Formal Verification of Interrupt Isolation for the TrustZone-based TEE

Leping Zhang[1,4], Qianying Zhang[1,2,✉], Xinyue Wang[1], Ximeng Li[1,3], Guohui Wang[1,2]
Zhiping Shi[1,3], *and* Yong Guan[1,3]

[1]*College of Information Engineering, Capital Normal University, Beijing, China*
[2]*Beijing Engineering Research Center of High Reliable Embedded System, Beijing, China*
[3]*Beijing Key Laboratory of Electronic System Reliability Technology, Beijing, China*
[4]*School of Computer Science and Engineering, Beihang University, Beijing, China*
zhangleping@buaa.edu.cn, {qyzhang, xywang, lixm, ghwang, shizp, guanyong}@cnu.edu.cn

***Abstract*—ARM TrustZone is a hardware security technology commonly used to implement the Trusted Execution Environment (TEE), which is a hardware-based isolated execution environment in which security-critical software can execute without interference and is widely applied to embedded systems. Isolation of interrupts in the two security domains of TrustZone is an important security mechanism that plays a vital role in protecting the execution of the TEE. This paper introduces a formal modeling and verification approach for the interrupt isolation mechanism of the TEE based on TrustZone. We propose a model formalizing the TrustZone-based TEE system at the instruction level, which focuses on modeling system behaviors of interrupt handling and captures crucial instructions related to interrupt isolation. We formally define three types of properties for the model: correctness, safety, and information flow security. The verification results in the theorem prover Isabelle/HOL show that the model satisfies the above properties, which indicates that the interrupt isolation mechanism of TrustZone correctly enforces the isolation of interrupts and protects the TEE from being interfered with non-secure interrupts.**

***Index Terms*—formal verification, interrupt isolation, TrustZone, TEE, security, Isabelle/HOL**

I. Introduction

Embedded systems are increasingly widely used, bringing great convenience to users' production and daily lives. Examples include smartphones and tablets used in daily life, onboard computers in automobiles, and industrial control systems in industrial production. With their widespread application, the amount of user data stored in embedded systems, especially security-sensitive data like bank account passwords, vehicle location information, and industrial production data, has gradually grown. The growth of sensitive data has led to increased attacks on embedded systems, necessitating urgent improvements in their security. In this case, the GlobalPlatform organization proposed the concept of the Trusted Execution Environment (TEE) [1], a secure area of the device processor that ensures sensitive data is stored, processed, and protected in an isolated and trusted environment. TEE isolates security-critical software from any software outside of the TEE and has been adopted by most device and chip manufacturers.

✉Corresponding author: Qianying Zhang, qyzhang@cnu.edu.cn

One of the most typical ways to implement a TEE is by leveraging the TrustZone technology [2] proposed by ARM, which separates system resources into two different execution environments: the secure world and the normal world. Trusted operating systems and applications run in the secure world, which acts as a TEE, while rich operating systems and normal applications run in the normal world. Software running in the normal world is restricted from accessing peripherals and memory regions designated by TrustZone as assigned to the secure world. To provide the above separation, TrustZone reconstructs the processor and enforces several mechanisms for the isolation of hardware resources, such as interrupt isolation and memory isolation. As a high-security system architecture, TrustZone technology has been widely applied to various fields [3]–[6], and has been used to provide secure foundations for commercial products [7], [8] and research projects [9], [10].

Unfortunately, several TEE systems based on the TrustZone technology have been found to have vulnerabilities [11]–[15]. According to the investigation conducted by Sandro Pinto and Nuno Santos in 2019 [16], over 130 vulnerabilities related to TrustZone and TrustZone-based TEE have been identified, most of which are existing bugs in the TEE kernel and TEE driver implementations. For instance, the Secure Execution Environment implemented by Qualcomm (QSEE) was revealed to have an integer overflow vulnerability [11], which enables an attacker with kernel-level privileges to execute arbitrary code in the context of QSEE and affects various Android devices equipped with Qualcomm Snapdragon SoC and enabled TrustZone technology. Vulnerabilities of TEE systems can be leveraged to compromise applications protected by TrustZone and leak sensitive data like user passwords and fingerprints, which may cause property losses to individuals or groups and in serious cases may even threaten lives. Therefore, it is very crucial to ensure the correctness of the TEE system including TrustZone hardware and TEE software, and then to guarantee the security of trusted applications in the TEE.

This paper exploits formal methods to ensure the correctness of TEE systems and focuses on the verification of the interrupt isolation of TrustZone-based TEE. Interrupt isolation is a

2023 30th Asia-Pacific Software Engineering Conference (APSEC) | 979-8-3503-4417-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/APSEC60848.2023.00041

2640-0715/23/$31.00 ©2023 IEEE
DOI 10.1109/APSEC60848.2023.00041

critical mechanism of TrustZone technology, which is used to divide and isolate the interrupts of the secure world and the normal world, and whose errors may cause the normal world to compromise the security of TEE using the interrupt mechanism. Currently, there is a lack of verification for whether the interrupt isolation mechanism satisfies the expected security properties.

There are three challenges in our work. The first challenge is the lack of modeling of the TEE interrupt isolation mechanism. For example, Komodo [10] implements a monitor that manages enclaves providing the TEE and proves the functional correctness of the monitor and the confidentiality and integrity of the enclaves. However, the model of Komodo does not include the interrupt isolation mechanism or verify any properties of the mechanism. Therefore, it is necessary to establish a formal model suitable for the verification of the interrupt isolation mechanism. The second challenge is the absence of formal definitions for the correctness and security properties of interrupt isolation, which cannot be described by properties defined in existing verification work, such as non-interference, confidentiality, and integrity. So, it is necessary to consider how to formulate security properties for verifying the interrupt isolation. Finally, the isolation mechanism of TrustZone is highly complex, involving hardware-related components and instruction-level actions, and has a large state space. The third challenge is how to construct a suitable method for verifying this complex mechanism. The paper aims to address these three challenges.

