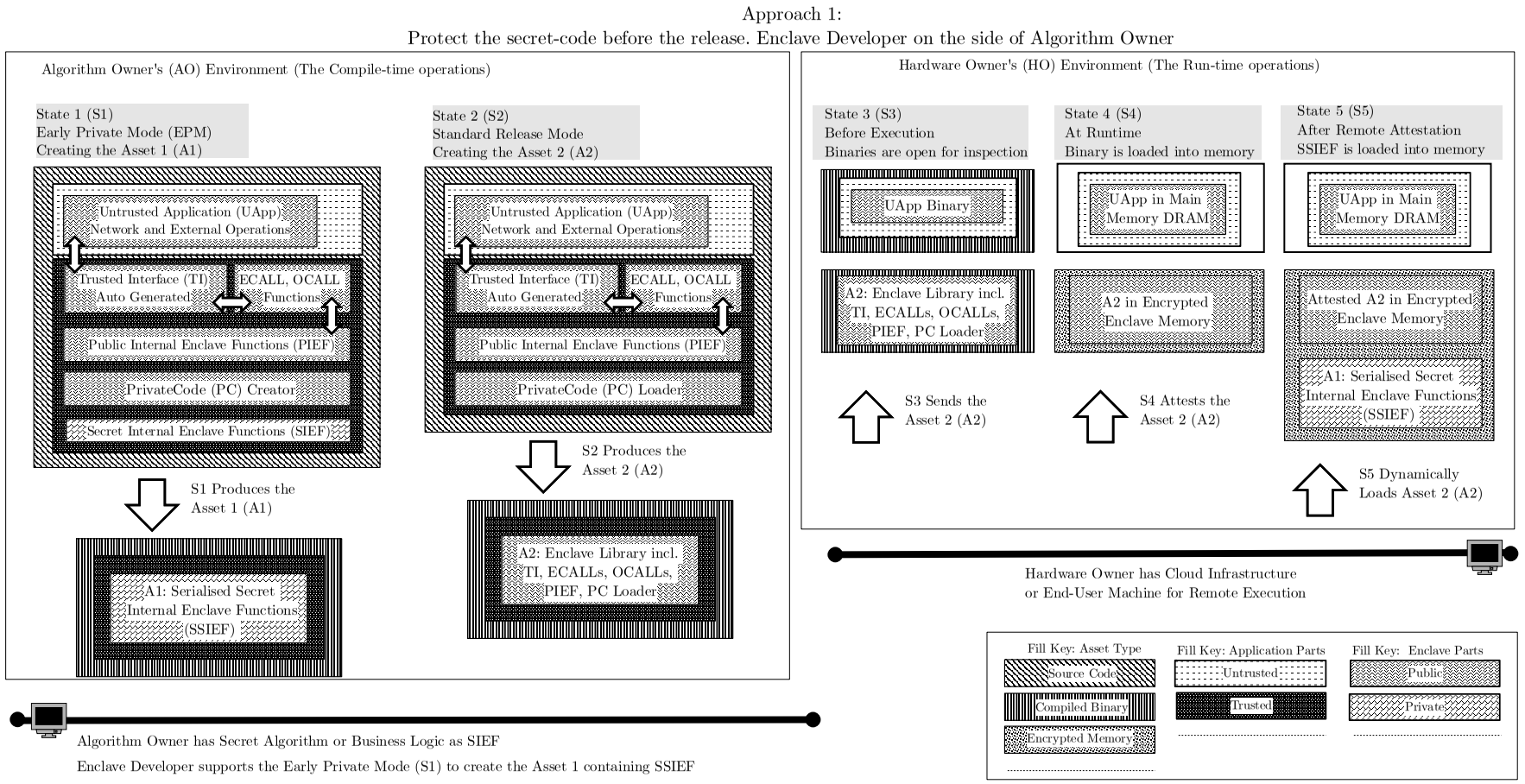

Figure 5.13: Secret-Code Execution (SCE) in Reduced TCB. Enabling the use of private algorithms inside SGX enclaves. Extend the enclave partitioning with private and public parts. The private part is processed in an early stage before the release of the enclave.

Part IV

Systematisation

Chapter 6

Fully General Domain Model of Confidential Remote Computing

Contents

Digital services have been offered through remote systems for decades. The questions of how these systems can be built in a trustworthy manner and how their security properties can be understood are given fresh impetus by recent hardware developments, allowing a fuller, more general, exploration of the possibilities than has previously been seen in the literature. Drawing on and consolidating the disparate strains of research, technologies and methods employed throughout the adaptation of confidential computing, we present a novel, dedicated Confidential Remote Computing model. CRC proposes a compact solution for next-generation applications to be built on strong hardware-based security primitives, control of secure software products' trusted computing base, and a way to make correct use of proofs and evidence reports generated by the attestation mechanisms. The CRC model illustrates the trade-offs between decentralisation, task size and transparency overhead. We conclude the chapter with six lessons learned from our approach, and suggest two future research directions explained in Part V.

This chapter is based on publication [10].

This thesis is on CRC modelling and implementation details. The novelty of this chapter is that we draw on existing approaches to confidential computing, and its development, and that here we extend those approaches and add extra concept to generalise the literature into our concept of *Confidential Remote Computing*. CRC has two main practical implications. Part II presents the support for many party applications, it contains a communication model, system model, threat model, and attestation model. Part III presents the model on private algorithms, computation models for multiple stakeholders, extended application partitioning

model and an extended threat model. Here in this chapter, we present novel concepts such as *Transparency Overhead* in connection with the *Task Size* and the *Decentralisation*, and a novel systematisation in three domains (hardware, attestation, development) for the CRC.

In the previous two chapters, we provided technical insights, development models and practices. Now in this chapter, we systematise the idea of CRC by defining its sub-domains and discussing the related concepts in more depth.

6.1 Confidential Remote Computing Model and the Index List:

The meaning of the word *model* can be ambigious for some readers, in the context of CRC, we use the model with a similar meaning to framework. The reason why we do not switch to word *framework* is because throughout the thesis we present multiple models of the theme. The aims of our model/framework are to evaluate solutions, to compare one solution/technology with another, to identify where more research is needed. For interested readers, the closest model we show is at Section 6.5 and the CRC model is drawn at the largest circle of the ring at Figure 6.4 with secure loaders (in development domain), secrecy and frequent updates (in attestation domain), TCB updates and private RAM (in hardware domain) and the fifth entity with the algorithm owners. We provide a brief list of the sections containing the pieces of the CRC model throughout the thesis.

Part II, Public Code and Private Data:

- System and Adversary Model: Section 4.2.
- Architecture Model: Section 4.5.5.
- Communication Model: Section 4.5.6.

Part III, Private Code and Private Data:

- Multiple Stake Holder Ownership Model: Section 5.1.
- Extended Adversary Model: Section 5.8.3.
- Extended Enclave Partitioning Model: Section 5.8.1.

- Late-Load Model for Private Algorithms: Section 5.8.2.
- Execution Model between a Hardware Owner and an Algorithm Owner, Computational Power as a Service Model: Section 5.9.1.
- Execution Model between a Hardware Owner, a Data Owner and an Algorithm Owner, Algorithm Querying as a Service Model (limiting Algorithm Exposure): Section 5.9.2.
- Execution Model between a Hardware Owner, a Data Owner and an Algorithm Owner, Data Querying as a Service Model (with Reduced Bandwith to Limit Data Leakages), Section 5.9.3

Part IV, Systematisation of Knowledge on CRC:

- Domain Model: Section 6.5.
- Trade-offs Model: Section 6.6.
- Unified Computing Model 6.6.5.
- Malware Infeasibility Model 7.5.

6.2 What is Confidential Remote Computing?

Computer programs contain computations. Simply put, the function $f(x) = xa + b$, a computation, or a NAND gate, consists of two elements, the data (input) and the algorithm (operation). These computations can require confidentiality for their assets. The basis of any confidentiality is one party's desire for protection from another's infringement on their assets. Although a computation can run locally, it turns into a remote computation between these two parties involved. This is why *Confidential Computing*[1] cannot be conceived without the key aspect of remoteness.

The problem we identify and response in *Confidential Remote Computing* has been around for decades. In fact, we shall discuss in Section 6.2.3 that it has been studied throughout the history of inter-connected computations. We shall highlight the development of old and new aspects of *Confidential Remote Computing*. Over time, new actors introduced new approaches, evolving requirements and

[1] `https://confidentialcomputing.io/`

corresponding security solutions. Thus, the multiplicity of actors in a computation poses a challenge for any emerging control decentralisation. The main issue is decentralisation of the computation, our solution approaches to this problem through its participants.

6.2.1 The Problem Statement

The problem of CRC can be expressed in various means. For simplification and inter-disciplinary understanding, we explain the problem by using a classic example with digitalised companies, *e.g.*, take two insurance companies. Both of these companies promise to perform advertised computational tasks remotely with your private information and promise you to delete all the information stored. One of them has certified infrastructure, isolated and confidential execution, and minimised software stack. The other one is infected by advanced malware, it may perform some hidden operations (*e.g.*, selling your data off in the background). Can you tell the difference between the two from your home computer? What proof could there be that you are communicating with the right insurance company? Similarly, consider a bank, a block-chain infrastructure, your government, and many other mission-critical online services you are using. How do you know that the behaviour of the remote system is, as promised, trustworthy, to whatever end you wish to employ it?

The service quality of a business depends on the transparency it offers. Being able to offer true trustworthy service is not only an added value to products, it is seen as a basic requirement by users. Gaining trust may come through formal validation and verification methods (*i.e.*, checking if a piece of code matches its specifications) in relatively small systems. In contrast, our focus in this chapter shall be placed on the *Confidential Computing* paradigm, in order to help us understand how the root of trust in hardware can be translated to humans through software components. For example, users can make sure that an election was counted correctly, cast-as-intended (*e.g.*, towards universal verifiability). Alternatively, a user can reverse-monitor[2] an

[2]Reverse-monitor; the action of observing the observer entity. *i.e.*, if monitoring takes place in one direction from a company towards their users, these users watch the company's actions in reverse direction.

online advertisement company to verify it does not invade her privacy.

6.2.2 The Ambiguity of Trust in Naming Convention

Trustworthy Systems	Used for security notions by verification
Trustworthy Computing	Used for security notions by measurement
Trusted Computing	Older term from DoD's Orange Book
Trustworthy Remote Computing	Entities with TPM/Enclave computing
Confidential Computing	Used by industry, Microsoft in Azure
Confidential Remote Computing	Term we model in the rest of the chapter
Secure Remote Computing	Term for the ideal case

Table 6.1: Overview of the terms with closely related meanings in the field.

Trustworthiness implies integrity guarantees among other security notions. Integrity can be observed through measurement or with the means of verification. Through software verification, other guarantees specified such as confidentiality can be obtained. Despite the presence of other security guarantees through verification, however, integrity is the key notion to trustworthiness. In table 6.1, we summarised these concepts. Although confidentiality properties can be derived by measurement or verification, the sub-notions such as privacy, data-secrecy, algorithm-secrecy, require additional attention based on threat modelling. Physical access, *e.g.*, through side-channel leakages, in a system can cause confidentiality issues. For example, software verification can guarantee that a piece of information is kept confidential throughout the life-cycle of the execution, avoiding software bugs leaking the information. In this chapter, we focus on *Confidential Remote Computing*, a concept which requires both strong integrity (due to remote execution) and strong confidentiality guarantees. The older term *trusted* was used in the Department of Defence's orange book [20], however, the current understanding replaces it with *trustworthy* where the users can evaluate the pieces of evidence present and make a trust decision. There are instances of the term *trustworthy systems*[3] used to imply *trust by verification*. *Trustworthy computing* is often used to describe the field of trusted computing, where the pieces of evidence are generated with/from

[3]Trustworthy Systems used by research groups. `https://ts.data61.csiro.au`

hardware *root of trust* primitives. There are secure systems built [2] with hardware technologies (such as TEE-based or TPM-based, etc.), where these systems address the security problems of *trustworthy remote computing*. The industry consortium of confidential computing also uses a similar naming convention, however, it implies a product name/group in cloud computing. Finally, the ideal case of *secure remote computing [165]* is an unsolved problem. While there may be small-scale proof-of-concept studies, *secure remote computing* is difficult to perform as a many-party, high-scale, generic computation.

6.2.3 Position of *Confidential Remote Computing* in Five Stages of Computing History

Similar challenges to ones found in *Confidential Remote Computing* have been around in computing history. Trust and confidentiality challenges have been around for decades, never fully solved, and will likely continue to evolve with new advances in computing by new participants. In this brief presentation of *Confidential Remote Computing*, we connect the past and future of remote computation. There is a progress in these stages on how the computations were carried. This list is not to say that every single computer followed them, **we justify this list with the advancement of the technologies.** Once a technology became available, *e.g.*, PKI, we move into the next stage. For example, with the introduction of hardware-assisted memory encryption, attestation and isolation features, we move into the fifth stage of computations where algorithms can be hidden while offering services in commodity/edge devices. These points are related to each other by the available technologies. Approximately each decade has a technology becoming the mainstream in how computations are done. This list helps us to create the systematic view around different domains in Figure 6.4.

- Trust Your Manufacturer in Confidential Computation:
  - Local Confidential Computation.
- The Use of Public Key Infrastructure in Remote Computing:
  - Remote Representation of Confidential Computation.

- Old-school Cloud Computing as Confidential Computing:
  - Partial Confidential Remote Computation in the Cloud.
- When Data Owners Claim Their Rights:
  - Data-Confidentiality in Remote Computation.
- Algorithms Matter:
  - Fully General Confidential Remote Computation.

From another perspective, *Confidential Remote Computing* can go back to remote procedure calls (RPC) in distributed computing. Any functions executing in remote nodes need to comply with security requirements. The following structure begins with local confidential computation (first stage), and the remote representation of confidential computation (second stage).

6.2.3.1 First Stage: Trust Your Manufacturer in Confidential Computation

Confidentiality requires at least two entities, one which protects an asset threatened by another. With regards to the earlier form of computation, it is possible to talk about confidential computation taking place in local machines. Individuals trust the manufacturers to build the correct computing device. Yet, the manufacturers sell hardware and no direct networking takes place. Individual computations take place locally and without external interactions in the early days.

6.2.3.2 Second Stage: The Use of Public Key Infrastructure in Remote Computing

Around the 1970s[4][5][6], secure communication through certificate-based authentication was initially developed/used by intelligence agencies[7], but only became available to the general public by the 1990s[166] through attempts of the public sector and

[4]Online, Last Accessed 26 May 2022, `https://archive.nytimes.com/www.nytimes.com/library/cyber/week/122497encrypt.html`

[5]Online, Last Accessed 26 May 2022, `https://web.archive.org/web/19980507105259/http://www.cesg.gov.uk/ellisint.htm`

[6]Online, Last Accessed 26 May 2022, `https://web.archive.org/web/19980507105439/http://www.cesg.gov.uk/cnellis.htm`

[7]*e.g.*, British Secret Service, or GCHQ, or considering some claims of NSA. Note added to clarify comments of one of the assessors of this work.

researchers, for example, the invention of WWW by Tim Berners Lee (1991) and SSL/TLS by Taher Elgamal (1994). Certification Authorities (CAs) introduced a new trust model based on public key infrastructures. With the help of this infrastructure, increasingly complex trust models began to facilitate interactions in the digital world. In those models, identity is provided with certificates. We still have to trust the manufacturers to build the right hardware, but in addition, we also need to trust certification authorities to provide the advertised services. While in the first stage local machine owners were only able to trust computations they performed themselves, now they can also validate and trust whether any given data, computation, application, or output stems from the expected entity.

6.2.3.3 Third Stage: Old-school Cloud Computing as Confidential Computing

Over time, entities with the ability of centralisation offered the first form of *Confidential Remote Computing*. These services satisfy customers who need more computational power in order to outsource the computational jobs. The set of hardware owned by those central authorities introduced a service we now know as cloud computing. The problems in the threat modelling, (1) *who protects what* and (2) *protect from who* never became clear. Two major problems appear after this stage: (1) Can the cloud providers trust their own infrastructure to do the right job? (2) Can customers trust the cloud provider's infrastructure to behave as promised? In this setting, the customers must be able to trust to the remote computation, for example, a sorting, compression, or other services. Private data is not necessarily involved yet. Algorithm secrecy is not a concern yet either due to a lack of awareness of the possible value of data and algorithms.

6.2.3.4 Fourth Stage: When Data Owners Claim Their Rights

Soon after the widespread takeover of the cloud systems around 2010, with their existing problems, the value of data becomes more important. The new evolving requirements bring new solutions together. While in the previous stage of old school cloud computing, the data in use is not encrypted and therefore its usage carries

high risks, at this stage, data owners require strong security guarantees for the data in use, besides data at rest and data in transit requirements.

6.2.3.5 Fifth Stage: Algorithms Matter

Recently, developers and companies asserting ownership on intellectual properties have become additional stakeholders in computations. The intellectual property in question is the private algorithm and/or the business logic itself. The algorithm security brings new challenges to private transactions, secret contracts, computer games, and many other domains. Besides the secrecy of the algorithm itself, a public algorithm processing a piece of private data is an attack vector at risk of leaking private data through side-channel attacks. It is not sufficient to keep algorithms private to mitigate the danger of side-channel attack, but data owners can benefit from secret algorithms depending on the decentralised trust model between hardware, data and algorithm owners. Private algorithms must be concealed from the hardware owner to contribute to data security, because potential collusion of private algorithms and hardware owners poses a high risk to data owners. The case-specific threat model, however, must have a micro-optimisation depending on the available security features of the underlying hardware.

6.3 Kernel and Hardware Assistance in Confidential Remote Computing

Confidential Remote Computing has not been fully modelled yet, we listed the existing similar concepts in the Setion 6.2.2. A way to achieve a comprehensive model can be by having the dimensions of *assets* and *demands* of mutually distrusting parties with conflicting interests. Hardware owners, data owners and algorithm owners are distinguishable in a many-party computation. These three parties are distinguished by the assets they bring to and the demands they expect of a computation. In practical terms, any combination of these parties could be a single entity. For example, a hardware owner and an algorithm owner may be represented by one company. Hardware owners are concerned with maintaining

their trustworthiness, maximising their revenue, and are honest-but-curious to learn anything about an algorithm (*i.e.*, business or game logic). The algorithm owners have interest in their logic (*i.e.*, formulated computation) secret, and they are curious to learn about the data they process. The data owners must protect the privacy and secrecy of the data set they manage, while aiming to maximise their revenue at the same time. Both data and algorithm owners are potentially interested in the outputs of computation.

In considering the security of distributed computing, we may trace a development path from the remote procedure call (RPC) forward. The earliest problem to be addressed was managing access control for the remote computational resource, followed by technologies for the security of communications to and from the remote platform. In the simple case, attestation adds further assurance to the initiating party about the service offered on the remote system. Moreover, attested service isolation helps to provide assurance for credentials passed to the remote service so that that service can read or write from other resources.

Much of our work has concentrated on a generalisation of multiple parties wishing to collaborate on a task, despite being mutually-distrusting (and distrusting the remote platform also). Attestation provides a mechanism to enable them to gain trust in the remote platform, but raises many challenges for the management of general-purpose platforms.

In this chapter, we explore the most general case, then, where the data *and* the computation are both secrets. Clearly, in this case, binary attestation may be sufficient to assure all parties that they are interacting with the same code entity, but crucial questions about the behaviour of the computation can only be addressed through some form of semantic remote attestation.

Our objective in this chapter is to explore these dimensions and provide an overview of potential strategies for trustworthy remote computation, leading towards a fully-general *Confidential Remote Computing* framework, always having regard for the concerns of TCB minimisation. We extend the framework to consider

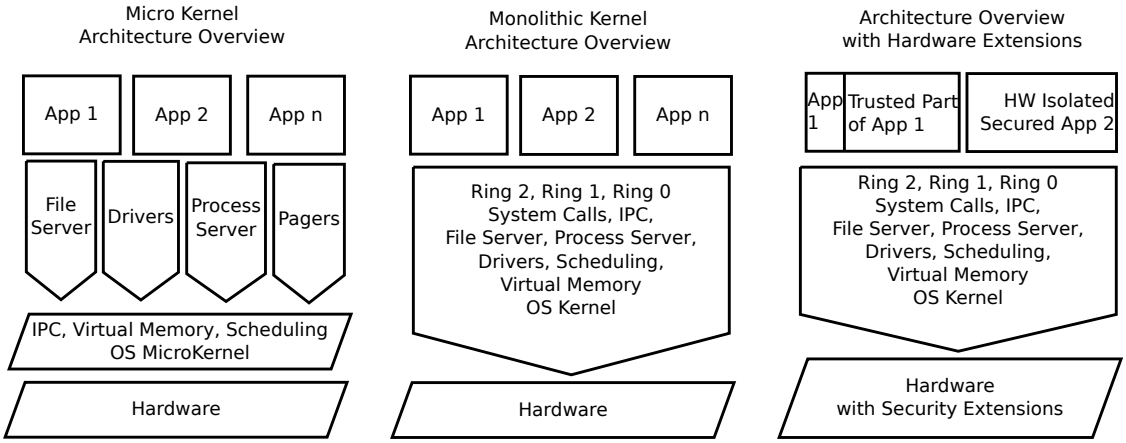

Figure 6.1: Overview of system architectures for security and performance. Monolithic kernels offer better performance, and, their security limitations can be mitigated with hardware extensions.

the notions of privacy protection explored elsewhere in the project as examples of the semantic properties to be attested.

6.3.1 What benefits can Hardware-based Root of Trust offer?

Basing the root of trust in hardware may offer stronger security guarantees than software-based mechanisms for attestation (local or remote) during a computation.

The core of security mechanisms in systems relies on components specified in the Root of Trust (RoT). Sub-definitions of RoT such as Root of Trust for Reporting, Root of Trust for Measurement or Root of Trust for Storage offer increased security with their independence and low size. Establishing trust in a system may require a piece of evidence (*e.g.*, hash, checksum) representing the state of the memory.

6.3.1.1 Is Software-based Root of Trust Insufficient?

Earlier, we introduced the arguments on software-based solutions in Section 2.2, hereby we further explain why is it not sufficient. In embedded systems or in computer peripherals, the software-based attestation mechanisms [167, 168] may generate time-based reliable reports about the trustworthiness of the system. General-purpose computers are more complex and have larger attack surfaces, and software-based attestation mechanisms fail [169, 170] to represent the state of

the system. Software-based attestation mechanisms cannot provide secure reports in rich execution environments if the attacker has physical access to the system. Alternatively, a software-based trust may fail if any exploit gives attackers equal privileges to attempt an attack on a lower level (towards hardware) than the software-based attestation mechanism runs at.

Hardware-based attestation mechanisms often run on the lowest level of systems and utilise private keys embedded inside the chip to generate trusted pieces of evidence representing the memory state. In our model, we focus on remote attestation mechanisms based on secure processors described in Section 6.5.4.

6.3.2 Failure or Success of the Micro-kernels?

The second architectural question for *Confidential Remote Computing* concerns the kernel and privilege structures responsible for performance and security.

Micro-kernel architectures provide software isolation for kernel functionalities, shown in Figure 6.1. In the execution phase, servers do not share resources or pointers and do not have direct access to the address space of other servers. Even in a traditional or layered kernel architecture, address spaces are isolated from each other with privilege levels; if one server passes any address space to another kernel module, changes must be tracked and controlled. The maintenance cost of memory tracking increases the performance overheads in micro-kernels. Even with an acceptable performance overhead, they require the utilisation of distributed programming models to share and maintain the resources between several modules. The distributed algorithms may not be sufficient to make the system available; if a kernel module fails, the system may fail too because of dependent resources, as occurs in distributed systems.

