IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 23, 2021, accepted May 25, 2021, date of publication June 7, 2021, date of current version June 15, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3087421

Demystifying Attestation in Intel Trust Domain Extensions via Formal Verification

MUHAMMAD USAMA SARDAR, (Member, IEEE), SAIDGANI MUSAEV, AND CHRISTOF FETZER
Chair of Systems Engineering, Faculty of Computer Science, Institute of Systems Architecture, Technische Universität Dresden, 01069 Dresden, Germany
Corresponding author: Muhammad Usama Sardar (muhammad_usama.sardar@mailbox.tu-dresden.de)

ABSTRACT In August 2020, Intel asked the research community for feedback on the newly offered architecture extensions, called Intel Trust Domain Extensions (TDX), which give more control to Trust Domains (TDs) over processor resources. One of the key features of these extensions is the remote attestation mechanism, which provides a unified report verification mechanism for TDX and its predecessor Software Guard Extensions (SGX). Based on our experience and intuition, we respond to the request for feedback by formally specifying the attestation mechanism in the TDX using ProVerif's specification language. Although the TDX technology seems very promising, the process of formal specification reveals a number of subtle discrepancies in Intel's specifications that could potentially lead to design and implementation flaws. After resolving these discrepancies, we also present fully automated proofs that our specification of TD attestation preserves the confidentiality of the secret and authentication of the report by considering the state-of-the-art Dolev-Yao adversary in the symbolic model using ProVerif. We have submitted the draft to Intel, and Intel is in the process of making the changes.

INDEX TERMS Formal verification, symbolic security analysis, ProVerif, trusted execution environment, trust domains, Intel TDX, remote attestation.

I. INTRODUCTION

Public cloud environments have proven themselves as a cost-effective solution for a variety of workloads. However, an important challenge that software stakeholders currently face in a public cloud is the confidentiality and integrity of sensitive data, such as private user data or intellectual property. Different hardware-based solutions, such as Trusted Execution Environments (TEEs) [1] and Trusted Platform Modules [2], [3], were offered. Moreover, many different TEE implementations have emerged in the past years for public clouds [4], [5]. These implementations are represented by two major categories: process-based and Virtual Machine (VM)-based. In process-based category, Intel released Software Guard Extensions (SGX) [4] in 2014. Intel SGX is supported by many different software platforms [6]–[8] and proved itself as a practical solution for security in a public cloud. VM-based solutions offer more flexibility, at the cost of managing a bigger software stack. In this category, Intel announced new extensions for its Instruction Set Architecture (ISA), namely Trusted Domain Extensions (TDX) [9] in August 2020. These extensions are a combination of Intel Virtual Machine Extensions, Intel Multi-Key Total Memory Encryption (MKTME) [10], and Intel CPU-attested software module. Intel TDX inherits some aspects of Intel SGX.

Compared with legacy Virtual Machine Monitor (VMM), TDX is supposed to help isolate resource management software into untrusted side, while the Virtual Machine (VM) itself is considered as trusted software. Compared to SGX, TDX has more control over the software stack and it also improves the management of CPU resources. Both SGX and TDX offer memory encryption, which guarantees confidentiality and integrity of the memory. Unlike SGX, TDX is designed to run legacy applications with the support of the secure, not-encrypted, digitally-signed service module, referred to as TDX module. This module is included in the Trusted Computing Base (TCB) and cannot be changed by untrusted software, including VMM. VMM can switch to TD, also known as Secure-Arbitration Mode (SEAM). In this mode, CPU is able to create, initialize and schedule different TDs. Assigning MKTME private encryption keys is done via TDX module (which is trusted), and integrity of these keys cannot be violated by the untrusted software, including VMM [9].

The associate editor coordinating the review of this manuscript and approving it for publication was Roberto Nardone.

One of the key features of Intel TDX is TD remote attestation, which extends the CPU-attestation from Intel SGX. The process of generation of a report in TDX is different from that of SGX, and thus TDX provides new instructions to generate and locally verify the attestation report [11]. Moreover, compared to SGX attestation, TD attestation provides more guarantee about the software stack running inside TEE, e.g., secure and shared pages in TD, MSRs state, initial TD measurements, and runtime TD measurements. This allows a challenger to remotely attest the full software stack with some intermediate runtime measurements. The attestation process still relies on the Quoting Enclave, just like SGX. The Quoting Enclave generates a remotely verifiable Quote with platform attestation key from the data generated by the specific TD. Quote is then sent to a challenger and can be verified by Attestation-Verification Service [9].

The security analysis of a TEE that supports attestation is typically performed informally using testing with a computational model of the adversary. Although such an analysis is suitable to show the presence of security errors, it cannot be used to prove their absence [12]. If left in the system, these errors may have serious consequences, for instance, significant damage to the finances and reputation of a company. To overcome this limitation of testing, rigorous and sound formal methods have been utilized to analyze the security goals on a formal model. For instance, Intel developed some formal tools [13]–[15] for establishing formal correctness of SGX. However, *the verification using these tools does not cover the attestation process*, which is a crucial part of Intel SGX for practical use [16]. Subramanyan *et al.* [17] presented a formalization of attestation for an idealized abstract platform and semi-automated verification approach to establish the identity of the enclave. However, *the authors assume axioms about the correctness of Intel SGX remote attestation*. We complemented these works by focusing on the symbolic verification of the cryptographic protocol in the remote attestation based on Enhanced Privacy ID (EPID) [18] and Data Center Attestation Primitives (DCAP) [19] in our recent work.

To the best of our knowledge, there is no publicly available formal specification and verification of TD attestation. The formal specification is essential to formally reason about the guarantees for security-critical use cases and to increase system designer's confidence. With this motivation, this work presents the formal specification of TD attestation based on a variant of applied pi-calculus. Interestingly, the process of formal specification reveals a number of subtle discrepancies in Intel's informal specifications. The discrepancies include not only the ambiguous names and missing fields in data structures, but also various instances of inconsistent information that could potentially lead to design and implementation flaws. Since TDX is currently in the design phase and the implementation is not yet released, we resolve these discrepancies based on our experience with its predecessor SGX and intuition. We then analyze the most relevant security properties, namely confidentiality and authentication, considering the state-of-the-art symbolic model of the adversary, i.e., Dolev-Yao model [20]. Our implementation is based on the state-of-the-art symbolic verification tool ProVerif [21] mainly because of its easy portability to other symbolic verification tools (e.g., Tamarin prover [22] via SAPIC), computational verification tools (e.g., CryptoVerif [23]). The main contributions of the work include:

- Precise specification of the TD attestation protocol, including all involved entities in ProVerif
- Identification of discrepancies including inconsistent information that could potentially lead to design and implementation flaws
- Automated verification of confidentiality of the shared secret and authentication of reports in ProVerif

The rest of the paper is organized as follows: We present the existing approaches for the formal verification of Intel SGX and its attestation in Section II. Section III presents an overview of our analysis approach for the specification and verification of the TD attestation. Then we present the formal specification of TD attestation process in Section IV. In Section V, we outline the discrepancies revealed in the Intel's specifications of TD attestation in TDX. Section VI presents the formalization of security properties and the verification results. Finally, we conclude the paper in Section VII with some future directions of research.

II. RELATED WORK

There is no publicly available formal specification and verification of Intel TD attestation. The most closely related work in general is the formal verification effort related to Intel SGX. Since SGX has been widely studied, we focus on the main approaches used for formal verification. In Section II-A, we give an overview of the formalization of secure remote execution by a group of researchers at the University of California, Berkeley and Massachusetts Institute of Technology, Cambridge [17]. Then we present the formalizations by Intel in Section II-B. We present related works on the use of symbolic security analysis tools for formal verification of remote attestation in Section II-C, pointing out the differences from the current work on TD attestation.

A. SECURE REMOTE EXECUTION

Subramanyan *et al.* [17] presented a formal definition for secure remote execution of enclaves and its decomposition into three main properties, namely confidentiality, integrity, and secure measurement. The proposed verification methodology is based on an idealized abstraction of enclave platforms, called Trusted Abstract Platform (TAP), along with a parameterized adversary model. Using an intermediate verification language, Boogie [24], the authors prove that TAP satisfies the requirements for secure remote execution. Additionally, they utilize stuttering simulation [25] to prove that Intel SGX and MIT Sanctum [26] implementations refine TAP under certain parameterizations of the adversary. Thus, the results can be used to compare security guarantees of different enclave platforms.