This paper proposes the formal verification of the interrupt isolation mechanism of the TEE, which consists of a formal model of the mechanism, the formalization of its expected properties, and the proof that the mechanism satisfies these properties. We model a TrustZone-based TEE system as a state machine and focus on the actions taken when the system responds to the requests of secure and normal interrupts. We define key components related to the interrupt isolation mechanism of TrustZone as elements of states in the state machine, such as the Secure Configuration Register (SCR) and the memory regions used to save context when switching between the two worlds of TrustZone. The actions of the system are defined as events that drive state transitions, such as hardware exceptions like the Fast Interrupt Request (FIQ) or software exceptions like the Secure Monitor Call (SMC). We define a run function to represent the execution of the state machine, and leveraging the function we define reachable states, which can represent any state in the machine. We formalize three types of properties: correctness properties that ensure TrustZone correctly implements interrupt handling in both worlds, safety properties that maintain certain unchanged relations within the TEE system, and information flow security properties that ensure the response to interrupts in the normal world does not affect the context in the secure world. By verifying that the reachable states satisfy the above properties, we prove that the interrupt isolation mechanism of TrustZone satisfies the expected properties. All proofs are performed using the theorem prover Isabelle/HOL [17]. The verification results demonstrate that all reachable states satisfy the three types of properties, indicating the correctness and security of the interrupt isolation mechanism of the TrustZone-based TEE.

The paper makes the following contributions.

1) We propose a formal model of the interrupt isolation mechanism of the TrustZone-based TEE, in which hardware-related components are involved and all events are defined at the instruction level. The model can be used as a reference for implementing the interrupt isolation mechanism of TrustZone.
2) We formalize the correctness and security properties of the TrustZone interrupt isolation mechanism. To the best of our knowledge, our work is the first effort to simultaneously formally verify correctness, safety, and information flow security properties w.r.t interrupt isolation.
3) We conduct a formal verification of the TrustZone interrupt isolation mechanism in Isabelle/HOL. We rigorously prove that the model satisfies the correctness and security properties through solving invariants, which ensures that interrupts are handled in the correct world and that the context in the secure world is not influenced by the response to interrupts in the normal world.

The rest of this paper is organized as follows. Section II introduces the background knowledge, while Section III details the formal model of the TrustZone-based TEE system, including states, event specifications, and system execution. In Section IV, we formalize three types of properties, which describe the correctness and security of the interrupt isolation of TrustZone. Section V presents the formal proofs in Isabelle/HOL, demonstrating that the model satisfies the defined properties. Section VI discusses the results and evaluates our work. Section VII depicts the related work and Section IX gives the conclusion.

II. Preliminaries

This section introduces the background knowledge. We first summarize the TrustZone technology, detail the interrupt isolation mechanism, and then introduce the theorem prover Isabelle/HOL used in our work.

A. ARM TrustZone Technology

The TrustZone architecture is shown in Fig. 1(a). TrustZone divides the processor's execution environment into a secure world and a normal world. The secure world provides the TEE for the trusted OS or trusted applications, and the normal world provides a Rich Execution Environment (REE) for the general-purpose OS and normal applications. The former can access all resources like memory and peripherals, but the latter can only access resources allocated to the normal world. A Non-Secure (NS) bit in the SCR register determines whether the current world is secure or normal: 0 represents the secure world, and 1 represents the normal world. To realize the world switch, TrustZone extends a mode from the original 7 modes of the ARM processor, called monitor mode. In the monitor mode, the value of the NS bit can be modified to change the current world to the other. The way the processor enters

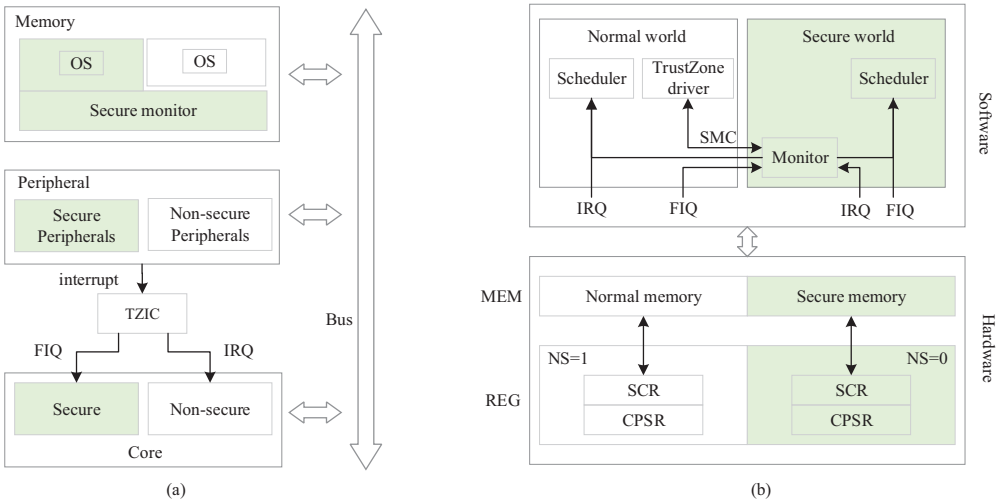

Fig. 1. (a) shows the TrustZone Architecture, and (b) shows the Interrupt Isolation Mechanism of TrustZone

monitor mode from the normal world is strictly controlled, including 1. by triggering the specific hardware exceptions. 2. by executing the Secure Monitor Call (SMC). The processor can enter monitor mode from the secure world in an additional way, that is, directly writing to the Current Program Status Register (CPSR). When in monitor mode, the processor is always executing in the secure world regardless of the value of the SCR NS bit [2]. A monitor is a security-critical software running in the monitor mode responsible for managing the switch between the two worlds: during world switching, it stores the context of the current world and then restores that for the other world.

In addition to dividing the execution environment of the processor, TrustZone divides and isolates the peripherals and memory into the secure and normal worlds. Correspondingly, the NS bit propagates down to buses bridging peripherals and memory.

