Thesis: Outline

Oberon Swings

May 7, 2022

Contents

Chapter 1

Introduction

Smartphone functionality. Smartphones have become an essential part of our daily lives and everyone is assumed to have one of their own. These devices can be spotted everywhere, people use them at home, on the bus or even at work. In most cases these phones are only occasionally used for text messaging or calling but very often for reading e-mails, surfing on the web or even for services like e-banking. Because of this wide range of functionality some people may even replace their personal computer by a smartphone entirely. The success of smartphones lies in their ease of use and always being accessible, people just carry them in their pocket. Besides the user having access to their phone all the time, a smartphone also has or could have access to the internet all the time. It is known that the internet is the gate to lots of services which are perceived as necessities lately.

IoT security. Devices that make lots of connections on the go while also utilizing online services for which sensitive data is required may introduce security vulnerabilities. The fact that people are using their smartphones for services like online banking or even consulting health related reports implies that some sensitive data must be stored on these devices or at least present while they are interacting with it. While this data should be protected very well it is present on an Internet of Things (IoT) device for which security solutions and standards present today are not adequate in terms of protection against the existing threats. The hardware similarities between smartphones and IoT devices are more prominent than one might expect, this is because the architecture of smartphones stems from that of IoT devices. The main issue here is that IoT devices are designed for performance, they only have a small number of tasks but these need to be executed as fast or as energy efficient as possible. This also applies to smartphones because while the functionality of a smartphone is close to that of a personal computer the hardware is not. This weak link in the hardware gives rise to multiple different attack strategies that adversaries can utilize to steal sensitive data from smartphone users. Lately improvements have been made in this area by

extending the processors of these devices with features that make it possible to setup a Trusted Execution Environment (TEE) on them. A TEE can increase the security of an IoT device, this is often achieved by utilizing core security services for critical operations. Examples of these critical operations are cryptographic operations, storing data in secure memory or accessing Input and Output (I/O) through trusted paths.

Our contributions. This thesis is based on the reproduction of existing work namely [Lin+21], the solution of this paper is replicated as closely as possible. The proposed system attempts to increase the security of a smartphone by having the TEE attest the code of the user and Operating System (OS) space. To allow for a direct comparison between the performance achieved in the paper and our implementation some experiments in the paper are redone. After this comparison more elaborate experiments are executed to give a better view on the trade offs between performance and increased security. To allow others to easily reproduce or review the work that has been done all code and experiment setups are made available in open source. The performance is of course only a small aspect of the analysis of the solution, to do the security analysis, this work (and the solution of the initial paper) are compared to similar work. In this comparison it is discussed whether the solution is the most effective out of the existing ones, which other alternatives may achieve more in terms of defended attacks or achieved security guarantees. Finally it is evaluated which type of solution is the most promising as the direction for future work based on the comparison with the similar alternatives.

Outline. In the next chapter more background information about among other things Remote Attestation and ARM TrustZone will be given. In the third chapter the methods to solve the problem will be explained. In chapter four and five the implementation of the attestation program are elaborated upon and the outcome of the experiments will be made clear respectively. The sixth chapter will conclude this thesis informing the reader about related work and future improvements. The final chapter will give a discussion on the presented work.

Chapter 2

Background

2.1 Remote Attestation

Goals

Supplying evidence about a target to a verifier is the most important goal for remote attestation. [Cok+11] defines attestation as follows:

> "Attestation is the activity of making a claim to an appraiser about the properties of a target by supplying evidence which supports that claim."

Making a claim in this context refers to stating whether the target is in a secure state or not, this is often done implicitly. The appraiser can be seen as the verifier, it receives the evidence (and the claim) and will decide based on those whether the claim is valid and the target is still trusted. Remote attestation is achieved when the verifier is a remote service provider which is accessed through a network.

Requirements

The target must adhere to certain constraints to provide the necessary abilities for correct attestation. First of all a trusted base is needed that can enforce an isolation mechanism to avoid it being tampered with. This isolation will make sure that the attestation can be executed even when the target is unreliable. Another important aspect of the attestation is the ability to measure useful aspects of the system, this means that there needs to be structure to these measurements to be able to understand them. When these measurements are requested, they need to be executed in a trusted manner and the results need to be sent to the verifier securely.

The verifier needs comprehensive and fresh information about the target to be able to correctly attest it. Based on this information the verifier makes

decisions on the reliability and trustworthiness of the target. These decisions are referred to as attestations, it should be possible to draw conclusions from multiple attestations or make predictions based on them. Besides a complete set of information about the target the verifier also needs proof of the trustworthiness of this information because it is used as evidence for the decision making process.

Techniques

Integrity Measurement Architecture implements attestation by measuring the code of programs before running them on the target. The verifier can check whether the program's code has been modified based on these measurements [YF08] [JSS06] [Dua+20] [KBC21]. According to [Ala+12] and many others Integrity Measurement Architecture (IMA) is inflexible and static because updates for programs are hard to take into account. The solution is also very limited because there are a variety of ways a program can misbehave without the code base being tampered with.

Attestation on Program Execution is an important step in the right direction, it measures the dynamic behavior of the program. [Gu+08] propose to observe the system calls the program makes to verify whether it adheres to the permitted control flow. Although it is a big improvement there are still weaknesses like the granularity of a system call might not give enough details and the behavior of a system is much more than the system calls alone. Many similar solutions that focus on the dynamic behavior of code have been proposed all with their weaknesses and shortcomings [Qin+20] [Ali+17] [SKU10] [Ba+17].

Combined strategies are being proposed more and more often due to them providing protection against a wider variety of vulnerabilities. Model-based Behaviour Attestation was proposed by [Ala+08] which is again mentioned by the same researchers in their analysis about existing techniques [Ala+12]. The platform is expected to enforce a certain security model and the attestation will verify whether the platform behaves accordingly. The behavior of the platform is monitored by a variety of techniques like IMA or Property Based Attestation (PBA) for a 'full picture' approach. PBA is focused on properties that the platform possesses, it is still very hard to map configurations of the platform to certain properties but it does provide lots of flexibility in terms of attestation. [MNP16] have also combined a variety of approaches to attest the trust of IoT devices and implemented multiple modules that attest certain platform properties based on different measurements.