This verification approach, however, is semi-automated as the user has to find the correct inductive invariants manually to prove the properties. In this case, 935 such annotations were required. This manual effort to re-prove each time a software policy is updated makes it difficult for runtime verification. Specifically, *for Intel SGX, the authors ignore the details of remote attestation and instead assume various axioms.*

B. FORMALIZATIONS BY INTEL

To the best of our knowledge, Intel has not published any work on the formalization of remote attestation of SGX or TDX. However, Intel has described some works, e.g., [27], on the formal verification of Intel SGX focusing on the enclave page management and page table translations tracking [28]. Intel addresses the verification in two steps: first, to prove the sequential correctness, and second, to prove that SGX is linearizable [15]. Some undiscovered bugs were identified in both steps [27].

Intel verifies the correctness of Intel SGX in the sequential (or single-threaded) setting by using Deductive Verification Framework (DVF), which is a language and Satisfiability Modulo Theories (SMT)-based deductive verification tool developed at Intel [13]. DVF models a system as a transition system consisting of states and transitions. An execution is an interleaving of transitions [29]. DVF supports Hoare-style and assume-guarantee reasoning. It maps proof obligations into the multisorted first-order logic supported by modern SMT solvers, e.g., Yices SMT solver. The properties verified using DVF are SGX invariants, security properties for enclave confidentiality and enclave data integrity, and lifecycle properties. Critical bugs were identified during this verification [27].

DVF has a couple of limitations. First, it does not model concurrency, whereas SGX has 22 instructions that share the concurrent data structure, some of which contain as many as 50 interleaving points [15]. Second, the verification is semi-automated and includes a painful process of manually generating the auxiliary invariants [30].

To address concurrency issues, Intel considers linearizability [31] as a correctness condition. Intel develops a framework called iPave [14] to prove SGX linearizability, and finds the linearization point using heuristics [15]. The properties, such as system invariants or per-state assertion, can be specified [27]. The graphical iFlow model is converted to XML representation. iPave contains two compilers. The first compiler translates XML representation to a new logical formalism, called a Logic Sequence Diagram (LSD) [14]. The second compiler translates the LSD to a symbolic finite state machine (FSM) with guarded transitions between states. Bounded model checking [32] for task-reachability and termination analysis can then be performed on the resulting finite-state machine. However, Intel provides no information about k-bound used for this verification.

Intel identified critical security bugs in SGX using iPave, proving the linearizability of SGX instructions using the assertion analysis and proving some SGX invariants, e.g., a pending page is never cached [27]. However, the framework has some limitations. Non-determinism cannot be modeled in iPave since disjunctive transitions are not currently supported due to performance reasons [14]. Moreover, the graphical input leads to a heavy translation burden which is frequently a source of modeling errors. Finally, because of the lack of abstraction mechanism, it is not easy to experiment with it using various SGX configurations [15].

To address the last two issues mentioned above, a compiler, known as Accordion [15], was developed at Intel to verify linearizability automatically via model checking. Accordion is implemented as an embedded domain-specific language in Haskell [33]. Each SGX instruction is specified as a function with necessary arguments. The syntax is close to the informal specification language used by the SGX architects. Intel reports that the bugs previously found by iPave could be replicated in Accordion. However, no new bugs were discovered.

C. SYMBOLIC SECURITY ANALYSIS

Various works have utilized the symbolic security analysis tools, such as Tamarin prover [22], Cryptographic Protocol Shapes Analyzer (CPSA) [34], and ProVerif [21], for the analysis of protocols. In the context of remote attestation, Tamarin prover was used in the analysis of Direct Anonymous Attestation (DAA) [35]. CPSA was used for the analysis of EPID-based remote attestation [36]. In our previous work, we utilized ProVerif for the proof of confidentiality and integrity properties in EPID-based [18] as well as DCAP-based [19] remote attestation. ProVerif was also used for the verification of frameworks based on Intel SGX, such as [37].

Compared to these and similar works on formalization of SGX remote attestation, the TD attestation is more generic, as it provides additional guarantees, such as runtime measurements. Moreover, the data structures and process of creation of attestation report is different in TDX as compared to SGX. We believe that TD attestation protocol will be used in the future as a unified approach for TD and SGX attestation. We finally remark that it is the first formal verification for TD attestation to the best of our knowledge.

III. OVERVIEW OF ANALYSIS APPROACH

A. SELECTION OF TOOL

Since the TDX implementation is not yet available, we focus on the design-level security analysis. Specifically, we create a formal (symbolic) model of the TD attestation. A wide range of tools [38] are available for symbolic security analysis. Since the problem is in general undecidable [39], a natural approach is to use the automated tools first, and then use the user-guided tools, such as Isabelle, only in case the properties could not be proved by automated tools. Since we are interested in trace-based properties, such as confidentiality and integrity, we focus on the unbounded trace-based

tools. Among such tools, ProVerif [21] and Tamarin [22] are considered the state-of-the-art. ProVerif is based on process calculus, whereas Tamarin is based on multiset rewriting. Both tools support trace-based and diff-equivalence properties, as well as user-defined equational theories.

We decided to use ProVerif due to various reasons. Firstly, in general ProVerif is more automatic while Tamarin relies more on user interaction. Tamarin accepts ProVerif-like input (via SAPIC [22]) but not vice versa. Thus, if the properties cannot be proved in ProVerif, they could be tried by importing the model in Tamarin via SAPIC. Secondly, ProVerif model can be easily converted to computationally sound model by using CryptoVerif [23], which supports the syntax of ProVerif. Finally, the performance analysis of both tools have mostly shown (cf. [40]) that ProVerif is faster than other tools for similar problems.

B. APPROACH

For the analysis of TD attestation in ProVerif, we extract the protocol from Intel's official documentation, mainly [9], [11], [41]–[46]. This turns out to be the most challenging part because of the presence of subtle flaws in the informal description. Some of them are described in Section V. The attestation mechanism is then formalized in ProVerif's input specification language, which is a variant of applied pi-calculus [47] enriched with cryptographic primitives. A formal specification for the attestation mechanism is generated by producing an unbounded replication of parallel composition of communicating processes along with an adversary with Dolev-Yao [20] capabilities. Such an adversary is in complete control of the communication network, has an unbounded computational power, but manipulates protocol messages according to some predefined rules with idealized cryptography [48]. The behavior of the adversary is non-deterministic in order to model different choices of actions available to the adversary, so all possible combinations of actions are covered. Exhaustive analysis of such non-determinism is impractical in functional testing-based analysis [21] and is one of the strengths of our approach.

The approach of ProVerif is sound but not complete. ProVerif automatically translates protocol specification into a set of Horn clauses [49]. Horn clauses are first-order logical formulas of the form $F_1 \wedge \ldots \wedge F_n \Rightarrow F$, where $F_1, \ldots, F_n$ and F are facts. The rationale is that Horn clause representation is more precise than tree-automata because it preserves the relational information on messages [21]. The desired properties expressed are also automatically converted to derivability queries on the Horn clauses. ProVerif uses a sound and complete (see [21] for a proof) algorithm based on *resolution with free selection* [50] to determine whether a fact is derivable from the clauses. If the fact is not derivable, then the desired security property is proved. If the fact is derivable, then ProVerif attempts to reconstruct an attack at the pi-calculus level. If this attack reconstruction succeeds, the attack against the considered property is detected. If the attack reconstruction fails because of the abstractions in the Horn clause representation, ProVerif cannot decide whether the property is satisfied or not. The abstractions are required because of the undecidability problem. However, soundness is not compromised by these abstractions. Moreover, in case it cannot decide, the additional information such as the attack derivation helps in understanding where the tool fails to prove and it is possible to use more advanced options to guide the tool to the proof in such cases.

IV. FORMAL SPECIFICATION OF TD ATTESTATION

We thoroughly explore available Intel TDX documentation, mainly [9], [11], [41]–[46], to extract the specification of the attestation mechanism in Intel TDX. In this section, we first give an informal overview of entities and protocol, and then present the formal specification of the protocol.

A. OVERVIEW OF CORE ENTITIES

There are six key entities in the TD attestation protocol: Quoting Enclave (QE), host Virtual Machine Manager (VMM), guest trust domain (TD), Intel TDX module, CPU hardware and challenger. QE represents Intel provided enclave that signs the report body (after its successful verification) to form a remotely-verifiable Quote. The VMM represents an untrusted hypervisor that manages Virtual Machines. The guest TD represents the enhanced Virtual Machine, which is initialized and measured at the boot time. The TDX module is an Intel provided software, which is provided and signed by Intel. It manages the interaction between the TD and the VMM. The TDX module is a part of attestation metadata. The CPU hardware generates and verifies the report. The challenger (also known as relying party) is the remote party that performs the attestation verification.

