Towards Interoperable Enclave Attestation: Learnings from Decades of Academic Work

Arto Niemi, Sampo Sovio, Jan-Erik Ekberg
Huawei Technologies Oy (Finland) Co Ltd.
Helsinki, Finland
firstname.surname@huawei.com

***Abstract*—Secure enclave technology has during the last decade emerged as an important hardware security primitive in server computer cores, and increasingly also in chips intended for consumer devices like mobile phones and PCs. The local or remote user of the enclave will rely on *attestation* protocols to confirm the isolation and other security properties of the enclave. In this paper, we analyze different attestation architectures and techniques as well as ways to present attestation evidence and metrics. We find that existing industry efforts to make attestation interoperable across platforms and trust roots can benefit from the wealth of research around remote attestation that has taken place since the 1990's.**

I. Introduction

An *enclave* is an isolated execution environment that protects code and data from the rest of system, including from privileged components such as the operating system or the hypervisor. Enclaves are widespread in today's computing: Arm TrustZone based enclaves are used on billions of consumer devices to protect applications such as DRM and user authentication [1], while Intel SGX and AMD SEV enclaves are used in the cloud to enable confidential computing [2]. Relying parties, such as cloud tenants or content providers, can establish trust in an enclave, its code and outputs using *attestation*—a process that allows trustworthiness of a target entity to be determined based on cryptographic evidence produced by a trusted on-device component.

Attestation was first studied in the context of Trusted Platform Modules (TPMs) [3], where its main use case is to provide trusted boot via the Linux integrity measurement architecture [4][5][6]. The most widespread form of enclave attestation to date is Android key attestation [7], which can be used to prove to a remote verifier that a cryptographic key, such as content decryption key, is confined to a TrustZone-based enclave. In the cloud, AMD SEV and Intel SGX enclaves are widely deployed and attested [8], while many academic proposals exists for attestation in the Internet of Things [9].

Recently, a number of academic surveys on remote attestation have been published [10][11][9][12]. In fact, attestation has been studied for almost twenty years, but many concepts around attestation have only recently received wider attestation in the industry [12, p. 1608], as exemplified by renewed standardization efforts: a substantial set of IETF standards for remote attestation procedures (RATS) [13][14][15] is under development and ETSI has published an attestation architecture for NFV [16]. With Intel TDX [17], IBM PEF [18] and ARM CCA [2], new hardware support for attestation is emerging, and open-source projects such as Veracruz [2] and Veraison [19] are addressing attestation in the enclave setting. We feel that there is a clear need for a survey that summarizes these developments, identifies the main issues in enclave attestation, and provides a set of recommendations for the industry.

To our knowledge, no significant surveys exists that specifically cover enclave attestation. Primarily, academic work has focused on the remote attestation of complete computing platforms, not on enclaves. Also, most publications on attestation tend to focus on one subtopic such as evidence generation and validation, and neglecting others such as authentication, endorsements and channel binding. Terminology in this field is also often inprecise and mutually conflicting between works.

In an attempt to be more comprehensive and precise, this study follows the basic architecture and terminology of RATS (which are similar those of the TCG [20]). We slightly tweak the RATS architecture to suit our needs, as shown in Fig. 1 and provide a more detailed discussion of topics such as identities and attestation metrics. Our approach resembles that of [12], which uses the RATS architecture as a basis for studying attestation for blockchain networks.

II. Attestation

In broad terms, attestation can be defined as a process that allows a *relying party* to assess properties of a *target entity*, which may be a system component or data. To make this possible, an *attester* produces *attestation evidence* on properties of the target entity. The properties are called *attestation metrics*. The evidence is evaluated by a *verifier*, which may be the relying party or a different entity, such as a *verification service*. If the verifier decides that the prover is trustworthy, it may issue an *attestation result*, which the relying party can use to make the final trust decision. The verifier's decision is based on the attestation evidence and possibly on *endorsements*—a set of attestation metrics that an *endorser* indicates to be good, for example, as reference values. Evaluation of attestation evidence and attestation results may also be influenced by *attestation policy*, which can consist of an *appraisal policy for evidence* used by a *verifier* and an *appraisal policy for attestation results* used by the relying party [13, p. 31]. In practice separation of *appraisal policies* means that policy for relying party might consider certain attestation metrics to be

ISSN 2305-7254

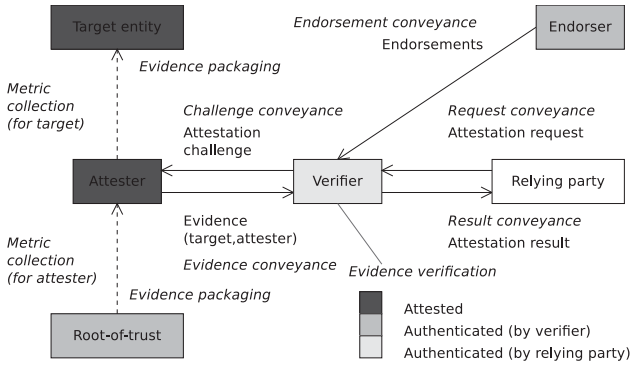

Fig. 1. Roles, messages and *processes* in the architectural model used in this article (adapted from the TCG and RATS architectures [20, p. 12][13, p. 8]. As discussed in the text, many variations are possible.

relevant even if these metrics are not relevant to the verifier. In *layered attestation*, multiple layers of attesters are possible. A *root-of-trust* is a last-level attester that, once authenticated, is unconditionally trusted by the verifier without attestation evidence. The *root-of-trust* may provide metrics about itself to the verifier, and these metrics are also trusted without attestation evidence. These processes are summarized in Fig. 1.

The end goal of attestation is to establish trust. Paraphrasing [21], we regard it as safe for a relying party to trust an entity when the following conditions hold:

1) The entity can be unambiguously identified
2) The entity operates unhindered and shielded from external influence
3) The relying party a) has first-hand evidence of consistent, good behaviour or b) trusts someone who vouches for consistent, good behaviour.

We would like to stress that all of these are necessary. For example, having proof that a program behaves according to expectations is of no use if an attacker manages to swap the good program with a compromised one or tamper with the program's execution at run-time. The first condition can be fulfilled by establishing the identity of the target either via attestation or cryptographic (public-key based) authentication. Running the target in an enclave allows to achieve the second condition, provided that the enclave isolation mechanisms are identified and trusted. To fulfill the third condition, three options are possible: (1) the relying party has a way of establishing trust in the target's behaviour directly, for example, because it has performed a semantic analysis of the program code; (2) the relying party matches the attestation metrics against endorsements by a trusted third-party; or (3) the relying party trusts a trusted compiler or interpreter that, before execution, transforms the program into a form that can is more trustworthy than the original. These options all rely on fundamentally different trust bases, respectively called *analytic trust*, *axiomatic trust* and *synthetic trust* in [22].