2.2 Trusted Execution Environment

Definitions

A definition of a Trusted Execution Environment is given by [SAB15]:

> "Trusted Execution Environment (TEE) is a tamper-resistant processing environment that runs on a separation kernel. It guarantees the authenticity of the executed code, the integrity of the runtime states (e.g. CPU registers, memory and sensitive I/O), and the confidentiality of its code, data and runtime states stored on a persistent memory. In addition, it shall be able to provide remote attestation that proves its trustworthiness for third-parties. The content of TEE is not static; it can be securely updated. The TEE resists against all software attacks as well as the physical attacks performed on the main memory of the system. Attacks performed by exploiting back door security flaws are not possible."

The requirements to achieve a secure and trusted execution environment are largely accomplished by the separation kernel. This kernel simulates a distributed system which divides the system into strongly isolated partitions with different security levels. For instance data in one partition cannot be leaked by shared resources because they are sanitized and cannot be read or modified by other partitions. A partition also needs to give explicit permission before others are able to communicate with it and a security breach in a partition cannot impact any other partitions.

Trust is a very important aspect of a TEE, there are multiple types of trust with different origins. Static trust is measured only once, before deployment in most cases and assumed to never change during the lifetime of the device. Dynamic trust on the other hand is based on the state of the system and this state changes continuously. In this latter case the trust needs to be measured periodically to have an up to date view on it. To be able to do these measurements a trusted entity is required, this is because trust cannot be created but needs to be transferred from the Root of Trust (RoT) to the component that is being measured. To reach this final target, a chain will be created which links intermediate components by having the next one build upon the trust of the previous component and this is how a Chain of Trust (CoT) is constructed.

Building blocks

The Root of Trust is the starting point for the secure boot process which assures that only code with certain properties is given control. For instance during the boot process checking the integrity of the succeeding component before loading it and giving it control will construct a Chain of Trust. This

CoT is necessary because that is the basis of the trust of the separation kernel. This RoT is often implemented using some hardware component that is trusted [MAT+21] [Fot+21] [Kin06] [Zha+21].

The separation kernel is a very important component of the architecture because it is responsible for the secure scheduling and information flow control. The secure scheduling makes sure that the TEE doesn't affect the rich OS too much to allow the latter to remain responsive and meet real-time requirements. Information flow control is tightly coupled with the inter-environment communication, this is an interface which allows communication between TEE and the rest of the system. To avoid security risks from this communication it needs to adhere to the following guidelines: reliable isolation, minimum overhead and protection of the communication structures. As shown in these papers [Van+19] [Guo+21] [KM20] [Mac+17] security risks in implementations of these components do exist and they need to be dealt with to make sure the TEE operates as it is expected to.

The TEE is where Trusted Applications (TA) run, it also has a trusted kernel which is kept as minimalistic as possible. The minimalism of the kernel is to avoid software bugs which could introduce security vulnerabilities. The trusted kernel provides services like secure memory and trusted I/O which are important features to allow the system to be used in a secure manner. Secure memory ensures confidentiality, integrity and freshness of stored data. The trusted I/O protects authenticity and confidentiality of communication between TEE and peripherals.

2.3 ARM TrustZone

Core principles

ARM TrustZone [ARM22a] implements the TEE on the processor level which means it runs below the hypervisor or OS [PS19]. This approach divides the system into two main partitions namely the Normal World (NW) and the Secure World (SW). The processor executes either in the NW or in the SW, these environments are completely isolated in terms of hardware and the SW is more privileged to make sure the NW doesn't behave in a malicious way. The partition between these worlds gives rise to new and better security solutions for applications running on these types of System on Chips (SoC) [Len+18] [Esk+18] [Cha+17] [Fer+17]. The SW can provide services like data storage, I/O and virtualization all with hardened security guarantees because of these hardware features.

Implementation

The Normal and Secure World are the two main environments in which the processor will be executing code. The Normal World shelters the rich OS (like Linux), it is mainly due to the size of these operating systems that they cannot be trusted to run in the secure world. The risk of there being implementation bugs that introduce security risks is too high. Also user level applications run in the NW, for peripherals for instance they rely on the rich OS and depending on the service the rich OS relies on the Secure World, some services can also be requested from the user application directly to the SW. The Secure World is where the trusted kernel runs, the implementation of this component is kept very minimal and needs to be designed and implemented very securely to avoid vulnerabilities.

The NS-bit is the 33rd bit (in a 32-bit architecture) that flows through the entire pipeline and can be read from the Secure Configuration Register (SCR) to identify the world in which the operation is being executed. The processor has a third state which is the monitor state, this is necessary to preserve and sanitize the processor state when making transitions between NW and SW states. The new privileged instruction Secure Monitor Call (SMC) allows both worlds to request a world switch and the monitor state will make sure this is handled correctly. The only other way of getting into the monitor state is with exceptions or interrupts from the Secure World.

TZ Address Space Controller (TZASC) can be used to configure specific memory regions as secure or non-secure, such that applications running in the secure world can access memory regions associated with the normal world, but not the other way around. Making these partitions is also performed by the TZASC which is made available through a programming interface only available from within the secure world. A similar approach is taken for off-chip ROM and SRAM, this is implemented using the TrustZone Memory Adapter (TZMA). Whether these components are available or not and how fine grained the memory can be partitioned depends on the SoC because they are optional and configurable.

TZ Protection Controller (TZPC) is in the first place used to restrict certain peripherals from worlds, for instance to only allow the secure world to access them. It also extends the Generic Interrupt Controller (GIC) with support for prioritized interrupts from secure and non-secure sources. This prioritization is important to avoid Denial of Service (DoS) attacks on the secure world.

2.4 PinePhone

ARM TrustZone is used in the System on Chip (SoC) of the PinePhone which is a popular solution amongst smartphone suppliers, Samsung KNOX [Sam22] and Android KeyStore [Goo22] are two examples of this. It is evident that this solution has a lot of potential and the industry believes in it's potential but often the technical details are not disclosed which gives the academic world little opportunities to build upon this technology [PS19].

Open source smartphone is the best way to describe the PinePhone, this is a powerful tool for researchers and developers to learn how to use the ARM TrustZone framework. The PinePhone is a pioneer in this aspect because their hardware developments are all open to inspect and they are take into account the development ideas of their community [PIN21]. Not only the hardware that is used is made 'open source' but the main operating system is Linux [PIN22] which enables the user to control every nook and cranny of the hardware.