B. INFORMAL OVERVIEW OF PROTOCOL

In this section, we briefly describe the protocol informally. In a nutshell, the challenger initiates the attestation request. The guest TD sends (the hash of) its public key to the Intel TDX module, which forwards the received public key along with the hash of assembled static and runtime measurements to the CPU hardware. The CPU hardware creates a report, so-called SEAMREPORT, and sends it over to the Intel TDX module, which extends the report by appending TDX-specific TEE information. This extended report, called TDREPORT, is sent over to TD QE via guest TD and host VMM. TD QE checks hashes and requests CPU hardware to verify the report. If the report is verified, QE signs the report body to form a Quote. The Quote is then sent over via host VMM and guest TD to the challenger. The mechanism for Quote generation is depicted in Fig. 1, which is based on [9], [11], [41]–[46].

The generation process for SEAMREPORT is elaborated in Fig. 2, which is based on [11]. The values of TDX module SVN `tsvn` and measurement of Intel TDX module `mrs` are extracted from the data structure `CRPL_TEE_TCB_INFO` and put into the data structure for measurement and configuration of TDX module `tcbi`. Then the CPU checks the Intel

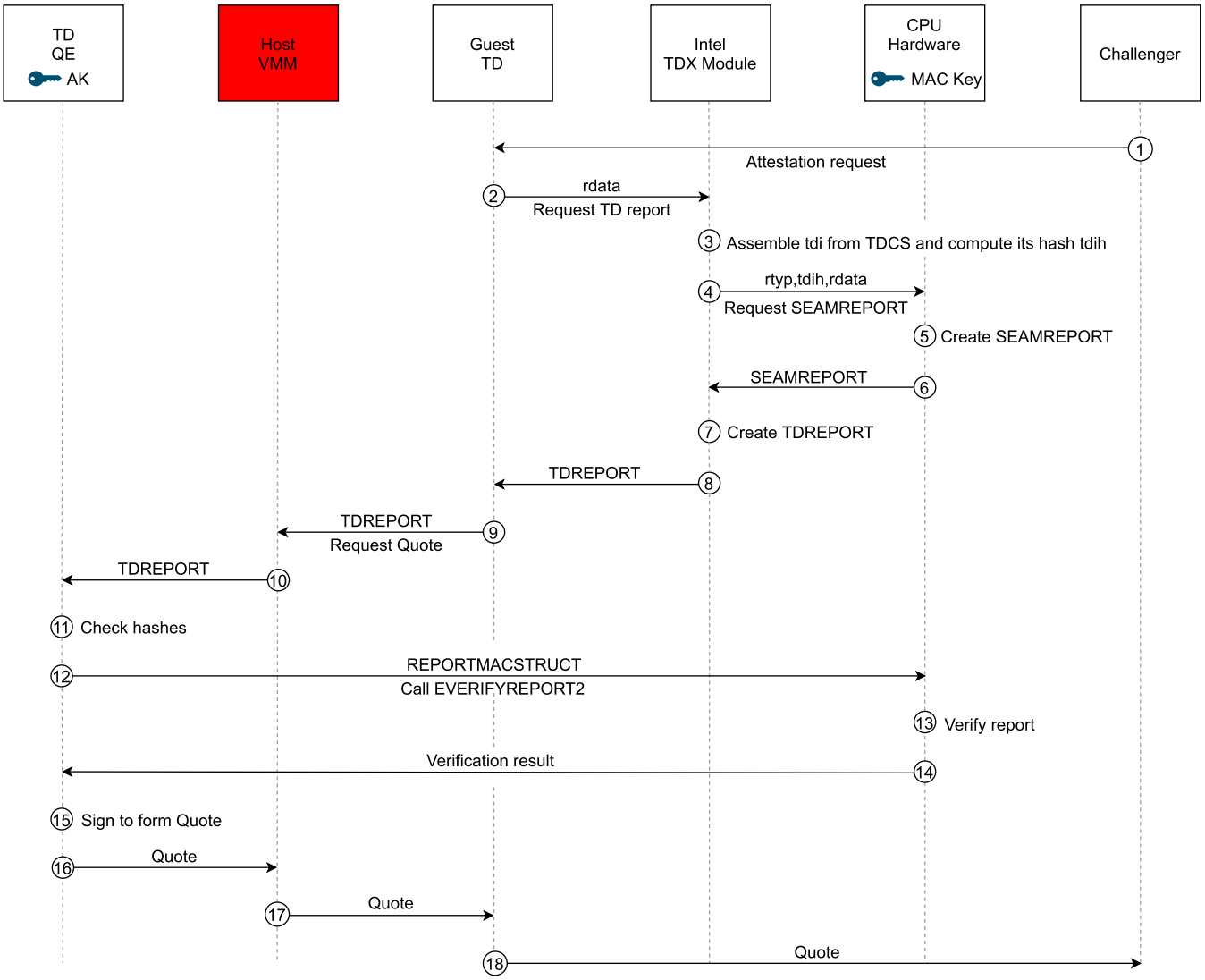

FIGURE 1. Process flow of the Quote generation for Intel TDX. Convention is that the text above the arrow represents data being sent over the channel and the text below the arrow represents requests or function calls. All the data shown in this figure is sent in unencrypted form. AK represents Attestation Key. Host VMM, shown in red, is untrusted.

SEAM flag. If it is false, then CPU also extracts the measurement of TDX module signer `mrss` and additional configurations `att` from the data structure `CRPL_TEE_TCB_INFO` and stores in the data structure for measurement and configuration of TDX module `tcbi`. In this case, the value of valid field `val` in `tcbi` is updated to 0xFFFF to indicate that the fields `mrss` and `att` are also valid in this case. If the Intel SEAM flag is true, the values `mrss` and `att` are not updated and the value of valid field `val` in `tcbi` is updated to 0x1FF to indicate that the fields `mrss` and `att` are not valid in this case. The reserved field `res3` for future use contains zeros.

Once the `tcbi` data structure is populated, its SHA384 hash is stored as the TEE_TCB_INFO_HASH `tcbh` in the data structure REPORTMACSTRUCT `rms`. The `rms` structure populates the fields report type `rtyp`, CPUSVN `csvn`, TEE_INFO_ HASH `tdih`, and report data `rdata` from RDX, CRPL_CPUSVN, R9 and R8, respectively. The reserved fields `res1` and `res2` for future use are all zeros. Finally, HMAC_SHA256 is computed over the fields of `rms` using a MAC key from CR_REPORT_KEY2, which is available to the CPU only. The resulting MAC `mac` is stored in the last field in the structure `rms` [11].

C. FORMAL SPECIFICATION

Inspired by the recommendation of Barbosa *et al.* [38], we clearly outline the trusted and untrusted parts of the system model for transparency and progress. The specification of TD attestation consists of defining three main aspects. First, it defines the core entities involved in the attestation mechanism and whether they are trusted. Each core entity is modeled by a *process*. Second, it defines the computations performed by the core entities and whether the adversary is