Interrupt Isolation. TrustZone provides an interrupt isolation mechanism that isolates secure and normal interrupt sources to the secure and normal worlds for responses. The implementation of interrupt isolation mainly relies on the TrustZone interrupt controller (TZIC). The TZIC component acts as the first-level interrupt source controller, and the interrupt requests of all interrupt sources are connected to the TZIC, and the interrupt request of the device is intercepted first. In general, it is recommended to set the FIQ exception as a secure interrupt source and the Interrupt Request (IRQ) exception as a normal interrupt source, to implement minimal modifications to existing software. In more detail, the programmer selects the specified interrupt sources as FIQ or IRQ exceptions by programming the relevant registers of the TZIC component. As shown in Fig. 1(b), regardless of whether the current world is secure or normal, FIQ and IRQ exceptions may occur. When an exception occurs in the corresponding world, the exception will be handled directly. Otherwise, the current world will be switched to the other through the monitor. An interesting phenomenon is that programmers can configure IRQ exceptions that occur in the secure world to be responded to or not. When configured not to respond, the IRQ exceptions are discarded. In this work, we consider the modeling and verification for both cases.

B. Theorem Prover Isabelle/HOL

Isabelle/HOL [17] is an interactive theorem prover for processing high-order logic, jointly developed by the University of Cambridge and the Technical University of Munich. We use it to perform our verification work. In Isabelle/HOL, the keyword **datatype** is used to define a new recursive type. For example, the natural number type nat is defined as **datatype** $nat = 0 \mid$ **Suc** nat, where **Suc** is a custom type constructor. If a type has several constructors with no parameters, the values of the type can be used as tags. The keyword **record** can define a record. For example, the following gives the record of a register reg:

$$\textbf{record}\ reg = r_1 :: 32\ word \\ r_2 :: 32\ word$$

where r_1 and r_2 are elements of reg, and their type are both 32-bit words defined in the Isabelle/HOL library. A value of reg can be defined as $(\!|r_1 = 1,\ r_2 = 2|\!)$ and the value can be updated like $reg(\!|r_1 := 2,\ r_2 := 3|\!)$. The keyword **type_synonym** is used to rename an existing type, and the keyword **definition** is used to define a constant or a function without recursive. For proofs, Isabelle/HOL provides a series of tactics, such as $auto$ and $simp$, which can automatically prove some simple goals. Furthermore, the prover defines several commands like **find**, **intro**, and **sledgehammer** to help us search for the tactics and lemmas needed for proof goals.

III. Formal Model of TrustZone Interrupt Isolation Mechanism

This section details the formal model of the interrupt isolation mechanism. We model a TrustZone-based TEE system, which focuses on the actions of handling interrupts. In our model, TrustZone is based on ARMv8 architecture, and the monitor is defined according to ARM Trusted Firmware (ATF) [18] which is a security solution for ARM architecture and provides implementations for security software including the monitor.

We define the TEE system as a state machine that simulates the execution of a system as a series of transitions of states. A state machine usually consists of a set of states, an initial state, a set of events, and a series of state transition functions. For the elements of the state, we consider registers related to world switching, such as SCR, CPSR, and memory in the secure world and normal world. The values of registers and memory determine the initial state before the TEE system starts. The events include the handling actions of secure interrupts and normal interrupts, the actions of the monitor, and several general instructions like Load/Store. We define a function to describe how the system executes an action sequence. Considering the definition of the information flow security properties, we give a reachable function to describe all possible states in the TEE system.

The following introduces states, state transitions, and system execution to show our model.

A. States

This section depicts the definition of states. Before that, we introduce several key components. Through the analysis of the TrustZone interrupt isolation mechanism, we model the components related to the process of handling FIQs and IRQs.

Worlds. According to the division of the execution environment, we define a type: **datatype** $World = sw \mid nw$, where sw represents the secure world and nw represents the normal world.

Registers. We model a series of registers related to the TrustZone interrupt isolation mechanism, including all general-purpose registers and several special-purpose registers. The interesting registers mainly include SCR, CPSR, SPSR, and so on. SCR is modified by the monitor during world switching, at the same time, CPSR and SPSR are set to correct values according to the current world. In addition, the stack pointer registers in the EL0 and EL3 layers (called SP_EL0 and SP_EL3) are considered in our model, and they are used to record the base address of the memory where the context is stored. These registers are of the same type 32 $word$ merged into a record type $RegState$. Referring to the implementation of ATF, an initial state $init\_regstate$ of these registers is given.

Memory. When switching from one world to another, the register data is backed up to memory. We define a type of memory: $MEM : 32\ word \rightarrow 32\ word$, i.e., a function that takes an address of type 32-bit word as a parameter and returns a value of type 32-bit word. By a given constant address, the whole memory is divided into secure and normal parts.

The state records the values of the above components, which are defined as follows:

record $State =$
$cur \ :: World$
$reg \ :: RegState$
$mem :: MEM$

where the variable cur records the current world. It can be seen that the value of cur corresponds to that for the NS bit in SCR, which is guaranteed by invariants subsequently.

Initial State. According to ATF, after a system starts up and completes the initialization operation, a practical initial state is established, where we abstract the operation as two functions namely set_mem and set_reg, for initializing the state fields in our model. The initial state s_0 is defined as follows:

$$s_0 = set\_mem\ (set\_reg\ iState)$$

where $iState$ is the state before the initialization, whose value is set to $(\!|cur = sw,\ reg = init\_regstate,\ mem = (\lambda x.\ 0)|\!)$. We use set_reg and set_mem to set the values of registers and memory. The memory setting mainly involves the backup of the normal world registers. For example, the register SCR in the normal world and the secure world is backed up to two locations in memory with different starting addresses but fixed offsets, and the starting addresses are 0x0000 and 0x0200, respectively.

B. State Transitions

In the ATF's implementation, the actions related to the TrustZone interrupt isolation mechanism include four categories, that is, handling secure interrupts FIQ, handling normal interrupts IRQ, switching world by the monitor, and several instructions under ARMv8 architecture. The following details these actions.

Handling FIQ Exceptions. Recalling the process of handling interrupts in Section II-A, that is, the TEE system first checks the current world. If the current world is secure, then the specific FIQ exception is handled directly; otherwise, the system switches from the current world to the secure world, further performing the handling for the exception. We give the formal definition of handling FIQ exceptions. As shown in Fig.2, when the current world is secure (Line 3), a concrete FIQ exception n is handled by the function $EL3\_handle : State \rightarrow nat \rightarrow State \times bool$ (Line 4), where the second item of type $bool$ in the return value indicates success or failure for this handling. When the current world is normal, switch the world (Lines 5-12) and then handle the exception (Line 13).