Besides the notion of availability, security is also characterised by the notion of non-compositionality [171]. Having individual secure kernel modules in the micro-kernel architecture approach does not prove the overall security of the kernel. The security properties of smaller modules may not make the system containing it secure by themselves. Micro-kernels were developed later than the monolithic

kernel approach. Current computer systems are mostly using monolithic kernel architectures, and deployment of systems based on micro-kernel architectures requires significant development effort.

The Information Flow Control (IFC) kernels [172, 173] can establish a trusted state for applications running on top of untrustworthy code. Their trust assumptions include hardware integrity and physical security against tampering.

The wimp-giant kernel architecture model [174] suggests an abstract isolation mechanism assuming to provide accurate and complete adversary definitions. In this approach, micro-hypervisors require a verifiable boot. Otherwise wimpy hypervisors and wimpy kernels cannot bring any security guarantee for wimp apps. A hardware-based remote attestation mechanism may satisfy the verifiable boot requirement of a wimpy micro-hypervisor. Nevertheless, this approach does not provide confidentiality and integrity properties for wimp apps. A malicious giant may read the contents of wimpy apps as they please. A giant may compromise the security of wimpy apps with a TOCTTOU attack [139] or an IAGO attack [116]. Due to incomplete adversary definitions, the dancing wimp-giant kernel architecture model is insufficient for establishing trust in the presence of the untrusted environment.

A micro-kernel alone may not be able to offer the security guarantees of hardware-assisted systems. But micro-kernels may be more suitable for formal verification than monolithic kernels.

The formally verified micro-kernels can offer isolation and confidentiality guarantees similar to enclave-based (hardware-assisted) systems. Besides, micro-kernels can be integrated with hardware extensions in a system for better security. For example, the seL4 [22] formally verified micro-kernel can be used with RISC-V [175] systems with the Keystone Enclaves [176]. Either the underlying system managing the enclaves can utilise the seL4, or the seL4 instance can be placed inside the enclaves for kernel support. However, one of the difficulties with formal verification of the micro-kernels remains unsolved, major updates or changes in the kernel may invalidate its proofs. Frequent modifications to proofs limit the implementation of

sophisticated functionalities in micro-kernels and thereby their wide deployment in general purpose computers.

6.3.3 Monolithic kernels are faster but offer no better security than micro-kernels

The X86 monolithic kernel architecture includes four privilege rings: operating system kernel level at ring 0, rest of the operating system at ring 1, device drivers at ring 2 and applications at ring 3. Most of today's systems use only ring 0 including the entire kernel, OS, and drivers, and ring 3 for user applications. Ignoring ring 1 and ring 2 brings performance benefits, as interrupts cause switch overheads between the rings. Since most of the existing operating systems in use are already designed to use only ring 0, it is impractical to re-implement all their software stack. The lack of intermediate privilege rings increases the security risk. If any user application can jump into ring 0, it takes full control of the operating system. If a malicious application hooks the system before a security mechanism running on ring 0 can do so, this means the security mechanism cannot detect the malicious application. This is one of the fundamental problems most anti-virus software products suffer from. A ring-3-malware may easily take control of the kernel in ring 0, and fully evade its anti-virus software.

6.3.4 Improving security of monolithic kernels with protected module architectures

Protected module architectures (*e.g.*, Intel's recent iteration) utilise the ring -3 (minus three) and increase the security of user-level applications with the ability to sandbox a memory region from the rest of the system. The microcode of the processor handles the isolation mechanisms of the user-level applications running on ring 3. This trend brings stronger security guarantees to existing systems and its development models allow existing applications to run with no modification or with small changes. The applications deployed on secure processors can also benefit from the memory encryption engine, remote attestation, confidentiality and integrity features.

6.4 Practical Implementations of Enclaves

Enclaves are the protected memory regions containing an application code. The protected module architectures provide the security features for these enclaves.

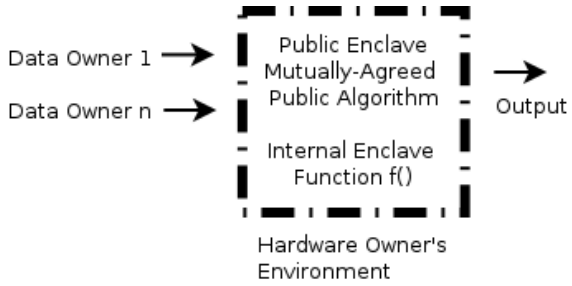

Figure 6.2: Many Data Owners join into a mutually agreed multi-party computation. An enclave can act as a trusted proxy for computations.

We describe two uses of enclaves: first grid computing, and second multi-party computing. In the grid computing shown in Figure 6.3, multiple hardware owners using enclaves can process a big data silo and prove that they did their job. In the multi-party computation shown in Figure 6.2, multiple data owners may jointly perform computations over their inputs.

6.4.1 Multi-Enclave Grid Computing

A long lasting question in grid computing can be answered by looking at the use of enclaves: how can we make sure that a given software processing a significant amount of non-sensitive data (*e.g.*, satellite, climate data) is running on non-tampered computers of end-users contributing to computation? Developers can send the compiled enclave binary to the nodes (who make revenue by running the software on their hardware), and developers can attest integrity of the enclaves with binary-based remote attestation mechanisms. If an enclave algorithm is public, and if the processed data is public, enclaves can provide great integrity guarantees for computations.

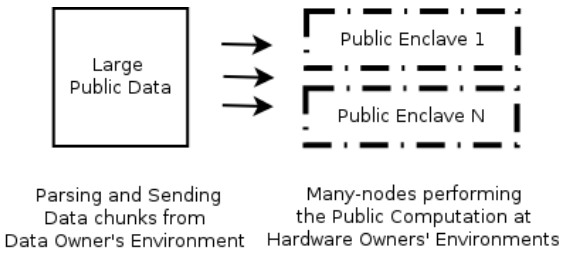

Figure 6.3: Multiple Hardware Owners contribute to grid Computation. Enclaves can prove the integrity of a completed job.

6.4.2 Trust in Grid Computing, Edge Computing, Fog Computing

In grid computing, the nodes are assigned a task containing a computation and must prove the integrity of the computation. During the lifecycle of the taken job, the computing farm must satisfy the integrity requirement. Confidentiality is also required, as the data is sent to the computing farm. On the other hand, in edge computing, edge devices' own data is processed in hardware before sending it to the remote or cloud environment. Later on, the remote environment can still trust the validity of received data and integrity of computation. For example, in the IoT domain, computational correctness is in the interest of the IoT device, the node itself or the system. Taking as an example a smart car system with a fog computing model, the system must display the correct behaviour while sending data to the remote entities. This is required because otherwise the computations can be altered, potentially leading to a loss of revenue. All these systems can benefit from trusted hardware primitives and satisfy their integrity requirements.

6.4.3 Multiple Data Owners using Enclaves

Similarly, multiple *Data Owners* can join a computation utilising the integrity guarantees of the enclaves. If the data is not privacy-sensitive or confidential, the *Data Owners* can retrieve cryptographic evidence about the data sets used in a mutually agreed on computation using a publicly visible algorithm. This way, each participant may trust the performed operations. However, if the data is privacy-sensitive, the question would be whose physical environment will be in use for the

computation? The physical environment of execution shows the entity who controls the physical hardware and the system used to define the threat model.

To process confidential data on public algorithms deployed in public enclaves, the enclave must be programmed carefully against non-trivial side-channel attacks (*e.g.*, page-level memory attacks) depending on local or remote *Physical Environment* of execution. Enclaves can provide integrity, but only limited confidentiality for a computation. Developers may provide additional confidentiality with secure programming.

6.4.4 Various Aims of Hardware Owners for Enclave Software

Hardware Owners differ by the attestation mechanisms they use. For example, the precise attestation information about a system or software is helpful for a company to identify the system version. But, on the other hand, it can tell adversaries the system version as well, thus informing them of existing vulnerabilities. Furthermore, home users require less precision of their attestation reports, although they may still need higher levels of precision to prove what system or software they are running. In the short Table 6.2, we highlight the differences of hardware owners in attestation aspects. A key difference is that, as opposed to home users, enterprises or military organisations often keep their attestation evidence and communications internal within their respective intranets, maximising control over their own systems. In contrast, home users interact frequently and obliviously with potential threats. This raises privacy concerns which can be addressed by requiring the attestation mechanisms to remove the identification information of users.

Type of Hardware Owner	Big Enterprise / Military	Home User
In Attestation	Prioritises Precise Information	Anonymous Hardware IDs
Privacy Aspect	No Privacy Concerns	Needs Privacy
Interaction	Internal	External

Table 6.2: Big enterprises and home users require different attestation evidences.

6.4.5 Confidential Remote Computing in the Real World

We give a few exemplary systems to map *Confidential Remote Computing* trade-offs in the real world. First, a smart grid system, the measurements can be collected every *30 minutes*, potentially for a town of *20.000 people*, for the privacy-preserving operations for billing, monitoring, and demand-response operations. Second, as another example, we may take a government election. During a voting day, the core task for a city of *10 million citizens* is to complete the election in *8 hours*, including the casting, verifying, and counting operations. These computations eventually take place in the remote computers where all participating entities can be satisfied with the level of decentralisation and the evidence of transparency. Apart from this core task (*e.g.*, vote-casting, verifying, counting), a system can also be initially defined by the level of decentralisation or transparency users (*e.g.*, voters) require. In a complex system of IoT deployment, *Confidential Remote Computation* can be defined with hundreds of computing devices running algorithms provided by different entities, and processing data of multiple users. In an industrial IoT deployment, the number of hardware devices can go up to millions. All these continuous computations (also called long-term jobs) represent one single *Confidential Remote Computing* system. Long-term jobs may contain private optimisation and minimisation algorithms for manufacturing complex geometries [177, 178]. The input parameters of desired geometries, algorithm itself, output models and machine behaviour data must remain secret.

The COVID-19 track and trace systems can be built in the CRC model. People provide their location, time and phone number per shop in exchange for a notification service, and are promised deletion of their sensitive data after its expiry. Despite these purported assurances, however, there is little or no effective privacy protection for users' behavioural data. The data collection points (*e.g.*, restaurants) and the data processing points (government health authority) provide no reliable evidence of secure data processing, deletion and refraining from abuse (using it for any other purposes, even anonymised). The computation time can be seen as *2 weeks per check-in* from an individual's perspective. The expected behaviour of the system is

to scan the collected data backwards for each data collection point and to notify relevant individuals whenever a new case or patient is detected. Although ideally this process could be set up with extensive security and privacy services at the cost of becoming more computationally expensive, in practice it is built in a straightforward manner, with its plain-text data processing taking place in a central server whose hardware owners can observe all documented user activities unhindered.

6.5 Co-Evolution of Requirements and Solutions through Five Entities

In order to model *Confidential Remote Computing*, we need to go through three domains as follows: **the hardware domain** as a root of trust, **the development domain** for software composition and **the attestation domain** for integrity representation. We systematise these domains by utilising five distinct parties directly or indirectly involved in the computation in the following manner:

1. **Manufacturer:** Local Confidential Computation
2. **Certification Authority:** Remote Presentation of Confidential Computation
3. **Hardware Owner:** Partial Confidential Remote Computation in the Cloud
4. **Data Owner:** Data-Confidentiality in Remote Computation
5. **Algorithm Owner:** Fully General Confidential Remote Computation

Each of these participants introduces new requirements and new trust models into a computation. We use each participant's respective requirements in chronological order for our model. This classification of computation types offers new concepts and solutions for the three domains outlined above.

6.5.1 Three Domains of Confidential Remote Computing

These three domains are aligned with the dimensions of *Confidential Remote Computing*: hardware features, programming models and attestation mechanisms. Figure 6.4 displays the three domains of *Confidential Remote Computing*, shows how we distinguish the methods, and provides an extendable picture for future methods under these domains. The elements of each domain are listed in order of

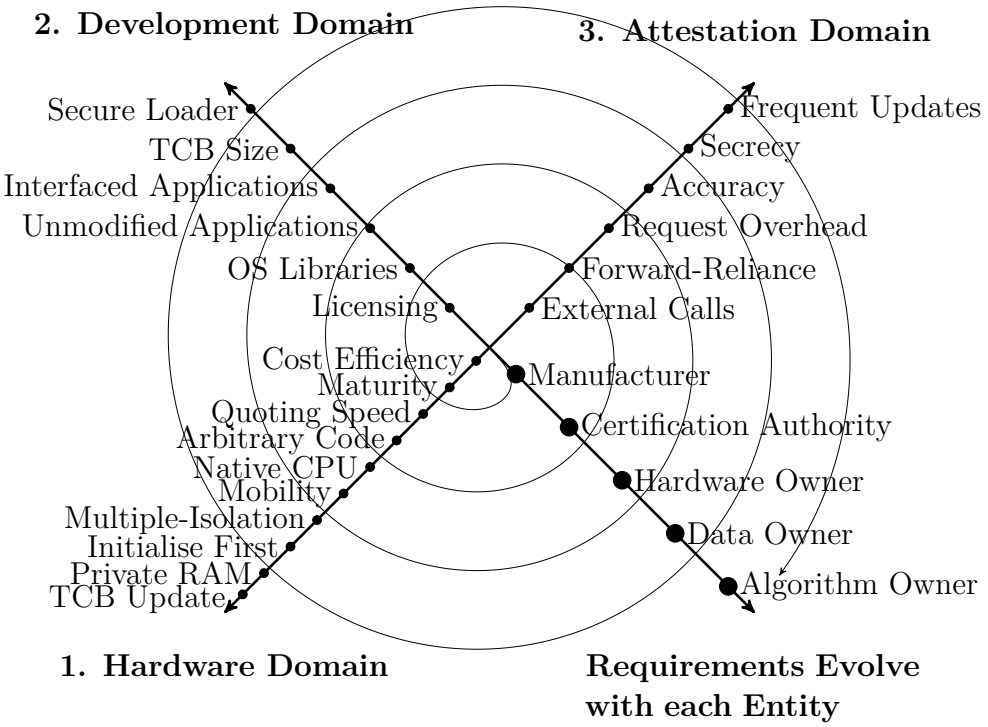

Figure 6.4: Confidential Remote Computing X-chart systematised around the increasing demands of five participants. Evolving concepts and solutions are classified under three domains. These domains show the orthogonal research directions of the field.

availability represented by points along the axes of the figure. Elements arranged closer to the centre have already been adopted at large by the industry at the time of writing. Elements closer to the outer circle may not be available in the market yet and/or have been only recently introduced, but are relevant nevertheless as we shall demonstrate shortly. Each circle completed through the dimensions represents a time period during which the included elements become available on the market as part of technology products.

Our X-chart in Figure 6.4 provides a systematised image of the domains of *Confidential Remote Computing* comprising the following nodes:

- Ten metrics for hardware technologies
- Five features of development techniques
- Six benefits of attestation mechanisms
- Five distinct roles of participating entities (with conflicting interests)

The following four sections elaborate on the four dimensions of this chart. In

Section 6.5.2, we describe the features of the **hardware domain**. This domain includes the practical hardware functionalities for *Confidential Remote Computing*. Each element described in this domain offers different benefits for a target system. They may improve performance, integrity or confidentiality of the system, reduce the cost of development, or give stronger cryptographic evidence of the trustworthiness of the system. Each feature is implemented in at least one hardware solution on the market, but no single hardware solution comprising all the features described has been introduced on the market. An intended *Confidential Remote Computing* system may use one or more hardware products in combination to address stronger adversary models. We describe how an ideal system could utilise each hardware functionality and identify the ideal trusted hardware to meet the demands of *Confidential Remote Computing*.

We describe the **development domain** in Section 6.5.3. For the development of *Confidential Remote Computing* applications, there are a few different possible programming models available in the field of trusted computing. Each method may offer different benefits such as providing a smaller trusted computing base, license-free development or a language defining the system's interaction with the outside world. Evaluating these benefits, we identify the ideal programming model for a *Confidential Remote Computing* system.

Section 6.5.4 addresses the **attestation domain**. We describe the benefits of different attestation mechanisms for *Confidential Remote Computing*. The various available mechanisms provide evidence of different aspects of a computation, different accuracy levels for trustworthiness, cause different overheads, handle different communication flows, and give different guarantees for attestation. We explain why a single attestation mechanism is not sufficient for *Confidential Remote Computing*. We also discuss the potential attestation domain aspects of a hypothetical ideal *Confidential Remote Computing* application.

We present **the conflicting interest of participants** in Section 6.5.5. The *Confidential Remote Computing* model involves several different participants, each holding different assets to protect and different demands to maximise. We distinguish

the participants by their role in the *Confidential Remote Computing* systems. There might be tens of thousands of parties of a certain type of participant group involved with any one *Confidential Remote Computing* system at higher scale. We identify three major participant types (*i.e.*, hardware, data, algorithm owners), aside the minor less numerous participants (*i.e.*, Certification Authorities and hardware manufacturers). We explain the characteristics of the major participant types, and introduce potential scenarios how each participant may maximise their revenue while still protecting their assets. We also explain why certain configuration models will lead to an inequitable arrangement.

6.5.2 Hardware Features

Trusted hardware provides a more secure environment for execution compared to standard hardware. However, any single trusted hardware alone is not sufficient for an ideal *Confidential Remote Computing* application. We compare three leading trusted hardware products by the features we identified for *Confidential Remote Computing* applications. Table 6.3 lists hardware domain elements useful for *Confidential Remote Computing* and compares three hardware technologies widely available on the market: Intel SGX [179], ARM Trustzone [113] and the TCG TPM [180] as three examples of available hardware. We pick Intel SGX for its availability on desktop computers, ARM Trustzone for its mobility on the IoT and smartphone market, and TPM because it is an independent external chip performing trustworthy operations.

For sake of brevity, we omit other secure processors such as XOM [181], Aegis [182], Bastion [183], Ascend [184], Phantom [185], Sanctum [186] in this chapter. Future work may evaluate these processors along the same hardware dimension we described. We also omit the technologies TDX [187, 188], Morello [189], AMD SEV (ES and SNP) [190] and IBM PEF [191], which run towards similar aims as SGX and Trustzone. A detailed analysis of these trusted hardware technologies may be explored elsewhere.

Multiple-Isolation: A secure application must be isolated from any untrusted component. As an external chip, TPM does not provide any isolation for the computation, but it provides secure storage for the measurements (Root of Trust for Storage) of a computation [180]. TPM itself does not isolate the system from applications, but DRTM mechanisms can be used to have software-based isolation mechanisms for applications. ARM Trustzone [113]-based processors provide strong isolation for a single secure world placed in a secure memory area which is different from main memory [192], but the hardware itself does not provide multiple memory regions for isolation. There are TEE [193, 194] software solutions such as Trustonic [195] for isolated multiple execution environments [196–198]. Even so, software-based solutions may not [199, 200] be sufficiently protective against powerful adversaries. SGX as implemented by Intel provides a multiple-isolation mechanism in microcode for its enclaves [179]; this provides multiple small regions for computation isolated from each other. However, Intel's solution does not provide a fully private memory; all enclaves sit on the main memory.

Private RAM: A secure system must have its own secure memory area, fully differentiated from any of the untrusted components. TPM has its own storage for measurements [180], but it does not offer any benefit for a computation loaded and executed in the main system. TPM can help to ensure software based memory encryption mechanisms are in place, but application's memory still sits in the main memory. ARM Trustzone has the best solution among trusted hardware solutions on the market. The secure world sits in a different memory area (depending on manufacturers' setting) than the main memory [192, 201]. The operating system and any untrusted component (normal world) of an ARM Trustzone enabled device has no control over the protected memory of the secure world [113, 192]. Intel's SGX has no private memory solution. All of the enclave memory uses pages within the main memory [202]. This leaves access patterns of memory open to inspection by potential attackers. By design, Intel's processors also have various shared resources which is a crucial shortcoming for security-critical applications.

Mobility: A secure IoT solution needs good mobility for deployment in the real world. TPM is an external chip accommodated on the system board [180]; TPM is widely included in most modern hardware. ARM Trustzone-equipped devices provide mobility and excellent security features in the mobile environment [113]. ARM processors are low-energy consuming and widely used in most of the currently available mobile devices. Intel's SGX offers almost no mobility for IoT yet, as it is only deployed in high energy-consuming processors which require more power than its competitors, and Intel machines are larger by size.

Fast Quoting: In order to attest a remote setting, hardware must generate cryptographic evidence about the trustworthiness of a system. TPM-based systems have quoting speed of 731ms [92]. This quoting speed may fulfil the requirements of many party applications, however, at larger scale, it may not suffice as shown in recent studies [2]. ARM Trustzone does not have any quoting or attestation mechanism for remote systems. Developers may implement a custom software-based attestation mechanism, and ARM Trustzone can manage a secure boot and measurement for software attestation [201]. However, a custom attestation implementation would not provide security guarantees of comparable strength to those enhanced within the hardware. Intel's SGX has the fastest quoting speed so far available on the market at around 20-30ms [2] speed per quote[8], which currently makes it the best candidate for deployment of *Confidential Remote Computing* applications.

Initialise First: Initialisation of the security module supporting applications should take place before the execution of any untrusted component in order to prevent any hook being placed against trusted components. Alternatively, the initialisation of the trusted component should not be dependent on untrusted components. TPM does not aim to provide direct assistance for the initialisation process of secure computations, but it runs externally and independently of the system; its features themselves are available for computations. ARM Trustzone offers the best solution so far presented on the market, as its hardware initialises

[8]even faster-quoting speed by the time of writing this document, thanks to recent fixes in Intel SDK

Ideal Feature/HW	(1)	(2)	(3)	Future
Multiple-Isolation	Y	N	N	Y
Private RAM	N	Y	N	Y
Mobility	N	Y	Y	Y
Fast Quoting	Y	N	N	Y
Initialize First	N	Y	Y	Y
Native CPU	Y	Y	N	Y
TCB Update	Y	N	N	Y
Arbitrary Code	Y	Y	N	Y
Secure Loader	N	N	N	Y
Maturity	N	Y	Y	Y
Cost Efficiency	N	N	Y	Y

Table 6.3: Comparing three trusted hardware models: Intel SGX (1) in-processor TEE, ARM Trustzone (2) mobile TEE, and TPM (3) external trusted hardware from TCG.

the *Secure World* before the *Normal World* [201]. Moreover, its *Secure World* application can [192] set a timer to trigger itself through an interrupt at any point in time without relying on a dependency on the *Normal World*. This feature is one of the most significant advantages of ARM hardware in deploying *Confidential Remote Computing* applications on Trustzone-enabled devices. Intel's SGX has no independence from the untrusted world. The operating system has to trigger and allocate memory for the enclaves [203]. Intel's SGX relies on their attestation mechanism to verify the integrity of the enclave later on. However, the OS and untrusted components still have full control over enclaves.

Native CPU: A secure application may require high processing power for its computations. TPM is not a processor and does not provide any computational resources for the execution environment. ARM Trustzone processors feature relatively good CPU power that a *Secure World* can utilise in mobile systems. Intel's SGX offers the highest CPU power for enclaves so far on the market; it is also available for server processors.