We formulate a list of questions to guide our survey of enclave attestation, inspired by a set of requirements presented in the literature. Coker et al. [4] presented five requirements for attestation in general. Addressing attestation in the context of virtualized systems, Lauer et al. [23][24] add two more requirements for layered attestation, and Eckel et al. [25] propose further implementation-specific requirements. Finally, Chen et al. [26] formulate three requirements specifically for the enclave attestation use case. Based on these, we assert that a secure and effective enclave attestation procedure must answer the following questions:

- **Q1 (Enclave identity).** What is the identity of the hardware and firmware that is responsible for the isolation of the enclave? What is the root-of-trust? How are these identities established?
- **Q2 (Target identity).** What is the identity of the target entity (software or data) that is protected by the enclave? How is the identity established?
- **Q3 (Metrics).** Which properties of the target entity and the enclave should are included in attestation evidence? Are they sufficient for a comprehensive appraisal of trustworthiness of the target and its running environment?
- **Q4 (Trust base).** Given a valid attestation evidence, what is the basis for the decision on whether to trust the evidence or not? Is the trust base axiomatic, analytic or synthetic?
- **Q5 (Interaction patterns).** How is attestation evidence requested? How are evidence and endorsements conveyed to the verifier? How is attestation result conveyed to the relying party?
- **Q6 (Packaging).** How are attestation evidence, endorsements and results protected and encoded for transmission? How is the freshness of attestation evidence guaranteed?
- **Q7 (Channel binding).** Is the target with the given identity and attested metrics the end-point of the current communication channel?
- **Q8 (Privacy).** Is the target entity able to constrain the amount of information attestation reveals to other parties? Can the target entity choose the attestation metrics based on the identity of the verifier?
- **Q9 (Linking).** If multiple attestation evidences are generated for the same system, how are these bound together?

In the rest of the paper, we attempt to survey how questions Q1-Q9 are addressed in literature and in industry open-source projects. In Sections III to XI we discuss concepts and issues related to each of the above questions in turn. Compared to previous surveys on attestation, we provide a more in-depth coverage of topics such as identities, interaction patterns and evidence packaging. Finally, in Section XII we show how the questions are addressed in open-source enclave projects.

III. Enclaves

Enclave attestation can be classified into *software*, *hardware* and *hybrid attestation*, depending on whether the attester is a software or hardware component, or whether both hardware and software attesters are used [10]. For example, in the

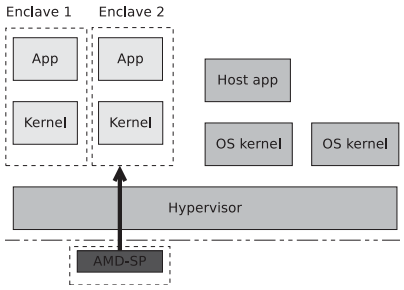

Fig. 2. In AMD SEV, enclaves are directly attested by a discrete security processor (AMD-SP). Enclaves are virtual machines, managed by an untrusted hypervisor.

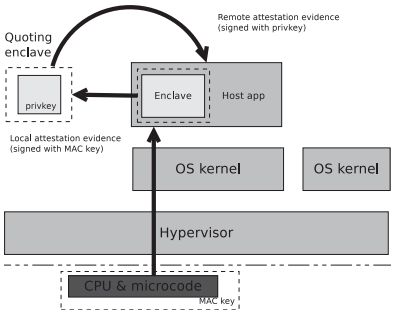

Fig. 3. In Intel SGX, enclaves are first locally attested by the CPU and microcode. An Intel-provided Quoting Enclave then signs the local attestation evidence using a device-specific key to produce the remote attestation evidence.

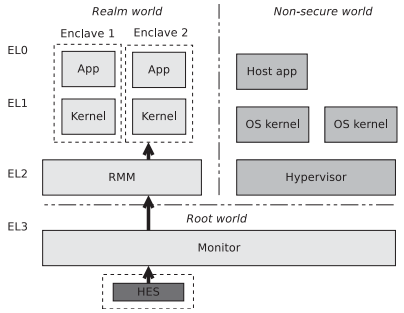

Fig. 4. In the upcoming Arm CCA architecture, attestation is hierarchically layered. There are three CPU-isolated worlds (realm, non-secure and root), each with multiple privilege levels. Enclaves run in the realm world and may be either virtual machines or more restricted applications.

layered attestation model, only the lowest level attester (the root-of-trust) may be a hardware component.

A. Currently available enclave hardware

Enclaves are currently supported in commercial hardware via the Intel SGX, AMD SEV and Arm TrustZone CPU/ISA extensions. Of these, TrustZone is the oldest, having been introduced already in 2005 [1].

Intel introduced Secure Guard Extensions (SGX) in 2015 for its 6th generation processors. SGX enclaves are CPU-protected, isolated sub-processes of a host application [27]. The enclave's identity is called *MRENCLAVE*. It consists of a hash over the enclave's memory pages (data and code), the relative positions of the pages in the enclave, and security flags associated with the pages [27]. MRENCLAVE is included as a metric in SGX attestation evidences. The root of trust for attestation is the CPU and its microcode. SGX enclaves are first attested locally by the CPU. The results are then converted into remote attestation evidence by an Intel-provided Quoting Enclave. Initially, SGX allowed attestation evidence to be verified only using an Intel-provided online verification service. Later, with the Data Center Attestation Primitives (DCAP) [28], it became possible to use custom verification servers. SGX has no limit on the number of running enclaves. SGX has now been deprecated from some Intel core series [1]. Its replacement, called Intel TDX, is described in the next section.

AMD SEV first became available in AMD EPYC processors in 2017. In contrast to SGX, SEV protects entire virtual machines, even from an untrusted hypervisor [28]. The enclave identity is called *MEASURE*. It is a HMAC over the SEV API and AMD-SP firmware versions, the launch hash, a nonce and a symmetric Transport Encryption Key (TIK), computed in a preceding key exchange in the remote attestation protocol [29, p. 459]. The root of trust is a separate isolated processor (AMD-SP). In SEV, enclaves are directly attested by the SP. SEV supports running at most 15 enclaves at the same time, which is considered as one of the main drawbacks of SEV in [30]. The first version of AMD SEV only provided boot-time attestation of enclaves, but the latest SEV-SNP variant, first included in the 3rd Generation AMD EPYC processors in 2021, allows attestation to be requested at any time [29]. Other improvements in SEV-SNP introduce memory integrity protection.

In contrast to SGX and SEV, Arm TrustZone only supports a single hardware-isolated enclave, called the secure world, which runs a separate operating system (secure OS). Relying on secure boot, the secure OS can provide additional, software-based isolation for secure world applications. TrustZone based enclaves, especially ones that follow the GlobalPlatform specifications, are prevalent in smartphones [31]. However, they are not ideally suited for virtualization and, as a consequence, are rarely used in the cloud. Furthermore, there is no native hardware or firmware support for attestation in TrustZone [1].

Trusted Platform Modules (TPMs) [3] do not, on their own, provide hardware support for enclaves. However, they can be used as root-of-trusts for a boot sequence where hashes of loaded components are added to a hash chain and the accumulated hash is included as a metric in attestation evidence. With such *measured boot*, it is possible for verifiers to receive evidence trusted software, such as a hypervisor, is running on the target device and providing isolation for enclaves. This is sometimes called *hypervisor-based isolation*.

B. Future enclave hardware

In the near future, two important new enclave hardware architectures are about to appear: Intel TDX and Arm CCA.

Like SEV, TDX can be used to protect virtual machines. Details have not yet been fully disclosed by Intel, but based

[1] https://cdrdv2.intel.com/v1/dl/getContent/655258

on available information, TDX attestation resembles SGX attestation in many aspects. For example, TDX also relies on a quotation enclave to convert local evidence to remotely verifiable evidence. The main difference is that more of the software stack can be attested in TDX [17].