Handling IRQ Exceptions. Considering that the IRQ exceptions that occur in the secure world have two cases: being responded to and not being responded to, we give definitions for both cases. When the exceptions are responded to, the process is like that for FIQ, except that the world where IRQ exceptions are handled is normal and the function name of handling the exceptions is $EL1\_handle$. When the exceptions are not responded to, if the current world is secure and an IRQ exception occurs, the state remains unchanged.

```
definition Handle_FIQ :: "State ⇒ nat ⇒ State × bool" where
"Handle_FIQ s n ≡
    (if cur s = sw
     then EL3_handle s n
     else let s' =
            (let r = reg s; m = mem s;
                base1 = (if cur s = sw then SP_EL0 r else SP_EL3 r);
                base2 = (if cur s = sw then SP_EL3 r else SP_EL0 r);
              in s(|mem := m(base1 + CTX_SCR_EL3 init_offset := SCR_EL3 r),
                    mem := m(base1 + CTX_SPSR_EL3 init_offset := SPSR_EL3 r),
                    reg := r(|SCR_EL3 := m(base2 + CTX_SCR_EL3 init_offset)|),
                    reg := r(|SPSR_EL3 := m(base2 + CTX_SPSR_EL3 init_offset)|)|))"
        in EL3_handle s' n)"
```

Fig. 2. Formal Definition of Handle_FIQ

Switching World. In addition to handling FIQ and IRQ exceptions that may cause world switching, there is also a special instruction SMC dedicated to switching worlds. Since the instruction is atomic as well as the execution of this instruction involves modifying registers and updating memory, we treat it as a separate class of events and model it.

Other Instructions. For completeness, we model two common instructions in the instruction set of ARM architecture (ISA), i.e., Load and Store, which are used to read and write memory, respectively. Both instructions help us to test the value of the specified memory. To ensure the effective isolation of memory, before executing these two instructions, the accessed address needs to be checked to see if it is within the allowed range.

C. System Execution

To describe the execution of the TEE system, we define a recursive function namely run, and the term $run\ s\ es$ obtains a new state after executing the event sequence es from the state s. The concrete definition is given as follows.

primrec $run :: "State \Rightarrow Event\ list \Rightarrow State"$ **where**
$"run\ s\ [] = s"\ |$
$"run\ s\ (e\#es) = run\ (step\ s\ e)\ es"$

Here, the function $step :\ State \rightarrow Event \rightarrow State$ obtains a new state after executing a single-step event from a state. With run, we give a function $\mathcal{R}$ to describe all reachable states in the state machine and define it as follows.

Definition 1 (Reachable State): State s is reachable if and only if there exists a sequence of events, the state obtained after executing the sequence from the initial state is equal to s, that is, $\mathcal{R}\ s \equiv \exists es.\ run\ s_0\ es = s$.

Later, we leverage the reachable state to prove safety and security properties.

IV. Properties

This section introduces properties of the interrupt isolation mechanism of TEE based on TrustZone. The properties include three categories: correctness, safety, and security. The correctness properties are built based on the reference manual [2], and the safety properties are mainly built based on our model. The security properties are defined based on some variants of information flow security properties.

A. Correctness

The correctness of the TrustZone interrupt isolation mechanism means that the secure interrupts are correctly isolated from the normal interrupts, which is reflected in the processor and memory.

First, the execution environment of the processor is divided into the normal world and the secure world, and the handling of interrupts is isolated into these two worlds. Secure interrupts (FIQ exceptions) are handled in the secure world and normal interrupts (IRQ exceptions) are handled in the normal world. Therefore, we must ensure that all interrupts are handled in the correct world, which is formalized as two correctness properties:

Property 1: executing a single-step FIQ event from any state will cause the current world of the successor state to be the secure world.

$$\forall s\ s'.\ s' = step\ s\ (FIQ\ i) \longrightarrow cur\ s' = sw$$

Property 2: executing a single-step IRQ event from any state will cause the current world of the successor state to be the normal world.

$$\forall s\ s'.\ s' = step\ s\ (IRQ\ i) \longrightarrow cur\ s' = nw$$

Second, the memory is divided into secure and normal memory, and handling for normal interrupts cannot influence the data in the isolated secure memory. Here, we focus on memory regions that back up critical registers like SCR rather than the whole memory and ensure that the values of these regions are not influenced by normal interrupts. For this, we define a property as follows.

Property 3: The context saved in the secure world remains unchanged after handling an IRQ exception in the normal world.

$$\begin{aligned}
&\forall s\ s'.\ s' = step\ s\ (IRQ\ i) \longrightarrow \\
&\quad (mem\ s)(SP_EL0\ (reg\ s) + \\
&\qquad CTX_SCR_EL3\ init_offset) = \\
&\quad (mem\ s')(SP_EL0\ (reg\ s') + \\
&\qquad CTX_SCR_EL3\ init_offset) \\
&\quad \wedge
\end{aligned}$$

$$(mem\ s)(SP\_EL0\ (reg\ s) + CTX\_SPSR\_EL3\ init\_offset) = (mem\ s')(SP\_EL0\ (reg\ s') + CTX\_SPSR\_EL3\ init\_offset)$$

Further, this property prevents errors in modifying these registers when handling interrupts. In addition, the description of this property is relevant to the concept of security. In fact, it is a foundation for the subsequent proof of security.

B. Safety

In this model, we represent safety properties through invariants, which can describe the relationship that remains unchanged in a system. The type of an invariant is defined as $inv: State \rightarrow bool$, i.e., a predicate indicating whether a state satisfies the specific invariant. This paper provides a total of 7 invariants, most of which are used to prove the subsequent properties of information flow security. These invariants are defined as follows.

Invariant 1: The value of the NS bit in the normal world is 1.

$$cur\ s = nw \longrightarrow read\_ns\ s$$

where $read\_ns\ s$ obtains the value of the NS bit in SCR and its return value is the type of **bool**.