2.5 OP-TEE

OP-TEE stands for Open Portable TEE [OPT22a], it is an open source implementation of the API's that are exposed to Trusted Applications (TA) and that communicate with the TEE. It was designed with ARM TrustZone in mind but is applicable to other realizations of TEEs as well. The main design principles applied when creating OP-TEE were isolation, small footprint and portability. While the two first principles seem logical and have to do a lot with the security of the final product the portability is not straight forward but a nice feature to allow a very diverse community to have a related framework which encourages collaborations.

2.6 Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes [Lin+21]

System overview

The threat model assumes that attackers have physical access to the IoT device and are able to launch a wide variety of attacks. The attackers are assumed to be able to tamper with the images of the secure and normal world (including that of the OS) before the device is booted up. Another assumed attack is one where the adversary injects malware into the normal world during runtime and tamper with the normal world applications. Only the security of the text section of a program is considered but on the other hand it is assumed that this code is on ROM that is protected from modification.

Lastly the secure world and remote attestation server are assumed to be trustworthy and secure.

The solution that is proposed uses a hybrid booting method to ensure the load-time integrity and remote attestation to ensure the runtime integrity of the system. Secure boot is used to load the kernel of the secure world, this provides strong guarantees that the secure world starts in a secure and known state. The normal world is booted with what is called trusted boot which uses attestation to provide proof of the integrity of the image that is being started. Before the control is given to the rich OS it's image is measured and after it has started it should send this to the remote attestation server to verify the measurement. The remote attestation service is implemented in the secure world. The memory pages of the rich OS are periodically measured, encrypted by an attestation key and sent to the remote attestation server for verification.

Hybrid booting

Secure boot starts with a Root of Trust (RoT), which in this case is achieved by using the OCROM and eFuse of the IoT device. The first-stage bootloader is encrypted with a private key and stored on the OCROM to verify the integrity during the boot phase. The hash of the public key is stored in the eFuse to verify it's integrity during the boot phase. The images of the second-stage bootloader and secure kernel are measured and signed before deploying the device. These measurements and signatures are stored in the flash memory while the hash of the public key of the secondary bootloader is stored in the eFuse. The hash of the public key of the trusted kernel is stored in the secondary bootloader to achieve an incremental chain.

The secure boot phase starts with the first-stage bootloader which locates the second-stage bootloader, the public key and its signature. Secondly the first-stage bootloader calculates the hash of the public key and verifies the integrity of the public key. After successful verification it uses the public key to obtain the measurement result for the second-stage bootloader. Finally the first-stage bootloader calculates the hash of the second-stage bootloader and verifies its correctness. The second-stage bootloader does this entire process for the secure kernel to complete the secure boot phase.

Trusted boot is setup by producing a hash chain, the image of the rich OS is hashed first and concatenated with the image of the file system and hashed again. This final hash value is stored in the remote attestation server, during run-time this hash value will need to be sent to the remote attestation server in a secure way. To achieve secrecy a symmetric key is used, storing this in the IoT device is not trivial so this is solved with the following method. The Cryptographic Acceleration and Assurance Module (CAAM) is used to execute cryptographic functions in a secure environment. This module is used

to generate a 256-bit blob key, this key is used to encrypt the attestation key. A MAC is calculated from the attestation key to ensure its integrity. The blobkey is itself encrypted with a Blob Key Encryption Key (BKEK) which is derived from the master key (MK) by the CAAM. The MK is stored in Secure Non-Volatile Storage (SNVS) which is assumed to be secure by default.

The trusted boot phase starts with the NW attestation client application establishing a TLS connection with the remote attestation server and requesting a nonce. The measurement Trusted Application (TA) restores the attestation key from the Blob using the CAAM. The TA measures the rich OS and filesystem images, appends the nonce to the final hash value and encrypts this combination with the attestation key. The encrypted text is put in shared memory to allow the client application to send it to the remote attestation server. On the remote attestation server the cyphertext is decrypted and verified to check for integrity violations and replay attacks.

Page-based attestation

The idea is to measure the code segments of the programs in the normal world on the IoT device, it is assumed that this code base does not change in the lifetime of the device. The secure world is trusted but the normal world is still vulnerable to attackers, that is why attesting the code in the normal world would increase the security for the applications running in the normal world. The measurement is done on pages of 4KB at a time so programs will have multiple tuples of the form {*process-name, page-hash*} which will later be used to verify the integrity of the process. The first measurement is done before deploying the IoT device and the results are stored on the remote attestation server to be able to compare the future measurements with.

Process integrity measurement starts with the measurement Trusted Application which resides in the Secure World, it requests the memory address of the initial process. The client application translates the virtual address of the *init_proc* into a physical address which is later translated to the virtual address in the secure world memory address space. With this address the measurement TA iterates over all processes and measures their code pages. This measuring method uses the *task_struct* which has a doubly-linked-list structure so it enables the TA to find all processes.

Process integrity attestation uses the measurement of the process integrity measurement stage. First the IoT device requests a nonce from the remote attestation server with which a TLS connection is established. The measurement TA encrypts the measurement results concatenated with the nonce using the attestation key. The measurements of the processes are encrypted individually meaning that a set of cyphertexts is sent to the remote attestation server. The remote attestation server decrypts the measurements

of the processes and checks whether integrity violations can be found, this means new software or old software that has been modified.

Evaluation

The effectiveness of the secure boot process is measured by whether the secure boot phase is able to detect any violations against the integrity of the images, the signatures or the public key which it does correctly. For the trusted boot the focus lies on whether the remote attestation server is able to identify an abnormal system status, this is the case because NW programs can still be executed but the remote attestation server will verify the system state. The process integrity attestation is tested by tampering with existing programs and inserting additional programs, both these cases are also picked up on by the attestation.

Performance of the boot procedures is measured by comparing the mean time of 30 iterations with secure and trusted boot and 30 iterations without it. The secure boot adds little overhead on the second-stage bootloader while the trusted boot almost doubles the time it takes for the secure kernel to boot. The main reason why the secure kernel takes this long is because the image of the filesystem and rich OS is rather large and takes some time to measure. The overhead of the measurement TA and the attestation CA is measured by calling rich OS services while these modules are running and with them disabled. The overhead these modules introduce is between -0.55% and $+0.67\%$.