In contrast, CCA can be used to protect both virtual machines and single processes, or even just a part of a process [2]. This reflects the fact that Arm processors are mainly used in end-user devices and embedded systems, but also marketed for cloud environments. Whereas TrustZone enclaves are primarily used to run OEM-installed first-party applications, Arm CCA aims to "democratize trusted computing" by providing enclaves where third parties can easily deploy their trusted software. Attestation in CCA is split into two parts: attestation of the platform and attestation of the enclave (called realm in CCA terminology) [32]. The root-of-trust is called Hardware Enforced Security (HES), which Arm recommends should be a discrete processor, but other kinds implementations are also possible [33]. Attestation is requested by the enclave (realm) code, and the request is passed all the way down to the HES. The HES attests the monitor, which is responsible for isolating the realm and the normal worlds from each other. The monitor attests the realm management module (RMM), which runs in the realm world and is a hypervisor for the enclaves [34]. Finally, the RMM attests the enclave, as shown in Fig. 4. The CCA architecture does not define an attestation protocol, but requires Arm PSA tokens to be used as the evidence format. According to the currently available specifications, like the early versions of SEV, Arm CCA seems to only support launch-time attestation of the enclave [32].

C. Security concerns

It should be stressed that attestation can, at best, only be as secure as the underlying hardware primitives. Unfortunately, numerous attacks against SGX [35] and SEV [36][29] have been presented in the literature, including ones that compromise attestation. The possibility of such vulnerabilities needs to be carefully taken into account when relying on attestation.

IV. Identities

Unambiguous identification of a target and its enclave are necessary preconditions for attestation (Q1 and Q2). An *identity* is something that can be used to distinguish an entity from other entities.

We classify identities into three groups according to their origin: *provisioned, derived and emergent*. We also distinguish between unique and non-unique run-time identities as well as functional (identifying a functionality of the entity) and non-functional identities.

1) Unique run-time identities: Enclaves are live, run-time entities. An identity that can be used to uniquely distinguish a run-time entity—such as a running instance of a program or a particular copy of data—is called a *unique run-time (URT) identity*.

2) Provisioned identities: A *provisioned identity* is fixed (e.g. fused) during manufacturing or provisioning and usually cannot be changed after assignment. Examples include serial numbers, private keys and random seed values, such as the Unique Device Secret (UDS) in the TCG's DICE architecture [20][37] or Chip Endorsement Key (CEK) in AMD SEV-SNP [38]. These are URT identities for the specific device; they cannot be used as URT identities for software components, but can form a basis from which component-specific URT identities can be derived. An example of provisioned non-URT identity is a program's binary code, which may be shared among multiple run-time entitities.

3) Derived identities: It is possible to derive new identities from an provisioned identity using a key-derivation function. For example, the Compound Devide Identifiers (CDIs) in DICE are derived from the UDS and the code hash of the attester, while in SEV-SNP new identities, called Versioned Chip Endorsement Keys (VCEKs), can be derived from the CEK and version numbers of the trusted firmware. In layered attestation, a parent layer (such as a hypervisor) may be able to assign a derived URT identity to its children, assuming that they are domain separated from each other (e.g. virtual machines or enclaves). The derived URT identity can then be included as one of the attestation metrics—this is called *authenticated attestation* [39]. A derived identity may also be short-term, such as a single-use keypair generated for secure channel establishment (a *channel end-point identity*).

4) Emergent identities: An *emergent identity* is not provisioned or derived, but emerges from non-functional properties of the entity—the main examples are physically unclonable functions (PUFs) [40]. In software attestation, timing profiles, such as network or memory access latencies, are also used as identitites [10, p. 142395].

5) Cryptographic identities: Most digital identities are easily copyable; *claiming an identity* therefore requires a process that does not allow verifiers to steal the identity. This can be done with cryptographic tools using authentication or identity attestation (Section V). Particularly useful are *cryptographic identities*. A cryptographic identity is a public key; the identity can be claimed by an entity by demonstrating that it has access to the corresponding private key, e.g. by signing a verifier-provided nonce. Cryptographic identities may be either provisioned or derived, usually depending on whether the entity exists during manufacturing (such as a secure hardware component) or is created at run-time (such as a virtual machine or enclave). Cryptographic identities are typically URT identities.

Related to identities, we define *authentication* as a process that allows an entity to assess the identity of a target entity. Authentication can be regarded as a form of attestation (*identity attestation*), however, in this paper, we make the following distinction: authentication is the process of claiming a *cryptographic identity*, i.e. that an entity is in possession of the private key corresponding to a certain public key.

All attestation schemes rely on authentication as defined above; attestation alone is insufficient to establish trust in

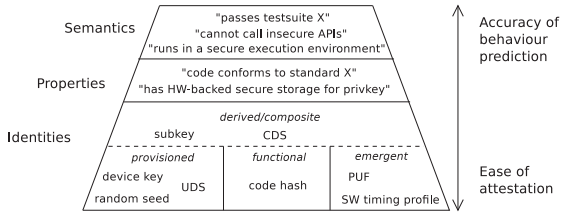

Fig. 5. Examples of attestation metrics

the target entity. Attestation always relies on a root-of-trust that cannot be attested by any other trusted on-device entity (strictly speaking, the root-of-trust can still *self-attest*, i.e. create attestation evidence regarding itself) The verifier's decision of whether to trust the root-of-trust must thus rely on *authentication*. In attestation, the root-of-trust is typically authenticated by the verifier, usually using asymmetric signatures. Similarly, the verifier usually authenticates the endorser, and the relying party authenticates the verifier.

V. Metrics

An attestation metric is a value that describes a property of the target entity. Good metrics allow verifiers to decide whether to trust a target; which metrics should be attested (Q3) is one of the main questions in trusted computing.

A. Attestation metrics can be categorized into *static* and *dynamic* metrics [10, Sec V] based on whether the metric value can legitimately change during the lifetime of the target component. For example, the hash of the binary code can usually be expected to stay the same, while hash of the program's memory space will not. As pointed out in [37, p. 4], dynamic enclave memory management—supported e.g. by the second generation SGX—may complicate things: loading or evicting code pages at run-time changes the code hash, which most verifiers expect to be static. This highlights the need to define enclave identities carefully. Dynamic evidence may include, for example, a log of system calls invoked by the enclave code, a monotonic counter or a timestamp. Dynamic evidence is usually much harder to collect than static evidence.

B. The most widely-used attestation metric is a cryptographic hash of a target component's binary code, taken by the attester before the component is loaded; this is called *binary attestation* or *hash-based attestation* [41]. Binary attestation is a form of *identity attestation*, because it attests an identity (binary code) of the target entity. Other kinds of identities, such as serial numbers, boot sequence hash values or public keys, can also be attested. One problem with binary attestation is that although it can be used to verify the identity of a program, it does not allow validating the trustworthiness of a program *instance*, since even programs with trustworthy code may be subjectible to various run-time attacks by e.g. an untrusted operating system.

C. In *key attestation*, a public key or a hash of a public key is used as an attestation metric and associated with a claim that describes how the private counterpart is protected. The claim can either be explicitly encoded as a metric or it can implicit, e.g. the attester is trusted to only attest a key if the private counterpart is securely stored. Key attestation can be used in layered attestation architectures to establish a chain-of-trust for attestation signing keys.

D. Property-based attestation (PBA) [42] proposes to attest properties of the software components instead of just hash-based identities. In property-based attestation, a trusted third party provides a mapping between identities and properties. In the architecture of Fig. 1, this can be done using the endorsements. For example, the endorsements could be X.509 certificates that cryptographically bind hashes to properties. The main difficulty in PBA is extracting the properties to be attested and deciding how the relying party can use these properties to actually predict program behaviour [6, p.41]. Another issue with both binary and property-based attestation is that trust decisions are done by fiat, relying on an external endorser [22].