Invariant 2: The value of the NS bit in the secure world is 0.

$$cur\ s = sw \longrightarrow \neg(read\_ns\ s)$$

where the symbol $\neg$ means negation.

Invariant 3: The offset address CTX_SCR_EL3 of the memory address that saves the context of the SCR register remains unchanged, and its value is 0.

$$CTX\_SCR\_EL3\ init\_offset = 0$$

Invariant 4: The stack pointer SP_EL0 used to save the base address of the context in the secure world remains unchanged, and its value is 0x0200.

$$SP\_EL0\ (reg\ s) = (0x0200 :: 32\ word)$$

Invariant 5: The stack pointer SP_EL3 used to save the base address of the context in the normal world remains unchanged, and its value is 0x0000.

$$SP\_EL3\ (reg\ s) = (0x0000 :: 32\ word)$$

Invariant 6: The NS bit of the SCR context saved in the secure world remains unchanged, and its value is 0.

$$\neg(test\_bit\ ((mem\ s)(SP\_EL0\ (reg\ s) + CTX\_SCR\_EL3\ init\_offset))\ 0)$$

where $test\_bit$ returns the value of the specified bit of a word.

Invariant 7: The NS bit of the SCR context saved in the secure world remains unchanged, and its value is 1.

$$test\_bit\ ((mem\ s)(SP\_EL3\ (reg\ s) + CTX\_SCR\_EL3\ init\_offset))\ 0$$

C. Information Flow Security

Information flow security properties mainly include classic noninterference [19], nonleakage [20], noninfluence [21], and their variants [22], [23]. Noninterference concerns the secrecy of the action, and nonleakage concerns the confidentiality of the data. Noninfluence is a combination of noninterference and nonleakage. For these variants, their main difference from the classical definitions is that the domain where the action is located depends on states. In this work, actions like Load/Store take place in the current world, i.e., relative to the current state. Therefore, we consider their variants as security properties to be verified and give their definitions as follows.

$noninterference \equiv$
$\forall es\ d.\ s_0 \lhd es \cong s_0 \lhd (ipurge\ s_0\ es\ d)\ @\ d$

$nonleakage \equiv$
$\forall s\ t\ es\ d.\ \mathcal{R}\ s \wedge \mathcal{R}\ t \wedge s \approx (sources\ s\ es\ d) \approx t \longrightarrow s \lhd es \cong t \lhd es\ @\ d$

$noninfluence \equiv$
$\forall s\ t\ es\ d.\ \mathcal{R}\ s \wedge \mathcal{R}\ t \wedge s \approx (sources\ s\ es\ d) \approx t \longrightarrow (s \lhd es \cong t \lhd (ipurge\ t\ es\ d)\ @\ d)$

Where the term $ipurge\ s\ es\ d$ filters out all actions whose domain have no information flow to d during the execution of the sequence es from the state s then returns a new action sequence, and the term $sources\ s\ es\ d$ collects all domains that can directly or indirectly flow information to d. $\_ \lhd \_ \cong \_ \lhd \_\ @\ \_$ is a symbolic representation of a function called state equivalence, and a term $s \lhd as \cong t \lhd bs\ @\ d$ means whether the two states obtained by executing as from s and executing bs from t are equivalent w.r.t d.

The above three properties involve four key elements, namely domain, domain of event, information flow policy, and observation function.

- The domain divides the execution environment into several different security levels, and threads with different security levels are executed in different domains. Naturally, in this work, we define the domain as:
  type_synonyms $Domain = World$
  which indicates that the domains include the secure world and the normal world.
- The domain of event is defined as a function that specifies the domain in which each event executes. The state-dependent domain of events is of type $State \rightarrow Event \rightarrow Domain$. Specifically, the event $Handle\_FIQ$ is defined in sw, while $Handle\_IRQ$ is defined in nw. Other events are defined in the current world.
- The information flow policy specifies whether information can be passed between domains and domains. In general, the policy is configured statically and has the type $domain \rightarrow domain \rightarrow bool$ (symbolized as $\_ \rightsquigarrow \_$). We give the concrete definition of the policy as $sw \rightsquigarrow nw$ and $\neg(nw \rightsquigarrow sw)$, which means that actions in the secure world can read and write the data of the normal world, but the normal world can neither read nor write that of secure world.
- The observation function specifies which data needs to be protected for each domain. A term $observe\ s\ d\ t$ (symbolized as $s \sim d \sim t$) determines whether states s and t are equivalent w.r.t domain d. In this work, the function is defined as follows.
  $observe :: State \Rightarrow Domain \Rightarrow State \Rightarrow bool$ **where**
  $s \sim d \sim t \equiv$
  (**if** $d = sw$

then (**if** $cur\ s = sw$
then $reg\ s = reg\ t \wedge mem\ s = mem\ t$
else $mem\ s = mem\ t$)
else (**if** $cur\ s = nw$
then $reg\ s = reg\ t \wedge$
$\lambda x.\ x < 0x200 \longrightarrow mem\ s = mem\ t$
else $\lambda x.\ x < 0x200 \longrightarrow mem\ s = mem\ t$))

Where the interesting point is that the data visible to a domain is related to the current state. For example, the secure world can observe the values of registers only if the current world is secure, and not otherwise. In addition, the secure world can observe the whole memory, while the normal world can only observe normal memory (the address is limited to the range of 0x0000-0x0200).

V. Formal Proofs

This section introduces the formal proofs of the TrustZone interrupt isolation mechanism, including correctness, safety, and information flow security proofs.

A. Correctness Proofs

According to Hoare [24], a function is totally correct if it is partially correct and terminates. The proof task for partial correctness involves the three properties proposed in Section IV-A. The proof steps for these properties are similar: Firstly, introducing or unfolding the definition of the events like $Handle\_FIQ$. Secondly, simplifying the proof goal using the tactic $simp$. Thirdly, searching the final tactic using **sledgehammer** due to the concise goal here. For proofs of termination, we take advantage of the automated proof mechanism of Isabelle/HOL, which automatically proves the termination of functions defined with the keywords **fun** and **primrec**. Once the function is non-terminating, the prover will give an error.