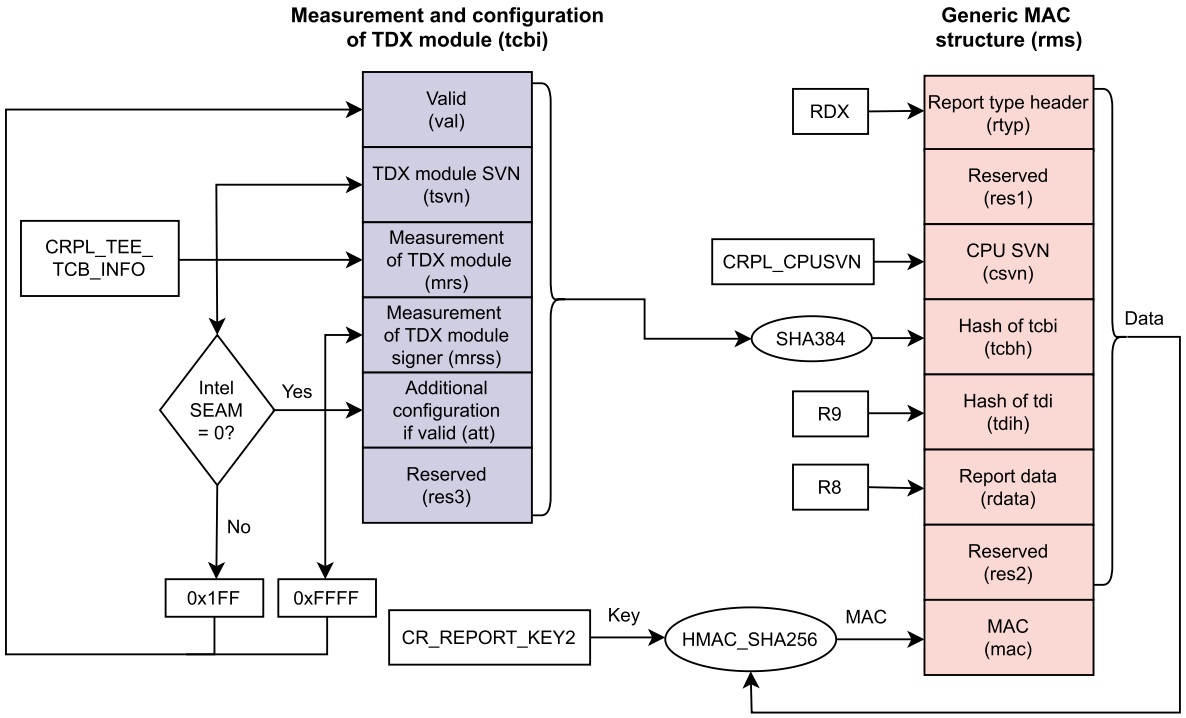

FIGURE 2. The SEAMREPORT structure `smr` generated by CPU using SEAMOPS[SEAMREPORT] leaf function in Intel TDX. Ovals represent functions.

able to perform these computations. The computations are modeled using *functions* [48]. Third, it defines the communications between the core entities and whether the channels used for communication are private. The communications are modeled using *channels*.

The core entities and their trust status is defined as a set of tuples of the form:

$$(eid, trust) \tag{1}$$

where *eid* represents the entity identifier, and $trust \in \{true, false\}$, where *true* represents *eid* is trusted, and *false* represents that *eid* is not trusted. For Intel TDX:

$$\{(QE, true), (VMM, false), (TD, true), (TDX, true), (CPU, true), (Ch, true)\}$$

where `QE` represents Quoting Enclave, `VMM` represents the host virtual machine manager that is untrusted, `TD` represents the guest trust domain, `TDXM` represents the TDX module, `CPU` represents the CPU hardware, and `Ch` represents the challenger, as briefed in Section IV-A.

The overall process *P* is defined as the parallel composition of the honest core processes and the arbitrary process for adversary, i.e.,

$$P \equiv !(QE \mid\mid TD \mid\mid TDXM \mid\mid CPU) \mid\mid !Ch \mid\mid !Adv, \tag{2}$$

where $\mid\mid$ represents the parallel composition, ! represents the replication, *Adv* represents the Dolev-Yao adversary in parallel to the core processes. Note that `VMM` is a part of *Adv* and therefore not explicitly mentioned.

The functions and their availability to the adversary are defined as a set of tuples of the form:

$$(func, adv) \tag{3}$$

where *func* represents the function name, and $adv \in \{true, false\}$, where *true* represents it is available to adversary, and *false* represents the adversary is unable to perform the computation done by the function. For Intel TDX, the set of major functions is:

$$\{(hash, true), (hmac, true), (sign, true), (verifysign, true), (aenc, true), (adec, true)\}$$

where `hash` is a unary function that computes the SHA384 hash of the input. `hmac` represents HMAC_SHA256 of the message under the specified key. If `hmac` is considered pseudorandom function (PRF), then the message is not extractable from the MAC, whereas if `hmac` is assumed to be unforgeable (UF-CMA) function, then the message is extractable from the MAC. `sign` represents the Elliptic Curve Digital Signature Algorithm (ECDSA) signature over the message with the specified signature key. Since ECDSA is signature with appendix [51], the verification function `verifysign` (implemented as a *destructor* in ProVerif) requires both message and signature as inputs for the verification, and gives a boolean result:

$$verifysign(spk(sk), m, sign(sk, m)) \rightarrow true \tag{4}$$

where function `spk` computes the public key corresponding to the private signature key `sk`, and `m` represents the message.

`aenc` represents the asymmetric encryption that computes the ciphertext corresponding to a secret message with the public key of the target and the function `adec` (implemented as a *destructor* in ProVerif) represents corresponding asymmetric decryption which retrieves the original message with the secret key:

$$adec(k, aenc(pk(k), m)) \rightarrow m \tag{5}$$

where function `pk` computes the public key corresponding to the secret key `k`, and `m` represents the message.

The channel is defined as

$$(chan, pub) \tag{6}$$

where *chan* represents the channel name, and $pub \in \{true, false\}$, where *true* represents it is a public channel, and *false* represents it is a private channel and thus the data sent over this channel is not available to the adversary. For Intel TDX:

$$\{(c\_QE\_CPU, false), (c\_TD\_TDXM, false), (c\_TDXM\_CPU, false), (c, true)\}$$

where `c_QE_CPU` represents the private channel between `QE` and `CPU`, `c_TD_TDXM` represents the private channel between `TD` and `TDXM`, `c_TDXM_CPU` represents the private channel between `TDXM` and `CPU`, and `c` represents the public channel for all other communications. The channel between TD and TDX module is private because TD communicates with TDX module via TDCALL without leaving the TD.

A possible specification of the protocol is as follows:

$$\begin{aligned}
&1: Ch \xrightarrow{\text{req}} TD\\
&2: TD \xrightarrow{\text{rdata}} TDXM\\
&3: TDXM: tdih = hash(tdi)\\
&4: TDXM \xrightarrow{\text{rtyp,tdih,rdata}} CPU\\
&5: CPU: smr = (rms, tcbi)\\
&6: CPU \xrightarrow{\text{smr}} TDX\\
&7: TDXM: tdr = (smr, res4, tdi)\\
&8: TDXM \xrightarrow{\text{tdr}} TD\\
&9: TD \xrightarrow{\text{tdr}} VMM\\
&10: VMM \xrightarrow{\text{tdr}} QE\\
&11: QE: tcbh = hash(tcbi) \;\&\&\; tdih = hash(tdi)?\\
&12: QE \xrightarrow{\text{rms}} CPU\\
&13: CPU: ReceivedMAC = mac(MACkey, rptbody)?\\
&14: CPU \xrightarrow{\text{result}} QE\\
&15: QE: Quote = (rptbody, Sig)\\
&16: QE \xrightarrow{\text{Quote}} VMM\\
&17: VMM \xrightarrow{\text{Quote}} TD\\
&18: TD \xrightarrow{\text{Quote}} Ch\\
&19: Ch: verifysign(spkAK, QuoteBody, Sig)?\\
&20: Ch \xrightarrow{\text{aenc(rdata,secret)}} TD
\end{aligned} \tag{7}$$

All the above symbols are defined in the corresponding step of the protocol below:

Step 1: The challenger process `Ch` sends an attestation request `req` to the guest TD `TD` that it wants to verify.

Step 2: On receiving the request for attestation, `TD` calls TDG.MR.REPORT leaf function [41] (to request TDREPORT `tdr`) by sending (the hash of) its public key in the argument report data `rdata` to the Intel TDX module `TDXM`.

Step 3: `TDXM` assembles the TD information data structure `tdi` from Trust Domain Control Structure (TDCS) [41], and computes its SHA384 hash `tdih` using `hash` function:

$$tdih \equiv hash(tdi) \tag{8}$$

where `tdi` is defined as:

$$tdi \equiv \langle tdatt, xfam, mrtd, mrc, mro, mroc, rtmr, res5\rangle \tag{9}$$

where `tdatt` represents attributes of TD, `xfam` represents extended features available mask, `mrtd` represents measurement of initial pages added to TD, `mrc` represents hash of ID for non-owner-defined configurations of TD, `mro` represents hash of ID of TD's owner, `mroc` represents hash of ID for owner-defined configurations of TD, `rtmr` represents an array of four runtime measurement registers, and `res5` represents reserved fields, as shown in Fig. 3, which is based on [41].

Step 4: `TDXM` then calls SEAMOPS[SEAMREPORT] leaf function by sending the arguments report type `rtyp`, hash `tdih` of TDINFO structure, and report data `rdata` to the CPU module `CPU` [11].