E. Semantic or *behavioral attestation* [43], attempts to fill the semantic gap between properties and and actual behaviour. It relies on a trusted execution service (TES), such as a virtual machine or an interpreter, to constrain and inspect a program running within. For example, the TES can restrict the system APIs the program may call, or instructions it may execute. Functioning as an attester, the TES can then include the constraints, or even the actual behaviour of the program, as attestation metrics in the evidence. Semantic attestation techniques can be divided into *analytic* methods, where program code is analyzed before execution, and *synthetic* methods, where a program is transformed prior to execution, producing an artifact that can be trusted in ways that the original could not [22]. The semantic attestation paradigm is very powerful. For example, the verifier could even send an arbitrary test suite to the attester, which runs the testsuite against the target entity and returns attestation evidence indicating the test result. Today, concrete and practical implementations of semantic attestation are still not widely available, as noted in [2, p. 136]. Identity, property-based and semantic attestation are illustrated in Fig. 5.

F. In *logical attestation* [22], attestation metrics consist of logic formulas. A goal statement is associated with every operation and resource on the system. A process requesting an operation is required to provide attestation evidence containing a proof that allows the goal statement to be reached. The proof is validated by a trusted operating system kernel. The authors of [22] describe an operating system called Nexus that supports logical attestation, using a TPM as the hardware trust root. Logical attestation provides a framework that supports all kind of attestation approaches, including semantic attestation without external endorsers. The drawback is that a rather large TCB is required, including an entire specialized operating system.

G. As discussed in II, the target entity may consist of software components, data or both. *Data attestation metrics* may, for example, indicate the origin of the data, how it was computed (e.g. whether it was computed in a TEE-isolated process) and when the data was created, e.g. by binding to a

value of a trusted timestamp or monotonic counter.

H. Metric collection or *measuring* refers to the process of determining the attestation metrics of a target entity:

1) Discrete and continuous metric collection: Metric collection may be *discrete* (collected at a particular time) or *continuous* (collected over a time period). Some attacks against discrete metric collection are listed in [10, Sec VI-A]. For example, an attacker with code injection capability may wait until after the target entity's memory code is hashed during metric collection before launching his attack. Continuous attestation can partially address such TOCTOU attacks. Metric collection is usually performed by a TCB.

2) Domain separation: Domain separation between the attester and the target [4, Sec 4.2.] is important in metric collection, because the target should be prevented from influencing the result of attestation. One way to achieve domain separation is to use hardware-based isolation. For example, the attester may be a hypervisor running on a more privileged CPU protection level (e.g. Ring -1 or EL2) and the target may be a virtual machine (running e.g. on Ring 0 or EL1). Ideally, the run-time status of the isolation mechanism should also be attested to ensure that the mechanism is working at the time of attestation. Another way to achieve domain separation is to measure a component (e.g. it's binary code) before it is started — before it can influence the attestation result.

VI. Trust base

Most attestation methods use an *axiomatic* trust base (Q4), where trust decisions are done by fiat [22]. *Endorsements* are attestation metrics whose trustworthiness is vouched for by a trusted party (an endorser) in some way, typically by appending digital signature to the metrics. Endorsements are needed in most variants of attestation, except in semantic attestation.

One interoperable way to store and convey endorsements is Attestation Transparency (AT) [44][45] which builds on top of Google's Certificate Transparency (CT) framework [46]. CT revolves around a public, append-only log that containing a whitelist of known-good public-key certificates. Consistency and trustworthiness of the log is maintained by continuous monitoring. A TLS extension defined in [46] can be used to prove that the sender's (TLS end-point) public-key certificate is included in the log. AT adds attestation metrics into these certificates, turning them into endorsements. The advantage of AT is that it builds on top of existing public-key infrastructure (PKI), which has been widely studied and deployed. However, only certificates signed by established CAs can be included in CT logs, and these CAs usually refuse to sign custom extensions, such as attestation evidence. To circumvent this, [44] suggests to use separate AT and CT logs, and bind attestation evidence to the certificate public keys by including the public key hash among the metrics.

Project Trillian[2] is an AT-like endorsement conveyance method that has found some adoption in practice (see Section XII). Project Trillian is an open-source implementation of the gRPC service[3], i.e. a repository of verifiable data structures described in a whitepaper by Eijdenberg et al. [47]. These data structures are Verifiable Logs, Verifiable Maps and Verifiable Log-Backed Maps. Verifiable Logs in Project Trillian [46] are implemented as Merkle Trees. The idea of Verifiable Logs is to provide a transparent ledger, where entities can publish data in append-only manner. Project Trillian represents hash values of all data elements in this tree. The signed Verifiable Logs structure allows clients to detect tampering of data, history of data and split-view attacks (service show only partial data). The most widely used application for Trillian is CT. Another application of Trillian is Firmware Transparency (FT), where Firmware metadata of "good firmware" are stored in Verifiable Logs. A verifier who receives claims containing firmware metadata associated with a target entity can compare such values against references published in Trillian logs.

In contrast to axiomatic trust, the *semantic attestation paradigm* provides the verifier the possibility to directly analyze the target's behaviour, providing an analytic trust base. For example, the verifier can, for example, transmit a testsuite, which the attester then executes on the target, including the test results in attestation evidence. Just-in-time (JIT) compilation, interpretation and sandboxing can be used to achieve *synthetic trust*, where the target is transformed into a more trustworthy (e.g. better isolated or constrained) form by a trusted compiler or interpreter. WebAssembly is used for this purpose in some enclave projects, see Section XII).

VII. Interaction patterns

In this section, we discuss the various topologies and interaction patterns in attestation (Q5).

A. In the *passport model* [13, Sec 5.1.], the relying party receives the attestation result from the attester instead of from the verifier, as shown in Fig. 6. This is similar to how people identify themselves by presenting a passport that has been issued by a trusted party. Successful identification (identity attestation) is a precondition for issuing the passport (attestation result), which then stored (cached) and used multiple times. The passport model is called *certificate-style* attestation in some sources [11].

B. In the *background check model* [13, Sec 5.2.], the attester delivers the evidence to the relying party, who then sends it to the verifier (e.g. an online verification service) for evidence verification, as shown in Fig. 7. The background check model is called "back-channel" in SAML 2.0 literature [12, p. 1597].