B. Safety Proofs

In Section IV-B, safety properties are represented as several invariants. We only need to prove that these invariants hold, that is, $\mathcal{R}\ s \longrightarrow inv\ s$, where $inv\ s$ is the intersection of all the invariants, and the implication is also the final proof goal for safety. Proving invariants usually takes the inductive approach. Therefore, we first introduce two lemmas to describe the properties of the reachable function $\mathcal{R}$ in a state-inductive manner, which are defined as follows.

Lemma 1: The initial state s_0 is reachable.

shows $(\mathcal{R}\ s_0)$

Lemma 2: For an arbitrary reachable state s and event e, the new state s' obtained after executing e from s is also reachable.

assumes $(\mathcal{R}\ s \wedge s' = step\ s\ e)$
shows $(\mathcal{R}\ s')$

Lemma 1 and Lemma 2 can be proved by introducing the definitions of $\mathcal{R}$ and $step$. With the tactic $induction$, Lemma 2 can further deduce the formula $\mathcal{R}\ s \wedge s' = run\ s\ es \longrightarrow \mathcal{R}\ s'$. Based on Lemma 1 and Lemma 2, the proof goal is transformed into the following two lemmas.

Lemma 3: The initial state s_0 satisfies the invariants.

shows $(inv\ s_0)$

Lemma 4: For an arbitrary state s that satisfies the invariant inv and an event e, the new state obtained after executing e from s satisfies inv.

assumes $(inv\ s \wedge s' = step\ s\ e)$
shows $(inv\ s')$

Lemma 3 is easily proved by introducing the definition of s_0 and inv. For the proof of Lemma 4, the tactic used is $cases$ on event e, which divides the goal into four subgoals for different events. Taking the event of handling an FIQ exception as an example, the subgoal is expressed as the following lemma.

Lemma 5: For an arbitrary state s that satisfies the invariant inv, the new state obtained after executing $Handle\_FIQ$ from s satisfies inv.

assumes $(inv\ s \wedge s' = fst\ (Handle\_FIQ\ s\ i))$
shows $(inv\ s')$

Where the function fst obtains the first item of pairs. Combined with the definition of $Handle\_FIQ$ and inv, Lemma 5 is proved by the tactic $auto$.

C. Information flow Security Proofs

In general, the properties of information flow security are proved by unwinding conditions [19], [25], which include $step\_consistent$ (SC) and $local\_respect$ (LR). Later, the former has a concise definition called $weak\_step\_consistent$ (WSC). There is a relationship between these conditions: $WSC \wedge LR \longrightarrow SC$. Therefore, we only need to prove that WSC and LR hold. The general definitions of both are shown below.

$$\begin{aligned} WSC \equiv \forall e\ d\ s\ t\ s'\ t'.\ & \mathcal{R}\ s \wedge \mathcal{R}\ t \wedge (dom\ s\ e) \rightsquigarrow d \wedge \\ & s \sim (dom\ s\ e) \sim t \wedge s \sim d \sim t \wedge \\ & s' = step\ s\ e \wedge t' = step\ t\ e \\ & \longrightarrow s' \sim d \sim t' \end{aligned}$$

$$\begin{aligned} LR \equiv \forall e\ d\ s\ s'.\ & \mathcal{R}\ s \wedge \neg((dom\ s\ e) \rightsquigarrow d) \wedge s' = step\ s\ e \\ & \longrightarrow s \sim d \sim s' \end{aligned}$$

For the proof of general unwinding conditions, we need to build and prove concrete unwinding conditions on all events in the state system. Taking the event $Handle\_FIQ$ as an example, both are defined as follows.

$$\begin{aligned} WSC\_FIQ \equiv \forall i\ d\ s\ t\ s'\ t'.\ & \mathcal{R}\ s \wedge \mathcal{R}\ t \wedge \\ & (dom\ s\ (FIQ\ i)) \rightsquigarrow d \wedge \\ & s \sim (dom\ s\ (FIQ\ i)) \sim t \wedge s \sim d \sim t \wedge \\ & s' = fst\ (Handle\_FIQ\ s\ i) \wedge \\ & t' = fst\ (Handle\_FIQ\ s\ i) \longrightarrow s' \sim d \sim t' \end{aligned}$$

$$\begin{aligned} LR\_FIQ \equiv \forall i\ d\ s\ s'.\ & \mathcal{R}\ s \wedge \neg((dom\ s\ (FIQ\ i)) \rightsquigarrow d) \wedge \\ & s' = fst\ (Handle\_FIQ\ s\ i) \longrightarrow s \sim d \sim s' \end{aligned}$$

So far, the proofs of information flow security properties as well as unwinding conditions cannot be done automatically. We manually prove all the above conditions in Isabelle/HOL, leveraging tactics such as $cases$ and $auto$ combined with our proposed invariants.

VI. Discussion

Results. We use Isabelle/HOL to define the formal model of the TrustZone-based TEE system, which focuses on the

actions of the interrupt isolation mechanism. We formalize three correctness and seven safety properties of the mechanism and prove that the model satisfies these properties, as well as information flow security properties. The proof script is described in a structured language Isar [26] provided by Isabelle/HOL. During the proofs, we use the command **sledgehammer** to automatically search for tactics, which reduces our proof burden and simultaneously merges dozens of lines of code (LOC) to one LOC guided by the keyword **by**. After a total effort of about 5 person-months, we developed approximately 3300 LOC, including 14 definitions (around 150 LOC) for modeling, 14 theorems (around 20 LOC) for properties, and the rest for proofs.

Evaluation. The following is an evaluation of our work. Firstly, this work is based on concrete low-level code to complete the modeling and verification. We build a finer-grain model of interrupt isolation of a TrustZone-based TEE system, in which all actions are at the instruction level. For completeness, we provide formal definitions of several instructions in ISAs, and we model the two cases for handling IRQ exceptions. Secondly, compared with other TEE verification work, the properties verified in this paper are more complete. We formalize and prove three types of properties for the TrustZone interrupt isolation mechanism, i.e., correctness, safety, and information flow security.