Step 5: In response to this, `CPU` generates a report called as SEAMREPORT `smr` [11], defined as:

$$smr \equiv \langle rms, tcbi\rangle \tag{10}$$

where `tcbi` represents the data structure for measurement and configuration of TDX module, and defined as:

$$tcbi \equiv \langle val, tsvn, mrs, mrss, att, res3\rangle \tag{11}$$

In (10), `rms` represents a generic MAC structure referred to as REPORTMACSTRUCT, and defined as:

$$rms \equiv \langle rtyp, res1, csvn, tcbh, tdih, rdata, res2, mac\rangle \tag{12}$$

where `tcbh` represents the SHA384 hash taken over `tcbi`:

$$tcbh \equiv hash(tcbi) \tag{13}$$

`mac` in (12) represents SHA256-based HMAC over the remaining fields of `rms` using the MAC key, i.e.,

$$mac \equiv hmac(MACkey, (rtyp, res1, csvn, tcbh, tdih, rdata, res2))$$

A visualization of these data structures along with the operations is presented in Figs. 2 and 3.

Step 6: `CPU` sends SEAMREPORT `smr` to `TDXM`.

Step 7: TDXM assembles all information to form TDREPORT tdr [41], defined as:

$$tdr \equiv \langle smr, res4, tdi \rangle \tag{14}$$

where smr represents SEAMREPORT as defined in (10), res4 represents reserved fields, and tdi represents TD information data structure, as defined in (9).

Step 8: TDXM sends TDREPORT tdr to TD.

Steps 9 and 10: TD forwards tdr to the TD QE process QE via host VMM requesting for a remotely verifiable Quote, because tdr cannot be remotely verified. Since host VMM is untrusted and therefore a part of the adversary, these two steps are equivalent to sending tdr from guest TD directly to TD QE over a *public* channel. This is implemented in ProVerif accordingly.

Step 11: QE, on receiving the report tdr, checks the hashes in the report.

Step 12: QE then calls ENCLU[EVERIFYREPORT2] leaf function [11] with the argument rms.

Step 13: CPU performs the verification of rms. The verification consists of three main steps: (i) verify that the header rtyp in the report is correct, (ii) verify that the CPUSVN csvn is a valid value, and (iii) compute the MAC over the fields in the report body rptbody using the MAC key MACkey, and verify that the computed MAC matches the value in the field mac of the received report (represented as receivedMAC) [11].

Step 14: CPU sends the verification result result back to QE. Each session between TD QE and CPU hardware is secure by the design of TDX [9].

Step 15: If hashes match (step 11) and the report is verified by CPU (step 13), QE replaces the MAC in the REPORTMACSTRUCT rms with the signature using the attestation key AK to form a Quote, defined as:

$$Quote \equiv \langle rptbody, Sig \rangle \tag{15}$$

where Sig represents the signature over rptbody, i.e.,

$$Sig \equiv sign(AK, rptbody) \tag{16}$$

Steps 16 and 17: The Quote is then sent back to TD via VMM. As in steps 9 and 10, this is equivalent to sending Quote directly to TD over a *public* channel. This is implemented in ProVerif accordingly.

Step 18: TD sends the Quote to the challenger Ch.

Step 19: Ch, on receiving the Quote, verifies the signature Sig on the Quote body QuoteBody. This assumes that it has the public key spkAK corresponding to the Attestation Key (AK). Ch can use the attestation-verification service to verify the Quote. This verification is based on the Data Center Attestation Primitives (DCAP) attestation mechanism [52], which has been independently formally verified in our recent work [19], and thus we skip the details here. After verification, it can analyze the static and runtime measurements as well as other configurations to make a decision whether to trust the TD.

Step 20: After verification, it sends a secret secret encrypted with the public key of TD contained in rdata using aenc function. TD can decrypt secret using its private key, and then the shared secret can then be used for secure communication.

V. DISCREPANCIES IN THE OFFICIAL LITERATURE OF INTEL TDX

One of the major challenges in the formal specification of Intel TD attestation is the presence of various discrepancies in the official literature of Intel TDX. An exhaustive list of discrepancies identified is out of the scope of this paper. We divide these discrepancies into three main categories: (i) ambiguous (re-)naming/undefined names, (ii) missing fields in data structures, and (iii) inconsistent information. For reference, we provide a few examples of each of them with pointers to the section numbers, wherever applicable, in Intel's literature. We also relate these examples to the symbols defined in Section IV and the most closely related entity of Section IV-A.

A. AMBIGUOUS (RE-)NAMING/UNDEFINED NAMES

The official literature of Intel TDX contains various undefined data structures, fields, instructions, and arrays. It is unclear why even the same document contains different names for the same entity. Our initial guess was that it might be because of different perspectives from the VM and VMM, but this did not help. We provide a few examples, classified according to the most closely related entity in which they occur, along with our intuition for each:

1) CPU HARDWARE

- SEAMINFO data structure (as used in, for example, Section 18.5.2 in [41], and SEAMREPORT leaf description in [11]) should be the same as TEE_TCB_INFO data structure (tcbi).
- The leaf functions EVERIFYTDREPORT2 (as used in, for example, Fig. 2.4 in [41]), EVERIFYREPORT (as used in, for example, SEAMREPORT leaf description in [11], and Section 2 in [9]), and VERIFYREPORT (as used in, for example, Fig. 8-2 in [44]) should be the same as EVERIFYREPORT2 leaf function. This is because EVERIFYREPORT2 leaf function is the only leaf function that is defined for the verification of the report in [11].
- tmp_tdreport, as used in SEAMREPORT leaf operation in [11], should instead be tmp_seamreport.

2) TDX MODULE

- TEE_INFO data structure (as used in, for example, Table 18.11 in [41], and SEAMREPORT leaf description in [11]) should be the same as TDINFO data structure (tdi).