Security analysis is executed on the hybrid booting approach and the page-based process attestation. In the booting method a Chain of Trust (CoT) is constructed from the Root of Trust (RoT) residing in the eFuse and OCROM. A successful secure boot ensures that the secure world can be seen as the secure base from which the normal world can be booted. If the normal world image is tampered with the remote attestation server will pick up on this threat. The execution of the measurement TA and the results it generates are both secure because of the isolation in the secure world. The results pass through the normal world but they are encrypted at this stage, the encryption key is also securely stored in the secure world giving the normal world no opportunity to get hold of the information. The main drawback of this approach is that the method relies on the rich OS to access the paging structure and process management kernel objects.

Chapter 3

Method

3.1 Detailed Problem

Smartphones often store, use and transmit sensitive data of their owners. Transmission will often happen on a 4G network because smartphones are frequently used on the go. This form of mobile computing implies that lots of people in the near surroundings of the smartphone can pick up on these signals. The security of these transmission methods like 4G are already extensively analyzed [Fer+18] [ZW21] [FZS22] but there is still room for improvements because vulnerabilities like a paging storm attack [FY20] or session hijacking [Lu+20] are still possible. 4G is of course not the only communication channel available on a smartphone, Bluetooth is a very popular option to connect devices to a Personal Area Network (PAN). While very popular there are still many security risks and possible attacks against it like reflection attack [CE21], cross technology pivoting attack (forcing a different protocol to be used) [Cay+21], Bluejacking, Bluesmacking and Bluesnarfing [PWR21] and finally entropy downgrade attacks which make brute force attacks on the keys possible [ATR20]. Certain security risks related to data transmission do not have a direct impact on the user but could have future implications as [Zhu+21] show, they found a way to identify the smartphone OS based on smartphone traffic which could give attackers insight in what vulnerabilities to use. Although very relevant, the transmission stage is not the only option for data to leak, it still needs to be protected during storage and when it is used at execution time.

Adversaries have many possibilities when it comes to stealing sensitive data from smartphone users while the data is being used during execution or stored on the device. [Jav+20] for instance proposed AlphaLogger which infers the letters being typed on a soft keyboard based on the vibrations and [Son+20] we're able to achieve identity theft through data cloning of auto login credentials. Besides custom attacks, there are plenty of well known risks as

well. [Set+20] have executed a review of various malicious software threats and mitigations, [GRA21] researched software attacks that take advantage of hardware resources to conduct fault injection or sidechannel analysis and [KKP20] reviewed the possible threats that can be introduced by smartphone providers altering the Android OS to add their signature flavor. These attacks are possible due to the fact that the design of smartphones is based on that of IoT devices, which means that lots of focus lies on the performance of the final product. This performance is often hard to achieve because the resources want to be kept to a minimum to lower the price or keep the device small. Security is still seen as a performance killer, which is very unfortunate because it should have an essential role in the design and implementation of a system. The security of a system like smartphones is even more crucial because lots of people are unaware of the possible threats they face.

3.2 System Model

The system model is a smartphone that is owned by a user but for which multiple software providers offer programs. These providers don't necessarily trust eachother but they do want guarantees that the execution of their software will not be interfered with by software of other providers. The user of the device is also it's owner, they have full control over what software should be able to be installed and run on the platform. This is very different from large smartphone companies where the company has a signature key which is kept secret from the user of the smartphone. Programs that are not signed with this signature key will be rejected for installation on the device. The smartphone in this case is a PinePhone which is equipped with 4 ARM Cortex A53 Cores that are TrustZone enabled. Lots of smartphones have ARM processors as System on Chip (SoC) which means that ARM TrustZone is the ideal candidate for the implementation. It provides additional hardware security features which make sure that the attack surface of the smartphone becomes very narrow and well defined. While providing additional security it only has a minimal impact on performance due to it being implemented on hardware [AS19] [Hua+21]. ARM TrustZone makes sure that there is a Trusted Execution Environment (TEE) available with capabilities like secure memory, trusted I/O and many others. The Trusted kernel is responsible for making partitions of the memory only accessible to the secure world which should then be able to provide secure data storage services to the normal world applications. Secure data storage is necessary to make sure that data from one application cannot be read or modified by another one. Trusted I/O paths on the other hand allow the user application to request I/O features from the Secure World (SW) instead of the rich OS. Because these connection go through the SW the rich OS is not able to inspect or modify the data that is transmitted to the I/O peripheral, this is important in cases where an adversary has gained control over the rich OS. For this TEE to work correctly a trusted kernel is required, for this the implementation of OP-TEE is used

which provides the interfaces to communicate with the TEE and call Trusted Applications (TA) from the normal world. The normal world runs a Mobian Linux distribution as kernel which is specifically written for a smartphone device. The system can be put together using the source of the Mobian distribution [Lin22], OP-TEE [OPT22b], U-boot [DEN22] and the ARM Trusted Firmware [ARM22b]. Unfortunately this was not realized in this work so the QEMU emulator [OPT22c] was used to test the code and run the experiments. QEMU allows to run OP-TEE on a desktop while emulating the TEE, because OP-TEE can run on the PinePhone the code and experiments should be reproducible on the PinePhone if correctly configured with OP-TEE.

3.3 Attacker Model

Physical access brings along lots of risk because the adversary has a variety of possible attacks they could launch from this position. The TEE can, when combined with some specific hardware make these attacks ineffective or a lot harder to execute. For instance reading from the hardware memory becomes ineffective because everything on there that is sensitive is encrypted by the TEE, only the TEE can decrypt this information with the help of a Trusted Platform Module (TPM). Tampering with memory can be made less effective by using secure boot, this ensures that the TEE is started up in a secure and known state from which a trusted base can be ensured. With this trusted base attestation could be run on the memory to check whether inconsistent memory pages can be found before they receive control in case of them being code pages. Another hardware attack is one where a physical back door is exploited, this is assumed to be impossible because the processor has been designed for security purposes and thus no back doors should be available. The main advantage defenders have in terms of hardware attacks is that they are often very hard, reading the physical memory is doable but this should be dealt with by the TEE combined with a TPM. There are still lots of hardware attacks for which TEE's are vulnerable, manipulation of RAM and eFuse bypassing secure boot [Gro+21], micro architectural structures leaking information [Rya19] and electro magnetic analysis of side channels [Buk+18]. These attacks are very advanced and would be really hard to execute but they do exist and not many defense mechanisms are present to protect against them.