TCB Update: The Trusted Computing Base of a system may need updates from time to time due to emerging vulnerabilities or newer functionalities. TPM may offer a TCB update with append-only storage of hashes assisting with the representation of the updated state of a certain system or programme. It offers fundamental RoT primitives to identify the system's state or the state of arbitrary

computations. System designers must define and load the software of the *Secure World* in advance, and its TCB update may be limited to patching the *Secure World*. Custom solutions can be implemented by the TEE developers. Intel's SGX provides TCB update mechanisms for both the SGX hardware and the enclaves [204]. TCB updates help patch the CPU Firmware (microcode) to fix the hardware bugs. Enclaves derive benefits from secure updates. For example, CPU can refuse executing outdated enclave software.

Arbitrary Code: A *Confidential Remote Computing* application may require arbitrary code execution on trusted hardware. Computations take place outside of TPM in the system. TPM does not handle the computations itself, but TPM-based solutions may assist applications in general-purpose computers. ARM Trustzone-based systems require developers to place the application code in the *Secure World* in advance, and the participants may not be able to load any arbitrary code remotely later on without third-party solutions. Intel's SGX allows [203] users to load any arbitrary code into the enclaves.

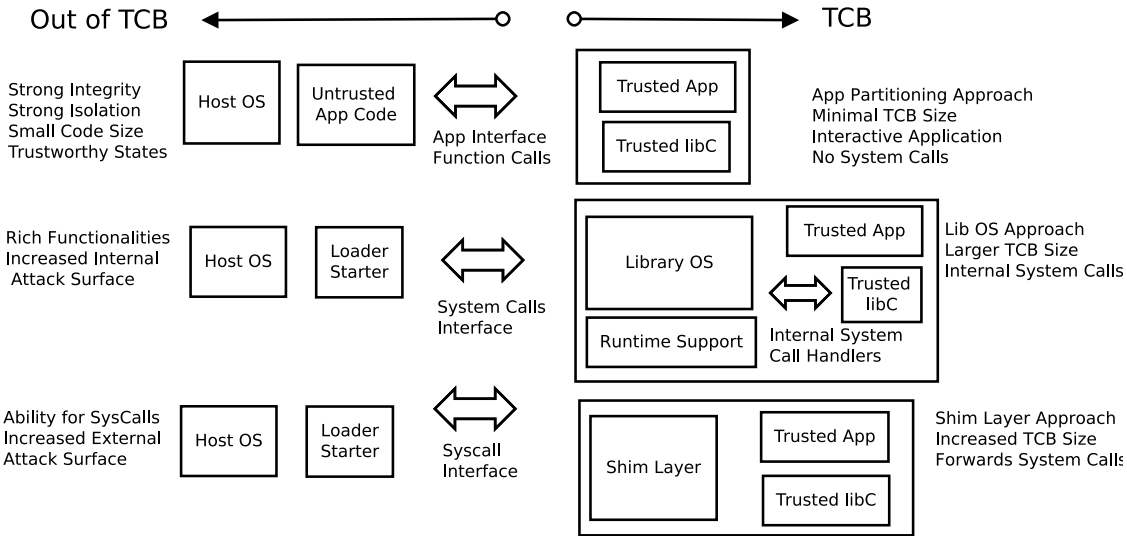

Figure 6.5: Overview of Development Models Security and Performance

Secure Loader: A *Confidential Remote Computing* application may require a secure code loader for loading code into the trusted execution environment. TPM offers fixed functionalities for system software. A separate software-based code loader in a system may be assisted by TPM. Trustzone developers must load the

initial secure-world code themselves, and they can fetch other code later on. They may place keys into the protected storage of Trustzone, which are required to load a secret code securely [192]. However, this may not scale for multiple code providers to cooperate yet. In the case of Intel's SGX, untrusted components are free to load any software into the enclave, but attestation mechanisms allow users to verify the content of an enclave before proceeding with execution. Attestation mechanisms utilise the binary hash of the enclave, compromising all secrecy of enclave code itself. Initially, there was no secure loader for the enclaves. Currently, Intel's SGX SDK [202, 203] offers integrated features as well as third-party solutions for secure code loaders. However, this raises other issues such as lack of property-based attestation and malicious secret behaviour in enclaves.

Maturity: A hardware candidate for *Confidential Remote Computing* should be mature enough to be a valid choice for commercial applications. TPM is the most accepted trusted hardware solution among current commodity systems, and it provides secure solutions for Root of Trust for Storage and Root of Trust for Reporting [180]. ARM Trustzone is another mature technology, widely deployed in most of the current mobile devices [113]. At the time of writing, Intel's SGX technology seems to require additional time to obtain the maturity necessary for enterprise-level solutions.

Cost Efficiency: The trusted hardware should be cheap and accessible for scalable deployment, considering the expected prevalence of IoT devices in the near future. TPM is one of the cheapest trusted hardware available in the market in terms of both hardware and development cost. In contrast, ARM Trustzone-based hardware are more expensive on the market, and incur additional high licensing cost for production. Intel's SGX hardware is also relatively expensive, but its license allows for free and open-source development.

6.5.3 Programming Models

In this section, we explain the programming models of secure applications. We evaluate four types of development using different approaches. Figure 6.5 shows three

approaches to enclave development with system support. These approaches have different interfaces and they result in different TCB sizes. Interfaced applications can provide strong integrity and isolation, smaller code size and trustworthy states. They follow the application partitioning approach: they can be interactive, and depending on the goals of the application the enclave may need to interact with the system in order to perform system calls or by design it may not require system calls. Enclave applications with embedded LibOSes feature a larger TCB size. They can offer rich functionalities through the libOS support; however, the increased TCB size increases the attack surface. System calls can be kept inside the enclave. The alternative approach to providing system support is the use of a shim layer, which also reduces the TCB size. This approach can provide a hybrid method for application development. TCB size can be kept relatively smaller than using a LibOS approach, but still provide a secure interface to the underlying system. We show the development models in Table 6.4 and evaluate them by five metrics: (1) the TCB size, (2) the communication interface of the enclave and the outside system, (3) the status of OS library support or use, (4) the status of development licenses (*e.g.*, open development), (5) the code loader for initiating the execution.

The development models could be expanded on in a separate study with other SDKs such as Microsoft OpenEnclave [205] supporting multiple TEEs, SGX-LKL [206] for SGX, Rust-EDP [207] for SGX, Google Asylo [208] SDK, Keystone [209] SDK for RISC-V, Sancus [210] SDK. Bulck *et al.* explored the security vulnerabilities of these SDKs [211].

Type \ Feature	TCB Size	Interface	OS Library	License	Loader
Intel's SGX SDK	Low	Interactive	Minimal	Open Source	Plain
GrapheneSGX SDK	High	No	Full	Open Source	Plain
Shim Containers	Medium	No	Shim Layer	Expensive	Plain
Trustzone	Low	Independent	No	Expensive	Secure
Ideal	Low	Independent	Minimal	Open Source	Secure

Table 6.4: Comparing development model features of different types of SDKs/systems.

TCB Size: Intel SGX SDK provides trusted libraries developed by Intel. The TCB size of the enclave is usually low. GrapheneSGX includes Library OS which adds substantial amount of code into the TCB. Shim Containers increase the TCB size more than regular development with Intel SGX SDK, but still less than a Library OS.

Interface: Intel SGX SDK allows developers to define the interface of an enclave via Enclave Description Language. The enclaves can interact securely with untrusted components through the interface. Enclave developers are responsible for designing and implementing a secure interface. GrapheneSGX provides no interaction with untrusted components; its design principle requires strong isolation between unmodified applications and untrusted components of the system. Applications can still make HTTP calls for communication with the outside world.

OS Library: Intel SGX SDK contains only minimised trusted libraries developed by Intel. GrapheneSGX SDK includes a full Library OS which helps any unmodified application execute inside their enclaves. Shim Containers use Shim Layers to pass calls on from the trusted environment to the untrusted environment.

License: Both Intel SGX SDK and GrapheneSGX SDK are free and open source. Container solutions are usually not open-sourced and may incur high expenses on the side.

Loader: In the initial model, both Intel SGX SDK and GrapheneSGX SDK load plain enclave code from the untrusted and inspectable environment. Thus, enclaves (*i.e.*, the loaded code) are visible to the owner of the execution environment. The loading mechanism in Intel's SDK has been updated to run with a Protected-Code Loader, and recent research shows [6, 79, 80] how the algorithms and enclave code can be protected.

6.5.4 Attestation Mechanisms

An attestation mechanism provides evidence to be presented to verifying entities. These pieces of evidence can consist of the hash of the binaries or any other means by which the summary of execution can be represented. The communication

Type \ Feature	Based On	Update	Accuracy	Flow	Secrecy	Forward
DH/SIGMA-based Remote Attestation	Quote	Changes	High	Complex	No	No
TLS-based Remote Attestation	Cert.	Changes	High	Simplified	No	No
Property-Based Attestation	Features	Static	Custom	Simplified	Yes	Yes
Final State Attestation	Hash	Changes	High	Simplified	No	Yes
Local Attestation	Report	Changes	High	Simplified	Yes	No

Table 6.5: Comparing different aspects of Attestation Mechanisms.

complexity of attestation schemes can cause an overhead between the participants. We summarise different attestation mechanisms in Table 6.5, evaluated in six aspects: (1) what the evidence is based on, (2) how the evidence is affected by software updates, (3) the level of accuracy presented by the evidence, (4) the complexity of the communication flow, (5) whether code secrecy is maintained by the evidence, (6) whether the scheme can provide forward reliance with rules and policies.

The Basis of Evidence: In Diffie-Hellman and SIGMA Protocol-based remote attestation, the QUOTE signed by the Quoting Enclave serves as evidence. In TLS-based remote attestation, the communication is additionally simplified with certificates. The property-based attestation mechanism uses features as a basis of the evidence. Final State Attestation relies on the hash that was previously trusted by verifiers. The local attestation relies on the REPORT generated by CPU firmware in an enclave itself.

The Impact on Software Change: If the software of the secure application changes in any way, presented evidence also changes in all attestation mechanisms except property-based attestation. In property-based attestation, the evidence would only change if the feature it is based on undergoes a major change, otherwise it will remain under the trustworthiness threshold defined by a verifying party.

The Accuracy of Attestation: Attestation evidence can tell little or more about attested software. In property-based attestation, the accuracy depends on the algorithm that analyses the secret code. High accuracy and secrecy are

mutually exclusive notions, and a trade-off between them has to be made. All other attestation mechanisms have high accuracy, as the evidence can give away the software version of the code used.

The Communication Flow of Attestation: The old type of remote attestation mechanism used by Intel SGX SDK had a relatively complex communication flow with approximately 14 calls. All other attestation mechanisms now offer simplified communication flow between the verifying and proving parties.

The Secrecy for Attested Code: Local attestation and property-based attestation can provide secrecy for the attested code. Local attestation can prove that two parties are located on a local machine. Based on a generated report, a verifying party can be informed that a prover is executing on the same hardware, without the proving party having to disclose the content of their assets. In property-based attestation, a verifier obtains as much of the feature list as the prover permits without learning the source code of prover. In other schemes, the verifier or another external party must approve that a certain piece of evidence belongs to a certain prover by replicating the same evidence generation method, for which they require delivery of the source code of a prover.

The Future Reusability of Evidence: In Final State Attestation, a verifier can rely on previously generated a piece of evidence, assuming that the prover does not change its state after the attestation. In property-based attestation, similarly, a verifier can trust a prover without requiring any further approval after each update of the prover's code. Once a verifier approves a list of features that the prover's code can access and perform, the prover can update its code as long as it satisfies the conditions of the verifier's previous approval.

6.5.5 Participant Roles in Confidential Remote Computing

Table 6.6 summarises the various roles of participants in *Confidential Remote Computing*. These roles can be fulfilled either by a single or any number of participant entities, and any given entity can act in more than one role. For example, in a blockchain-based system the hardware owner may utilise millions of devices owned

by different participants, and combine their computational power to offer a unified infrastructure. On the other hand, a homomorphic encryption scheme can represent a single hardware owner, operating with a single algorithm owner, but receiving input from multiple data owners. In an ideal configuration, high numbers of these entities would need to be able to join and leave the computation at any point in time.

What do they demand? *Hardware Owners* (or hardware resource providers) demand to be able to make more revenue off their existing infrastructure. The *Algorithm Owners* need large data sets to run their algorithms and perform analysis. They either demand the result of the computation for themselves, or they are interested in making revenue with the algorithm as a provided service. *Data Owners* request new algorithms to process their data from algorithm owners. Like algorithm owners, they too are interested in computational results, or in making revenue by allowing the use of their data. *Data Owners* are also interested in computational results of other *Data Owners*.

What are their assets? *Data Owners* store large confidential data sets, they also protect the privacy of individuals in the data set. *Algorithm Owners* maintain a private algorithm they developed. Such an algorithm is commercially valuable and requires secrecy. *Hardware Owners* have a level of trustworthiness and reputation that they must maintain in order to maximise their revenue.

What do they threaten? *Data Owners* and *Hardware Owners* are curious about the secret algorithm. They pose a potential threat against the algorithm; they may for example collaborate to leak the secret algorithm. *Hardware Owners* and *Algorithm Owners* are curious about data sets. They take any opportunity to threaten private data sets, and they may collaborate to diminish the privacy

Feature / Role	Demands For	Assets	Threat Against
Hardware Owner	Revenue	Computation Platform	Data, Algorithm
Data Owner	Results, Revenue	Private Data	Algorithm
Algorithm Owner	Results, Revenue	Secret Algorithm	Data

Table 6.6: Comparing Participant Roles

of individuals in data sets. As the algorithm processes the data, it is easier for a *Hardware Owner* to collaborate with the *Algorithm Owner* against the *Data Owner* to leak or signal the private data. In addition, *Hardware Owners* are also curious about both secret algorithms and private data.

6.6 Trade-offs in working towards Ideal Confidential Remote Computation

We define three requirements for the ideal case. Because these requirements are difficult to satisfy all at once, application developers usually have to make trade-offs according to their preferences. A system might be defined by how much time its execution takes, the number of required participants to complete a task, and the rules or the instructions to operate it. The task size then determines the amount of resources contributed by the resource providers. These resource providers are called hardware owners, data owners, and algorithm owners. The providers can consist of multiple parties or a single entity. If decentralisation is required in the system, this can increase the communication overhead between these entities. Transparency can be provided through attestation, communication and verification processes. If more transparency is required to convince the participating entities, then this reduces the portion of the task that can be completed in a given time. In Figure 6.6, we show that full decentralisation, complex task size, and full transparency are difficult to achieve at the same time. Each of these parameters complicates the next one and causes an additional overhead. For example, a larger task size requires more resources. The resources can be provided by a centralised authority, but a decentralised resource provider may not be able to fulfil the demand in restrictions. A real-world example can be that visa cards can handle a high volume of transactions centrally, but block-chain technologies cannot reach the same level of throughput owing to their decentralised nature. The second challenge is that high decentralisation makes transparency operations even more difficult. However, it is required in order to safeguard decentralised activities with transparency, preventing creeping centralisation. Transparency then increases the

overheads in communication, attestation or verification processes. The accumulation of these overheads render the completion of a task effectively impossible.

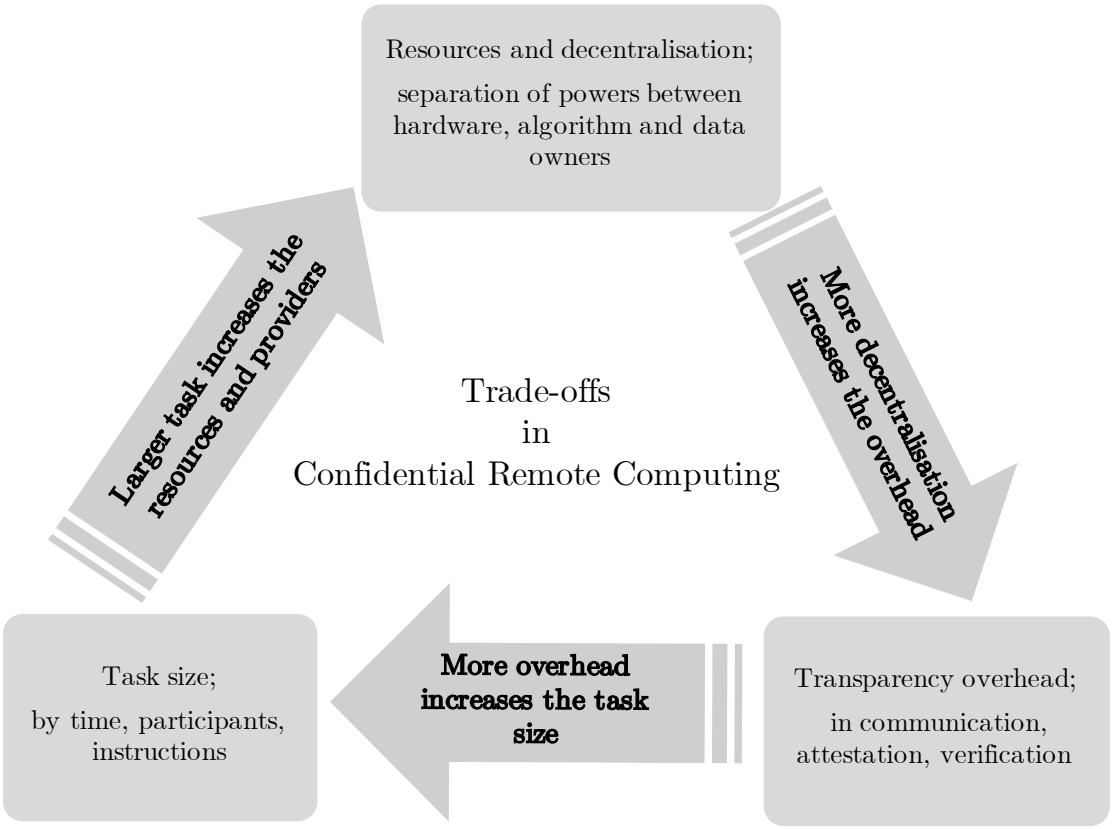

Figure 6.6: *Confidential Remote Computing* and its trade-offs. The initial requirements can begin from any of given node, are in turn complicated by the other parameters, increasing their cost.

6.6.1 Control Decentralisation of Computations

Today's cloud services retain and accumulate their control of computations and become centralised powers. Data and algorithms are not hidden from most cloud providers. The first step to break the authoritarian execution models of the cloud is to keep data confidential in-use to answer increasing privacy concerns. For a better decentralisation, the computations must be independent of the underlying hardware pool. While manufacturers do not control what is computed using their devices, most of the cloud hardware owners are selective on what can be computed. Hardware

owners must remain neutral, lest they collude with either data or algorithm owners in alternative channel attacks on each other.

6.6.2 Scalability under Limitations of Time, Participants and Instructions

Ceteris paribus, a cloud system offering no security may scale higher at a lesser cost. One way to quantify scalability is the number of participants being served in an hour. Cryptographic protocols can offer formal security properties, however, its performance is limited by the number of participants and operations. In a fully scalable setting, systems must be able to support multiple hardware, data and algorithm owners having a joint computation. The number of distinct participants may go up to millions or billions with general-purpose computation abilities. Participants must be able to enter and exit the computations dynamically.

6.6.3 Communication Overhead in Attestation Methods

Automating the attestation mechanisms can help to reduce communication overheads. For example, a frequently updated system with binary attestation can slow down the entire process. Attestation reports can be verified asynchronously, or attestation quotes can be reused when possible, the number of properties of the report can have variables to tolerate certain changes.

6.6.4 Challenges on Keeping TCB Minimal

While decentralising control, allowing for a high number of participants and automating the attestation process, the whole system should have the minimal possible trusted computing base. An often repeated mistake is that TCB size goes dramatically high, up to hundreds of thousands (or even millions) of lines of code. This causes difficulties in formalisation and often results in an obscure failure of security analyses.

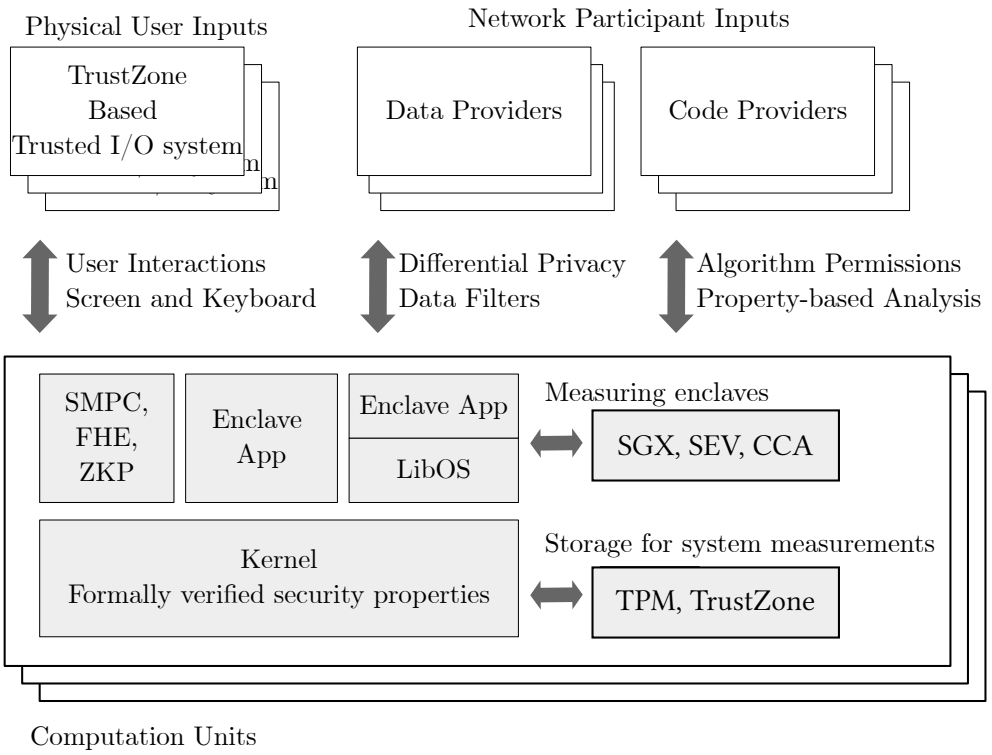

Figure 6.7: A unified architecture for *Confidential Remote Computing* systems.

6.6.5 The Unified Computing Model

We use the lessons learned to build a unified architecture for *Confidential Remote Computing* units, shown in Figure 6.7. Multiple hardware technologies and techniques are integrated into the ideal setting. A micro-kernel may not necessarily improve the security of the system by itself. Still, a formally verified micro-kernel for this architecture can improve the security of the overall system (one candidate for such a kernel is seL4 [22]). A subset of the model can be used in smart grid systems, in e-voting systems or in hub/gateway systems of the Internet of Things (IoT). End-to-end cryptography can be used in isolated enclave applications. Enclave code can also contain applications utilising zero-knowledge proofs (ZKP), secure multi-party computation (SMPC) and fully homomorphic encryption (FHE) schemes. Enclave measurement might be done by SGX instructions, SEV instructions, ARM's CCA or future instructions.

6.7 Closing Remarks

In this chapter, we introduced the CRC model[9] under three domains and we explained how it evolved to its current form. We presented security practices for CRC for non-malicious intentions, however, some researchers may argue that the hardware instructions used in CRC can also be used to strengthen malware. In the next chapter, we clarify those concerns by providing the most compherensive systematisation to date about the malware in enclaves.