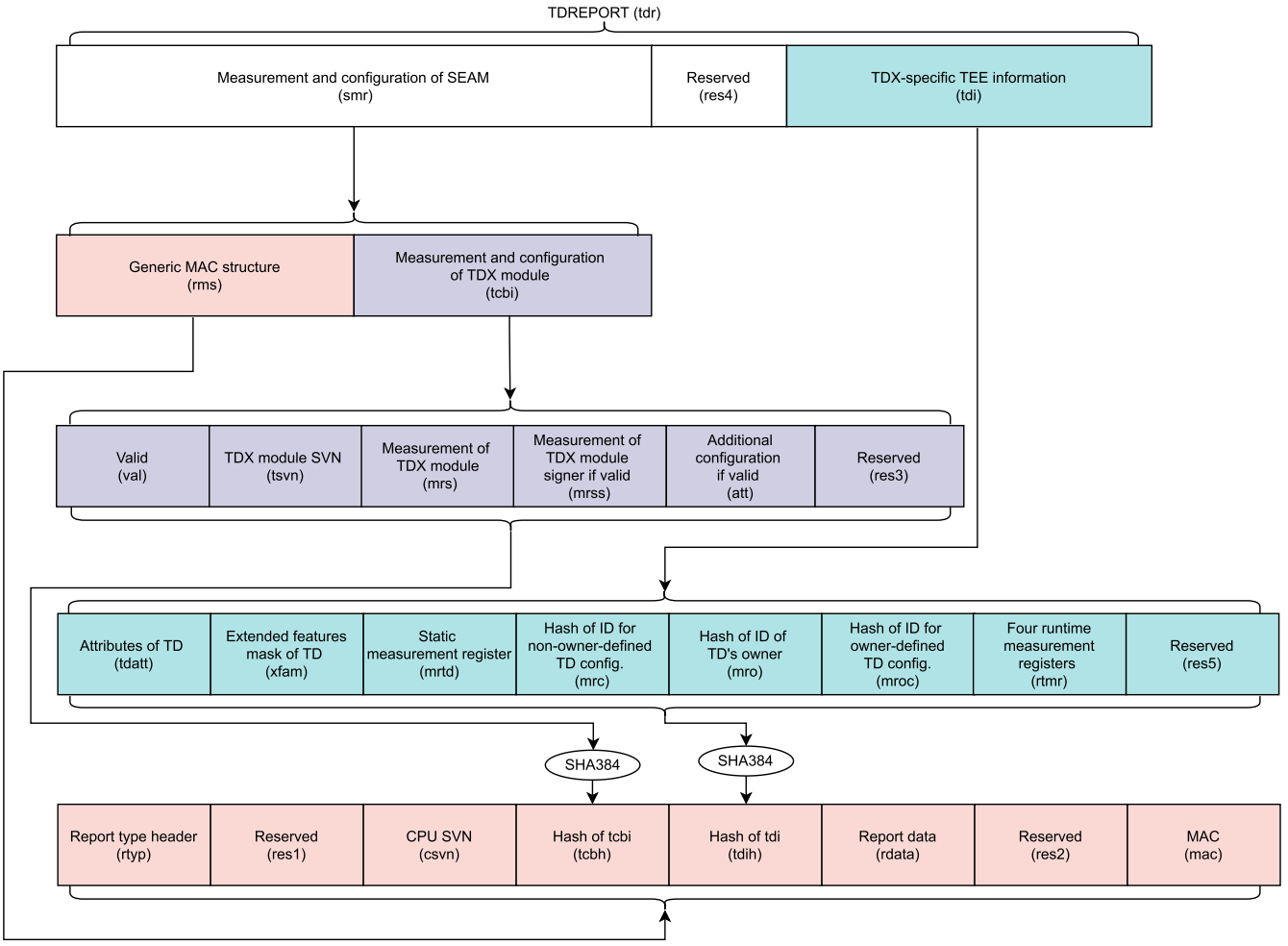

FIGURE 3. An overview of the components of TDREPORT data structure generated by TDG.MR.REPORT instruction in Intel TDX. Same structures are represented by same colors. Colors are consistent with Fig. 2.

B. MISSING FIELDS IN DATA STRUCTURES

A few fields are missing in some figures in the specification documents of Intel TDX. So, instead of clarifying the concept, the figures unfortunately add more to the confusion of the reader. We couldn't find any good reason not to include these fields in the figures. We provide a couple of instances below:

- *TDX Module:* MROWNERCONFIG field (`mroc`) is missing in TDINFO data structure `tdi` in Fig. 10.1 of [41]. Although it is not clear whether it is part of the SHA384 hash of the data structure, we find no good reason why this should be excluded from the hash, because there is no other cryptographic protection for this field. Moreover, similar fields MROWNER (`mro`) and MRCONFIGID (`mrc`) are included in the hash. Therefore, we believe hash is also computed over this field.
- *Quoting Enclave:* All the fields of measurements and configuration of TDX module as well as TDX-specific TEE information data structures (e.g., static `mrtd` and runtime `rtmr` measurements) are missing in the Quote structure in Fig. 10.1 of [41]. Since Quote is the *only* final structure which is sent over by QE to the challenger, it is not clear how the challenger is supposed to get these fields. We think that either the two data structures `tcbi` and `tdi` should be sent along with the Quote, or else these structures should be part of the Quote. In the first case, the hashes of these data structures are there in the Quote and signed, and it is still protected. From our experience with SGX, we believe that the second option is more likely, since static measurement (represented as MRENCLAVE) in SGX is sent as part of the Quote. It is important to emphasize here that verifying static and runtime measurements is a critical part of the remote attestation.

C. INCONSISTENT INFORMATION

The most critical part is that there are a number of descriptions in the official literature which are not only ambiguous but also contradict the information provided at other places, even within the same document. This is critical because it can lead to potential specification errors that may be translated

into errors in the design and implementation of the technology. We provide a few examples from the CPU module:

- Fig. 10.1 of [41] shows that the SHA384 hash is taken over four fields of the TEE_TCB_INFO data structure (`tcbi`) in a specific order, whereas the SEAMREPORT leaf operation in [11] shows that the SHA384 hash is taken over the whole data structure in a completely different order given in Table 2-3. It is worth pointing that order matters for hash computation.
- Fig. 10.1 of [41] shows that the reserved field (`res5`) in the TDINFO data structure (`tdi`) is not included in the SHA384 hash stored in REPORTMACSTRUCT (`rms`) and Quote, whereas for the verification (for reference, Section 18.5.3 in the same document), it is implied that the hash is over the complete TDINFO data structure (`tdi`). Moreover, the order of fields in TDINFO data structure (`tdi`) in Fig. 10.1 of [41] is inconsistent with the order in Table 18.13 of the same document.
- REPORTTYPE.TYPE field (inside `rtyp`) as described in Table 2-4 of [11] requiring *only* the high bit of TYPE to be 1 (i.e., allowing all values from 0×80 until 0xFF) is inconsistent with the operation of EVERIFYREPORT2 leaf function in the same document, where it is explicitly required to be of value 0×81, and any other value leads to an error. We believe that TDX should have the TYPE value 0×81.
- The description of REPORTMACSTRUCT `rms` (for reference, Section 18.5.3 in [41]) states: ''Software verifying a TEE report structure (for TDX, this includes TEE_TCB_INFO_STRUCT and TDINFO_STRUCT) should first confirm that its REPORTMACSTRUCT. TEE_TCB_INFO_HASH equals the hash of the TEE_TCB_INFO_STRUCT (if applicable) and that REPORTMACSTRUCT.TEE_INFO_HASH equals the hash of the TDINFO_STRUCT.'' Our intuition is that REPORTMACSTRUCT.TEE_TCB_INFO_ HASH (`tcbh`) should always be equal to the hash of the TEE_TCB_INFO (`tcbi`), because the TDX module measurements and configurations are always to be verified in the report. So the ''if applicable'' here does not make sense. If additionally, a TD is to be verified then REPORTMACSTRUCT.TEE_INFO_HASH (`tdih`) should match the hash of the TDINFO (`tdi`). Thus, the phrase ''if applicable'' should be with the second statement.
  We have submitted the draft to Intel for correcting the identified discrepancies in the documentation. Intel is in the process of making the changes.

VI. FORMAL VERIFICATION OF TD ATTESTATION

In this section, we first describe the sanity checks for the formal model in ProVerif. We then describe the properties that we proved in ProVerif.

A. SANITY CHECKS

We perform many sanity checks to exclude the modelling issues and gain confidence in the formal model that we actually model what we intend to model. First, we analyze that all parts of the code are reachable. Secondly, we intentionally break the protocol at each point to see if we could detect the break. Specifically, for confidentiality, we make each of the private channels `c_QE_CPU`, `c_TD_TDX`, and `c_TDX_CPU` as public one by one and then ensure that it breaks the confidentiality of the shared secret in each case.

B. SECURITY SERVICES

In the context of TD attestation, the most important security services are confidentiality and authentication.

1) CONFIDENTIALITY

After the successful verification of the Quote, the challenger sends a secret encrypted with the public key of the guest TD that it obtains from the `rdata` field in the Quote. This secret can be a session key to establish a secure channel with the guest TD. Therefore, it is important to ensure that the adversary is not able to decrypt this. This can be formalized as a reachability property, where we inspect whether it is possible to reach a state where the adversary can derive the secret in the plaintext. After overcoming all the discrepancies mentioned in the Section V, we prove the confidentiality of the secret in ProVerif, i.e., a state in which the adversary is sent the secret in plaintext is not reachable. Internally, ProVerif represents the model using 76 Horn clauses.

2) AUTHENTICATION

For authentication, the formal model is enriched with events. It is important to note that these events do not affect the adversary knowledge. We place the event `CPUsentSMR` just before the CPU sends `smr` to TDX (step 6), and the event `QuoteVerified` just after the successful verification of the Quote by the challenger (step 19). We additionally define the arguments to these events. Adding arguments to events in authentication property ensures that values of these arguments remain *unchanged*. We then verify the following property in ProVerif:

$$\begin{aligned}&\forall\, rtyp, res1, csvn, tcbh, tdih, rdata, res2.\\&\exists\, mac, tcbi.\\&event(QuoteVerified(rtyp, res1, csvn, tcbh, tdih, rdata,\\&res2)) \Rightarrow\\&event(CPUsentSMR(rtyp, res1, csvn, tcbh, tdih, rdata,\\&res2, mac, tcbi))\end{aligned}$$

where all argument symbols are as defined in Section IV. Essentially there is universal quantification for all fields of Quote except signatures. The parameter `mac` is existentially quantified because QE replaces the `mac` with signatures to be remotely verified by the challenger. Also the parameter `tcbi` is existentially quantified because `tcbi` is not part of Quote, according to Intel's documentation [41]. The existential quantification of `tcbi` preserves security because

its hash `tcbh` is universally quantified and authenticated. Informally, the query proves that if a Quote with the given parameters is verified by the challenger (step 19), then a report `smr` with the *same* parameters was previously generated by the CPU with some values of `mac` and `tcbi`. It is important to emphasize that our authentication property does *not* need to change with any future definitions of the reserved fields `res1` and `res2`, because these fields are already accounted in the theorem. Thus the property ensures that authentication holds from sending report (step 6) up to receiving Quote (step 19). Internally, ProVerif represents the model using 77 Horn clauses.