OS/Firmware attacks have the ability to compromise all user level applications because the OS is the 'trusted' layer on which these user level applications rely. This is the main area where ARM TrustZone and other TEE implementations make a very big difference in security. ARM TrustZone for instance is implemented below the rich OS which means that the trusted kernel has more privileges than the rich OS, the rich OS has of course more functionality but to achieve this functionality it will in some cases have to rely on the trusted kernel. A TEE increases the security when an OS attack has succeeded by shielding the user level applications from this OS, this is done by

allowing the user level application to request services like trusted I/O and secure memory from the TEE itself without interference of the OS. Examples of this are a container using ARM TrustZone [Hua+21], securing camera and location peripherals [SD21] and checking whether the OS executes system services correctly [Gua+17]. It does not make claims about making OS attacks harder because the vulnerabilities in the rich OS are still there, the normal world could be attested which would allow the detection of an OS attack but the TEE doesn't have special protection mechanisms to avoid it from happening. This is not surprising of course, OS attacks are often a consequence of software bugs in the implementation that give rise to security vulnerabilities which are very hard (if not impossible) to avoid.

Software attacks try to tamper with the control flow of certain program executions or get hold of sensitive data through malicious code. Most of these attacks can be defended against by a TEE implementation or with the help of a TEE. Applications can be isolated from each other making it a lot more difficult for a malicious application to tamper with the execution or data of another one. This isolation can be enforced using virtualization or specialized Trusted Applications (TA), they make sure the application can only be interacted with using a very well defined interface. The virtualization needs to be implemented correctly to make sure no data is leaked in between the partitions, in the case of the TA it can use the TEE functionality to store its data securely. Software attack defense mechanisms based on ARM TrustZone come in many forms, isolate application and secure communication [ZY20], control flow integrity scheme [Kaw+20] and reverse engineering protection [BZ19].

3.4 Solution

As discussed in the attacker model, the TEE (ARM TrustZone in this case) will provide a certain level of security on its own by providing secure and trusted services to user applications. These services can be utilized to achieve secret encryption, trusted I/O paths and secure data storage [PS19]. These services rely on the assumption that the TEE itself is secure and trusted, this can be achieved through secure boot [Jia+17]. Secure boot ensures that the device starts in a known secure state, to achieve this a Root of Trust (RoT) is needed from which a Chain of Trust (CoT) is constructed. The RoT is often implemented by using a Trusted Platform Module (TPM) along with hardware memory specifically designed for secure storage like an eFuse for instance. The CoT ensures that only code that is verified will be able to execute and get control over the device during boot time. When the device is successfully started up using secure boot it can be confidently assumed that the trusted kernel and TEE will work as intended. This of course is only the beginning, a TEE provides a framework which needs to be used to implement secure solutions and defense mechanisms against known attack strategies.

An important application that is built upon the TEE framework is one where the integrity of the control flow, code and data is guaranteed. Achieving this level of security is rather hard, weakening this constraint in the sense of making sure violations to this integrity are detected is achievable. Attestation can be used to check the integrity of an application or running system depending on what properties are looked at to measure the reliability of the target. Remote attestation seems like a weird solution in the context of a smartphone, but the user (which in our case is seen as the owner of the device) could be alerted about the attestation results. This attestation process can run within the TEE on the device and because of this will be tamper proof against software and OS attacks. Notifying the user will also need to be done in a secure manner, for this the trusted I/O paths that ARM TrustZone provides can be used.

Chapter 4

Implementation

4.1 Attestation TA

Trusted application

Hashing the memory pages of the text section of a program is the very first step the TA does when attesting a program.

Storing reference values needs to be done when the device is in a known secure state, the memory pages will be hashed and these hashes are stored for later comparison.

Comparing the hash of a memory page with its reference value is the actual attesting step, from this comparison it should be clear whether the integrity of the memory page has been violated.

Notifying the user is the final step to inform them about the problem to allow them to take action, this could be rebooting to ensure a secure known state again or ask for help from a specialist.

NW OS dependencies

Retrieving address needs to be done from within the NW OS at the moment, this is because the datastructures that contain this information are owned by the NW OS.

Translating address is another functionality for which the rich OS is used, this is also due to the fact that these translations are easily determined from the NW OS datastructures.

Extensions

Trusted IO could be used to inform the user of the problem (which program has been tampered with for instance), it could also take on a more coarse grained form that an led licht signals the user that some piece of software has failed the attestation which is easier but less usefull.

Becoming independent from the rich OS in the normal world seems like a very important step because otherwise OS/Firmware attacks are still a threat.

Detailed attestation is necessary, lots of software attacks are based on the used datastructures and don't impact the text section of code of programs.

Chapter 5

Experiments

5.1 Performance

Reproduction

Trusted boot is based on the attestation of the normal world before giving it control. In the paper they talk about 107 MB of filesystem image that is being measured so this could be a valuable starting point to compare their performance with the performance achieved in this thesis.

Overhead is measured in the paper by executing system services from the linux kernel and running this experiment with and without the attestation.

Additional

Attestation time should be measured for a program with a certain size to be able to evaluate how often this attestation should be executed to have a balance between performance overhead and security assurance.

5.2 Performance Evaluation

Comparison between the performance achieved in the paper and the performance measured in this thesis is important to be able to interpret the results correctly.

Balance between performance overhead and security assurance depends on the usecase but in the context of the smartphone some statements could be made.

5.3 Security Properties

Integrity of the measurement execution is of utmost importance when it comes to attestation, in remote attestation this is achieved because of a hardened server but here the trusted execution environment needs to take care of this.

Secure storage of results is also important, first of all to make sure the reference values are not tampered with and second of all to correctly react to results that indicate a violation.

5.4 Security Evaluation

Security guarantees that can be made are the integrity of the measurement execution and the security of the results that are being stored. These are achieved due to secure boot enabling the trusted execution environment but are key assumptions in the field of remote attestation.