Contributions

This chapter expresses the most comprehensive model of *Confidential Remote Computing* yet. The *Confidential Remote Computing* paradigm brings new opportunities to the computing world. Our model aims to provide a structured view of the technologies and methods behind the digitalisation trend of the concept of trust. Throughout the chapter, we make the following contributions with the novel *Confidential Remote Computing* model:

(1) CRC resolves the ambiguity of *trust* in relevant domains, and we connect the former understanding of *trust* in computations (local and certificate-based) to future computing models (with distinct cloud providers, data providers, and algorithm providers).

(2) We provide an extensive analysis on how *digital trust* can be derived from alternative methods such as micro-kernels, software-based attestation, and the verification technologies, followed by use of the hardware enclaves in the grid, edge and fog computing use cases.

(3) We consolidate the CRC model in the X-chart in three domains; hardware, attestation and development, presented in Section 6.5. Each of these domains is surveyed and systematised with applied techniques and methods in the literature.

(4) We present the trade-offs in the CRC model, due to the large task size, more decentralisation and more transparency.

[9]Some others may call it the CRC framework.

Chapter Recap

In this chapter, we attempt to consolidate the disparate research approaches into one *Confidential Remote Computing* model, from past to future. We begin by explaining the participating entities in chronological order, divided into five subsequent stages. The model then is structured in three main sections. First, we begin with the role of hardware technologies in *Confidential Remote Computing*. Then we introduce our model demonstrating the orthogonal research structured by each entity. Lastly, we demonstrate the key trade-offs providing an overview of the deployed systems.

Chapter 7

Twelve Misconceptions about Enhancing Malware with Hardware Enclaves

Contents

We introduced the related work on where malware research and hardware-based security research intersect in Section 7.1. In Section 5.9.1.3, we discussed why our solution in Chapter 5 (with private algorithms; see Section 7.4.1) would not conflict with our arguments presented here. The problem definition of CRC on distinguishing an infected machine in Section 6.2.1, is extended in this chapter as to why the CRC would not give a malware more tools to boost their characteristics presented in Section 7.2.2.

Besides Intel's SGX technology, there are long-running discussions on how trusted computing technologies can be used to cloak malware. Past research showed example methods of malicious activities utilising Flicker, Trusted Platform Module, and recently integrating with enclaves. We observe two ambiguous methodologies of malware development being associated with SGX, and it is crucial to systematise their details. One methodology is to use the core SGX ecosystem to cloak malware; potentially affecting a large number of systems. The second methodology is to create a custom enclave not adhering to base assumptions of SGX, creating a demonstration code of malware behaviour with these incorrect assumptions; remaining local without any impact. We examine what malware aims to do in real-world scenarios and state-of-art techniques in malware evasion. We present multiple limitations of maintaining the SGX-assisted malware and evading it from anti-malware mechanisms. The limitations make SGX enclaves a poor choice for achieving a successful malware campaign. We systematise twelve misconceptions (myths) outlining how an overfit-malware using SGX weakens malware's existing abilities. We find the differences by comparing SGX assistance for malware with non-SGX malware (*i.e.*, *malware in the wild* in our chapter). We conclude that the use of hardware enclaves does

not increase the preexisting attack surface, enables no new infection vector, and does not contribute any new methods to the stealthiness of malware.

This chapter was presented at SGX Community Day 2020 [11] and at HASP 2022, and it is based on this publication [12].

The theme of this chapter is to evaluate the TEE assisted malware development from the perspective of malware developers (bad actors), understand the malware characteristics and requirements of the malware and evaluate systematically whether TEE assistance can fulfill the implementation details of a stronger malware. Section 7.5 relates to this theme by comparing a TEE assisted malware with a wild malware in a system, The relation of Section 7.6 to the chapter's theme is to discuss the remaining points on which open points are left for bad actors to be considered in future trusted hardware designs.

7.1 On Enhancing Malware with Trusted Hardware

There are example studies in the literature [212–216] for bad CPU design choices, micro-architectural attacks and the known-to-be-vulnerable software chunks used in enclaves. When CPU design choices cause a security issue, microcode can receive a Trusted Computing Base update [217]. In fact, the system's microcode gets patched at each boot cycle. If an attack requires full kernel and OS control, this can be mitigated by a measured boot of a formally verified kernel. The applications containing vulnerable software must update their enclave code base and revoke any execution permissions for the older versions.

We examine the question of whether developing a fine-grained malware by utilising TEE makes it stronger or weaker [218–221]. There are long-lasting discussions based on these questions:

- Can SGX help to deliver malware payloads?
- Can TEE help ransomware (*i.e.*, for operations of data copy/ encryption/ delete, key generation/ storage, persistence, communication)?

- What secret operations can an enclave handle/execute inside an isolated memory region?

The purpose of enclaves is to allow secret operations; nevertheless, any secret operations with malicious intentions are similar to independent, remote operations. We shall further discuss these questions in detail in the scope of this chapter.

Enterprise-level anti-malware solutions with advanced memory protections can apply to enclave-based malicious software. Memory Exploit Mitigation (MEM) from Symantec [222], and other defensive techniques as Data Execution Prevention (DEP), Address Space Layout Randomization (ASLR), Structured Exception Handling Overwrite Protection (SEHOP) and Return-oriented Programming (ROP) protections are effective for detection and mitigation purposes [223].

With these questions above in mind, we present our systematisation in this chapter.

7.2 Why Enclave-based Malware is Infeasible?

Today it is highly likely that the majority of commodity systems are exposed to malicious software due to their highly complicated software stack. One of the revolutionising techniques in fighting malware is to utilise trusted hardware enclaves. However, using enclaves has also provoked discussions, and misconceptions leading to myths about whether attackers can utilise these enclaves for adversarial purposes. Similar to end-user computers, home IoT deployments suffer from the same problem. Nowadays, most home Internet of Things (IoT) deployments contain insecure, untrusted, outdated devices installed in houses, permanently spying on users. We consider the malware living in commodity systems and home IoT deployments as the *malware in the wild*.

An example of malware in the wild is a light bulb in a smart house. The light bulb may contain a full Linux software stack, always online in the network, sniffing and spying on the home WiFi network, and constantly leaking user assets. We group these types of harmful IoT devices as *Evil Light Bulbs (ELB)*, as the

malware living in a highly noisy commodity environment. Trusted hardware technologies, *e.g.*, Trusted Execution Environments, also fight back against ELB by securing the IoT deployments[1]. We examine whether malware in ELB can utilise trusted hardware to be more powerful in attacks towards industrial IoT and smart home IoT deployments. Our arguments and the misconceptions discussed in Section 7.4 for wild malware and enclave-based malware in computers are also valid for ELB in IoT deployments.

Concepts: Throughout the chapter, we may use the following synonyms. Wild malware refers to malware in commodity systems, malware in the untrusted world, ELB in IoT deployments, or malware in a highly noisy system. Enclave-based malware refers to a malicious code placed in a custom developed enclave, SGX-malware, SGX-based malware, malware based on trusted hardware, or malware in TEE. In order to distinguish the core SGX ecosystem from custom developed enclaves, we provide a detailed definition in Section 7.2.3.

7.2.1 Scope of Malware

Malware presence can be in various forms; *virus, worm, trojan, zombie/botnet, phishing/spam, spyware, keylogger, sniffer* and *ransomware*. Throughout this chapter, we refer to this section for the precise scope of the *malware*. We take into account specifically the malicious software targeting Windows PCs. Nevertheless, most of our systematisation can also be valid for Linux distributions and OS X systems (Intel, non-ARM). Malware targeting other commodity operating systems, advanced persistent threats (APT), and other relatively harmful software can be explored outside of this chapter. We keep relatively badly behaving software outside of the malware definition. A bad program is one, when found and analysed by an AV company, is subsequently detected by their malware detection systems. Finally, we consider malware as malicious software tailored for disruption, damage and unauthorised access to the commodity computer system. We describe the characteristics of malware in more detail in Section 7.2.2. Similar to malware

[1] `https://www.iotsecurityfoundation.org/best-practice-guide-articles/device-secure-boot/`

in a single computer, an IoT deployment with tens of devices in a house suffers from the same problems due to ELB.

7.2.2 Characteristics of Malware

The ideal malware must[2] have the characteristics listed below for a successful malware campaign. However, one of the fundamental issues is that these characteristics can also be seen in benign and legitimate software. Having these characteristics in benign software makes malware detection a challenging task for anti-malware mechanisms.

7.2.2.1 Persistence across boot cycles/load at startup

The malware must be able to reload itself whenever the system is available. This can be done via scheduled events, registry entries, drivers and functions in other processes. Section 7.4.5 presents the persistence aspect of enclave-assisted malware.

7.2.2.2 Communication on demand

Ideally, the malware must be able to communicate with the author to receive commands or leak information whenever required. Communication might be necessary for reselling the network of victims, fetching a new payload or ready-to-use new exploits, or equipping malware with up-to-date zero days to be exploited. A secure communication channel is also required for the key management; whether the keys are managed locally or through external servers, the malware must have a direct or indirect key management channel. We present the communication aspects in Section 7.4.3 and Section 7.4.6, and key management aspects in Section 7.4.2 and Section 7.4.4.

7.2.2.3 Full Un-Detection (FUD)

The malware must remain fully undetected throughout its life cycle against any possible detection mechanisms. At a minimum, this can be done via resilience to all anti-virus software products in the market. FUD checks can be done offline via

[2]Others may structure these characteristics or malware patterns in an alternative way.

automated environments without leaking the know-how of malware. We present the arguments on detection in Section 7.4.1 and Section 7.4.7.

7.2.2.4 Access to system calls/APIs

Ideal malware must be able to access kernel functionalities, system calls, and Application Programming Interfaces (APIs) present for low-level operations. Access to the full system is especially necessary for targeted attacks where a victim user's core assets are extracted. Section 7.4.8 presents the arguments on syscalls.

7.2.2.5 The highest possible privileges

The malware must escalate into privilege levels present in a system as high as possible. Most attacks may begin at the lowest level of permissions; however, successful malware must be equipped with the necessary payloads to exploit the rest of the system. Section 7.4.5 and Section 7.4.9 present the arguments on malware privileges.

7.2.2.6 Unrestricted resources and assets

Ideally, malware must be able to use all system resources without restrictions. Resources may refer to all devices/components other than computational abilities and full memory access. These resources may also contain the target user assets and valuable information. We present the arguments on resources in Section 7.4.10 and on user assets in Section 7.4.4.

7.2.2.7 Infect more targets, audience and availability

Ideally, malware must aim to infect more victims in a campaign. Infection rate can be a success measure for malware. A malicious attack targeting one specific system might fall into the APT category. For compatibility, the target system must fulfil the requirements of the malware at infection time. Further, if the malware has specific dependencies to operate, these dependencies must continue to be available. We present the arguments on target victims in Section 7.4.11 and malware availability in Section 7.4.3.

7.2.2.8 Maximise revenue/profit and long-term maintenance

The goal of malware should be to increase its profit above all. The monetisation of a campaign might be a way to see the profit, but the overall damage or the information extracted can also be of profit to the malware authors. The malware would require maintenance for long terms jobs or while reselling or in case of an ownership transfer. Section 7.4.6 and Section 7.4.12 present arguments on malware maintenance.

7.2.3 Focused TEE: The SGX Ecosystem

TEE provide hardware-assisted new security capabilities as specified in Global Platform documentations[3]. We focus on Intel's Software Guard eXtensions, widely available for commodity systems today in the market. SGX-enabled hardware products allow applications to be executed in a protected memory region called an enclave. Enclaves aim to protect sensitive workloads, process secret information and provide strong isolation for applications. To examine better whether enclaves can fundamentally help to cloak malware in contrast to its design goals, we briefly describe the SGX ecosystem. SGX was introduced in new hardware instructions (host + user level[4], and VMM instructions added), and is a hardware-level feature, implemented as xucode[5] in the CPU package. The implementation can be considered more or less as executing on the ring level minus three (-3), and SGX can be configured in the firmware. The memory amount allocated for SGX is predefined and known at the system start. The amount of encrypted memory pages that applications can benefit from depends on the enclave's requirements at run time. We refer readers to more background on Intel SGX provided in Section 2.3 and in Section 3.3.

Intel provides a native Software Development Kit (SDK) for enclave development, which contains trusted libraries (nowadays absolutely required in any enclave) to be imported into enclaves. There are also other SDKs for enclave development, but

[3] `https://globalplatform.org/technical-committees/trusted-execution-environment-tee-committee/`

[4] 13 host + 5 user, but it depends on if we count actual instructions or SGX leaf instructions

[5] Online. `https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/xucode-implementing-complex-instruction-flows.html`

we keep them out of the scope of the core SGX ecosystem. For example, Fortranix Enclave Development Platform (EDP)[6], is one of the most advanced environments to develop secure enclaves, utilising RUST language. Enclave developers are responsible for keeping their development environment malware-free, and only trusted libraries and trusted SDKs must be utilised. Intel's SGX SDK provides privileged architectural enclaves such as Launch Enclave, Quoting Enclave and Platform Service Enclave (LE, QE, PSE) and others. These are assistive enclaves for the custom enclaves developed; they are built with Intel's SGX SDK. Enclaves can have manufacturing, provisioning and attestation life cycles in their production and execution periods. Finally, there are crypto libraries and protocols (*e.g.*, for communication) utilised in enclaves. These points define the SGX ecosystem, where SGX itself can be held responsible for a bad design choice. After all, the custom enclave code is outside of the SGX ecosystem, and outside of a TEE ecosystem. If the custom enclave contains a vulnerable piece of software code programmed by the enclave developer, the security guarantees offered by the hardware may be diminished, and SGX instructions should not be blamed for such cases. The composition of the underlying hardware and the developed software plays a crucial role in secure application development. Some of the important questions:

- Is persistent-malware taking advantage of any of these features in the core SGX ecosystem?

- Does malware gain superior features or become weaker by utilising these features?

- Where is the actual location of malware (*e.g.*, in the microcode, or in developer-defined enclave-code)?

[6]Online. `https://edp.fortanix.com/`

7.3 Existing Difficulties in Malware Detection

Malware detection in a high entropy system has been a difficult challenge for decades. The complexity of an end-user's commodity system and the similar patterns of benign and malicious software make malware detection more difficult. We show example methods of non-SGX malware for malware stealthiness and malware delivery (*i.e.*, malware without enclaves, malware without trusted execution environments) in Section 7.3.1 and Section 7.3.2. These methods continue to achieve successful malware campaigns. Throughout our discussions, we examine whether an enclave-assistance can make a malware campaign more successful, or result in its quicker failure. We explain the misconceptions in Section 7.4. The reason why we show a non-SGX malware is to be able to compare it with a potential SGX-based malware systematically and draw a clear picture of their comparison of whether the existing capabilities increase or decrease.

7.3.1 Case Study: (Non-SGX) Malware Infection in Memory

Malware in a commodity system (*e.g.*, in Windows PC) can escalate its privileges to have full memory access. We present three common techniques for a malicious payload to spread and continue to execute with benign processes. Depending on how sophisticated implementation a malware has, detection might be very difficult. Most of these actions can be seen in legitimate software as well. Thus the malicious behaviour can often bypass the detection mechanisms.

Software-based memory encryption and obfuscation techniques can hide malware in run-time and make reverse engineering difficult. We show three of these commonly used techniques. Due to real-time mutations, *e.g.*, malware polymorphism, old-time static code analysis and detection techniques (signature-based) can fail. Failure of static analysis techniques and the rise of dynamic malware identity motivated the industry to bring promising behaviour-based dynamic analysis techniques, even if the memory is encrypted or obfuscated. Today, anti-malware companies use advanced memory protection and analysis solutions explained in Section 7.1 to fight

back the memory-based evasion techniques. We highlight the following techniques to comprehend the myths in Section 7.4; SGX run-time memory encryption does not give the anti-malware industry a new challenge.

This chapter assumes that users (implementing or using anti-malware solutions) will not opt for old-time static code analysis or signature-based detection. If they rely on signatures only (old-fashioned), some readers may think that enclaves can hide the code with memory encryption. The code loaded into an enclave is initially a static binary and is open for inspection or reverse engineering. The potential threat with enclaves might come at runtime where an enclave fetches dynamic content (potentially malicious); this is where we argue that the threat vector is not new for the anti-malware industry. Section 7.4.1 on memory encryption and Section 7.4.10 on memory access are related to this section.

RunPE: Windows computers utilise portable executable (PE) format. In the RunPE technique, the malware replaces the memory content of a legitimate process (*i.e.*, belonging to a benign system service) with the malicious payload. The payload does not need to be extracted all at once; malware can extract small pieces of malicious content depending on its goals and revert back to the original memory. Linux systems use Executable and Linkable Format (ELF); a similar technique can be called RunELF, where the content of a legitimate process is replaced with a malicious payload at runtime.

PE/ELF injection: Malware (*e.g.*, a C binary) in a commodity system with low-level privileges can access the full content of the memory. Instead of replacing the process memory, malware can also allocate a new memory region, inject its payload (PE or ELF) and align/compute the memory addresses so that the execution can continue. Similar to the RunPE method, the payload extraction happens at runtime and does not utilise the persistent storage.

DLL/SO injection: Alternatively, malware can prepare a modified library, Dynamically Linked Library (DLL) or Shared Object (SO), containing the malicious functions ready in the persistent storage to be called. Often in commodity systems,

even if the system had measured boot, the libraries in the disk can be replaced by different versions highly likely containing malicious payloads.

7.3.2 Case Study: (Non-SGX) Drive-by Malware Distribution

Besides the memory operations for stealthiness, a malware can be capable of escalating to higher privileges. It is crucial to examine how it can spread in the first instance and escalate its privileges. We implemented a scalable malware delivery campaign for this case study and reported it with responsible disclosure steps. Note that it is no longer possible to follow the exact steps as (1) Facebook does not allow the inclusion of third-party video players anymore, and (2) Adobe no longer supports Flash Player since the end of 2020[7]. No real accounts were used in the proof-of-concept (PoC) experiment, only the test accounts generated by Facebook for bug bounty purposes were used.

The following PoC is an example of the malware in wild; it gives us tangible facts to establish a comparison with potential TEE-assisted malware. We discuss more details of the enclave-assisted malware and the malware in wild in Section 7.5.1. Section 7.4.3 on malware delivery and Section 7.4.11 on target victims are related to this experiment.

Scalable drive-by malware delivery campaign: At the time of our experiment, Facebook was allowing the embedding of externally hosted ShockWave Flash (SWF) video players on users' walls. We developed a custom video player in ActionScript 3.0, streaming legitimate content, hosted on our HTTPS-enabled external web server. Having a TLS connection on the host server was the only condition by Facebook to cache the custom video player and stream the videos on users' walls inline. Once the source link has been distributed through a high number of shares (*e.g.*, to millions of users), we updated the initially benign video player containing *known* exploits (for OS X, Linux, Windows) available in the public

[7]Although flash technology is no longer available, we anticipate that a similar set of issues can be seen with WebAssembly applications and other client-side containers.

space[8]. Although it is expected for most of the target users to have up-to-date (non-vulnerable) versions of flash and sand-boxed browsers, there are a sufficient number of users with vulnerable software.

We demonstrated a proof-of-concept video to the Facebook security team, triggering other existing vulnerabilities in the system (*e.g.*, enabling a drive-by Java downloader, triggering a local media player's vulnerabilities, calling system applications and accessing kernel functions). To infect a system[9], it is sufficient for victim users to watch a video on their Facebook feed. Initially, the Facebook security team responded that (1) the nature of the platform is to allow third-party content to be circulated (2) their internal whitelisting/blacklisting mechanism can block the source link as soon as it is detected. Later, (and currently) the functionality of embedding third-party video players is completely removed. Crucially, we did not need any unknown zero-day vulnerabilities. As Facebook provided the highly scalable distribution network, we were able to demonstrate this attack with known exploits that were not patched by all users. Having known exploit payloads in SWF video players also proved to us that Facebook's internal scanning mechanisms before caching any external resource were inadequate. Finally, although the inline video embedding/playing feature is removed, the danger of distributing malware in social media continues. It is always possible to change the content of an external link once it has been shared millions of times. The worst part is that media contents can be changed depending on user location and personal views (*e.g.*, political), giving an extended ground for manipulation. Once a user's computer is infected, all of the content in a social media account can be tampered with locally at the client side through malware. Associating users' true identities with infected machines gives attackers a highly valuable asset in malware reselling.

[8]Known CVEs, existing exploit where the patch is available. The development details of the exploit itself are out of scope for our chapter.

[9]A similar simplicity of infection can also be seen in an IoT deployment; ELB can cause infection once the lights are turned on.

7.4 Misconceptions about enclave assisted malware

Although the public launch of SGX technology was in 2015, SGX-assisted malware discussions within the research and security community go back to 2013 [218, 219] at least[10]. Our goal in this section is to stretch these claims to study and understand them correctly; this may result in some of the myths being read as hand-waving. We aim to give readers an understanding of how dangerous SGX-assisted malware can be and how future TEE designers can benefit from our systematisation in order to avoid TEE features being abused by bad actors to strengthen their malware. Besides our work, Symantec has also published [222] a written statement that claims behind SGX malware may not be as bad as it is *believed*, and they may remain as myths only. Here in our systematisation, we present them as myths or misconceptions interchangeably. In accumulation and conciseness of these claims, we benefited from past SGX Community Day discussions and presentations [11].

The structure we create may serve as a template for future discussions, and may evolve with support or criticism from the community. In this section, we structure[11] the claims about malware utilising enclaves especially crafted for SGX-enabled trusted hardware. Similar characteristics and claims are valid for ELB utilising trusted hardware capabilities to attack IoT deployments internally. The claims and arguments behind these myths are derived from ITL resources[12] [13], industry/practitioners conferences[14] [221], industrial vendor resources [15], and are compiled from Intel's SGX Community Days discussions and presentations[16].

[10]Researchers may be able to trace TEE-based malware discussions considering older technologies with DRTM, TXT or similar.

[11]Through SGX research community days, community's written statements, and other references found in this document.

[12]`https://theinvisiblethings.blogspot.com/2013/08/thoughts-on-intels-upcoming-software.html`

[13]`https://theinvisiblethings.blogspot.com/2013/09/thoughts-on-intels-upcoming-software.html`

[14]`https://www.zdnet.com/article/researchers-hide-malware-in-intel-sgx-enclaves/`

[15]`https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/sgx-malware-explainer`

[16]`https://www.intel.com/content/dam/www/public/us/en/documents/research/kubila`

7.4.1 Will Enclave's memory encryption hide the malware?

Relates to the FUD characteristic of malware in Section 7.2.2.3. Section 7.4.7 presents further arguments on malware detection.

Supporting Arguments: SGX provides a memory encryption engine for trusted applications. The content of the memory pages of enclaves is not visible at runtime. For example, the memory of enclaves cannot be inspected directly. Comparing it to an untrusted part of the memory, enclaves can help to hide the malware (*e.g.*, malicious payload or malicious behaviour).

Counter Arguments: Enclave binaries are inspectable in the disk, and anyone can dump the initial content of the enclave, as shown in [6] and in Section 5.3. The memory locations of enclaves are the most visible part of the system. The network traffic into the enclaves is visible; although traffic is encrypted, most of the network analysis techniques still apply. All of the disk operations, input and output operations on main memory, the interface operations with the untrusted part, usage patterns and CPU utilisation of the enclaves are visible. These points make enclave-based malware even more visible than malware in the wild. Hardware-assisted encryption instead of software-based encryption may give stronger security guarantees for malware; nevertheless, the actual detection of malware is only possible through the behavioural patterns. In other words, detection mechanisms do not necessarily require decryption of the malware. Tracking malware in a complex and noisy environment is more difficult than tracking the behaviour of an enclave which is entirely dependent on resources managed by the OS.