We prove a similar property for TDREPORT `tdr`. In this case, we place the event `TDXMsentTDR` with suitable arguments just before the TDX module sends `tdr` to TD (step 8), and the event `QEaccepted2` with the same arguments just after the successful verification of the `smr` (step 14). We then verify the following property:

$$\forall\, smr, tdi.$$
$$event(QEaccepted2(smr, tdi)) \Rightarrow$$
$$event(TDXMsentTDR(smr, tdi))$$

where `smr` represents the measurement and configuration of SEAM, and `tdi` represents TDX-specific TEE information, as defined in (10) and (9), respectively. Informally, the query proves that for every `tdr` that is accepted by QE with the arguments `smr` and `tdi`, there is a previous `tdr` created by the TDX module with the same arguments. Thus, we prove the authentication of `smr` and `tdi` in `tdr`.

It is important to note that adding a parameter `res4` in the last property makes it false, because it is not protected.

$$\forall\, smr, res4, tdi.$$
$$event(QEaccepted3(smr, res4, tdi)) \not\Rightarrow$$
$$event(TDXMsentTDR3(smr, res4, tdi))$$

where $\not\Rightarrow$ represents that the property does not hold and the events `QEaccepted3` and `TDXMsentTDR3` are placed at same positions as `QEaccepted2` and `TDXMsentTDR`, respectively. Informally, this means that an adversary could change these reserved bytes and still get the report accepted by QE.

Finally, we ensure that the events `QuoteVerified`, `QEaccepted2`, `QEaccepted3` are indeed reachable in the formal model, and thus the results of the authentication properties can be trusted.

The verification is performed using ProVerif version 2.02pl1 on Ubuntu 18.04 LTS on an Intel Core i7-6700 quad-core machine with a processor base frequency of 3.40 GHz with 32 GB of RAM. The average verification time to prove all the above properties (including reachability of all events) in this experimental setup is 0.5 s. This shows the scalability of the proposed approach, and the convenience to schedule the verification task in the design process. Another key feature of the approach is automation, i.e., all properties are proved automatically. The user involvement is only required for the specification of the protocol (Section IV) and the properties.

VII. CONCLUSION

The new architecture extensions, called TDX, announced by Intel in August 2020 promise a new and exciting era of VM-based TEE for Intel. In this work, we presented the formal specification of one of the security-critical processes of Intel TDX, namely TD remote attestation, using state-of-the-art symbolic analysis tool ProVerif. The process of formal specification revealed various subtle discrepancies in the Intel TDX specifications, including inconsistent information that could potentially lead to design and implementation errors. After corrections based on our experience and intuition, we proved confidentiality of the shared secret, and authentication of reports (`smr` and `tdi`). Interesting future directions include verification of equivalence properties, computational security analysis and the generation of formally verified implementation of Intel TD attestation with a focus on side-channel resistance that is correct-by-construction.

ACKNOWLEDGMENT

The authors would like to thank Anna Galanou and Amna Shahab for their helpful comments on the presentation of this work.

REFERENCES

[1] Confidential Computing Consortium. (Jan. 2021). *A Technical Analysis of Confidential Computing, V1.1*. [Online]. Available: https://confidentialcomputing.io/wp-content/uploads/sites/85/2021/03/CCC-Tech-Analysis-Confidential-Computing-V1.pdf

[2] W. Arthur and D. Challener, *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*. Berkeley, CA, USA: Apress, 2015.

[3] G. Proudler, L. Chen, and C. Dalton, *Trusted Computing Platforms: TPM2.0 in Context*. Cham, Switzerland: Springer, 2014.

[4] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptology ePrint Archive*, vol. 2016, no. 86, pp. 1–18, 2016. [Online]. Available: https://eprint.iacr.org/2016/086.pdf

[5] AMD. (2020). *AMD SEV-SNP: Strengthening VM Isolation With Integrity Protection and More*. [Online]. Available: https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf

[6] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell, and D. Goltzsche, "SCONE: Secure linux containers with Intel SGX," in *Proc. USENIX Symp. Operating Syst. Design Implement.*, 2016, pp. 689–703.

[7] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A practical library OS for unmodified applications on SGX," in *Proc. USENIX Annu. Tech. Conf.*, 2017, pp. 645–658.

[8] C. Priebe, D. Muthukumaran, J. Lind, H. Zhu, S. Cui, V. A. Sartakov, and P. Pietzuch, "SGX-LKL: Securing the host OS interface for trusted execution," Aug. 2019, *arXiv:1908.11143*. [Online]. Available: http://arxiv.org/abs/1908.11143

[9] Intel. (2020). *Intel Trust Domain Extensions*. [Online]. Available: https://software.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf

[10] Intel. (2021). *Intel Intel Architecture Memory Encryption Technologies: Specification, Revision 1.3*. [Online]. Available: https://software.intel.com/sites/default/files/managed/a5/16/Multi-Key-Total-Memory-Encryption-Spec.pdf