Shortcomings in terms of security are the OS/firmware attacks because the solution still relies on the rich OS rather heavily.

Extensions in terms of additional aspects of the system that can be attested are necessary to protect against software attacks.

Chapter 6

Discussion

6.1 Related work

Secure boot, Trusted boot and remote attestation for ARM TrustZone-based IoT Nodes is the paper on which the implementation and experiments are based.

DAA-TZ: An Efficient DAA Scheme for Mobile Devices Using ARM TrustZone implements Direct Anonymous Attestation on a mobile ARM TrustZone device.

SecTEE: A Software-based Approach to Secure Enclave Architecture Using TEE implements enclaves on a CPU with ARM TrustZone technology.

TZ-MRAS: A Remote Attestation Scheme for the Mobile Terminal Based on ARM TrustZone uses ARM TrustZone to protect the attestation service on the mobile device from being tampered with.

TrustShadow: Secure Execution of Unmodified Applications with ARM TrustZone utilizes the functionality of the secure world to shield applications from untrusted OSes.

6.2 Comparison of Approaches

Effectiveness

The goal of these papers are all a little different but it is important to evaluate which ones actually realized their goal and how this compares to the goal set out by this thesis.

Most variety of attacks that the solution defends against is a clear measure on how effective the solution is in the field.

The strongest security guarantees that were made and achieved also indicate how well the solution works.

Assumptions

The least assumptions that were made by the authors of the paper the more widely applicable the solution is because there is enormous hetergeneity among devices and assumptions put restrictions on the devices for which the paper is usefull.

The most realistic assumptions are of course also important to look at, if the assumptions are not realistic they are not practical to adhere to and the solution will be worthless if it can't be applied to the real world.

6.3 Future Improvements

Tradeoffs

- a hash for every page vs one hash for the entire process

Weaknesses

Rich OS dependency is very undesirable, it is thus important to look at different solutions that achieve similar outcomes to avoid this aspect of the current solution.

Uncomplete attestation introduces a fake sense of security because not all possible attacks are checked, for instance modified data structures that influence the control flow of a program. (inspiration from Lightweight and Flexible Trust Assessment Modules for the Internet of Things)

Additional features

- Based on the related work papers some additional features could be stated or solutions could be combined to achieve protection against a wider variety of attacks.

Chapter 7

Conclusion

Solution overview

Reproduction of the solution provided in the paper is the starting point of this thesis.

Extensions on this reproduced solution are necessary because the original solution does not achieve all goals.

Shortcomings

Attacker possibilities that are not accounted for are still present, the security measures taken in the solution are not adequate for the wide variety of attacks that it claimed to protect against.

Desired guarantees the solution should be able to provide are not entirely met.

Positives

Code for this thesis is made available as open source code to make it easier to reproduce the experiments and continue work on this research topic by other researchers.

Thorough comparison has been executed on the provided solution and solutions of related work to give an overview of what direction is most promising to achieve the security goals.

Bibliography

[Ala+08] Masoom Alam et al. “Model-based behavioral attestation”. eng. In: *Proceedings of ACM Symposium on Access Control Models and Technologies, SACMAT*. SACMAT ’08. ACM, 2008, pp. 175–184. ISBN: 9781605581293.

[Ala+12] Masoom Alam et al. “Analysis of existing remote attestation techniques”. eng. In: *Security and communication networks* 5.9 (2012), pp. 1062–1082. ISSN: 1939-0114.

[Ali+17] Toqeer Ali et al. “Design and implementation of an attestation protocol for measured dynamic behavior”. eng. In: *The Journal of supercomputing* 74.11 (2017), pp. 5746–5773. ISSN: 0920-8542.

[ARM22a] ARM Ltd. *TrustZone TrustZone for cortex-a*. 2022. URL: `https://www.arm.com/technologies/trustzone-for-cortex-a` (visited on 04/26/2022).

[ARM22b] ARM-Software. *ARM Software arm trusted firmware*. 2022. URL: `https://github.com/ARM-software/arm-trusted-firmware/` (visited on 04/29/2022).

[AS19] Julien Amacher and Valerio Schiavoni. “On the performance of ARM trustzone: (Practical experience report)”. eng. In: *Lecture Notes in Computer Science*. Vol. 11534. Distributed Applications and Interoperable Systems. Springer International Publishing, 2019, pp. 133–151. ISBN: 3030224953.

[ATR20] Daniele Antonioli, Nils Tippenhauer, and Kasper Rasmussen. “Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy”. eng. In: *ACM transactions on privacy and security* 23.3 (2020), pp. 1–28. ISSN: 2471-2566.

[Ba+17] Haihe Ba et al. “Runtime Measurement Architecture for Bytecode Integrity in JVM-Based Cloud”. eng. In: *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. Vol. 2017-. IEEE, 2017, pp. 262–263. ISBN: 9781538616796.

[Buk+18] Sebanjila Kevin Bukasa et al. “How TrustZone Could Be Bypassed: Side-Channel Attacks on a Modern System-on-Chip”. eng. In: *Information Security Theory and Practice*. Vol. 10741. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 93–109. ISBN: 9783319935232.

[BZ19] Raz Ben Yehuda and Nezer Jacob Zaidenberg. “Protection against reverse engineering in ARM”. eng. In: *International journal of information security* 19.1 (2019), pp. 39–51. ISSN: 1615-5262.

[Cay+21] Romain Cayre et al. “Cross-protocol attacks: Weaponizing a smartphone by diverting its bluetooth controller”. eng. In: *WiSec 2021 - Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 2021, pp. 386–388. ISBN: 9781450383493.

[CE21] Tristan Claverie and Jose Lopes Esteves. “BlueMirror: Reflections on Bluetooth Pairing and Provisioning Protocols”. eng. In: *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 339–351. ISBN: 1728189349.

[Cha+17] Rui Chang et al. “MIPE: a practical memory integrity protection method in a trusted execution environment”. eng. In: *Cluster computing* 20.2 (2017), pp. 1075–1087. ISSN: 1386-7857.

[Cok+11] George Coker et al. “Principles of remote attestation”. eng. In: *International journal of information security* 10.2 (2011), pp. 63–81. ISSN: 1615-5262.