C. In *attestation by proxy* [2][48], the verifier receives from the target an evidence of type A, and forwards it to a verification proxy. The proxy evaluates the evidence and issues a new evidence, of type B, based on the original evidence and returns the evidence to the main verifier. The primary benefit of this approach is that it allows to abstract over various kinds and formats of attestation evidence. A verifier does not have to support all possible kinds of attestation evidence. For example,

[2]https://github.com/google/trillian

[3]https://www.grpc.io/

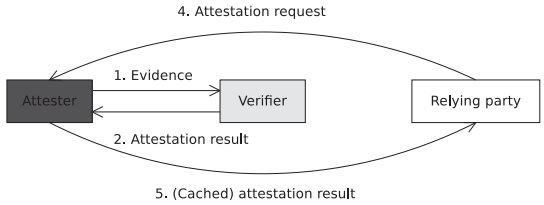

Fig. 6. Passport model

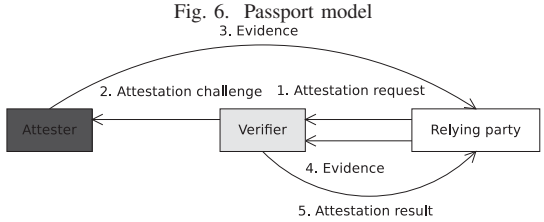

Fig. 7. Background check model

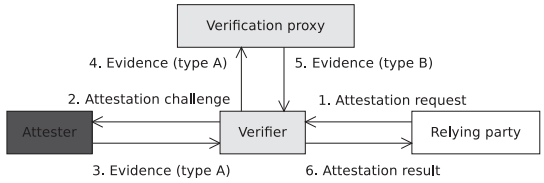

Fig. 8. Attestation by proxy

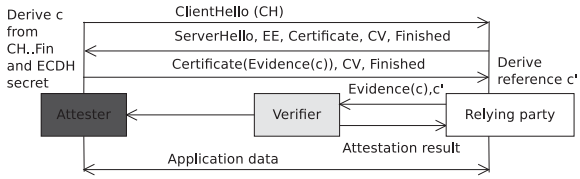

Fig. 9. Trusted channel model with one-sided intra-handshake attestation

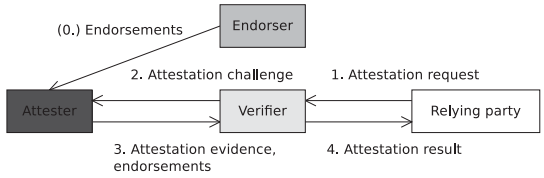

Fig. 10. Bundled endorsement model

there might exist a separate proxy for each evidence type. The attester also only needs to communicate with a single verification server. Project Veracruz uses attestation by proxy for this purpose [2]. Arm Ltd. has also applied for a patent that covers attestation by proxy [49].

D. In *trusted channel model*, attestation is bound to a secure channel, such as a TLS 1.3 session. There are three fundamental approaches for this: *pre-handshake* (evidence is genered before the handshake), *intra-handshake* (evidence is generated during the handshake) and *post-handshake* (evidence is generated after secure channel establishment). According to [50], the intra-handshake approach is the best, because it avoids the extra round-trip needed for the post-handshake approach and allows stronger channel binding (see Section IX) than the pre-handshake approach. Furthermore, the intra-handshake approach is conceptually convenient, because it makes secure channel establishment conditional on successful attestation, avoiding an intermediate "secure, but not attested" channel. Fig. 9 shows the trusted channel approach, following the proposal of [50], but with one-sided instead of of mutual attestation. The challenge c is influenced by the transmitted handshake messages as well as the derived ECDH secret, providing strong channel binding.

E. In the *bundled endorsement model*, the endorsements are obtained by the attester and bundled with the attestation evidence. For example, when using binary attestation, the endorser may be the provider of the the target and attester components. When provisioning these components, the provider may provide a signed approval (endorsement) of their hashes as auxiliary data. The attester can then concatenate these to the evidence.

F. In *challenge-response attestation*, the verifier sends a challenge value (nonce) to the attester. The verifier expects the attester to include the nonce as one of the attestation metrics. The goal is to ensure freshness of attestation evidence and prevent relay and replay attacks.

G. Attestation protocols are usually interactive to guarantee freshness of the attestation evidences. However, interaction is not always possible between the attester and the verifier, for example, when store-and-forward communication is used. *In uni-directional attestation*, the verifier is not able to supply a fresh nonce to the attester. One solution is to use a timestamp in place of a nonce if a secure clock is available.

H. In the *mobile agent model*, the target entity is a mobile agent running in a trusted execution service (TES) such as a virtual machine. Mobile agents have the ability to migrate from one TES to another without losing execution state. Thus, instead of letting the TES perform metric collection and packaging, it is possible to migrate the target entity to a TES on the verifier's (or relying party's) device, and attest it there locally. Migration can also be performed via a store-and-forward channel. The mobile agent model has received little attention in the literature; the closest work is probably [51].

I. In standard attestation, targets are typically attested only when their services are needed. In contrast, *streaming attestation* attempts to detect target compromise as soon as it occurs. For example, some streaming attestation methods use heartbeats to detect when a device leaves the network, which may be an indication for a possible physical attack. [52]

J. Attestation can be *local* or *remote*, depending on whether the verifier is on the same or a different device as the prover. In some sense, local attestation is harder: if the device is under the control of an attacker, how can the verifier be trusted to operate correctly? Typically, a local verifier must be part of the TCB. Local attestation is used for enclave-to-enclave interaction on the same device, or when a host application needs to establish trustworthiness of the enclave it has launched. A special case of local attestation is *secure boot*, where the first-stage bootloader (combining the attester, verifier and relying party roles) is trusted to measure the next-stage bootloader's binary code. If the hash matches a hard-

coded endorsement value, the next bootloader is loaded. The process continues iteratively until the operating system kernel. In contrast to secure boot, *trusted boot* is based on remote attestation: no component is prevented from being loaded, but the cumulative hash of all loaded components is reported to the relying party in attestation evidence.

VIII. Packaging

Evidence packaging (Q6) refers to the process of encoding the attestation metrics and protecting them. All message types (evidence, endorsement, result) require specifications and encoding formats.

A *data definition language* can be used to define data types. Examples include ASN.1, CCDL and JSON. In the TCG context, ASN.1 is typically used, but e.g. RATS uses CDDL instead. The *encoding* of a type should be uniquely decodable, compact to make transmission efficient, and canonical to allow it to be digitally signed. Some formats such as JSON and CBOR [53] combine definition and encoding into one, while others, such as ASN.1 [54], provide separation between the two, allowing different encoding rules to be used for the type definition. For example, ASN.1 types can be encoded using the unique Distinguished Encoding Rules (DER), the compact Packed Encoding Rules (PER) or the human-readable XML Encoding Rules (XER). DER is used to encode X.509 certificates, cryptographic keys and many TCG attestation data types. The RATS architecture prefers the CBOR encoding format and CDDL types. CBOR has been designed to produce a very compact encoding, and small-footprint codecs such as tinycbor are available as open source. The comparison of DER, PER and CBOR encoding for common attestation data types is an interesting research topic; a survey of various encoding formats, including these, is available as a pre-print [55]. However, the authors do not take security relevant aspects, such as canonicality into account.

We regard *evidence protection* to be part of evidence packaging. Evidence must be protected against modification as well as replay and relay attacks. The origin of the evidence (attester) must also be authenticated in some way. Both integrity protection and origin authentication is typically achieved by letting the attester sign the metrics using its private (asymmetric) or secret (symmetric) key, such that the verifier has access to or can authenticate the corresponding verification key. It is convenient to include an identity of the attester (SignerInfo) in the attestation evidence to make verification easier. The identity may be, for example, a public-key certificate or a public key hash. Standard formats for signed data, such as CMS [56] and COSE [57] can be used for this purpose. Protecting against replay attacks requires either a timestamp or a nonce (*attestation challenge*) to be included in the attestation metrics. To protect against relay attacks, the nonce value should be provided by the verifier and be dependent on a unique identifier of the evidence conveyance channel, as analyzed in [50].

In order to protect the evidence, an attester needs an *evidence protection key*. The key must be known only to the attester and trusted (e.g. attested) parties. How the key is securely provisioned to the attester is an important problem. In many layered attestation methods, the current layer derives a new keypair for the next layer based on its own and the next layer's identity (usually code hash) [37]. This is done, for example, in DICE [20].