[11] Intel. (2020). *Intel Trust Domain CPU Architectural Extensions*. [Online]. Available: https://software.intel.com/content/dam/develop/external/us/en/documents/intel-tdx-cpu-architectural-specification.pdf
[12] A. Zeller, *Why Programs Fail: A Guide to Systematic Debugging*. Amsterdam, The Netherlands: Elsevier, 2009.
[13] A. Goel, S. Krstic, R. Leslie, and M. Tuttle, "SMT-based system verification with DVF," in *Satisfiability Modulo Theories*, vol. 20. EasyChair, 2013, pp. 32–43.
[14] R. Fraer, D. Keren, Z. Khasidashvili, A. Novakovsky, A. Puder, E. Singerman, E. Talmor, M. Y. Vardi, and J. Yang, "From visual to logical formalisms for SoC validation," in *Formal Methods Models for Codesign (MEMOCODE)*. New York, NY, USA: ACM, 2014, pp. 165–174.
[15] R. Leslie-Hurd, D. Caspi, and M. Fernandez, "Verifying linearizability of Intel software guard extensions," in *Proc. Int. Conf. Comput. Aided Verification*. Springer, 2015, pp. 144–160.
[16] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proc. Int. Workshop Hardw. Architectural Support Secur. Privacy*, vol. 13. New York, NY, USA: ACM, 2013. [Online]. Available: https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing
[17] P. Subramanyan, R. Sinha, I. Lebedev, S. Devadas, and S. A. Seshia, "A formal foundation for secure remote execution of enclaves," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 2435–2450.
[18] M. U. Sardar, D. L. Quoc, and C. Fetzer, "Towards formalization of enhanced privacy ID (EPID)-based remote attestation in intel SGX," in *Proc. 23rd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2020, pp. 604–607.
[19] M. U. Sardar, R. Faqeh, and C. Fetzer, "Formal foundations for Intel SGX data center attestation primitives," in *Formal Methods and Software Engineering* (Lecture Notes in Computer Science), vol. 12531, S.-W. Lin, Z. Hou, and B. Mahoney, Eds. Cham, Switzerland: Springer, 2020, pp. 268–283.
[20] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 2, pp. 198–208, Mar. 1983.
[21] B. Blanchet, "Modeling and verifying security protocols with the applied Pi calculus and ProVerif," *Found. Trends Privacy Secur.*, vol. 1, nos. 1–2, pp. 1–135, 2016.
[22] D. Basin, C. Cremers, J. Dreier, and R. Sasse, "Symbolically analyzing security protocols using tamarin," *ACM SIGLOG News*, vol. 4, no. 4, pp. 19–30, Nov. 2017.
[23] B. Blanchet, "CryptoVerif: A computationally-sound security protocol verifier," Tech. Rep., 2017.
[24] M. Barnett, B.-Y. E. Chang, R. DeLine, B. Jacobs, and K. R. M. Leino, "Boogie: A modular reusable verifier for object-oriented programs," in *Formal Methods for Components and Objects*. Springer, 2005, pp. 364–387.
[25] M. C. Browne, E. M. Clarke, and O. Grümberg, "Characterizing finite Kripke structures in propositional temporal logic," *Theor. Comput. Sci.*, vol. 59, nos. 1–2, pp. 115–131, Jul. 1988.
[26] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. USENIX Secur. Symp.*, 2016, pp. 857–874.
[27] Intel. (Jun. 2015). *Intel Software Guard Extensions (Intel SGX), Revision 1.1*. Accessed: Nov. 5, 2020. [Online]. Available: https://software.intel.com/sites/default/files/332680-002.pdf
[28] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel software guard extensions (Intel SGX) support for dynamic memory management inside an enclave," in *Proc. Hardw. Architectural Support Secur. Privacy (HASP)*, 2016, pp. 1–9.
[29] M. R. Tuttle and A. Goel, "Protocol proof checking simplified with SMT," in *Proc. Int. Symp. Netw. Comput. Appl.*, 2012, pp. 195–202.
[30] S. Conchon and M. Roux, "Reasoning about universal cubes in MCMT," in *Formal Methods and Software Engineering*. Springer, 2019, pp. 270–285.
[31] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Trans. Program. Lang. Syst.*, vol. 12, no. 3, pp. 463–492, Jul. 1990.
[32] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. Design Autom. Conf.*, 1999, pp. 317–320.
[33] A. Gill, "Domain-specific languages and code synthesis using Haskell," *Queue*, vol. 12, no. 4, pp. 30–43, Apr. 2014.
[34] M. Ramsdell, J. D. Guttman, and J. D. Liskov. (2016). *CPSA: A Cryptographic Protocol Shapes Analyzer*. [Online]. Available: http://hackage.haskell.org/package/cpsa
[35] J. Whitefield, L. Chen, R. Sasse, S. Schneider, H. Treharne, and S. Wesemeyer, "A symbolic analysis of ECC-based direct anonymous attestation," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS P)*, Jun. 2019, pp. 127–141.
[36] J. D. Guttman and J. D. Ramsdell, "Understanding attestation: Analyzing protocols that use quotes," in *Security and Trust Management*. 2019, pp. 89–106.
[37] G. Chen, Y. Zhang, and T.-H. Lai, "OPERA: Open remote attestation for Intel's secure enclaves," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: ACM, Nov. 2019, pp. 2317–2331.
[38] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, and B. Parno, "SoK: Computer-aided cryptography," in *Proc. 42nd IEEE Symp. Secur. Privacy*, May 2021. [Online]. Available: https://eprint.iacr.org/2019/1393.pdf
[39] N. Durgin, N. Durgin, P. Lincoln, P. Lincoln, J. Mitchell, J. Mitchell, A. Scedrov, and A. Scedrov, "Undecidability of bounded security protocols," *Proc. Workshop Formal Methods Secur. Protocols (FMSP)*, Trento, Italy, 1999.
[40] P. Lafourcade and M. Puys, "Performance evaluations of cryptographic protocols verification tools dealing with algebraic properties," in *Foundations and Practice of Security*. 2016, pp. 137–155.
[41] Intel. (2020). *Architecture Specification: Intel Trust Domain Extensions (Intel TDX) Module*. [Online]. Available: https://software.intel.com/content/dam/develop/external/us/en/documents/intel-tdx-module-1eas.pdf
[42] Intel. (2020). *Guest-Host-Communication Interface (GHCI) for Intel Trust Domain Extensions (Intel TDX)*. [Online]. Available: https://software.intel.com/content/dam/develop/external/us/en/documents/intel-tdx-guest-hypervisor-communication-interface.pdf
[43] Intel. (2020). *Intel Trust Domain Extensions—SEAM Loader (SEAMLDR) Interface Specification*. [Online]. Available: https://software.intel.com/content/dam/develop/external/us/en/documents/intel-tdx-seamldr-interface-specification.pdf
[44] Intel. (Oct. 2020). *Intel TDX Virtual Firmware Design Guide*. [Online]. Available: https://software.intel.com/content/dam/develop/external/us/en/documents/tdx-virtual-firmware-design-guide-rev-1.pdf
[45] N. Smith, S. Ortiz, M. Castelino, and M. Ylinen. (2019). *Technologies for Fast Launch of Trusted Containers*. [Online]. Available: https://www.freepatentsonline.com/y2019/0042759.html
[46] L. Kida, K. Zmudzinski, R. Lal, P. Pappachan, A. Basak, and A. Trikalinou. (2019). *Technologies for Filtering Memory Access Transactions Received From One or More I/O Devices*. [Online]. Available: https://www.freepatentsonline.com/y2019/0138755.html
[47] M. Abadi, B. Blanchet, and C. Fournet, "The applied Pi calculus: Mobile values, new names, and secure communication," *J. ACM*, vol. 65, no. 1, pp. 1–41, Jan. 2018.
[48] V. Cortier and S. Kremer, "Formal models and techniques for analyzing security protocols: A tutorial," *Found. Trends Program. Lang.*, vol. 1, no. 3, pp. 151–167, 2014.
[49] C. Weidenbach, "Towards an automatic analysis of security protocols in first-order logic," in *Proc. Int. Conf. Automated Deduction*. Springer, 1999, pp. 314–328.
[50] L. Bachmair and H. Ganzinger, "Resolution theorem proving," in *Handbook Automated Reasoning*. Amsterdam, The Netherlands: Elsevier, 2001, pp. 19–99.
[51] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. [Online]. Available: http://cacr.uwaterloo.ca/hac/
[52] V. Scarlata, S. Johnson, J. Beaney, and P. Zmijewski, "Supporting third party attestation for Intel SGX with Intel data center attestation primitives," White Paper, 2018.

MUHAMMAD USAMA SARDAR (Member, IEEE) received the B.S. degree in electronics engineering from the Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi, Pakistan, in 2009, and the master's degree from the School of Electrical Engineering and Computer Sciences (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan, in 2015, with 2nd position in his batch. He is currently pursuing the Ph.D. degree with the Faculty of Computer Science, Technical University of Dresden, Germany.

His master's dissertation was in collaboration with the Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany. In 2017, he was a recipient of the prestigious DAAD Research Grant to pursue his Ph.D. degree. He has vast work experience, including research as well as teaching. He was a Research Internee with the Chair of Embedded Systems, Karlsruhe Institute of Technology, Germany, and a Research Assistant with the System Analysis and Verification Laboratory, NUST, Pakistan. He is currently a Tutor for the master's courses, including Systems Engineering 1, Principles of Dependable Systems, and Software Fault Tolerance, at the Technical University of Dresden. His research work has resulted in publications at top international forums, such as the *Journal of Parallel and Distributed Computing*, the NASA Formal Methods Symposium, and *Journal of Automated Reasoning*. His current research focus is on the development of tools integrating the formal methods to complement the existing simulation techniques for trusted execution environments.

Mr. Sardar is also serving as a Volunteer at CAVlinks. He has received the South Asia Triple Helix Association (SATHA) Innovation Award, in 2018, the Best Researcher of the Year Award in System Analysis and Verification Lab in Pakistan, in 2017, and the best speaker awards at Workshop on Applications in ASIC Design, in December 2016, and Workshop on Recent Trends in Theorem Proving, in April 2016.

SAIDGANI MUSAEV received the B.Sc. degree from the Faculty of Applied Mathematics and Computer Science, Moscow State University, Dushanbe Branch, in 2016, and the M.Sc. degree from the Faculty of Computer Science, Vrije Universiteit Amsterdam, in 2019.

Since 2019, he has been a Research Assistant with the Chair of Systems Engineering, Technical University of Dresden.

CHRISTOF FETZER received the Diploma degree in computer science from the University of Kaiserslautern, Germany, in December 1992, and the Ph.D. degree from UC San Diego, in March 1997.

He joined AT&T Labs-Research, in August 1999, and had been a Principal Member of Technical Staff until March 2004. Since April 2004, he heads the Endowed Chair (Heinz-Nixdorf endowment) in systems engineering at the Department of Computer Science, TU Dresden, where he is currently the Chair of the Distributed Systems Engineering (International Master's Program) at the Computer Science Department. He has published over 150 research articles in the field of dependable distributed systems.

Dr. Fetzer has been a member of more than 50 program committees. He has won three best paper awards at DEBS 2013, LISA2013, and SRDS 2014, his Ph.D. students have won two best student paper awards at IEEE Cloud 2014 and DSN 2015, and the EuroSys Roger Needham Award, in 2014. As a student, he received a two-year scholarship from DAAD and won two best student paper awards at SRDS and DSN. He was a Finalist of the 1998 Council of Graduate Schools/UMI Distinguished Dissertation Award and received the IEE Mather Premium, in 1999.