[DEN22] DENX Software Engineering. *Pine64 U-Boot*. 2022. URL: `https://gitlab.com/pine64-org/u-boot` (visited on 04/29/2022).

[Dua+20] Jialiang Duan et al. “Integrity Measurement Based on TEE Virtualization Architecture”. eng. In: *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*. IEEE, 2020, pp. 370–376. ISBN: 9780738105208.

[Esk+18] Saba Eskandarian et al. “Fidelius: Protecting User Secrets from Compromised Browsers”. eng. In: (2018).

[Fer+17] Andrew Ferraiuolo et al. “Komodo: Using verification to disentangle secure-enclave hardware from software”. eng. In: *Proceedings of the 26th Symposium on operating systems principles*. SOSP ’17. ACM, 2017, pp. 287–305. ISBN: 9781450350853.

[Fer+18] Mohamed Amine Ferrag et al. “Security for 4G and 5G cellular networks: A survey of existing authentication and privacy-preserving schemes”. eng. In: *Journal of network and computer applications* 101 (2018), pp. 55–82. ISSN: 1084-8045.

[Fot+21] Georgios Fotiadis et al. “Root-of-Trust Abstractions for Symbolic Analysis: Application to Attestation Protocols”. eng. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 13075. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 163–184. ISBN: 9783030918583.

[FY20] Kaiming Fang and Guanhua Yan. “Paging storm attacks against 4G/LTE networks from regional Android botnets: rationale, practicality, and implications”. eng. In: *Proceedings of the 13th ACM Conference on security and privacy in wireless and mobile networks*. WiSec ’20. ACM, 2020, pp. 295–305. ISBN: 9781450380065.

[FZS22] Jasim Khalid Fadhil, Ghafoor Kayhan Zrar, and Maghdid Halgurd S. “Analysis of Encryption Algorithms Proposed for Data Security in 4G and 5G Generations”. eng. In: *ITM web of conferences*. Vol. 42. EDP Sciences, 2022, p. 01004.

[Goo22] Google. *Android keystore system*. 2022. URL: `https://developer.android.com/training/articles/keystore` (visited on 04/26/2022).

[GRA21] Joseph GRAVELLIER. *Remote Hardware Attacks on Connected Devices*. eng. 2021.

[Gro+21] Mathieu Gross et al. “Breaking TrustZone memory isolation and secure boot through malicious hardware on a modern FPGA-SoC”. eng. In: *Journal of cryptographic engineering* (2021). ISSN: 2190-8508.

[Gu+08] Liang Gu et al. “Remote attestation on program execution”. eng. In: *Proceedings of the 3rd ACM workshop on scalable trusted computing*. STC ’08. ACM, 2008, pp. 11–20. ISBN: 9781605582955.

[Gua+17] Le Guan et al. “TrustShadow: Secure Execution of Unmodified Applications with ARM TrustZone”. eng. In: *MobiSys 2017 - Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys ’17. ACM, 2017, pp. 488–501. ISBN: 1450349285.

[Guo+21] Pengfei Guo et al. “Research on Arm TrustZone and Understanding the Security Vulnerability in Its Cache Architecture”. eng. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 12382. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 200–213. ISBN: 9783030688509.

[Hua+21] Zhichao Hua et al. “TZ-Container: protecting container from untrusted OS with ARM TrustZone”. eng. In: *Science China. Information sciences* 64.9 (2021). ISSN: 1674-733X.

[Jav+20] Abdul Rehman Javed et al. “AlphaLogger: detecting motion-based side-channel attack using smartphone keystrokes”. eng. In: *Journal of ambient intelligence and humanized computing* (2020). ISSN: 1868-5137.

[Jia+17] Hang Jiang et al. “Implementing a ARM-Based Secure Boot Scheme for the Isolated Execution Environment”. eng. In: *2017 13th International Conference on Computational Intelligence and Security (CIS)*. Vol. 2018-. IEEE, 2017, pp. 336–340. ISBN: 9781538648223.

[JSS06] Trent Jaeger, Reiner Sailer, and Umesh Shankar. “PRIMA: policy-reduced integrity measurement architecture”. eng. In: *Proceedings of the eleventh ACM symposium on access control models and technologies*. SACMAT ’06. ACM, 2006, pp. 19–28. ISBN: 1595933530.

[Kaw+20] Tomoaki Kawada et al. “TZmCFI: RTOS-Aware Control-Flow Integrity Using TrustZone for Armv8-M”. eng. In: *International journal of parallel programming* 49.2 (2020), pp. 216–236. ISSN: 0885-7458.

[KBC21] Michal Kucab, Piotr Borylo, and Piotr Cholda. “Remote attestation and integrity measurements with Intel SGX for virtual machines”. eng. In: *Computers & security* 106 (2021), p. 102300. ISSN: 0167-4048.

[Kin06] Steven Kinney. *Trusted platform module basics: using TPM in embedded systems*. Mbedded technology series. Amsterdam ; Boston: Elsevier Newnes, 2006. ISBN: 1280637005.

[KKP20] Sudesh Kumar, Lakshmi Jayant Kittur, and Alwyn Roshan Pais. “Attacks on Android-Based Smartphones and Impact of Vendor Customization on Android OS Security”. eng. In: *Information Systems Security*. Vol. 12553. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 241–252. ISBN: 3030656098.

[KM20] Fatima Khalid and Ammar Masood. “Hardware-Assisted Isolation Technologies: Security Architecture and Vulnerability Analysis”. eng. In: *1st Annual International Conference on Cyber Warfare and Security, ICCWS 2020 - Proceedings*. IEEE, 2020, pp. 1–8. ISBN: 9781728168401.

[Len+18] Matthew Lentz et al. “SeCloak: ARM Trustzone-based Mobile Peripheral Control”. eng. In: *MobiSys 2018 - Proceedings of the 16th ACM International Conference on Mobile Systems, Applications, and Services*. MobiSys ’18. ACM, 2018, pp. 1–13. ISBN: 9781450357203.

[Lin+21] Zhen Ling et al. “Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes”. eng. In: *Journal of systems architecture* 119 (2021), p. 102240. ISSN: 1383-7621.

[Lin22] Linux community. *Mobian sunxi64-linux*. 2022. URL: `https://gitlab.com/mobian1/devices/sunxi64-linux` (visited on 04/29/2022).