IX. Channel binding

Secure communication with an enclave-bound target is a common requirement. Typically, a secure channel protocol, usually TLS, is used for this purpose. The critical consideration is prevention of relay attacks, where a TLS end-point on a compromised system relays the attestation challenge to an uncompromised enclave on another system, and presents the received evidence as its own. In other words, it is important to ensure that the entity that is being attested really is the end-point of the current communication channel—the process that accomplishes this called *channel binding* [58]. Channel binding (Q7) is a non-trivial problem, as discussed in [50]. One technique is to compute a unique identifier for the current secure channel session, include it in the attestation metrics, and transmit and validate the evidence during secure channel setup. When secure channel setup is dependent on mutual attestation, the result is called a trusted channel [59] (see also Section VII).

X. Privacy

An important concern is *attestation privacy* (Q8)—the ability of the target entity or its author to choose which attestation metrics are revealed to 1) the verifier, 2) to observers listening to network traffic. There are essentially three ways to achieve attestation privacy using cryptography. The first is encryption: the target encrypts the attestation evidence using either the verifier's public key or a secure channel (such as TLS) session key. This requires the target be pre-provisioned with the verifier's public key or to trust the verifier's public key or certificate (for secure channel end-point authentication). The second approach is to use a zero-knowledge proof (ZKP), which allows convincing a remote verifier of an attestation metric without revealing the metric value to the verifier or potential eavesdroppers. The ZKP approach is used in the TCG's Direct Anynymous Attestation (DAA) [60], used together with TPMs. The third approach, used in Intel SGX' Enhanced Privacy ID (EPID) [27] attestation, is to rely group identities to protect attestation evidence. An alternative or supplement to cryptographic protection is to allow the target and verifier to negotiate a minimum set of metrics in the attestation protocol, ensuring that only the minimum amount of information is revealed.

XI. Linking

Unless the root-of-trust can attest the target entity directly, several attestation evidences are needed to establish trust and the evidences need to be linked (Q9). There are two basic approaches for this: composition and layering [12, p. 1598]. In *composite attestation*, each target generates an attestation evidence and forwards it to a *lead attester*, which concatenates

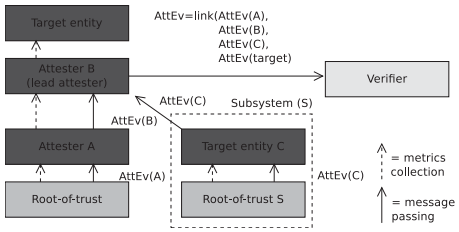

Fig. 11. Layered and composed attestation

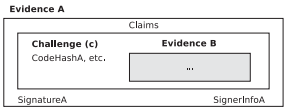

Fig. 12. Nested evidences

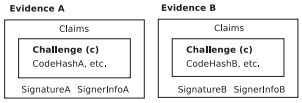

Fig. 13. Chained evidences with challenge-based binding

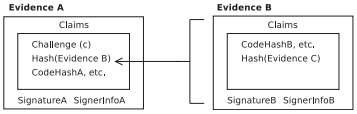

Fig. 14. Chained evidences with hash-based binding

and sends them together to the verifier, possibly adding its own attestation into the mix. In *layered* attestation, each attester attests the next one, with the root-of-trust at the bottom, as shown on the left side of Fig. 11. In the context of virtualized systems, layered attestation is called *deep attestation* [23][39]. In layered attestation, attestations evidences are either nested or chained. Hardjono et al. [12, p. 1599] list four requirements for layered attestations: i) each layer must be unambiguously distinguishable; ii) the next layer must be inspectable by the current layer; iii) there must be a way to achieve layer sequencing; iv) there must be a way for each layer to generate an attestation evidence of itself, while cryptographically binding the evidence to evidences of previous layers. The requirements may be difficult to fulfill in some use cases, however. Layered attestation typically assumes that each attested component is part of a consecutive boot sequence. If this is not the case, a combination of composite and layered attestation may be sometimes needed, as shown in Fig. 11.

In layered attestation, each layer may produce a separate attestation evidence. All evidences must then be conveyed to the verifier. Some form of binding should exist between the evidences so that the verifier can validate whether they all come from the same source and are connected to the attestation of the same target entity. There are two basic ways to do this: nesting and chaining. In nesting, the target entity's evidence contains the attester's evidence as one of the attestation metrics, as shown in Fig. 12 (or the other way around, see Section XI-A. The benefit of nested evidence is that the outer-level signature cryptographically binds all evidences together and there is no need to match evidences against each other. Evidence chaining may be easier to implement. One way to do chaining is to include the same challenge value as a metric in each evidence as in Fig. 13. This ensures that every evidence is fresh, but prevents evidence caching, which is useful e.g. when the last-level evidence is generated by a slow hardware root-of-trust. This approach provides only weak binding [25], and depends on the secrecy of the challenge value. Another way is to include the hash of the next evidence as one of the claims, as in Fig. 14. The problem of evidence binding in layered attestations is called *layer linking* in some sources such as [39] and [23]. Temporal order of attestations is also an important concern, as discussed in Section XI-A.

A. Temporal order of attestations

The *temporal order of attestations* is important to prevent TOCTOU attacks [61]. Consider a layered attestation with 3 layers. Assume that an attacker cannot compromise the root-of-trust (layer 1), but is able to compromise the next attestation layer (layer 2) at the time of his choosing. Then he can wait until the root-of-trust has generated (a valid) attestation evidence for layer 2 before launching his attack. A compromised layer 2 will naturally create a valid attestation evidence for layer 3. Now, the whole chain of attestation evidences will be valid, even though layers 2 and 3 are compromised. A critical consideration is the order in which the layers should be attested: *bottom-up*, starting from the most lowest (most privileged) layer, or *top-down*, from the highest (least privileged layer)? The answer depends on the difficulty for an attacker to *repair* a component, i.e. to convert it from compromised back to uncompromised state in order to receive a valid attestation evidence. If repairing is hard or impossible, top-down may be more secure [23]. If repairing is easy, then *bottom-up* is optimal [11]. In any case, the layer evidences must be bound together. This can be done, for example, with evidence nesting, using the first layer as the inner-most evidence, or with chaining, where each evidence includes a hash of the previous layer's evidence among the attestation metrics. It should be noted that the approach of Fig. 13, where evidences are bound together by including the same challenge in each, cannot be used to prove temporal order of attestation. Nested, top-down approach potentially also has the befit that it allows each layer to understand what it is signing. Each evidence then not only vouches for the original metrics of the target, but also that those metrics were valid when the target attested some other entity. The Copland language and formal semantics can be used to define and negotiate attestation processes, including the order in which layer attestations are generated [11].

B. Scalability of evidence binding

Attestation by the root-of-trust can be very slow in some cases (e.g. when the root-of-trust is a TPM). Therefore, it may be unacceptable to require a fresh attestation evidence from the root-of-trust in every layered attestation, especially if the number of attested components is large. This is the

case, for example, when a hypervisor attests a number of virtual machines, and the hypervisor is attested by a root-of-trust. Evidence caching can used in this situation, but then the problem becomes how to link the lower-level evidences with the top-level evidence. The work [39] proposes one solution for this: the hypervisor creates a set of n key pairs $(k_1, K_1), ..., (k_n, K_n))$ in advance, with n being the expected maximum number of virtual machines (VMs) in the system. When a VM is launched, the hypervisor gives it the next unused keypair, e.g. (k_i, K_i). The hypervisor is attested by the root-of-trust only once, but the public keys $K_1, ..., K_n$ are all included in the hypervisor's attestation metrics. This provides linkage between hypervisor and VM attestation evidences. A similar solution is proposed for scalable attestation in the Arm CCA architecture [33, Section 9.4].