Reference and Rebuttal: [224] states that enclave can hide the malware, they conclude that malware can benefit from SGX. In fact, they only download static plain-text data into an enclave, with no proof of execution or malicious activity where SGX directly contributes. They ignore multiple facts how the initial enclave was launched, where did this enclave connect or who supervised it.

---

y-kucuk-malware-infeasibility-sgx.pdf

7.4.2 Can Enclaves generate encryption keys for each malware payload?

These arguments relate to malware key management requirements in Section 7.2.2.2. We discuss the scalability and ransomware aspects of key management in Section 7.4.4.

Supporting Arguments: Enclaves can generate private keys inside encrypted memory regions. These keys never leave the private memory. This can help malware to maintain its key generation and key storage problems. Further, malware can generate unique keys for each victim.

Counter Arguments: The instruction used for key generation `EGETKEY` is bound to the enclave binary identity. Enclave is measured at load time once, and any payload that the enclave fetches later does not change the enclave identity. The source code of the initial binary is inspectable on the disk. Enclave ID, therefore, enclave-based derived keys are based on initial binary (object code can be dumped). The background on Quoting Enclave assisting on signing an enclave report and identity was given in Section 2.3. Generated sealing keys can be derived from initial enclave measurement and can be derived from root sealing key as well. An attacker can produce unique keys for the victims, bound to hardware and software. But, generating independent keys in an enclave is similar to generating them at a remote location. The key point is that the dynamically fetched payload is not part of the key generation through SGX, it is done through already existing manual efforts. Overall, this does not provide the malware author with anything new or any superior feature. The main enclave identity remains the same as the initial binary. Nevertheless, SGX's newer Key Sharing and Separation (KSS) feature must be examined separately. Although KSS does not change the enclave ID for attestation purposes, optionally, KSS can include the identity of the newly fetched payload for sealing key generation. This may give a unique key based on the payload.

Reference and Rebuttal: [225] shows that attackers can generate public-private key pair inside an enclave and attackers can use them for encrypting user files with the public key outside of the enclave. The paper makes many assumptions

far from reality that all conditions must be set for a successful attack. Further, it tries to solve a problem that does not exist anymore; ransomware gangs changed how they carry out their attacks. The use of an enclave for key generation makes the job of attackers more difficult.

7.4.3 Will Enclaves secretly deliver malware?

These arguments relate to communication aspects in Section 7.2.2.2 and availability requirements in Section 7.2.2.7. Section 7.4.6 presents communication of malware towards establishing secure channels.

Supporting Arguments: Enclaves can fetch secret payloads into the private memory regions at runtime. This may allow enclaves to be a point of malware distribution. Enclaves might be used as stateless malware carrying embassies.

Counter Arguments: Enclaves do not receive the payloads out of anywhere arbitrary. The network patterns and their behaviour can be observed. Further, enclaves are not always available in a system. An enclave's responsiveness (*i.e.*, opportunity to real-time respond to an attacker) is relatively low compared to the system's other services. If an enclave allows malicious payload at runtime, it must utilise public key infrastructure (PKI) to ensure that only the attacker can fetch a payload. Otherwise, malware reverse engineers can also inject healing code inside an enclave and extract its private keys. Overall, attackers utilise classical PKI techniques for malware delivery as usual, and do not profit directly from SGX's features. We explained the past use of PKI, for potentially malicious or benign purposes in Section 6.2.3.2 and in [10]. SGX enclaves are just another point they might choose for malware delivery, but their low availability makes them a bad choice for high-scale infection campaigns. Launch Enclave (LE) is a special enclave to account for malware in an enclave. It can be used to determine whether the enclave may be launched on the platform, utilising launch tokens and policies.

Reference and Rebuttal: [224] states that stalling mechanism can delay the malware behaviour and help to delivery. Time measurement inside an enclave is difficult, they develop a mechanism to measure the time, however, the paper does

not show any solid proofs for malware delivery. They ignore the actual infection point in practice and jump to the behavioural detection discussion.

7.4.4 Will Enclaves scale and ease the ransomware operations and key management?

The following arguments relate to key management in Section 7.2.2.2 and access to user assets in Section 7.2.2.6.

Supporting Arguments: Ransomware key management (generating and storing the keys, or regenerating the keys at a later stage) can be easily achieved within enclaves. Every victim can have a unique key based on their machine identity. Similar to any digital rights management software designed for enclaves, ransomware can utilise the same features.

Counter Arguments: Using enclaves for a ransomware leaves more behaviour patterns in a system. Ransomware must still copy the data into enclaves if files will be used for encryption purposes. If the enclave generates the keys and gives the secret keys to the untrusted world for encryption, this would be considered a design failure for the ransomware. Either the encryption can take place outside the enclave with the released public key or inside the enclave. Outside encryption is susceptible to existing detection mechanisms. Technically, carrying all of the user's data into the malicious enclave is not a feasible operation. The reason is; data transfer is dependent on outside operations requiring multiple assumptions. Furthermore, enclave-based key management adds an extra step to the current ransomware key management mechanisms. We refer readers to the local and remote interactions of our enclaves in providing remote services in Section 4.5.6. The healthiest way to maintain ransomware keys still remains as remote resources. Additionally, a ransomware must encrypt a considerable amount of data over time without being detected, making the existing noise level of the untrusted world essentially a better choice for malware authors.

Reference and Rebuttal: [225]'s threat model starts with the assumptions that malware delivery and installations are already done, system has no anti-

virus and no anti-ransomware mechanisms, they only consider key capturing mechanisms [226] which attackers already solved since five years. They argue that block-chain based mechanisms will automate the ransom payments. In overall, their ransomware is supported more by their assumptions and block-chain based methods, there is no evidence that use of enclaves resulted a stronger ransomware.

7.4.5 Can Enclave-assisted malware be persistent in the system?

These arguments relate to persistence in Section 7.2.2.1 and malware privileges in Section 7.2.2.5. Section 7.4.9 presents the arguments on privileges at the custom enclave code (ring 3) and privileges of the micro-code level (ring -3).

Supporting Arguments: SGX-enabled machines can start-up an enclave whenever it is triggered. This can allow private operations to be completed on demand. Malware can live in a system as persistent through enclaves. Lower-level hardware assistance can enable persistent rootkits in a system.

Counter Arguments: Enclaves themselves are not persistent pieces of software applications. They have a life cycle of `ECREATE` where the enclave is created and `EDESTROY` to kill the enclave. Such operations were explained and measured in [2] and in Section 4.4. Enclaves operate at the user level (ring 3) and although their execution is supported by the low-level hardware features, they do not operate at any lower privilege levels in the system. Enclave-based malware cannot be persistent on its own. Outside system support for persistence would create an extra burden. If a malware can be persistent in the untrusted world, trying to utilise an enclave may only make it less persistent.

Reference and Rebuttal: [227] states that their malicious operations can reach persistence through SGX enclaves. They assume custom enclave code to be vulnerable, not exploiting anything in the core SGX ecosystem. They claim persistence, but it is completely dependent on user-level host enclave, in reality this offers a very weak persistence.

7.4.6 Can a malware inside an enclave communicate independently?

These arguments relate to the communication requirement in Section 7.2.2.2 and the malware control in Section 7.2.2.8. Section 7.4.12 presents further arguments on malware maintenance with ownership transfer.

Supporting Arguments: Malicious code executing inside a private memory can communicate with the outside world through HTTP calls and secure channels. This can help to leak information from the user's system.

Counter Arguments: For communication, enclaves rely on untrusted channels. Although enclave-based malware can send arbitrary data to the outside world, the classical method applies before malware fetches or copies any private data from the disk or memory. This is not a new issue; most of the applications in a commodity computer can steal users' private files at any moment. Enclaves, in fact, have more dependencies for their communication with the outside world. The technical dependencies and libraries an enclave minimally may need is presented in [2] and in Section 4.6.1. An alternative but larger option can be library OSes, *e.g.*,with interpreter enclaves as explained in [10] and in Section 5.5. They can also leave clear traces of their communication pattern. Monitoring the whole system's (host) communication is more demanding than monitoring an enclave.

Reference and Rebuttal: [224] performs attestation with a remote server and fetch a payload from outside. They ignore the fact that all communication channels were dependent on untrusted channels where in reality they would not be able to freely communicate with any malicious server.

7.4.7 Can TEE based malware be FUD (Fully Un-Detected)?

These arguments relate to malware evasion in Section 7.2.2.3.

Supporting Arguments: Antivirus software products cannot directly perform a scan on the enclaves' memory. Enclaves can keep malicious payloads away from scanners. This can cause a shortcut in achieving FUD malware.

Counter Arguments: An encrypted piece of the non-executing payload does not harm the system. Enclave software or non-enclave software can both have pre-encrypted payloads. As soon as a piece of code starts to execute, the same malware detection mechanisms apply. A malware inside an enclave must extract instructions to an untrusted world or perform operations in the main memory in order to reach valuable user assets (or towards its campaign goals). Existing research and the techniques on malware detection apply for malware behaviour in the untrusted world.

Reference and Rebuttal: [228] states that their malware in an SGX enclave remains **completely/entirely invisible** to anti-virus software and even to ring 0. The argument is ambigious and expected as their attack does not leave the SGX ecosystem nor enclaves, it crosses the boundaries of enclave isolation through caches. The attack can be successful only with the specific requirements and assumptions. In reality, being invisible to the antivirus mechanisms means that the malware remains invisible while being actively inspected and being in operation against the full system resources or assets. If their malware attacked to the actual system, it would be then visible; their invisibility is relying upon having no attack to the kernel or similar components, hence the kernel does not see anything.

7.4.8 Will SGX based malware access System APIs?

These arguments relate to system calls capabilities in Section 7.2.2.4.

Supporting Arguments: Malware can reach system calls, WinAPIs and kernel functionalities from the enclave. Accessing ring 0 level resources can boost the capabilities of the malware. Within the enclave memory, malware can remain hidden and still perform a scan for vulnerabilities in the system to reach the kernel.

Counter Arguments: Enclaves operate at user level privileges (ring 3), a brief background on Intel SGX can be found in Section 3.3. In order to escalate into higher privileges or to access any system calls, the malware must go through the untrusted parts of applications, or perform operations on the main memory [144]. The malicious behaviour from the enclave towards the system can be monitored and restricted. Depending on the programming model and the software development

kit, the enclave may have no access to the system at all. The enclave may be given a LibOS to utilise dependencies internally without needing to make calls to the outside system.

Reference: [222] answers the questions on what SGX malware may need to do in order to access system APIs. The arguments remain valid.

7.4.9 Will malware have the highest privileges through SGX?

These arguments relate to malware privileges in Section 7.2.2.5.

Supporting Arguments: SGX hardware is a microcode implementation and has the highest privileges in a system. A malware utilising enclaves can benefit from these high privileges. If malware operates at ring level minus three, no detection mechanism can catch its malicious activities.

Counter Arguments: Having a malware inside the core SGX ecosystem is not the same as having a malware in a custom developed enclave. Enclaves operate at the user level, and their abilities are supported by the extended CPU instructions. If a malware exploits CPU-level bugs on SGX, it might have the highest privileges. Otherwise, malware in an enclave can only benefit from the user-level privileges.

7.4.10 Will enclave assistance give malware full memory access?

These arguments relate to malware resources in Section 7.2.2.6.

Supporting Arguments: Untrusted applications cannot see the memory of trusted enclave applications. But enclaves can see the memory of the all main memory content. This gives a malware inside an enclave a superior power.

Counter Arguments: Untrusted applications can already see the full main memory content. The ability of enclaves to see the full memory is not a new feature on top of any other usual application. If an enclave performs operations on the main memory, the input/output traffic is visible to the operating system.

7.4.11 Will TEEs help malware to target more victims?

These arguments relate to malware delivery in Section 7.2.2.7.

Supporting Arguments: SGX-enabled hardware products are standardised in Intel CPUs. Since the end of 2015, most of the new laptops, desktops, and other machines have SGX bit in their firmware. Users may not be well informed about SGX features and it might be left default enabled. The applications utilising enclaves can also drop malware in users' machines. Having SGX enabled computers can increase the number of victims infected by malware.

Counter Arguments: From a malware author's point of view, non-SGX malware can target more users than a malware tailored for SGX. A malware targeting the general audience (both SGX-enabled and non-SGX machines) can infect a higher number of victims. SGX may be unavailable at any moment; therefore, relying on SGX being enabled is a poor design choice for malware. The host can kill an enclave with suspicious behaviour (executing for an unnecessarily long time, memory operations, suspicious network traffic) from outside (other than the internal destruction function). If a malware requires an enclave to be restarted in order to be in operation, this will make it a very ineffective malware. The malware must be independent as much as possible; utilising enclaves give malware a sizeable hump to carry at all times. Intel SGX is no longer available in consumer devices, unfortunately. Only servers have Intel SGX at the moment.

7.4.12 Is malware inside an enclave easier to maintain?

These arguments relate to malware maintenance in Section 7.2.2.8.

Supporting Arguments: Enclaves aim to minimise the trusted computing base. They contain a relatively small amount of code to make formal verification feasible. Enclaves can fetch a new malware payload whenever necessary and act as an update mechanism for malware. This can make malware maintenance cheap and more sustainable.

Table 7.1: Malware in the wild (untrusted high noise system, non-SGX) in comparison to malware in an enclave utilising SGX features. ⊕ Enclave enhances/strengthens the malware. ⊘ Enclave has no impact. ⊖ Enclave weakens the malware.

Characteristics in Misconceptions	**Malware in Enclave (ME)**	**Malware/ELB in wild**	**Conclusion** (⊕, ⊘, ⊖): and Reason
7.2.2.1 **Persistence** in 7.4.5	Needs to trigger	Services, drivers	⊖: ME is less persistent.
7.2.2.2 **Communication** in 7.4.3, 7.4.6	Needs untrusted channels	In high noise	⊖: ME has longer path to communicate.
Key management in 7.4.2 7.4.4	Extra steps for payload	Existing PKI	⊖: ME has more crypto operations.
7.2.2.3 **Detection** in 7.4.1, 7.4.7	Visible resources and I/O	Existing methods	⊖: ME is more visible by resources.
7.2.2.4 **System calls** in 7.4.8	Needs untrusted assistance	Direct access	⊖: ME has longer path to syscalls.
7.2.2.5 **Privileges** in 7.4.5, 7.4.9	User level/ring 3	Kernel/ring 0	⊖: ME is less privileged.
7.2.2.6 **Resources** in 7.4.10	Limited/restricted	Unlimited	⊖: ME has access to less resources.
Access to user assets in 7.4.4	Indirectly	Direct Access	⊖: ME has longer path to user assets.
7.2.2.7 **Audience** in 7.4.11	CPU-specific	Independence	⊖: ME has less number of targets.
Availability in 7.4.3	Low availability	Always On	⊖: ME is less available.
7.2.2.8 **Maintenance** in 7.4.6, 7.4.12	Specific dependencies	Easy-to-resell	⊖: ME is more expensive to maintain.

Counter Arguments: Enclave development has many dependencies and it is an evolving ecosystem[17]. A malware designed for enclaves would actually require additional engineering work to be always compatible. Additional compatibility issues can cause malware to fail to operate in a high number of systems. Crafting a malware for SGX brings additional development costs, maintenance requirements, and lowers the number of target machines to infect or continue to operate on.

7.5 The Limitations of SGX-Malware

An SGX-based malware is not impossible to produce, but in practice, such malware would become weaker as explained in Section 7.4. Further, we now compare SGX-based malware with a malware in the wild for their characteristics listed in Section 7.2.2. Now in this section, we present systematic limitations (or disadvantages[18] for the malware developers/bad actors) of utilising a TEE for developing malware. ELB in IoT deployment can deploy attacks against user assets similar to wild malware in commodity computers. We argue that using TEE in the development of malicious IoT devices (in the domain of ELB) will restrict their capabilities for the reasons listed in the following section.

[17] `https://download.01.org/intel-sgx/latest/linux-latest/docs/Intel_SGX_SDK_Release_Notes_Linux_2.16_Open_Source.pdf`

[18] All malware has disadvantages to anyone unfortunate to be infected with it, but we refer here to disadvantages for a successful malware campaign carried out by malware authors.

7.5.1 Enclave-Assisted Malware vs. Malware in the Wild

Table 7.1 summarises the malware characteristics examined within the misconceptions presented in Section 7.4. An application from an outside, untrusted world must trigger the enclave-based malware (*i.e.*, the enclave binary portion). Malware can only communicate with the outside world through untrusted channels. The behaviour patterns of input and output streams are visible to detection mechanisms. For system APIs, calls, lower-level support, an enclave needs assistance from the untrusted world. Malware in an enclave runs at the user level privileges (ring 3 in monolithic systems). System resources such as memory, threading, timers and CPU features are restricted and limited for enclaves. The audience is limited to SGX-enabled (or TEE enabled) machines only. Key management requires extra steps to manage the payload with PKI as usual, and enclave identity does not change based on the payload. Enclave availability is lower than other system services; enclaves are not in constant or persistent operation. Enclaves cannot directly access user assets, but must indirectly utilise outside resources for completing an attack. Malware in the enclave requires specific dependencies in development and makes updates and reselling more challenging.

7.5.2 Can SGX boost any characteristics of the malware?

A wild malware in an untrusted system can place various attacks against enclaves. In contrast, enclaves are limited and continue to be more limited in performing attacks on other enclaves and towards the untrusted world. The design goal of enclaves is to protect the critical operations from wild malware. Utilising an enclave for a malware weakens its abilities dramatically.

7.6 Discussion on Malware and Trusted Execution Environment

TEE are introduced to limit the existing abilities of wild malware. Moving more applications into the TEE domain will eventually leave wild malware outside the

trusted ecosystem. Crucially, TCB minimisation plays a key role in leaving malicious software outside of the trusted applications. Including a fully capable operating system together with the user assets inside an enclave may simply recreate the untrusted world and its problems again.

7.6.1 Zero-day SGX vulnerabilities in Malware as a Service (MaaS)

SGX and TEE raised awareness of micro-architectural attacks. Due to microcode weaknesses, an advanced malware can temporarily take advantage of vulnerable systems. These zero-day vulnerabilities may be equipped in APTs. However, TCB updates on microcode and revocation mechanisms make such attacks infeasible in the long term [217].

To implement an advanced malware with SGX, attackers must utilise the vulnerabilities in the core SGX ecosystem defined in Section 7.2.3. Intel can prevent such attacks by TCB updates. If not patched, bad actors may sell the security problems of the core SGX ecosystem in exploit markets for MaaS.

7.6.2 Potential malware planted inside SGX ecosystem

In the unlikely event of a core SGX ecosystem including critical vulnerabilities, a sophisticated malware may be planted into benign enclaves (*e.g.*, during compiling time, in a development environment, in SDKs, or trusted libraries). Due to the increased use of hardware enclaves (*i.e.*, moving of critical user assets into enclaves), attackers may target the core ecosystem more. Future questions can be around malware in the core SGX ecosystem, not in a custom enclave but on a microcode level.

- Are more attacks possible on micro-architectural level towards malware deployment, similar to attacks in ring minus two or three (ring -2, ring -3) [229–236], System Management Mode (SMM)?
- Can malware authors infect/exploit the architectural enclaves with no one noticing?

After all, every software brings an attack vector; however, breaking the enclave's security may have a high impact as more critical assets move into enclaves.

7.6.3 Malware capabilities in wild, without a TEE

In contrast to SGX-based malware, a typical malware in wild, commodity and high noise systems, can utilise all of the system resources such as registry configurations, services, triggered events, drivers and other hardware devices for spreading and persistence. Wild malware can hide its communication patterns in high noise, and distinguishing what belongs to malware behaviour remains an open challenge. Malware detection techniques with memory analyses apply, but continue to be a challenge as wild malware can abuse any software package in an untrusted world. Malicious applications in an untrusted world can often access system calls directly as they are available to use and not distinguishable from benign use. Most of the benign software already operates at the kernel level and malware in the wild continues to abuse any of the existing primitives. System resources accessing main memory, and other CPU features are not limited for a wild malware. Enclave independent malware can target any victim with a commodity computer (*e.g.*, x86). Malware authors utilise PKI and other crypto techniques in place in order to manage the master keys of their malware. Availability of malware in the wild is up as long as the system operates as normal. Malware in the wild operates in the same environment where user assets are placed, often giving direct access to valuable information. Maintenance, updating, and reselling a malware in the wild can be straightforward by pulling another payload from a new malware author, or simply via passing/delegating keys to a buyer in a remote environment.

Malware with or without SGX can be detected with similar capabilities. For example, in a cloud server, bad actors have the freedom to deploy malicious operations as a client of a cloud company. Company admins may watch the network traffic for suspicious and criminal activities or due to complaints. Normally, the attacker or the server tenant is free to run any software in the rented machine. Using or not using SGX does not bring any difference to this model. Alternatively,

attackers may want to utilise enclaves for private operations such as domain name generation (to be used in a campaign), or for planning criminal activities. Private operations, however, require an active connection and persistent communication to serve. These are similar possibilities to be handled in remote servers and enclave utilisation does not bring any immediate benefit to these use cases.

Contributions

This study presents the most comprehensive collection of misconceptions about malware in enclaves to the date.

(1) Our novel contribution is that we systematically show why malware becomes weaker due to trusted hardware.

(2) We also clarify the ambiguity of *malware in an SGX-enclave* and the *malware in the core SGX ecosystem*.

Chapter Recap

We structured this chapter into four main categories. First, we describe the good-to-have characteristics of an ideal malware. Second, we demonstrate existing malware evasion techniques and high-scale, effortless malware distribution techniques. We show non-TEE malware evasion techniques and a delivery campaign for two reasons; (1) to define the assumptions and review the scope of malware detection, (2) to demonstrate a real-world scenario on scalable infection to give readers an understanding of malware in the wild. Third, we systematically evaluate twelve misconceptions on malware in TEE and present why these myths are far from the truth in practice. Finally, we compare the malware in the wild with enclave-based malware and see if utilising enclaves provides any additional benefits to malware in practice.

Part V

Conclusions and Future Directions

Chapter 8

Thesis Conclusions

Contents

The term *trust* does not imply good or bad behaviour. Neutrality of trusted computing primitives gave our research a fundamental challenge. A malicious or a benign application may equally utilise available security features towards their end-goal. In this research we created a novel computing model to strengthen mission-critical applications against malware whereas a malware cannot be strengthened with the same model. This thesis as whole described our computing model with the following parts: Part II: protecting private data processed by public algorithms in scalable systems, Part III: protecting private algorithms along the private data

in multi stakeholder scenarios, Part IV: systematisation with the computing model and the malware aspects.

We demonstrated in Part II and Part III practical uses of *Confidential Remote Computing* with high number of participants where applications in an attestable infrastructure can scale to tens of thousands of participants per hour. We described our practical implementations through the Section 4.5 in Chapter 4 and the Section 5.8 in Chapter 5. By utilising the hardware instructions, CRC applications can protect users' data, preserve their privacy and enable running secret algorithms in remote computers. In CRC model, we presented how enclave developers can correctly translate hardware security features into software domain and bring distrustful asset owners into an agreement. Although it seemed paradoxical prior to our research, we showed the research community why fundamentally the novel CRC model will not enhance malicious applications.

In the following sections, we present our detailed conclusions.