[Lu+20] Yu-Han Lu et al. “Ghost calls from operational 4G call systems: IMS vulnerability, call DoS attack, and countermeasure”. eng. In: *Proceedings of the 26th Annual International Conference on mobile computing and networking*. MobiCom ’20. ACM, 2020, pp. 1–14. ISBN: 9781450370851.

[Mac+17] Aravind Machiry et al. “BOOMERANG: Exploiting the Semantic Gap in Trusted Execution Environments”. eng. In: *NDSS*. ndss symposium, 2017.

[MAT+21] Tsutomu MATSUMOTO et al. “Secure Cryptographic Unit as Root-of-Trust for IoT Era”. eng. In: *IEICE transactions on electronics* E104.C.7 (2021), pp. 262–271. ISSN: 0916-8524.

[MNP16] Jan Tobias Mühlberg, Job Noorman, and Frank Piessens. “Lightweight and Flexible Trust Assessment Modules for the Internet of Things”. eng. In: *Computer Security – ESORICS 2015*. Vol. 9326. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 503–520. ISBN: 9783319241739.

[OPT22a] OP-TEE. *OP-TEE About OP-TEE*. 2022. URL: `https://optee.readthedocs.io/en/latest/general/about.html` (visited on 04/26/2022).

[OPT22b] OP-TEE. *OP-TEE optee_os, optee_client*. 2022. URL: `https://github.com/OP-TEE/` (visited on 04/29/2022).

[OPT22c] OP-TEE. *OP-TEE QEMU v8*. 2022. URL: `https://optee.readthedocs.io/en/latest/building/devices/qemu.html#qemu-v8` (visited on 04/29/2022).

[PIN21] PINE64. *PINE64 PinePhone*. 2021. URL: `https://www.pine64.org/pinephone/` (visited on 04/26/2022).

[PIN22] PINE64. *PinePhone Software Releases*. 2022. URL: `https://wiki.pine64.org/wiki/PinePhone_Software_Releases` (visited on 04/26/2022).

[Pot+12] Rahul Potharaju et al. “Plagiarizing Smartphone Applications: Attack Strategies and Defense Techniques”. eng. In: *Engineering Secure Software and Systems*. Vol. 7159. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 106–120. ISBN: 9783642281655.

[PS19] Sandro Pinto and Nuno Santos. “Demystifying Arm TrustZone: A Comprehensive Survey”. eng. In: *ACM computing surveys* 51.6 (2019), pp. 1–36. ISSN: 0360-0300.

[PWR21] Nishitkumar Patel, Hayden Wimmer, and Carl M Rebman. “Investigating Bluetooth Vulnerabilities to Defend from Attacks”. eng. In: *2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. Piscataway: IEEE, 2021, pp. 549–554. ISBN: 9781665449304.

[Qin+20] Yu Qin et al. “RIPTE: Runtime Integrity Protection Based on Trusted Execution for IoT Device”. eng. In: *Security and communication networks* 2020 (2020). ISSN: 1939-0114.

[Rya19] Keegan Ryan. “Hardware-Backed Heist: Extracting ECDSA Keys from Qualcomm’s TrustZone”. eng. In: *Proceedings of the 2019 ACM SIGSAC Conference on computer and communications security*. CCS ’19. ACM, 2019, pp. 181–194. ISBN: 9781450367479.

[SAB15] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. “Trusted Execution Environment: What It is, and What It is Not”. eng. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. IEEE, 2015, pp. 57–64. ISBN: 9781467379526.

[Sam22] Samsung. *KNOX Stay connected, protected, and productive*. 2022. URL: `https://www.samsungknox.com/en` (visited on 04/26/2022).

[SD21] Ammar S Salman and Wenliang (Kevin) Du. “Securing Mobile Systems GPS and Camera Functions Using TrustZone Framework”. eng. In: *Intelligent Computing*. Vol. 285. Lecture Notes in Networks and Systems. Cham: Springer International Publishing, 2021, pp. 868–884. ISBN: 9783030801281.

[Set+20] Rico Setyawan et al. “A brief review of attacks and mitigations on smartphone infrastructure”. eng. In: *IOP Conference Series: Materials Science and Engineering* 852.1 (2020), p. 12141. ISSN: 1757-8981.

[SKU10] B Stelte, R Koch, and M Ullmann. “Towards integrity measurement in virtualized environments - A hypervisor based sensory integrity measurement architecture (SIMA)”. eng. In: *2010 IEEE International Conference on Technologies for Homeland Security (HST)*. IEEE, 2010, pp. 106–112. ISBN: 1424460476.

[Son+20] Wenna Song et al. “Android Data-Clone Attack via Operating System Customization”. eng. In: *IEEE access* 8 (2020), pp. 199733–199746. ISSN: 2169-3536.

[Van+19] Jo Van Bulck et al. “A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes”. eng. In: *Proceedings of the 2019 ACM SIGSAC Conference on computer and communications security*. CCS ’19. ACM, 2019, pp. 1741–1758. ISBN: 9781450367479.

[YF08] Aimin Yu and Dengguo Feng. “BBACIMA: A trustworthy integrity measurement architecture through behavior-based TPM access control”. eng. In: *Wuhan University journal of natural sciences* 13.5 (2008), pp. 513–518. ISSN: 1007-1202.

[Zha+21] Shijun Zhao et al. “Research on Root of Trust for Embedded Devices based on On-Chip Memory”. eng. In: *2021 International Conference on Computer Engineering and Application (ICCEA)*. Piscataway: IEEE, 2021, pp. 501–505. ISBN: 9781665426169.

[Zhu+21] Ye Zhu et al. “Towards Smartphone Operating System Identification”. eng. In: *IEEE transactions on dependable and secure computing* 18.1 (2021), pp. 411–425. ISSN: 1545-5971.

[ZW21] Rana M Zaki and Hala Bahjat Abdul Wahab. “4G Network Security Algorithms: Overview”. eng. In: *International journal of interactive mobile technologies* 15.16 (2021), pp. 127–143. ISSN: 1865-7923.

[ZY20] Diming Zhang and Shaodi You. “iFlask: Isolate flask security system from dangerous execution environment by using ARM TrustZone”. eng. In: *Future generation computer systems* 109 (2020), pp. 531–537. ISSN: 0167-739X.