XII. Survey of attestation in open-source enclave projects

A. Veracruz is "a framework for designing and deploying collaborative privacy-preserving computations" [2]. It currently supports TrustZone and SGX based enclaves, AWS Nitro enclaves, and enclaves based on seL4/IceCap virtual machines. Support for CCA is on the project's backlog. The main component of Veracruz is a trusted runtime, which also provides attestation and TLS-based trusted channels. The enclave services are compiled into WebAssembly, providing sandboxing and executed at run-time, providing a form of synthetic trust (see Section VI). Veracruz uses attestation by proxy, where the various "native" attestation evidence formats are converted into CBOR-encoded Arm PSA attestation tokens by a proxy. The communication between the proxy verifier and the relying party is based on the Arm PSA attestation protocol. [48]

B. Since the Arm CCA ecosystem is expected to comprise various stakeholders and supply chain actors, the verification of the trustworthiness of the Arm CCA enclaves is likely to be complex. Therefore, Arm has initiated the open-source *Project Veraison* [19], whose goal is to implement an Attestation Verification Service that is flexible enough to support various endorsements, interaction models, Appraisal Policies and RoTs. Project Veraison developers have seen the opportunity to utilize firmware transparency through the Verifiable Logs provided by the Project Trillian (see Section VI). This allows vendors from supply chain to endorse firmware binaries.

C. The open-source *Open Enclave* (OE) SDK originating from Microsoft, focuses on APIs (both run-time and interaction ones) for platform agnosticity. OpenEnclave applications are written in C/C++, so portability without recompilation is not supported. From the start, OpenEnclave has been supported on Intel SGX, where APIs to attest OpenEnclaves using SGX methods have been supported. With a recent OpenEnclave port to ARM OP-TEE, alternative approaches to attestations have had to be considered [62]. The new plan includes a generic list of evidences such as platform and enclave code UIDs, code version and validity as well as platform indicator (SGX or OP-TEE). The attestation API is formalized as a plugin mechanism, and no formatting proposal to unify evidence structures has yet been proposed.

D. Similar to OpenEnclave, Google *Asylo* [63] is a C++ framework for enclave implementation, also initially only running on Intel SGX. Asylo focuses on the easy construction of host - enclave application pairs, by formalizing the enclave as a Remote Procedure Call (RPC). In a sense, Asylo can be considered an abstraction running over SGX, and its attestation support is proposed as an alternative to SGX Quoting Enclaves by Intel. [64]. The *Asylo Assertion Generator Enclave* (AGE) adds mutual device attestation to the RPC secure channel between two enclave applications, or between an enclave application and a non-enclave application. AGE attestation relies on SGX identities as evidences.

E. Red Hat's *Enarx*[4] is an open-source application deployment system that allows applications to run inside enclaves without source code changes; only recompilation into WebAssembly (WASM) bytecode is needed. Enarx enclaves (called Keeps) are currently based on Intel SGX and AMD SEV hardware. Each enclave contains a microkernel, the WASM runtime, the WASM System Interface (WASI) and the enclave application bytecode. By using WebAssembly, Enarx allows a synthetic trust base, like Veracruz. Attestation in Enarx has not yet been fully implemented, but some details are available in the project documention. The attested metrics include the SGX/SEV hardware and firmware versions and a hash of the Enarx runtime. An external verifier service provided by Red Hat must be used to validate the attestation evidence. A CDDL-defined and CBOR-encoded types are used to wrap SGX or SEV attestation evidences. Attestation is supposed to be fully transparent to the application. No channel bindings to standardized protocols such as TLS are provided, but AMD SEV's proprietary secure channel protocol (see e.g. [36, p. 4]) can be used. All metrics are coalesced into a single attestation evidence, and no evidence linking methods are described in the documentation.

F. Project *Oak*[5] is an open-source project from Google aiming to provide specification and a reference implementation for the secure transfer, storage and processing of data. The Oak provides isolated WASM runtime to execute code units called Oak Nodes. Oak Enclave consists of set of Oak Nodes called Oak application and Oak runtime. Clients can communicate with Oak application using Google gRPC protocol. When Oak Client establish connection to Oak Application they first agree ephemeral session key and then they provide assertions to each other. The client will receive exact hash of the Oak Runtime, which is cryptographically bound to gRPC session and signed by HW RoT. Current version of the Oak supports Intel SGX based enclaves and their attestations. The session key agreement and attestation is implemented using Enclave Key Exchange Protocol (EKEP) which is part of Google Asylo framework.

G. Amazon's *AWS Nitro Enclaves* [65] depend on a trusted

[4]https://enarx.dev/docs/Technical/Introduction
[5]https://github.com/project-oak/oak

TABLE I. ATTESTATION IN THE SURVEYED PROJECTS. HEADINGS Q1 TO Q9 REFER TO THE QUESTIONS IN SECTION II. ABBREVIATIONS: AXIOMATIC TRUST (AX), SYNTHETIC TRUST (ST), ATTESTATION BY PROXY (PROXY), USER-DEFINED (UD), BINARY ATTESTATION (BA)

Project	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
Veracruz	SGX/SEV	MRENCLAVE/MEASURE	BA	AX+ST	PROXY	PSA token	TLS	No	None
Veraison	Cortex-M/Dice	Hash-based	BA	AX	UD	PSA/DICE token	None	No	None
Open Enclave	SGX	MRENCLAVE	BA	AX	UD	Opaque blob	None	No	None
Enarx	SGX/SEV	MRENCLAVE/MEASURE	BA	AX+ST	PROXY	CBOR wrapper	None	No	None
Oak	SGX	MRENCLAVE	BA	AX	UD	CBOR wrapper	EKEP	No	None
AWS Nitro	N/A	Hash-based	BA	AX	UD	COSE/CBOR	None	No	None

hypervisor (Nitro Hypervisor) to enforce isolation and provide to provide attestation. The hypervisor is also the root-of-trust for attestation. AWS Nitro enclaves are currently a software-only solution and do not offer hardware-backed enclaves. Attestation evidence is encoded using COSE and CBOR. Attestation metrics include ID of the attester (a Nitro hypervisor) and its public-key certificate, binary hashes and an optional public key [65, pp. 33-34]. As in TPM-based attestation, Amazon's documentation calls the binary hashes "PCRs", but there is no evidence that the PCR values would actually be provided by a TPM. Verification process includes matching the hashes against endorsements, validating the signature and validating the certificate using the AWS Nitro Attestation PKI. No channel binding mechanism or guidance on how to do channel binding is provided in the documentation.

H. Other examples of open-source enclave projects that at least partially address attestation include Apache Teaclave[6], MarbleRun[7] and KubeTEE[8]. We do not review these here as either their design is still under development or they have not garnered significant industry mindset yet.