8.1 Conclusion on Verifiable Third Party

The SGX-based TRE addressed a stronger attacker model than the one described for TPM-based TRE. Using Intel SGX as the basis for a TRE made several advantages. Specifically, removing several components from TCB and providing the necessary TRE functionalities with our architecture reduced the TCB size even further than the previous TPM implementation. A smaller TCB is not destined when switching from TPM to SGX for a system. These two hardware solutions offer completely different fundamental instructions. A system with another hardware requires rearchitecturing and redesinging with the available primitives. In fact, many solutions and related work shown in Section 5.5.4 and Section 6.5.3 using SGX has reached to x5 to x10 times larger TCB size due to their architectural choices of including larger OS libraries. The smaller TCB size is likely to have fewer vulnerabilities and makes it easier to audit or perform formal verification. In other trusted hardware, the *Software TCB* size may not have impact on performance, however SGX provides secure execution in encrypted memory pages therefore smaller TCB size is important

also for reducing the cost of execution. We show the increasing execution cost due to TCB size due to binary size, stack and heap memory allocations per enclave life cycle and critical operations in Section 4.4, Section 4.4.1 and in Table 4.2, and we present it as a new insight. The host Application is an intermediate point between all message exchanges. An adversary may have physical access to the platform. Nevertheless, SGX's hardware based isolation mechanism and hardware secured attestation instruction provides services to build sufficient confidentiality and integrity against addressed adversaries. We implemented the TRE prototype in a single enclave. SGX supports running enclaves concurrently, therefore different enclave structures can be explored in future work.

The overall performance of non-optimized TRE solution on SGX hardware shown in Section 4.6.4and Section 4.6.2 is nearly same as TPM-based optimized TRE solution shown in Section 3.6.3.2. Therefore, SGX has great potential as a trust brokerage between distrustful parties. We expect that optimizations of the design and implementation can provide further improvements and allow the SGX-based TRE to scale to even larger many-party applications.

In terms of cost of development, the TRE has two parts; one-time development cost of TRE and placing application specific code into TRE and application specific part of TRE. If an ISV prefers to use an existing TRE model and implementation, rest of deployment cost can be even negligible. Therefore, TEE solution has less cost than secure multiparty computation and data de-identification solutions.

Any application specific code and computation can be placed into the related section of an existing TRE.

Once a TRE is implemented, there is no need for a new design per data or new design per application. Only deployment cost would be adapting an application into the TRE's model which does not require any challenging knowledge and can be considered same as usual software development process.

Overall, our results show that Intel SGX can be used to build a trustworthy remote entity for use in many-party applications. Compared to the previous TPM-based design, our SGX-based TRE has two main advantages. First, it requires

significantly fewer lines of code, which reduces the burden on the developer, decreases the likelihood of code defects, and minimizes the amount of code that must be trusted by a relying party. Secondly, SGX memory protection enables the TRE to resist a stronger adversary, who has physical access to the platform's memory. Our initial SGX-based implementation provides the same performance (and thus scalability) as the highly optimized TPM-based TRE. However, we identified that improving the performance the SGX remote attestation protocol could benefit the SGX-based TRE.

8.2 Conclusions on Private Algorithms

In summary, our work presented a security analysis on interpreter enclaves and has aimed to demonstrate in Section 5.6 and Firgure 5.5 how a third-party commodity software can become the weakest link in a security chain. We provided a new design for secret-code execution in remote computers, and demonstrated such design in three practical generic templates. Our model reduced the TCB size by a power of ten, in comparison to the alternative design approach using interpreters. Approach 1 brings an extra step in development, requiring additional enclave partitioning (public and private parts). In Approach 2, we showed it is difficult to hide the called functions, thus requiring an additional strong sandbox for function secrecy. This sandbox would, ideally, need to call a set of similar functions for each function. It is very likely to have an exponential computational overhead, and this overhead may grow with the size of the interpreter. Despite its shortcomings on usability, our approach provides stronger security guarantees.

8.3 Conclusions & Insights Gained on Confidential Remote Computing

In this section, we provide a list of the lessons learned through *Confidential Remote Computing* model. We also list some of the outstanding questions and aspects of system design requiring more attention.

8.3.1 Lesson I: Underlying insecure system components enabling side-channel attacks

In case of TEEs, many of the software-based side channel attacks become possible due to architectural choices. The key lesson here is that when designing enclaves and TEE applications, manufacturers initially claimed that OS can be fully insecure and enclaves will be still isolated and fully protected from the malicious OS, we learned that this is not possible in practice, and the importance of the underlying insecure system components should not be ignored. Fully adversarial systems can deploy side-channel attacks against the applications running on top of them. Hardening techniques can mitigate side-channel attacks, but in order to avert the risk completely it would be necessary to remove the fundamental conditions inviting them. A system supporting enclave execution can be built with formally verified kernels (*e.g.*, seL4) and utilise measured and attested boot features with TPM. This can ensure that the enclaves operate in benign and trusted systems. Electronic voting systems and IoT gateway systems managing a high number of entities and deploying mission-critical operations can benefit from such secure structures.

8.3.2 Lesson II: Algorithm status and physical location for security

One of the requirements of software-based side-channel attacks targeting the enclave software from an insecure operating system is having the enclave code (algorithm) publicly available for the attacker. Hiding the algorithms through dynamic code loaders at run-time can help reduce leakages from attacks. Nevertheless, keeping algorithms hidden is not sufficient by itself to entirely avert side-channel attacks. The key point to somewhat mitigate attacks is that the algorithms must be hidden from the hardware owner. If a malicious admin or hardware owner has access to the enclave code, this can help them deploy sophisticated attacks more easily. Furthermore, if potentially malicious hardware and algorithm owners collude, they could develop covert channels to signal and leak the data. This collusion could then invalidate the security guarantees offered by the underlying secure hardware.

Therefore, enclave developers must be independent of the infrastructure providers, why we say *must be* independent is because the collusion would ruin the security as we discussed in Section 6.6 and 5.6. Suppose a company (or an entity) claims to provide both enclave software and secure hardware resources. In that case, this may lead to a high risk of data signalling due to explicit collusion, even if the enclave code is presented to or attested by the data owners. As a countermeasure, the whole system must be attested as designed with secure kernels, rather than attesting only the enclave code. Moving the physical trusted execution environment to the data provider's environment and deploying physical network restrictions can help in terms of confidentiality; however, such a system setting would face the scalability challenges of multi-party computation solutions. This has been a major insight because when the TEEs were initially offered in commodity hardware, the researchers and manufacturers simply did not know enough to define the right threat model.

8.3.3 Lesson III: Role of enclaves and verification in secure computing

Enclave technology can help verify remotely that a given piece of code purported to have run indeed did run. In this manner, trusted hardware generates evidence for an attestation process. However, the actual behaviour of the code, whether it matches its expected security properties, is a question for verification technologies. In other words, formal verification technology helps prove that a piece of code does what it is supposed to do. These two steps are ideally both necessary and not here to replace one another. While designing the enclave applications, partitioning can give enclave developers an opportunity to enable verification of the code base. For example, migrating the security-critical parts into the isolated memory can allow this. Design choices, unnecessarily large code base, and poor partitioning would ruin this chance. Interested readers are referred back to the Sections 4.5.5, 5.8, and 6.5.

8.3.4 Lesson IV: Potential issues with a model suggesting to initialise enclaves first in a system

What should we expect to happen if enclaves were initialised independently of the host computer, before the system starts? Parallel to the model of SMM, which inadvertently yielded highest system privileges to malicious actors, if enclaves were to be initialised first, they could be expected to draw similar attacks [237]. Nevertheless, SGX memory for enclaves is in fact allocated by default at the earliest stage of the system start. The firmware knows in advance how much memory will be given for SGX enclaves, but the content of enclaves is not persistent and binaries are loaded later. That is why early initialisation of SGX memory does not cause similar problems as it used to with SMM.

In contrast to its characteristics in SGX enclaves, content in a Trustzone secure world remains persistent. Trustzone applications are initialised first in the system, but they can trigger the system to take control on demand via interrupts, for example to protect their integrity. Trustzone promises to mitigate potential exploits compromising its persistent content via secure TCB updates.

8.3.5 Lesson V: There are now Multiple Root(s) of Trust in a system

A novel use of the TXT/DRTM, as introduced in Flicker [18], evolved into the use case of hardware enclaves with SGX. Multiple measurements in a system are stored in different Platform Configuration Registers. Incorrect use of PCR already has difficulties representing the system state correctly. With SGX hardware implementation and a TPM in a system, an additional problem arises alongside that of wrong use of PCR values. This development places the important new responsibility to securely combine Root of Trust for Measurement on system designers and security architects. Use of SGX in production without merging its Root of Trust for Measurement with that of TPM-enabled systems threatens to fork the Root of Trust for Measurement concept into two independent measurement mechanisms. This can be avoided via using the same trusted firmware on the

bottom in future hardware designs. For example, when ARM or RISC-V introduces enclaves similar to SGX for user level applications, they may use the same *trusted firmware* to manage the enclaves and the Trust-Zone.

8.3.6 Lesson VI: TCB minimisation should not be neglected

Researchers put library operating systems with hundreds of thousands of lines of code inside enclaves. This creates a large and insecure software stack. To avoid such an outcome, enclaves must be kept as minimal as possible. LibOSes can boost functionalities of enclaves, but sacrifice the security guarantees derived from trusted hardware. In-enclave OS support introduces the same security risks of an OS in the untrusted world. The TCB will be potentially vulnerable to attacks unless the new LibOS implementations give formal security guarantees. Otherwise, enclaves with LibOSes may not be able to patch their systems, which are generally insecure due to being built upon a weak kernel ring structure (Ring 0 and 3). It would be preferable to keep enclaves clean from the clutter of unnecessary, untrusted code.

8.4 Conclusions about Malware in Enclaves

We have examined why malware in a trusted execution environment will become much weaker than operating in a wild, high noise system. We considered *frequently seen characteristics* of an ideal malware. With the use of TEE, these characteristics are either still the same as any typical malware in a system, or they are more restricted than before. In the case studies, we revisited how a fully undetected and scalable malware infection campaign can already be in place, without any assistance of trusted hardware. Malware distribution via social media platforms remains crucial compared to malware delivery through enclaves. Section 7.4 showed the misconceptions about how enclaves operate and why relying on enclaves for adversarial purposes is a poor design choice. In contrast, enclaves continue to be a powerful mechanism for protecting highly valuable user assets against malware. Finally, in the near future, we may see more enclave-based malware, but they will

be practically weaker than a malware in high entropy, and they will not be more superior than the malware samples in untrusted environments. Known techniques and methods must be applied to eliminate these attempts. We highlight that the communication and key management characteristics of the malware are seen in four myths in this thesis. We conclude that for these characteristics, the use of TEE makes the malware weaker in commodity systems. Systematisation, categorisation and the definition of the myths can be done in many ways. In future work, the myths and the characteristics can be extended and studied in more depth to see whether the arguments hold for future TEE or not. Considering the new hardware prototypes supporting enclave-like isolated containers, our structured discussions on malware can help mitigate potential TEE abuse at the design stage.

Chapter 9

Future Research

Contents

9.1 Future Work

In the upcoming years, we anticipate that consumer devices (ARM/RISC-V) will be equipped with new user-level realms (similar to enclaves). These devices' wide availability, mobility and decentralised setting can enable a new era of inter-connected, remote computations. Thanks to the rich commodity software stack and libraries available for ARM architecture today, system designers and developers may offer Confidential Remote Computing applications for consumer devices. We encourage developers to work on solutions from e-voting systems to decentralised financial applications and data or algorithm rental services independent of centralised third-party servers. Towards a more decentralised setting for computations, we may see legitimate **digital embassies** of individuals running on devices of other consumers.

9.1.1 Configuration Security, Modularity and Composition

In complex systems, the configuration of systems with modular structures remains an open research challenge. Security properties of sub-modules alone cannot serve as a representation of the properties of the entire composed system. Formally verified kernels may be difficult to keep up to date in the frequently-changing (or update-receiving) world of general-purpose systems. On the other hand, relatively smaller industrial-embedded systems and IoT devices can benefit from formal guarantees.

9.1.2 Using Multiple Trusted-hardware Chips on a Single Consumer Device

Taking measurements (*e.g.*, checksum of code and data) in order to understand the trust status of a system is generally accepted as a useful strategy. Today, a single commodity computer can have both TPM and SGX enabled. This may potentially mean having two different Root of Trust for Measurement (platform firmware and CPU xucode) by which to describe the system and represent its trustworthiness or lack thereof. This poses the question which one or both of the two are to be actively used in representing a whole system, and in the latter case how their differential information output should be combined. In order to evaluate the trustworthiness of a system, reports, quotes and evidences must be securely composed and presented in a unified form. Using the right mechanism, for example the *trusted firmware* for the right purpose in secure composition remains the crucial challenge.

Index

References

[1] Olusola Akinrolabu, Robin Ankele, Ahmad Atamli, Ranjbar Balisane, Ravishankar Borgaonkar, Pardeep Kumar, Kubilay Ahmet Küçük, Yudhistira Nugraha, Piers O'Hanlon, Thomas Spoor, Tina Wu, and Andrew Martin. "Trustworthy Systems". In: Oxford Cyber Security Open Day, 2017.

[2] Kubilay Ahmet Küçük, Andrew Paverd, Andrew Martin, N Asokan, Andrew Simpson, and Robin Ankele. "Exploring the use of Intel SGX for Secure Many-Party Applications". In: *Proceedings of the 1st Workshop on System Software for Trusted Execution - SysTEX '16*. SysTEX '16. Trento, Italy: ACM Press, 2016, 5:1–5:6. URL: http://doi.acm.org/10.1145/3007788.3007793.

[3] Robin Ankele. "Addressing syntactic privacy for privacy-preserving data analysis and data release". PhD thesis. University of Oxford, 2020. URL: https://ora.ox.ac.uk/objects/uuid:fdbfe37f-4860-4d8c-8f5f-e18ce68136cd.

[4] Robin Ankele, Kubilay Ahmet Küçük, Andrew Martin, Andrew Simpson, and Andrew Paverd. "Applying the trustworthy remote entity to privacy-preserving multiparty computation: Requirements and criteria for large-scale applications". In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ ATC/ ScalCom/ CBDCom/ IoP/ SmartWorld)*. IEEE. 2016, pp. 414–422. URL: https://ieeexplore.ieee.org/abstract/document/7816873.

[5] Robin Ankele and Andrew Simpson. "On the performance of a trustworthy remote entity in comparison to secure multi-party computation". In: *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE. 2017, pp. 1115–1122. URL: https://ieeexplore.ieee.org/document/8029564.

[6] Kubilay Ahmet Küçük, David Grawrock, and Andrew Martin. "Managing Confidentiality Leaks Through Private Algorithms on Software Guard eXtensions (SGX) Enclaves: Minimised TCB on Secret-Code Execution With Early Private Mode (EPM)". In: *EURASIP Journal on Information Security, Special Issue on Recent Advances in Software Security, Springer* 2019.14 (2019). URL: https://doi.org/10.1186/s13635-019-0091-5.

[7] Kubilay Ahmet Küçük and Andrew Martin. "Framework of secret differential privacy on private data". In: ORA, https://ora.ox.ac.uk/objects/uuid:afaa9c13-8630-431d-862a-bf04bf4f4663/download_file?safe_filename=ssg-oxfordv2.pdf. Secure

Systems Annual Demo Day, Aalto University, Finland, 2017. URL: https://wiki.aalto.fi/pages/viewpage.action?pageId=117688774.

[8] Kubilay Ahmet Küçük and Andrew Martin. "Enabling the Use of Strongly-Private Algorithms". In: Open-Source Enclaves Workshop (OSEW) 2019, Wozniak Lounge - Soda Hall, UC BERKELEY, 2019. URL: https://keystone-enclave.org/open-source-enclaves-workshop/slides/OSEW19_AhmetKucuk_Oxford.pdf.

[9] Kubilay Ahmet Küçük. "Managing Private Algorithms in SGX Enclaves". In: *CordaCon 2019, R3 Corda Conference DevDay, London UK*. R3. 2019. URL: https://www.r3.com/videos/managing-private-algorithms-in-sgx-enclaves-university-of-oxford/.

[10] Kubilay Ahmet Küçük and Andrew Martin. "CRC: Fully General Model of Confidential Remote Computing". In: OpenAccess, arXiv:2104.03868 PrePrint, https://arxiv.org/abs/2104.03868. 2021.

[11] Kubilay Ahmet Küçük. "How infeasible the malware deployment in SGX is in real-life? Can Malware benefit from SGX?" In: Intel's 2nd SGX Community Day. Last Accessed 28 Feb 2021. Intel, Hillsboro, Portland OR, 2020. URL: https://www.intel.com/content/dam/www/public/us/en/documents/research/kubilay-kucuk-malware-infeasibility-sgx.pdf.

[12] Kubilay Ahmet Küçük, Steve Moyle, Andrew Martin, Alexandru Meceacre, and Nicholas Allott. "SoK: How 'Not' to Architect Your Next-Generation TEE Malware". In: *Hardware and Architectural Support for Security and Privacy (HASP) 2022*. ACM, 2022.

[13] Victor Costan and Srinivas Devadas. *Intel SGX Explained*. Tech. rep. Cryptology ePrint Archive, Report 2016/086, 20 16. http://eprint.iacr. org.

[14] Shay Gueron. *A Memory Encryption Engine Suitable for General Purpose Processors*. Cryptology ePrint Archive, Report 2016/204. https://eprint.iacr.org/2016/204. 2016.

[15] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. "Using Innovative Instructions to Create Trustworthy Software Solutions". In: *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '13. visited on 03/Jun/2019. Tel-Aviv, Israel: ACM, 2013, 11:1–11:1.

[16] Simon P Johnson, Vincent R Scarlata, Carlos V Rozas, Ernie Brickell, and Frank McKeen. "Intel SGX: EPID provisioning and attestation services". In: *Intel* (2016).

[17] Strackx Strackx. "Security Primitives for Protected-Module Architectures". PhD thesis. KU Leuven, 2014. URL: https://kuleuven.limo.libis.be/discovery/fulldisplay?docid=lirias1656259&context=SearchWebhook&vid=32KUL_KUL:Lirias.

[18] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. "Flicker: an execution infrastructure for tcb minimization". In: *EuroSys* 42.4 (2008), p. 315. URL: http://dl.acm.org/citation.cfm?id=1357010.1352625$\backslash$nhttp://portal.acm.org/citation.cfm?doid=1352592.1352625.

[19] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. "Trust visor: Efficient TCB reduction and attestation". In: *Proceedings - IEEE Symposium on Security and Privacy*. 2010, pp. 143–158.

[20] Donald C Latham. "Trusted Computer System Evaluation Criteria". In: *Department of Defense* (1985). URL: https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/dod85.pdf.

[21] Ira Winkler and Araceli Treu Gomes. "Chapter 7 - Adversary Infrastructure". In: *Advanced Persistent Security*. Ed. by Ira Winkler and Araceli Treu Gomes. Syngress, 2017, pp. 67–79. URL: https://www.sciencedirect.com/science/article/pii/B9780128093160000075.

[22] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. "seL4: Formal verification of an OS kernel". In: *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 2009, pp. 207–220.

[23] Siani Pearson. "Trusted computing platforms, the next security solution". In: *HP Labs* 177 (2002).

[24] Siani Pearson and Boris Balacheff. *Trusted computing platforms: TCPA technology in context*. Prentice Hall Professional, 2003.

[25] Andrew Martin. "The ten page introduction to trusted computing". In: *Computing Laboratory, Oxford University Oxford* (2008).

[26] GJ Proudler. "Concepts of trusted computing". In: *Trusted computing* 6 (2005), pp. 11–27.

[27] Chris Mitchell. *Trusted computing*. Vol. 6. Iet, 2005.

[28] Sean W Smith. *Trusted computing platforms: design and applications*. Springer, 2005.

[29] Mike Bursell. *Trust in Computer Systems and the Cloud*. John Wiley & Sons, 2021.

[30] Peter Lipp and Ahmad-Reza Sadeghi, eds. *Trusted Computing-Challenges and Applications: First International Conference on Trusted Computing and Trust in Information Technologies, TRUST 2008 Villach, Austria, March 11-12, 2008 Proceedings*. Vol. 4968. Springer Science & Business Media, 2008.

[31] Liqun Chen, Chris J Mitchell, and Andrew Martin, eds. *Trusted Computing: Second International Conference, Trust 2009 Oxford, UK, April 6-8, 2009, Proceedings*. Vol. 5471. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, p. 218. URL: https://books.google.co.uk/books?id=l3aqCAAAQBAJ.

[32] Alessandro Acquisti, Sean W Smith, and Ahmad-Reza Sadeghi, eds. *Trust and Trustworthy Computing: Third International Conference, TRUST 2010, Berlin, Germany, June 21-23, 2010, Proceedings*. Vol. 6101. Springer, 2010.

[33] Jonathan McCune, Boris Balacheff, Adrian Perrig, Ahmad-Reza Sadeghi, Angela Sasse, and Yolanta Beres, eds. *Trust and Trustworthy Computing: 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011, Proceedings*. Vol. 6740. Springer, 2011.

[34] Stefan Katzenbeisser, Edgar Weippl, L Jean Camp, Melanie Volkamer, Mike Reiter, and Xinwen Zhang, eds. *Trust and Trustworthy Computing: 5th International Conference, TRUST 2012, Vienna, Austria, June 13-15, 2012, Proceedings*. Vol. 7344. Springer, 2012.

[35] Michael Huth, N Asokan, Srdjan Capkun, Ivan Flechais, and Lizzie Coles-Kemp. *Trust and Trustworthy Computing: 6th International Conference, TRUST 2013, London, UK, June 17-19, 2013, Proceedings*. Vol. 7904. Springer, 2013.

[36] Mauro Conti, Matthias Schunter, and Ioannis Askoxylakis. *Trust and Trustworthy Computing: 8th International Conference, TRUST 2015, Heraklion, Greece, August 24-26, 2015, Proceedings*. Vol. 9229. Springer, 2015.

[37] Michael Franz and Panos Papadimitratos. *Trust and Trustworthy Computing*. Springer, 2016.

[38] Will Arthur, David Challener, and Kenneth Goldman. *A practical guide to TPM 2.0: Using the new trusted platform module in the new age of security*. Springer Nature, 2015.

[39] Bryan Parno, Jonathan M McCune, and Adrian Perrig. *Bootstrapping trust in modern computers*. Springer Science & Business Media, 2011.

[40] David Grawrock. *Dynamics of a Trusted Platform: A building block approach*. Intel Press, 2009.

[41] David Grawrock. *Trusted Computing Infrastructures, MSc course at Oxford University*. Oxford University. URL: `https://www.cs.ox.ac.uk/softeng/subjects/TCI.html`.

[42] Sean Smith. "Hardware security modules". In: *Handbook of Financial Cryptography and Security*. Chapman and Hall/CRC, 2010, pp. 283–304.

[43] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Nguyen Anh Quynh. "Trusted boot and platform trust services on 1CD Linux". In: *2008 Third Asia-Pacific Trusted Infrastructure Technologies Conference*. IEEE. 2008, pp. 64–71.

[44] Seiji Munetoh. *Open Platform Trust Services (OpenPTS) User's Guide*. 2011.

[45] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, Nguyen Anh Quynh, Megumi Nakamura, and Seiji Muhetoh. "TPM+ Internet Virtual Disk+ Platform Trust Services= Internet Client". In: *Trusted Infrastructure Technologies Conference, 2008. APTC '08. Third Asia-Pacific* (Nov. 2008).