Limitations. In this work, we assume the consistency between our model and the source code, as the model is built manually rather than automatically translated into the theorem prover. In addition, the current model does not involve the cache component. After our analysis, this component is a factor that violates security since malicious attackers can easily use it to obtain confidential data, despite TrustZone's design efforts to provide a lot of protection. In the future, we will consider incorporating this component into our work.

VII. Related Work

Verification of TrustZone. Over the past few years, there has been growing interest in verifying TrustZone. For the Komodo project [10], Ferraiuolo et al. implemented and verified a prototype based on TrustZone that delegates some core hardware mechanisms to a software monitor, which in turn implements enclaves. They define the functional correctness of the monitor and the integrity and confidentiality of enclaves, which ensures the correctness of the monitor's implementation. In the same year, they verified a simplified multi-core prototype [27] of the TrustZone architecture using a hardware description language with annotations, where TrustZone acts as an information flow policy. It can be divided into three situations: a normal world processor cannot read or write to secure world memory, a normal world cannot modify NS bits, and a normal world cannot modify TrustZone control registers. Through static information flow analysis, they proved that the information interaction between the secure world and the normal world in the prototype satisfies the weak noninterference property, and detected some hardware vulnerabilities in commercial processors. Sun et al. [28] proposed a TEE design that uses a narrow interface to communicate with REE, which is convenient for TEE security property verification. They proved the correctness of the security monitor and scheduler and proposed a verification framework to verify end-to-end security in C and assembly language. Miao et al. [29] proposed CVTEE, the first compatible TEE architecture to securely manage sensitive resources and proved its functional correctness and information flow security properties. Miao et al. [30] proposed an access control framework based on verification tags REAL and verified the correctness and security of the framework. They tested it on OP-TEE, and the results showed that it can effectively alleviate some malicious attacks. Ma et al. [31] modeled the memory isolation of the TEE system based on TrustZone and verified the functional correctness of memory management and the information flow security properties of memory isolation. Building upon this, Jin et al. [32] proposed a refinement-based verification of memory isolation mechanism for a trusted execution environment. They added the modeling of context saving during world switch, MMU components, and other functions, defined and proved the refinement relationship between the abstract model and the concrete model, and proved that the concrete model also satisfies the security properties.

Verification of TEE. In addition to TrustZone, TEEs encompass other implementations such as SGX. These are also some verification works regarding SGX and enclaves. Sinha et al. [33] modeled several adversaries and relevant aspects of SGX, including a formal model of enclave programs and a formal model of x86 and SGX instructions. They introduced a tool called Moat, which is used to formally verify that enclave programs running on SGX satisfy confidentiality properties. The verification results demonstrated that the four enclave programs they evaluated satisfied confidentiality. Subramanyan et al. [34] presented a method for designing applications in securely isolated regions in a way that enables verifying their confidentiality, and the evaluation suggested that the method scales to real-world applications. Shinde et al. [35] proposed the first file system interface to protect the integrity of enclaves from malicious operating systems, named BesFS, and proved the security of the BesFS specification.

Verification of Information Flow Security. SeL4 is the first general-purpose operating system kernel that has been formally verified. The basic idea of verifying seL4 [36] is to divide it into three layers: the abstraction layer, the implementation layer, and the C code layer. The abstraction layer contains all the functions of seL4. The proof of information flow security is established in the abstract layer, and the verified properties include noninterference and nonleakage. Subramanyan et al. [37] proposed a verification method based on a Trusted Abstract Platform (TAP), which includes an ideal enclave platform and a parameterized adversary. They formally defined the concept of secure remote execution and proved that TAP satisfies three key security properties of secure remote execution: integrity, confidentiality, and secure measurement. They also proved that Sanctum and Intel SGX are improvements of TAP in the case of adversary parameterization

using the refinement method. Zhao et al. [23], [38] proposed a general security model, defined information flow security properties and a reasoning framework, and used this security model to verify the security of the separation kernel of the ARINC 653 standard. They developed a two-level functional specification: The top-level specification formalizes kernel initialization, partition scheduling, partition management, and inter-partition communication; the second-level specification is refined based on the first-level specification, incorporating processes, process scheduling, and process management. Finally, they verified that the model satisfies information flow security properties.

VIII. Conclusion

This paper presents the formal verification of interrupt isolation for the TrustZone-based TEE. We model a TrustZone-based TEE system at the instruction level, which involves both hardware and software components related to the interrupt isolation mechanism of TrustZone. We formalize three types of properties, which ensure secure interrupts and normal interrupts are correctly handled in their corresponding worlds, that the response to interrupts in the normal world does not influence the context in the secure world, and that there is interrupt isolation between the normal world and the secure world. Using Isabelle/HOL, we prove that the model satisfies these properties, thereby demonstrating the satisfiability of the TrustZone-based TEE system w.r.t correctness, safety, and information flow security. Furthermore, our work provides correctness and reliability guarantees for the interrupt isolation of TrustZone.

IX. Acknowledgment

This work was supported by the National Natural Science Foundation of China (62272458, 62372311, 62002246).

References

[1] GlobalPlatform, "Introduction to Trusted Execution Environments," https://globalplatform.org/resource-publication/introduction-to-trusted-execution-environments/.

[2] ARM Limited, "ARM security technology - building a secure system using TrustZone technology," ARM White Paper, 2009.

[3] K. Kostiainen, J.-E. Ekberg, N.Asokan, and A. Rantala, "On-board credentials with open provisioning," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security (ASIACCS)*. ACM, 2009, pp. 104–115.

[4] H. Sun, K. Sun, Y. Wang, J. Jing, and H. Wang, "TrustICE: Hardware-assisted isolated computing environments on mobile devices," in *Proceedings of the 45th International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society, 2015, pp. 367–378.

[5] R. Liu and M. Srivastava, "PROTC: PROTeCting drone's peripherals through ARM TrustZone," in *Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications (DroNet)*. ACM, 2017, pp. 1–6.