[46] TCG Infrastructure Working Group. "Platform Trust Services Interface Specification (IF-PTS)". In: *https://trustedcomputinggroup.org/wp-content/uploads/IWG-IF-PTS_v1.pdf* 1 (Nov. 2006).

[47] Thomas Haines, Rajeev Goré, and Bhavesh Sharma. "Did you mix me? Formally Verifying Verifiable Mix Nets in Electronic Voting." In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1114.

[48] Ralph C Merkle. "A digital signature based on a conventional encryption function". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1987, pp. 369–378.

[49] US Department of Defense. "A Guide to Understanding Audit in Trusted Systems". In: *The 'Orange Book'Series* (1985), pp. 734–760.

[50] Jakob Mökander, Jessica Morley, Mariarosaria Taddeo, and Luciano Floridi. "Ethics-based auditing of automated decision-making systems: Nature, scope, and limitations". In: *Science and engineering ethics* 27.4 (2021), pp. 1–30.

[51] Jakob Mökander and Luciano Floridi. "Ethics-based auditing to develop trustworthy AI". In: *Minds and Machines* 31.2 (2021), pp. 323–327.

[52] Jakob Mökander and Maria Axente. "Ethics-based auditing of automated decision-making systems: intervention points and policy implications". In: *AI & SOCIETY* (2021), pp. 1–19.

[53] Tigist Abera, N Asokan, Lucas Davi, Farinaz Koushanfar, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. "Things, trouble, trust: on building trust in IoT systems". In: *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2016, pp. 1–6.

[54] Yanlin Li, Yueqiang Cheng, Virgil Gligor, and Adrian Perrig. "Establishing software-only root of trust on embedded systems: facts and fiction". In: *Cambridge International Workshop on Security Protocols*. Springer. 2015, pp. 50–68.

[55] Sandeep Tamrakar, Jian Liu, Andrew Paverd, Jan-Erik Ekberg, Benny Pinkas, and N. Asokan. "The Circle Game: Scalable Private Membership Test Using Trusted Hardware". In: *CoRR* abs/1606.01655 (2016). URL: `http://arxiv.org/abs/1606.01655`.

[56] Craig Gentry. "Fully homomorphic encryption using ideal lattices". en. In: *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09*. Bethesda, MD, USA: ACM Press, 2009, p. 169. URL: `http://portal.acm.org/citation.cfm?doid=1536414.1536440` (visited on 08/27/2020).

[57] Oded Goldreich. "Secure multi-party computation". In: *Manuscript. Preliminary version, and Foundations of Cryptography in Cambridge University Press* (1998), pp. 86–97.

[58] Yansong Gao, Said F Al-Sarawi, and Derek Abbott. "Physical unclonable functions". In: *Nature Electronics* 3.2 (2020), pp. 81–91.

[59] Charles Herder, Meng-Day Yu, Farinaz Koushanfar, and Srinivas Devadas. "Physical unclonable functions and applications: A tutorial". In: *Proceedings of the IEEE* 102.8 (2014), pp. 1126–1141.

[60] Michael Neve, Eric Peeters, David Samyde, and J-J Quisquater. "Memories: A survey of their secure uses in smart cards". In: *Second IEEE International Security in Storage Workshop*. IEEE. 2003, pp. 62–62.

[61] David Basin, Ralf Sasse, and Jorge Toro-Pozo. "Card Brand Mixup Attack: Bypassing the {PIN} in {non-Visa} Cards by Using Them for Visa Transactions". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 179–194.

[62] Cas Cremers. "Compositionality of Security Protocols: A Research Agenda". In: *Electronic Notes in Theoretical Computer Science* 142 (2006). visited on 02/Jun/2019. Proceedings of the First International Workshop on Views on Designing Complex Architectures (VODCA 2004), pp. 99–110. URL: https://doi.org/10.1016/j.entcs.2004.12.047.

[63] Pieter Ceelen and S Mauw UdL. "Compositionality of Security Protocols". In: *Master's Thesis, Technische Universiteit Eindhoven* ().

[64] John Lyle. "Trustworthy Services Through Attestation". In: *Philosophy* (2010).

[65] John Lyle and Andrew Martin. "On the feasibility of remote attestation for web services". In: *2009 International Conference on Computational Science and Engineering*. Vol. 3. IEEE. 2009, pp. 283–288.

[66] John Lyle and Andrew Martin. "Engineering Attestable Services". In: *Trust and Trustworthy Computing*. 2010.

[67] George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian O'Hanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen. "Principles of remote attestation". In: *International Journal of Information Security* 10.2 (2011), pp. 63–81.

[68] Ioannis Sfyrakis and Thomas Gross. "A survey on hardware approaches for remote attestation in network infrastructures". In: *arXiv preprint arXiv:2005.12453* (2020).

[69] Alexander Sprogø Banks, Marek Kisiel, and Philip Korsholm. "Remote attestation: a literature review". In: *arXiv preprint arXiv:2105.02466* (2021).

[70] Dominic P Mulligan, Gustavo Petri, Nick Spinale, Gareth Stockwell, and Hugo JM Vincent. "Confidential Computing—a brave new world". In: *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE. 2021, pp. 132–138.

[71] Ernie Brickell, Jan Camenisch, and Liqun Chen. "Direct anonymous attestation". In: *Proceedings of the 11th ACM conference on Computer and communications security*. ACM. 2004, pp. 132–145.

[72] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. "Intel® software guard extensions: Epid provisioning and attestation services". In: *White Paper* 1 (2016), pp. 1–10.

[73] Kari Kostiainen, N. Asokan, and Jan-Erik Ekberg. "Practical property-based attestation on mobile devices". In: *Proceedings of the 4th international conference on Trust and trustworthy computing - TRUST'11*. 2011, pp. 78–92. URL: http://dl.acm.org/citation.cfm?id=2022245.2022254.

[74] Ahmad-Reza Sadeghi and Christian Stuble. "Property-based attestation for computing platforms: caring about properties, not mechanisms". In: *Proceedings of the 2004 workshop on New security paradigms (NSPW'04)*. New York, NY, USA: ACM Press, 2004, pp. 67–77.

[75] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. "Innovative Instructions and Software Model for Isolated Execution". In: *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '13. visited on 03/Jun/2019. Tel-Aviv, Israel: ACM, 2013, 10:1–10:1.

[76] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. "Innovative Technology for CPU Based Attestation and Sealing". In: *HASP - Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy* (2013), pp. 1–7.

[77] Hugo Krawczyk. "SIGMA: The 'SIGn-and-MAc'approach to authenticated Diffie-Hellman and its use in the IKE protocols". In: *Advances in Cryptology-CRYPTO 2003*. Springer, 2003, pp. 400–425.

[78] Confidential Computing Consortium. "Confidential Computing: Hardware-Based Trusted Execution for Applications and Data". In: *A Publication of The Confidential Computing Consortium July 2020* (2020). URL: https://confidentialcomputing.io/wp-content/uploads/sites/85/2020/06/ConfidentialComputing_OSSNA2020.pdf.

[79] Rodolfo Silva, Pedro Barbosa, and Andrey Brito. "Dynsgx: A privacy preserving toolset for dinamically loading functions into intel (r) sgx enclaves". In: *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2017, pp. 314–321.

[80] Erick Bauman, Huibo Wang, Mingwei Zhang, and Zhiqiang Lin. "Sgxelide: enabling enclave code secrecy via self-modification". In: *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. 2018, pp. 75–86.

[81] Fahmida Y Rashid. "The rise of confidential computing: Big tech companies are adopting a new security model to protect data while it's in use-[News]". In: *IEEE Spectrum* 57.6 (2020), pp. 8–9.

[82] Alexander Frank. *System and method for confidential remote computing*. US Patent App. 14/214,936. Sept. 2015.

[83] Ken Naganuma. *Confidential computation system, confidential computation method, and confidential computation program*. US Patent 9,276,734. Mar. 2016.

[84] Dan R. K. Ports and Tal Garfinkel. "Towards Application Security on Untrusted Operating Systems". In: *Proceedings of the 3rd Conference on Hot Topics in Security*. HOTSEC'08. San Jose, CA: USENIX Association, 2008, 1:1–1:7. URL: https://doi.acm.org/10.1145/1496671.1496672.

[85] Lenin Singaravelu, Calton Pu, Hermann Härtig, and Christian Helmuth. "Reducing TCB complexity for security-sensitive applications". In: *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*. Vol. 40. 4. visited on 03/Jun/2019. Leuven, Belgium: ACM, 2006, pp. 161–174.

[86] F. Piessens, D. Devriese, J. T. Mühlberg, and R. Strackx. "Security Guarantees for the Execution Infrastructure of Software Applications". In: *2016 IEEE Cybersecurity Development (SecDev)*. visited on 03/Jun/2019. Nov. 2016, pp. 81–87.

[87] Jonathan M. McCune, Bryan Parno, Adrian Perrig, Michael K. Reiter, and Arvind Seshadri. "How Low Can You Go?: Recommendations for Hardware-supported Minimal TCB Code Execution". In: *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XIII. visited on 03/Jun/2019. Seattle, WA, USA: ACM, 2008, pp. 14–25.

[88] Raoul Strackx and Frank Piessens. "Developing Secure SGX Enclaves: New Challenges on the Horizon". In: *Proceedings of the 1st Workshop on System Software for Trusted Execution - SysTEX '16*. visited on 03/Jun/2019. Trento, Italy: ACM Press, 2016, 3:1–3:2.

[89] Cullen Linn and Saumya Debray. "Obfuscation of Executable Code to Improve Resistance to Static Disassembly". In: *Proceedings of the 10th ACM Conference on Computer and Communications Security*. CCS '03. visited on 03/Jun/2019. Washington D.C., USA: ACM, 2003, pp. 290–299.

[90] Zhenyu Wu, Steven Gianvecchio, Mengjun Xie, and Haining Wang. "Mimimorphism: A New Approach to Binary Code Obfuscation". In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS '10. visited on 03/Jun/2019. Chicago, Illinois, USA: ACM, 2010, pp. 536–546.

[91] Ziad A. Al-Sharif, Mohammed I. Al-Saleh, Luay M. Alawneh, Yaser I. Jararweh, and Brij Gupta. "Live forensics of software attacks on cyber–physical systems". In: *Future Generation Computer Systems* (July 2018). visited on 02/Jun/2019.

[92] Andrew Paverd. "Enhancing Communication Privacy Using Trustworhy Remote Entities". 2016. URL: `https://ora.ox.ac.uk/objects/uuid:66ce7364-9562-4441-957a-a940e4a50784`.

[93] Richard O Sinnott, Prem Chhetri, Yikai Gong, Angus Macaulay, and William Voorsluys. "Privacy-preserving Data Linkage through Blind Geo-spatial Data Aggregation". In: *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*. IEEE. 2015, pp. 1381–1386.

[94] RO Sinnott, O Ajayi, and AJ Stell. "Data privacy by design: digital infrastructures for clinical collaborations". In: (2009).

[95] Patrick Koeberl, Vinay Phegade, Anand Rajan, Thomas Schneider, Steffen Schulz, and Maria Zhdanova. "Time to rethink: Trust brokerage using trusted execution environments". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9229. 2015, pp. 181–190.

[96] Seongmin Kim, Youjung Shin, Jaehyung Ha, Taesoo Kim, and Dongsu Han. "A First Step Towards Leveraging Commodity Trusted Execution Environments for Network Applications". In: *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*. ACM. 2015, p. 7.

[97] Prerit Jain, Soham Desai, Seongmin Kim, Ming-Wei Shih, J Lee, Changho Choi, Youjung Shin, Taesoo Kim, Brent Byunghoon Kang, and Dongsu Han. "OpenSGX: An Open Platform for SGX Research". In: *Proceedings of the Network and Distributed System Security Symposium, San Diego, CA*. 2016.

[98] Rohit Sinha, Sriram Rajamani, Sanjit Seshia, and Kapil Vaswani. "Moat: Verifying Confidentiality of Enclave Programs". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 1169–1184.

[99] Rohit Sinha, Manuel Costa, Akash Lal, Nuno Lopes, Sanjit Seshia, Sriram Rajamani, and Kapil Vaswani. "A Design and Verification Methodology for Secure Isolated Regions". In: (2016).

[100] Ahmad Atamli-Reineh and Andrew Martin. "Securing Application with Software Partitioning: A Case Study Using SGX". In: *Security and Privacy in Communication Networks*. Springer, 2015, pp. 605–621.

[101] Dawson R Engler, M Frans Kaashoek, et al. *Exokernel: An operating system architecture for application-level resource management*. Vol. 29. 5. ACM, 1995.

[102] Jonathan Poritz, Matthias Schunter, Els Van Herreweghen, and Michael Waidner. "Property attestation—scalable and privacy-friendly security assessment of peer computers". In: (2004).

[103] Andrew C Yao. "Protocols for secure computations". In: *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*. IEEE. 1982, pp. 160–164.

[104] Andrew Paverd, Andrew Martin, and Ian Brown. "Privacy-enhanced bi-directional communication in the Smart Grid using trusted computing". In: *2014 IEEE International Conference on Smart Grid Communications, SmartGridComm 2014*. 2015, pp. 872–877.

[105] Jared Schmitz, Jason Loew, Jesse Elwell, Dmitry Ponomarev, and Nael Abu-Ghazaleh. "TPM-SIM: a framework for performance evaluation of trusted platform modules". In: *Proceedings of the 48th Design Automation Conference*. ACM. 2011, pp. 236–241.

[106] Michael O Rabin. "How To Exchange Secrets with Oblivious Transfer." In: *IACR Cryptology ePrint Archive* 2005 (2005), p. 187.

[107] Andrea Bartoli, Juan Hernández-Serrano, M Soriano, Mischa Dohler, Apostolous Kountouris, and Dominique Barthel. "Secure lossless aggregation for smart grid M2M networks". In: *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*. IEEE. 2010, pp. 333–338.

[108] Jens-Matthias Bohli, Christoph Sorge, and Osman Ugus. "A privacy model for smart metering". In: *Communications Workshops (ICC), 2010 IEEE International Conference on*. IEEE. 2010, pp. 1–5.

[109] Andrew Paverd, Andrew Martin, and Ian Brown. "Privacy-enhanced bi-directional communication in the smart grid using trusted computing". In: *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*. IEEE. 2014, pp. 872–877.

[110] Chi-Yin Chow, Mohamed F Mokbel, and Xuan Liu. "A peer-to-peer spatial cloaking algorithm for anonymous location-based service". In: *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*. ACM. 2006, pp. 171–178.

[111] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. "Privacy-friendly aggregation for the smart-grid". In: *Privacy Enhancing Technologies*. Springer. 2011, pp. 175–191.

[112] Alfredo Rial and George Danezis. "Privacy-preserving smart metering". In: *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*. ACM. 2011, pp. 49–60.

[113] T Alves and D Felton. "Trustzone: Integrated Hardware and Software Security". In: *ARM white paper* 3.4 (2004), pp. 18–24. URL: `https://www.techonline.com/electrical-engineers/education-training/tech-papers/4129205/TrustZone-Integrated-Hardware-and-Software-Security`.

[114] Cynthia Dwork. "Differential privacy". In: *Automata, languages and programming*. Springer, 2006, pp. 1–12.

[115] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems". In: *Proceedings of the 2015 IEEE Symposium on Security and Privacy*. SP '15. visited on 03/Jun/2019. Washington, DC, USA: IEEE Computer Society, May 2015, pp. 640–656.

[116] Stephen Checkoway and Hovav Shacham. "Iago attacks". In: *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems - ASPLOS '13*. New York, New York, USA: ACM Press, 2013, p. 253. URL: `http://dl.acm.org/citation.cfm?doid=2451116.2451145`.

[117] Rajat Subhra Chakraborty and Swarup Bhunia. "Hardware protection and authentication through netlist level obfuscation". In: *2008 IEEE/ACM International Conference on Computer-Aided Design*. 2008, pp. 674–677.

[118] Brooks Olney and Robert Karam. "WATERMARCH: IP Protection Through Authenticated Obfuscation in FPGA Bitstreams". In: *IEEE Embedded Systems Letters* 13.3 (2021), pp. 81–84.

[119] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. "Oblivious Multi-party Machine Learning on Trusted Processors". In: *Proceedings of the 25th USENIX Conference on Security Symposium*. SEC'16. visited on 03/Jun/2019. Austin, TX, USA: USENIX Association, Aug. 2016, pp. 619–636. URL: `https://doi.acm.org/10.1145/3241094.3241143`.

[120] Fahad Shaon, Murat Kantarcioglu, Zhiqiang Lin, and Latifur Khan. "SGX-BigMatrix: A Practical Encrypted Data Analytic Framework With Trusted Processors". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. visited on 03/Jun/2019. Dallas, Texas, USA: ACM, 2017, pp. 1211–1228.

[121] Rodrigo J. Riella, Luciana M. Iantorno, Laerte C. R. Junior, Dilmari Seidel, Keiko V. O. Fonseca, Luiz Gomes-Jr, and Marcelo O. Rosa. "Securing Smart Metering Applications in Untrusted Clouds with the SecureCloud Platform". In: *Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems*. W-P2DS'18. visited on 03/Jun/2019. Porto, Portugal: ACM, 2018, 5:1–5:6.

[122] L. Coppolino, S. D'Antonio, G. Mazzeo, G. Papale, L. Sgaglione, and F. Campanile. "An Approach for Securing Critical Applications in Untrusted Clouds". In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. visited on 02/Jun/2019. Mar. 2018, pp. 436–440.

[123] Leandro Ventura Silva, Rodolfo Marinho, Jose Luis Vivas, and Andrey Brito. "Security and Privacy Preserving Data Aggregation in Cloud Computing". In: *Proceedings of the Symposium on Applied Computing*. SAC '17. visited on 03/Jun/2019. Marrakech, Morocco: ACM, 2017, pp. 1732–1738.

[124] F. Kelbert, F. Gregor, R. Pires, S. Köpsell, M. Pasin, A. Havet, V. Schiavoni, P. Felber, C. Fetzer, and P. Pietzuch. "SecureCloud: Secure big data processing in untrusted clouds". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. visited on 03/Jun/2019. Mar. 2017, pp. 282–285.

[125] F. Campanile, L. Coppolino, S. D'Antonio, L. Lev, G. Mazzeo, L. Romano, L. Sgaglione, and F. Tessitore. "Cloudifying Critical Applications: A Use Case from the Power Grid Domain". In: *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. visited on 02/Jun/2019. Mar. 2017, pp. 363–370.

[126] Global Platform. *Introduction to Trusted Execution Environments*. Online Resource In Global Platform, Inc. Website. `https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Environment-15May2018.pdf`. visited on 03/Jun/2019. Non-Profit Association: Global Platform, Inc., May 2018,

[127] Joshua Lind, Christian Priebe, Divya Muthukumaran, Dan O'Keeffe, Pierre-Louis Aublin, Florian Kelbert, Tobias Reiher, David Goltzsche, David Eyers, Rüdiger Kapitza, Christof Fetzer, and Peter Pietzuch. "Glamdring: Automatic Application Partitioning for Intel SGX". In: *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*. USENIX ATC '17. visited on 03/Jun/2019. Santa Clara, CA, USA: USENIX Association, 2017, pp. 285–298. URL: `https://doi.acm.org/10.1145/3154690.3154718`.

[128] Robert Krahn, Bohdan Trach, Anjo Vahldiek-Oberwagner, Thomas Knauth, Pramod Bhatotia, and Christof Fetzer. "Pesos: Policy Enhanced Secure Object Store". In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys '18. visited on 03/Jun/2019. Porto, Portugal: ACM, 2018, 25:1–25:17.

[129] Stefan Brenner, Tobias Hundt, Giovanni Mazzeo, and Rüdiger Kapitza. "Secure Cloud Micro Services Using Intel SGX". In: *Distributed Applications and Interoperable Systems*. Ed. by Lydia Y. Chen and Hans P. Reiser. Cham: Springer International Publishing, 2017, pp. 177–191.

[130] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Mark L Stillwell, David Goltzsche, David Eyers, Peter Pietzuch, and Christof Fetzer. "SCONE: Secure Linux Containers with Intel SGX". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. visited on 02/Jun/2019. Savannah, GA, USA: USENIX Association, 2016, pp. 689–703. URL: http://doi.acm.org/10.1145/3026877.3026930.

[131] Pierre-Louis Aublin, Florian Kelbert, Dan O'Keeffe, Divya Muthukumaran, Christian Priebe, Joshua Lind, Robert Krahn, Christof Fetzer, David Eyers, and Peter Pietzuch. "LibSEAL: Revealing Service Integrity Violations Using Trusted Execution". In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys '18. visited on 02/Jun/2019. Porto, Portugal: ACM, 2018, 24:1–24:15.

[132] Stefan Brenner, David Goltzsche, and Rüdiger Kapitza. "TrApps: Secure Compartments in the Evil Cloud". In: *Proceedings of the 1st International Workshop on Security and Dependability of Multi-Domain Infrastructures*. XDOMO'17. visited on 02/Jun/2019. Belgrade, Serbia: ACM, 2017, 5:1–5:6. URL: https://doi.acm.org/10.1145/3071064.3071069.

[133] Dmitrii Kuvaiskii, Oleksii Oleksenko, Sergei Arnautov, Bohdan Trach, Pramod Bhatotia, Pascal Felber, and Christof Fetzer. "SGXBOUNDS: Memory Safety for Shielded Execution". In: *Proceedings of the Twelfth European Conference on Computer Systems*. EuroSys '17. visited on 03/Jun/2019. Belgrade, Serbia: ACM, 2017, pp. 205–221.

[134] Chia-Che Tsai, Donald E Porter, and Mona Vij. "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX". In: *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*. USENIX ATC '17. visited on 03/Jun/2019. Santa Clara, CA, USA: USENIX Association, 2017, pp. 645–658. URL: https://doi.acm.org/10.1145/3154690.3154752.

[135] Feng Chen, Michelle Dow, Sijie Ding, Yao Lu, Xiaoqian Jiang, Hua Tang, and Shuang Wang. "PREMIX: PRivacy-preserving EstiMation of Individual admiXture". In: *American Medical Informatics Association Annual Symposium* (Feb. 2016). visited on 02/Jun/2019, pp. 1747–1755. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5333197/pdf/2500255.pdf.

[136] Feng Chen, Shuang Wang, Xiaoqian Jiang, Sijie Ding, Yao Lu, Jihoon Kim, S Cenk Sahinalp, Chisato Shimizu, Jane C Burns, Victoria J Wright, Eileen Png, Martin L Hibberd, David D Lloyd, Hai Yang, Amalio Telenti, Cinnamon S Bloss, Dov Fox, Kristin Lauter, and Lucila Ohno-Machado. "PRINCESS: Privacy-protecting Rare disease International Network Collaboration via Encryption through Software guard extensionS". In: *Bioinformatics (Oxford, England)* 33.6 (Mar. 2017). visited on 02/Jun/2019, 871—878. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5860394/pdf/btw758.pdf.

[137] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. "Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. visited on 03/Jun/2019. Savannah, GA, USA: USENIX Association, 2016, pp. 533–549. URL: https://doi.acm.org/10.1145/3026919.

[138] Rohit Sinha, Sriram Rajamani, Sanjit Seshia, and Kapil Vaswani. "Moat: Verifying Confidentiality of Enclave Programs". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*. CCS '15. visited on 03/Jun/2019. Denver, Colorado, USA: ACM Press, 2015, pp. 1169–1184.

[139] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. "AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves". In: *21st European Symposium on Research in Somputer Science*. Springer, Cham, 2016, pp. 440–457. URL: http://link.springer.com/10.1007/978-3-319-45744-4_22.

[140] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. "VC3: Trustworthy Data Analytics in the Cloud Using SGX". In: *Proceedings of the 2015 IEEE Symposium on Security and Privacy*. Vol. 2015-July. SP '15. visited on 03/Jun/2019. IEEE Computer Society, 2015, pp. 38–54.

[141] David Goltzsche, Colin Wulf, Divya Muthukumaran, Konrad Rieck, Peter Pietzuch, and Rüdiger Kapitza. "TrustJS: Trusted Client-side Execution of JavaScript". In: *Proceedings of the 10th European Workshop on Systems Security*. EuroSec'17. visited on 03/Jun/2019. Belgrade, Serbia: ACM, 2017, 7:1–7:6.

[142] Asier Rivera Fernandez. "Integrity and confidentiality for web application code execution in untrusted clients. Promoting a Trust Relation in Web-Applications". visited on 03/Jun/2019. MA thesis. Göteborg, Sweden: Department of Computer Science and Engineering (Chalmers), Chalmers University of Technology, 2017. URL: http://publications.lib.chalmers.se/records/fulltext/252354/252354.pdf.

[143] Victor Costan, Ilia Lebedev, and Srinivas Devadas. "Sanctum: Minimal Hardware Extensions for Strong Software Isolation". In: *Proceedings of the 25th USENIX Conference on Security Symposium*. SEC'16. visited on 02/Jun/2019. Austin, TX, USA: USENIX Association, Aug. 2016, pp. 857–874. URL: https://doi.acm.org/10.1145/3241094.3241161.

[144] Intel. *Intel® Software Guard Extensions Enclave Writer's Guide v1.02*. Revision 1.02. visited on 03/Jun/2019. Intel Corporation. Portland, OR, USA, 2015. URL: https://software.intel.com/sites/default/files/managed/ae/48/Software-Guard-Extensions-Enclave-Writers-Guide.pdf.

[145] Intel. *Intel® Software Guard Extensions Programming Reference*. Ref. #329298-002. visited on 03/Jun/2019. Intel Corporation. Portland, OR, USA, Oct. 2014. URL: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[146] Intel. *Intel® Software Guard Extensions (Intel® SGX) Developer Guide v2.1*. Revision 2.1 Linux. visited on 03/Jun/2019. Intel Corporation. Portland, OR, USA, May 2019. URL: https://download.01.org/intel-sgx/linux-2.1/docs/Intel_SGX_Developer_Guide.pdf.

[147] King John and Magoulas Roger. *2016 Data Science Salary Survey*. visited on 03/Jun/2019. 2016. URL: https://www.oreilly.com/data/free/2016-data-science-salary-survey.csp.

[148] Shweta Shinde, D Le Tien, Shruti Tople, and Prateek Saxena. "PANOPLY: Low-TCB Linux Applications with SGX Enclaves". In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. visited on 03/Jun/2019. Reston, VA: The Internet Society, 2017. URL: https://dblp.org/rec/bib/conf/ndss/ShindeTTS17.

[149] Marcus Hähnel, Weidong Cui, and Marcus Peinado. "High-resolution Side Channels for Untrusted Operating Systems". In: *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*. USENIX ATC '17. visited on 03/Jun/2019. Santa Clara, CA, USA: USENIX Association, 2017, pp. 299–312. URL: https://doi.acm.org/10.1145/3154690.3154719.

[150] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. "Software Grand Exposure: SGX Cache Attacks Are Practical". In: *Proceedings of the 11th USENIX Conference on Offensive Technologies*. WOOT'17. visited on 02/Jun/2019. Vancouver, BC, Canada: USENIX Association, 2017, pp. 11–11. URL: https://doi.acm.org/10.1145/3154768.3154779.

[151] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. "Cache Attacks on Intel SGX". In: *Proceedings of the 10th European Workshop on Systems Security*. EuroSec'17. visited on 03/Jun/2019. Belgrade, Serbia: ACM, 2017, 2:1–2:6.

[152] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. "CacheZoom: How SGX Amplifies the Power of Cache Attacks". In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Cham: Springer International Publishing, 2017, pp. 69–90.

[153] Intel, Simon Johnson. *Intel® SGX and Side-Channels*. Developer Zone. published on March 16th 2017, updated February 27th 2018, visited on 03/Jun/2019. Portland, OR, USA: Intel Corporation. URL: https://software.intel.com/en-us/articles/intel-sgx-and-side-channels.

[154] Intel. *Intel ® Software Guard Extensions Developer Reference (Intel ® SGX) SDK for Linux * OS*. Revision 2.1. visited on 03/Jun/2019. Intel Corporation. Portland, OR, USA, Dec. 2017. URL: https://download.01.org/intel-sgx/linux-2.1/docs/Intel_SGX_Developer_Reference_Linux_2.1_Open_Source.pdf.

[155] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. "T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs". In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. visited on 03/Jun/2019. Reston, VA: The Internet Society. URL: https://dblp.org/rec/bib/conf/ndss/ShihOKP17.

[156] Yangchun Fu, Erick Bauman, Raul Quinonez, and Zhiqiang Lin. "SGX-LAPD: Thwarting Controlled Side Channel Attacks via Enclave Verifiable Page Faults". In: *Research in Attacks, Intrusions, and Defenses*. Cham: Springer International Publishing, 2017, pp. 357–380.

[157] Swarup Chandra, Vishal Karande, Zhiqiang Lin, Latifur Khan, Murat Kantarcioglu, and Bhavani Thuraisingham. "Securing Data Analytics on SGX with Randomization". In: *Computer Security – ESORICS 2017*. Ed. by Simon N. Foley, Dieter Gollmann, and Einar Snekkenes. visited on 02/Jun/2019. Cham: Springer International Publishing, 2017, pp. 352–369. URL: https://doi.org/10.1007/978-3-319-66402-6_21.

[158] Jaebaek Seo, Byoungyoung Lee, Seong Min Kim, Ming-Wei Shih, Insik Shin, Dongsu Han, and Taesoo Kim. "SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs". In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. visited on 03/Jun/2019. Reston, VA: The Internet Society, 2017. URL: http://dblp.org/rec/bibtex/conf/ndss/SeoLKSSHK17.

[159] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. "Path ORAM: An Extremely Simple Oblivious RAM Protocol". In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS '13. visited on 03/Jun/2019. ACM. Berlin, Germany: ACM, 2013, pp. 299–310.

[160] T. P. Thao, A. Miyaji, M. S. Rahman, S. Kiyomoto, and A. Kubota. "Robust ORAM: Enhancing Availability, Confidentiality and Integrity". In: *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. visited on 03/Jun/2019. Christchurch, New Zealand: IEEE, Jan. 2017, pp. 30–39.

[161] Ahmad Atamli-Reineh and Andrew Martin. "Securing Application with Software Partitioning: A Case Study Using SGX". In: *Security and Privacy in Communication Networks*. Ed. by Bhavani Thuraisingham, XiaoFeng Wang, and Vinod Yegneswaran. Cham: Springer International Publishing, 2015, pp. 605–621.

[162] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. *Integrating Remote Attestation with Transport Layer Security*. Tech. rep. visited on 03/Jun/2019. Portland, OR, USA: Intel Corporation, 2018. URL: https://arxiv.org/abs/1801.05863.

[163] Michael Steiner, Thomas Knauth, Li Lei, Bin Xing, Mona Vij, and Somnath Chakrabarti. "Technology For Establishing Trust During A Transport Layer Security Handshake". Pat. visited on 03/Jun/2019. US Patent App. 16/174,337. Intel Corporation. In Google Patents. Feb. 2019. URL: https://patents.google.com/patent/US20190065406A1/en.

[164] Michael Schwarz, Samuel Weiser, and Daniel Gruss. "Practical Enclave Malware with Intel SGX". In: *arXiv preprint arXiv:1902.03256* (2019). visited on 03/Jun/2019. arXiv: 1902.03256. URL: http://arxiv.org/abs/1902.03256.

[165] Victor Costan and Srinivas Devadas. "Intel SGX Explained." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 86.

[166] Simon Singh. *The code book: the science of secrecy from ancient Egypt to quantum cryptography*. Anchor, 2000.

[167] Arvind Seshadri, Adrian Perrig, Leendert Van Doom, and Pradeep Khosla. "SWATT: SoftWare-based ATTestation for embedded devices". In: *Proceedings - IEEE Symposium on Security and Privacy* 2004 (2004), pp. 272–282.

[168] Yanlin Li, Jonathan M. McCune, and Adrian Perrig. "SBAP: Software-based attestation for peripherals". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6101 LNCS (2010), pp. 16–29.

[169] Markus Jakobsson and Karl Anders Johansson. "Practical and secure software-based attestation". In: *Proceedings - 2011 Workshop on Lightweight Security and Privacy: Devices, Protocols, and Applications, LightSec 2011* (2011), pp. 1–9.

[170] Claude Castelluccia, Aurélien Francillon, Daniele Perito, and Claudio Soriente. "On the difficulty of software-based attestation of embedded devices". In: *Proceedings of the 16th ACM conference on Computer and communications security - CCS '09* (2009), p. 400. URL: http://portal.acm.org/citation.cfm?doid=1653662.1653711.

[171] Cas Cremers. "Compositionality of security protocols: A research agenda". In: *Electronic Notes in Theoretical Computer Science* 142.SPEC. ISS. (2006), pp. 99–110.

[172] Nickolai Zeldovich. "SECURING UNTRUSTWORTHY SOFTWARE USING INFORMATION FLOW CONTROL". In: *PhD Thesis* (2007). URL: http://www.scs.stanford.edu/{~}nickolai/papers/thesis-phd.pdf.

[173] Thomas F J Pasquier. "Towards practical information flow control and audit". In: *Technical reports* 893 (July 2016). URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-893.pdf.

[174] Virgil Gligor. "Dancing with the Adversary: A Tale of Wimps and Giants". In: *Security Protocols XXII*. Ed. by Bruce Christianson, James Malcolm, Vashek Matyáš, Petr Švenda, Frank Stajano, and Jonathan Anderson. Cham: Springer International Publishing, 2014, pp. 100–115.

[175] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. *The Rocket Chip Generator*. Tech. rep. UCB/EECS-2016-17. EECS Department, University of California, Berkeley, Apr. 2016. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html.

[176] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. "Keystone: An open framework for architecting trusted execution environments". In: *Proceedings of the Fifteenth European Conference on Computer Systems*. 2020, pp. 1–16.

[177] HA Zschippang, Sascha Weikert, Kubilay Ahmet Küçük, and Konrad Wegener. "Face-gear drive: Geometry generation and tooth contact analysis". In: vol. 142. Elsevier, 2019, p. 103576. URL: https://doi.org/10.1016/j.mechmachtheory.2019.103576.

[178] H Andreas Zschippang, Natanael Lanz, K Ahmet Küçük, Sascha Weikert, and Konrad Wegener. "Face-gear drive: Assessment of load sharing, transmission characteristics and root stress based on a quasi-static analysis". In: vol. 151. Elsevier, 2020, p. 103914. URL: https://doi.org/10.1016/j.mechmachtheory.2020.103914.

[179] Intel Corporation. "Intel Software Guard Extensions Programming Reference". In: *Intel* (2014). URL: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[180] Trusted Computing Group. "Trusted Platform Module (TPM) Main Specification". In: *TCG* (2003). https://trustedcomputinggroup.org/tpm-main-specification/ Last Accessed 26 Feb 2018.

[181] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. "Architectural support for copy and tamper resistant software". In: *Acm Sigplan Notices* 35.11 (2000), pp. 168–177.

[182] G Edward Suh, Charles W O'Donnell, and Srinivas Devadas. "Aegis: A single-chip secure processor". In: *IEEE Design & Test of Computers* 24.6 (2007), pp. 570–580.

[183] David Champagne and Ruby B Lee. "Scalable architectural support for trusted software". In: *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE. 2010, pp. 1–12.

[184] Christopher W Fletcher, Marten van Dijk, and Srinivas Devadas. "A secure processor architecture for encrypted computation on untrusted programs". In: *Proceedings of the seventh ACM workshop on Scalable trusted computing*. 2012, pp. 3–8.

[185] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. "Phantom: Practical oblivious computation in a secure processor". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 311–324.

[186] Victor Costan, Ilia Lebedev, and Srinivas Devadas. "Sanctum: Minimal hardware extensions for strong software isolation". In: *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 2016, pp. 857–874.

[187] Reshma Lal, Luis S Kida, and Soham Jayesh Desai. *Technologies for establishing secure channel between i/o subsystem and trusted application for secure i/o data transfer*. US Patent App. 16/369,303. July 2019.

[188] Santosh Ghosh, Kirk Yap, and Siddhartha Chhabra. *Systems, methods and apparatus for low latency memory integrity mac for trust domain extensions*. US Patent App. 16/021,496. Feb. 2019.

[189] Arm Limited. *Arm Architecture Reference Manual Supplement - Morello for A-profile Architecture. DDI0606*. Tech. rep. Retrieved 2020-12-17 from https://documentation-service.arm.com/static/5fc52df386375544ec188e19, 2020.

[190] AMD SEV-SNP. "Strengthening VM isolation with integrity protection and more". In: *White Paper, January* (2020).

[191] Guerney Hunt and Ram Pai. *IBM Protected Execution Facility (PEF)*. IBM Research. Last Accessed 17 Dec 2020 https://www.platformsecuritysummit.com/2019/speaker/hunt/. Oct. 2019.

[192] ARM. "ARM Security Technology. Building a Secure System using TrustZone Technology ARM". In: *ARM white paper* (2009), p. 108. URL: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.

[193] N Asokan, Jan-erik Ekberg, and Kari Kostiainen. "The Untapped Potential of Trusted Execution Environments on Mobile Devices". In: *Financial Cryptography*. Vol. 7859. 2013, pp. 293–294. URL: http://link.springer.com/10.1007/978-3-642-39884-1_24.

[194] Global Platform. "Technical Device Specifications, Trusted Execution Environment (TEE)". In: *Global Platform Online* (2017). https://www.globalplatform.org/specificationsdevice.asp Last Accessed 26 Feb 2018.

[195] Trustonic. " Trustonic Technical Resources and White Papers". In: *Trustonic Documents* (2017). https://www.trustonic.com/solutions/downloads/ Last Accessed 26 Feb 2018.

[196] Kinibi and Trustonic. "Kinibi v311A Security Target". In: *Kinibi Security Target ST* (2017), pp. 0–92. URL: https://www.ssi.gouv.fr/uploads/2017/02/anssi_cc-2017_03-cible-publique.pdf.

[197] Cuihtlauac Alvarado. *Telecommunication Terminal Comprising Two Execution Spaces*. 2008. URL: https://patentimages.storage.googleapis.com/15/26/4f/4b602337534c78/US20080032668A1.pdf.

[198] Axelle Apvrille, Roquefort Les Pins, and Frey. *System and method for securing data*. 2011. URL: https://patentimages.storage.googleapis.com/3f/2e/50/2bbce3a6d70362/US20110162083A1.pdf.

[199] Ahmad Atamli-Reineh, Ravishankar Borgaonkar, Ranjbar A Balisane, Giuseppe Petracca, and Andrew Martin. "Analysis of Trusted Execution Environment usage in Samsung KNOX". In: *Proceedings of the 1st Workshop on System Software for Trusted Execution - SysTEX '16*. 2016, pp. 1–6. URL: https://dl.acm.org/doi/10.1145/3007788.3007795.

[200] Gal Beniamini Google Project Zero. " Trust Issues: Exploiting TrustZone TEEs ". In: *Google Project Zero Blog* (2017). https://googleprojectzero.blogspot.ch/2017/07/trust-issues-exploiting-trustzone-tees.html Last Accessed 26 Feb 2018.

[201] ARM. "TrustZone Technology for the ARMv8-M Architecture Version 2.0". In: *ARM Online* (2016). URL: https://static.docs.arm.com/100690/0200/armv8m_trustzone_technology_100690_0200.pdf.

[202] Intel Corporation. "Intel Software Guard Extensions Enclave Writer's Guide version 1.02". In: *Intel* (2015). URL: https://software.intel.com/sites/default/files/managed/ae/48/Software-Guard-Extensions-Enclave-Writers-Guide.pdf.

[203] Intel Corporation. "Intel Software Guard Extensions Developer Guide v2.1". In: *Intel* (2018). URL: `https://download.01.org/intel-sgx/linux-2.1/docs/Intel_SGX_Developer_Guide.pdf`.

[204] Ittai Anati. "TEE - More than just a secure container The importance of TCB updates". In: (2016). URL: `https://systex16.ibr.cs.tu-bs.de/slides/anati-systex16-slides.pdf`.

[205] Microsoft Research. *Microsoft OpenEnclave SDK*. Microsoft Research. Last Accessed 17 Dec 2020 https://openenclave.io/sdk/.

[206] Christian Priebe, Divya Muthukumaran, Joshua Lind, Huanzhou Zhu, Shujie Cui, Vasily A Sartakov, and Peter Pietzuch. "SGX-LKL: Securing the host OS interface for trusted execution". In: *arXiv preprint arXiv:1908.11143* (2019).

[207] Fortanix Research. *Fortanix Enclave Development Platform*. Rust EDP. Last Accessed 17 Dec 2020 https://edp.fortanix.com/.

[208] Asylo Authors. *Asylo: An open and flexible framework for enclave applications.* Google Asylo. Last Accessed 17 Dec 2020 https://asylo.dev/.

[209] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Dawn Song, and Krste Asanović. "Keystone: A framework for architecting TEEs". In: *arXiv preprint arXiv:1907.10119* (2019).

[210] Job Noorman, Jo Van Bulck, Jan Tobias Mühlberg, Frank Piessens, Pieter Maene, Bart Preneel, Ingrid Verbauwhede, Johannes Götzfried, Tilo Müller, and Felix Freiling. "Sancus 2.0: A low-cost security architecture for IoT devices". In: *ACM Transactions on Privacy and Security (TOPS)* 20.3 (2017), pp. 1–33.

[211] Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D Garcia, and Frank Piessens. "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 1741–1758.

[212] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution". In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2019, pp. 142–157.

[213] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. "Spectre attacks: Exploiting speculative execution". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 1–19.

[214] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. "Meltdown: Reading kernel memory from user space". In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 973–990.

[215] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. "Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution". In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 991–1008.

[216] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. "Malware guard extension: Using SGX to conceal cache attacks". In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2017, pp. 3–24.

[217] Intel Corporation. *Performance Monitoring Impact of Intel Transactional Synchronization Extension Memory*. Document Number: 604224. Last Accessed 28 Feb 2021. Mar. 2019. URL: http://cdrdv2.intel.com/v1/dl/getContent/604224.

[218] Joanna Rutkowska. "Thoughts on Intel's upcoming Software Guard Extensions (Part 1)". In: The Invisible Things Lab's blog. Last Accessed 28 Feb 2021. ITL, 2013. URL: https://theinvisiblethings.blogspot.com/2013/08/thoughts-on-intels-upcoming-software.html.

[219] Joanna Rutkowska. "Thoughts on Intel's upcoming Software Guard Extensions (Part 2)". In: The Invisible Things Lab's blog. Last Accessed 28 Feb 2021. ITL, 2013. URL: http://theinvisiblethings.blogspot.com/2013/09/thoughts-on-intels-upcoming-software.html.

[220] Marion Marschalek. "The Wolf In SGX Clothing." In: Bluehat IL. Jan. 2018. URL: https://www.infosecurity.us/blog/2018/2/7/bluehat-il-2018-marion-marschalek-s-the-wolf-in-sgx-clothing.

[221] Michael Schwarz, Samuel Weiser, and Daniel Gruss. "Practical enclave malware with Intel SGX". In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2019, pp. 177–196.

[222] Tamas Rudnai. *Dispelling Myths Around SGX Malware*. Symantec Enterprise Blogs / Threat Intelligence. Last Accessed 28 Feb 2021. Apr. 2019. URL: https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/sgx-malware-explainer.

[223] Broadcom Technical Documents. *Symantec Endpoint Protection Memory Exploit Mitigation techniques*. Symantec Endpoint Protection Installation and Administration Guide. Last Accessed 28 Feb 2021. Feb. 2021. URL: https://techdocs.broadcom.com/us/en/symantec-security-software/endpoint-security-and-management/endpoint-protection/all/Using-policies-to-manage-security/hardening-windows-clients-against-memory-tampering-v123149922-d3859e12879/memory-exploit-mitigation-techniques-v114221375-d53e13441.html.

[224] J van Prooijen. "The design of malware on modern hardware: Malware inside Intel SGX enclaves". In: *University of Amsterdam, Tech. Rep* (2016), pp. 2015–2016.

[225] Alpesh Bhudia, Daniel O'Keeffe, Daniele Sgandurra, and Darren Hurley-Smith. "RansomClave: Ransomware Key Management using SGX". In: *The 16th International Conference on Availability, Reliability and Security*. 2021, pp. 1–10.

[226] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. "Paybreak: Defense against cryptographic ransomware". In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 2017, pp. 599–611.

[227] Flavio Toffalini, Mariano Graziano, Mauro Conti, and Jianying Zhou. "SnakeGX: a sneaky attack against SGX Enclaves". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2021, pp. 333–362.

[228] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. "Malware Guard Extension: abusing Intel SGX to conceal cache attacks". In: *Cybersecurity* 3.1 (2020), pp. 1–20.

[229] Rafal Wojtczuk and Joanna Rutkowska. "Following the White Rabbit: Software attacks against Intel VT-d technology". In: 2011. URL: https://invisiblethingslab.com/resources/2011/Software%20Attacks%20on%20Intel%20VT-d.pdf.

[230] Rafal Wojtczuk. "A stitch in time saves nine a stitch in time saves nine: A case of multiple os vulnerability". In: Citeseer, 2012. URL: https://media.blackhat.com/bh-us-12/Briefings/Wojtczuk/BH_US_12_Wojtczuk_A_Stitch_In_Time_WP.pdf.

[231] Alexander Tereshkin and Rafal Wojtczuk. "Introducing ring-3 rootkits". In: 2009. URL: https://invisiblethingslab.com/resources/bh09usa/Ring%20-3%20Rootkits.pdf.

[232] Rafal Wojtczuk and Alexander Tereshkin. "Attacking intel bios". In: 2009. URL: https://invisiblethingslab.com/resources/bh09usa/Attacking%20Intel%20BIOS.pdf.

[233] Rafal Wojtczuk and Joanna Rutkowska. "Attacking Intel TXT via SINIT code execution hijacking". In: 2011. URL: https://invisiblethingslab.com/resources/2011/Attacking_Intel_TXT_via_SINIT_hijacking.pdf.

[234] Rafal Wojtczuk, Joanna Rutkowska, and Alexander Tereshkin. "Another way to circumvent Intel trusted execution technology". In: 2009, pp. 1–8. URL: https://invisiblethingslab.com/resources/misc09/Another%20TXT%20Attack.pdf.

[235] Rafal Wojtczuk and Joanna Rutkowska. "Attacking intel trusted execution technology". In: vol. 2009. 2009, pp. 1–6. URL: https://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20paper.pdf.

[236] Joanna Rutkowska and Rafał Wojtczuk. "Preventing and detecting Xen hypervisor subversions". In: 2008. URL: https://invisiblethingslab.com/resources/bh08/part2-full.pdf.

[237] Rafal Wojtczuk and Joanna Rutkowska. "Attacking SMM memory via Intel CPU cache poisoning". In: *Invisible Things Lab* (2009), pp. 16–18.