[6] S. Pinto, A. Tavares, and S. Montenegro, "Space and time partitioning with hardware support for space application," in *Proceedings of the 2016 Conference on Data Systems in Aerospace (DASIA)*, 2016.

[7] Samsung, "White paper: An overview of Samsung Knox platform," http://www.samsung.com/es/business-images/resource/white-paper/2014/02/Samsung_KNOX_whitepaper-0.pdf, 2013.

[8] Google for Developers, "Android KeyStore," https://developer.android.com/training/articles/keystore.html, 2022.

[9] Linaro, "OP-TEE," https://www.op-tee.org/, 2014.

[10] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno, "Komodo: Using verification to disentangle secure-enclave hardware from software," in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*. ACM, 2017, pp. 287–305.

[11] D. Rosenberg, "QSEE TrustZone kernel integer overflow vulnerability," in *Proceedings of the 2014 Black Hat Conference*, 2014.

[12] D. Shen, "Exploiting TrustZone on Android," in *Proceedings of the 2015 Black Hat Conference*, 2015.

[13] Huawei Technologies Co., Ltd, "Security warning: Two security vulnerabilities regarding privilege escalation in Huawei Mate7 mobile phones," http://www.huawei.com/cn/psirt/security-advisories/hw-432807, 2020.

[14] Y. Zhang, Z. Chen, H. Xue, and T. Wei, "Fingerprints on mobile devices: Abusing and leaking," in *Proceedings of the 2015 Black Hat Conference*, 2015.

[15] A. Atamli-Reineh, R. Borgaonkar, R. A. Balisane, G. Petracca, and A. Martin, "Analysis of Trusted Execution Environment usage in Samsung KNOX," in *Proceedings of the 1st Workshop on System Software for Trusted Execution (SysTEX)*. ACM, 2016, pp. 7:1–7:6.

[16] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[17] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002.

[18] Linaro, "Trusted Firmware-A," https://github.com/ARM-software/arm-trusted-firmware, 2019.

[19] J. Rushby, *Noninterference, transitivity, and channel-control security policies*. SRI International, Computer Science Laboratory, 1992.

[20] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," *IEEE Journal on selected areas in communications*, vol. 21, no. 1, pp. 5–19, 2003.

[21] D. Von Oheimb, "Information flow control revisited: Noninfluence= noninterference + nonleakage," in *Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS)*. Springer, 2004, pp. 225–243.

[22] T. Murray, D. Matichuk, M. Brassil, P. Gammie, and G. Klein, "Noninterference for operating system kernels," in *Proceedings of the 2nd International Conference on Certified Programs and Proofs (CPP)*. Springer, 2012, pp. 126–142.

[23] Y. Zhao, D. Sann, F. Zhang, and Y. Liu, "Reasoning about information flow security of separation kernels with channel-based communication," in *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 2016, pp. 791–810.

[24] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[25] J. A. Goguen and J. Meseguer, "Unwinding and inference control," in *Proceedings of the 5th IEEE Symposium on Security and Privacy (SP)*. IEEE, 1984, pp. 75–75.

[26] M. Wenzel, "The Isabelle/Isar Reference Manual," http://isabelle.in.tum.de/doc/isar-ref.pdf, 2022.

[27] A. Ferraiuolo, R. Xu, D. Zhang, A. C. Myers, and G. E. Suh, "Verification of a practical hardware security architecture through static information flow analysis," in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2017, pp. 555–568.

[28] H. Sun and H. Lei, "A design and verification methodology for a TrustZone Trusted Execution Environment," *IEEE Access*, vol. 8, pp. 33 870–33 883, 2020.

[29] X. Miao, R. Chang, J. Zhao, Y. Zhao, S. Cao, T. Wei, L. Jiang, and K. Ren, "CVTEE: A compatible verified TEE architecture with enhanced security," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 20, no. 1, pp. 377–391, 2023.

[30] X. Miao, F. Zeng, R. Chang, C. Yu, Z. Zhang, L. Jiang, and Y. Zhao, "Is your access allowed or not? A verified tag-based access control framework for the multi-domain TEE," in *Proceedings of the 13th Asia-Pacific Symposium on Internetware (Internetware)*. ACM, 2022, pp. 252–261.

[31] Y. Ma, Q. Zhang, S. Zhao, G. Wang, X. Li, and Z. Shi, "Formal verification of memory isolation for the TrustZone-based TEE," in *Proceedings of the 27th Asia-Pacific Software Engineering Conference (ASPEC)*. IEEE, 2020, pp. 149–158.

[32] C. Jin, Q. Zhang, Y. Ma, X. Li, G. Wang, Z. Shi, and Y. Guan, "Refinement-based verification of memory isolation mechanism for

Trusted Execution Environment," *Journal of Software*, vol. 33, no. 6, pp. 2189–2207, 2022.

[33] R. Sinha, S. Rajamani, S. A. Seshia, and K. Vaswani, "Moat: Verifying confidentiality of enclave programs," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2015, pp. 1169–1184.

[34] P. Subramanyan, R. Sinha, I. Lebedev, S. Devadas, and S. A. Seshia, "A formal foundation for secure remote execution of enclaves," in *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 2435–2450.

[35] S. Shinde, S. Wang, P. Yuan, A. Hobor, A. Roychoudhury, and P. Saxena, "BesFS: A POSIX filesystem for enclaves with a mechanized safety proof," in *Proceedings of the 29th USENIX Security Symposium (USENIX Security)*. USENIX Association, 2020, pp. 523–540.

[36] T. Murray, D. Matichuk, M. Brassil, P. Gammie, T. Bourke, S. Seefried, C. Lewis, X. Gao, and G. Klein, "seL4: from general purpose to a proof of information flow enforcement," in *Proceedings of the 34th IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2013, pp. 415–429.

[37] P. Subramanyan, R. Sinha, I. Lebedev, S. Devadas, and S. A. Seshia, "A formal foundation for secure remote execution of enclaves," in *Proceedings of the 2017 ACM Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 2435–2450.

[38] Y. Zhao, D. Sanán, F. Zhang, and Y. Liu, "Refinement-based specification and security analysis of separation kernels," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 16, no. 1, pp. 127–141, 2017.