XIII. CONCLUSIONS

Table I summarizes our survey results, as applied to industrial and open-source projects. We can see that the projects in terms of attestation properties remain close to each other, differing mainly when it comes to packaging formats. However, the questions in Section II point to requirements that will have to be considered carefully before these systems reach widespread utilization: Will the mechanisms support (live) migration between platforms? Will attestation work in a proxy-less distributed manner, or at the very least, how can the existing proxies interact to enable cross-domain attestation. Is there not a necessity for channel binding, e.g. to push confidential data to (and from) enclaves as part of the attestation procedure? We note that none of the projects provides privacy as defined in Section II, because the target has no control over the set of metrics revealed to the verifier, such as the template-based MRENCLAVE or MEASURE identities, that are always included. Also, we believe that the emergence of VM-based enclaves will highlight the necessity of proper security linkage between multi-layered attestation schemes. All of these issues have been identified and solutions have been proposed in the body of surveyed academic work. As a conclusion, we now see theory turning into practice.

[6]https://teaclave.apache.org/

[7]https://marblerun.sh/

[8]https://github.com/SOFAEnclave/KubeTEE

REFERENCES

[1] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A comprehensive survey," *ACM Computing Surveys*, vol. 51, pp. 1–36, Feb. 2019.

[2] D. P. Mulligan, G. Petri, N. Spinale, G. Stockwell, and H. J. M. Vincent, "Confidential computing—a brave new world," in *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE Computer Society, 2021, pp. 132–138.

[3] A. Segall, *Trusted Platform Modules: Why, when and how to use them*. London, United Kingdom: Institution of Engineering and Technology, 2017.

[4] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, H. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen, "Principles of remote attestation," *International Journal of Information Security*, vol. 10, pp. 63–81, 2011.

[5] M. Alam, T. Ali, S. Khan, M. Ali, M. Nauman, A. Hayat, M. K. Khan, and K. Alghathbar, "Analysis of existing remote attestation approaches," *Security and Communication Networks*, vol. 5, pp. 1062–1082, 2012.

[6] J. Lyle, "Trustworthy services through attestation," Ph.D. dissertation, Oxford University, 2011.

[7] B. Prünster, G. Palfinger, and C. P. Kollmann, "Fides: Unleashing the full potential of remote attestation," in *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications, SECRYPT'19*. SciTePress - Science and Technology Publications, 2021, pp. 314–321.

[8] O. Demigha and R. Larguet, "Hardware-based solutions for trusted cloud computing," *Computers & Security*, vol. 103, p. 102117, Jan. 2021.

[9] B. Kuang, A. Fu, W. Susilo, S. Yu, and Y. Gao, "A survey of remote attestation in internet of things: Attacks, countermeasures and prospects," *Computers & Security*, vol. 112, p. 102498, 2022.

[10] W. A. Johnson, S. Ghafoor, and S. Prowell, "A taxonomy and review of remote attestation schemes in embedded systems," *IEEE Access*, vol. 9, pp. 142 390–14 210, 2021.

[11] S. C. Helble, I. D. Kretz, P. A. Loscocco, J. D. Ramsdell, P. D. Rowe, and P. Alexander, "Flexible mechanisms for remote attestation," *ACM Transactions on Privacy and Security*, vol. 24, no. 4, 2021.

[12] T. Hardjono and N. Smith, "Towards an attestation architecture for blockchain networks," *World Wide Web*, vol. 24, pp. 1587–1615, 2021.

[13] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, "Remote attestation procedures architecture," draft-ietf-rats-architecture, 2021.

[14] H. Birkholz, M. Eckel, W. Pan, and E. Voit, "Reference interaction models for remote attestation procedures," draft-ietf-rats-reference-interaction-models-05, 2021.

[15] ——, "The entity attestation token (eat)," draft-ietf-rats-eat-11, 2021.

[16] "Report on NFV remote attestation architecture," Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG), Sophia Antipolis Cedex, FR, Group report, 2019.

[17] M. U. Sardar, S. Musaev, and C. Fetzer, "Demystifying attestation in Intel Trust Domain Extensions via formal verification," *IEEE Access*, vol. 9, pp. 83 067–83 079, 2021.

[18] G. D. H. Hunt, R. Pai, M. V. Le, H. Jamjoom, S. Bhattiprolu, R. Boivie, L. Dufour, B. Frey, M. Kapur, K. A. Goldman, R. Grimm, J. Janakirman, J. M. Ludden, P. Mackerras, C. May, E. R. Palmer, B. B. Rao, L. Roy, W. A. Starke, J. Stuecheli, E. Valdez, and W. Voigt, "Confidential computing for OpenPOWER," in *EuroSys '21: Proceedings of the Sixteenth European Conference on Computer Systems*. New York, NY, USA: ACM, Apr. 2021, pp. 294–310.

[19] ARM, "Attestation verification service /veraison)," https://github.com/veraison/veraison, 2021.

[20] *DICE Attestation Architecture*, Trusted Computing Group, Mar. 2021, rev. 0.23.

[21] A. Martin, "A ten-page introduction to trusted computing," Oxford University Computing Laboratory, Tech. Rep., 2008.

[22] E. G. Sirer, W. de Bruijn, P. Reynolds, A. Shieh, and K. Walsh, "Logical attestation: an authorization architecture for trustworthy computing," in *SOSP '11: Proceedings of the Twenty-Third ACM Symposium on Operating System Principles*. ACM, Oct. 2011, pp. 219–229.

[23] H. Lauer and N. Kuntze, "Hypervisor-based attestation of virtual environments," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, 2016, pp. 333–340.

[24] H. Lauer, A. Salehi, C. Rudolph, and S. Nepal, "User-centered attestation for layered and decentralized systems," in *Proceedings of the 2018 Workshop on Decentralized IoT Security and Standards (DISS)*, 2018.

[25] M. Eckel, A. Fuchs, J. Repp, and M. Springer, "Secure attestation of virtualized environments," in *35th IFIP International Conference on ICT Systems Security and Privacy Protection (SEC)*. Springer International Publishing, Sep. 2020, pp. 203–216.

[26] G. Chen and Y. Zhang, "MAGE: Mutual attestation for a group of enclaves without trusted third parties," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA, USA: USENIX Association, Aug. 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/chen-guoxing

[27] I. Anati, S. Gueron, S. P. Johnson, and V. R. Scarlata, "Innovative technology for CPU based attestation and sealing," Intel Corporation, Tech. Rep., 2013.

[28] 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.

[29] L. Wilke, J. Wichelmann, F. Sieck, and T. Eisenbarth, "undeSErVed trust: Exploiting permutation-agnostic remote attestation," in *2021 IEEE Security and Privacy Workshops (SPW)*, 2021, pp. 456–466.

[30] J. Gu, X. Wu, B. Zhu, Y. Xia, B. Zang, H. Guan, and H. Chen, "Enclavisor: A hardware-software co-design for enclaves on untrusted cloud," *IEEE Transactions on Computers*, vol. 70, pp. 1598–1611, Oct. 2021.

[31] K. Suzaki, K. Nakajima, T. Oi, and A. Tsukamoto, "Library implementation and performance analysis of GlobalPlatform TEE Internal API for Intel SGX and RISC-V Keystone," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020, pp. 1200–1208.

[32] "Introducing Arm Confidential Compute Architecture," Arm Limited, Tech. Rep. Den0125, 2021.

[33] "Arm CCA security model 1.0," Arm Limited, Tech. Rep. Den0096, 2021.

[34] "Arm Confidential Compute Architecture software stack," Arm Limited, Tech. Rep. Den0127, 2021.

[35] S. Fei, Z. Yan, W. Ding, and H. Xie, "Security vulnerabilities of SGX and countermeasures: A survey," *ACM Computing Surveys*, vol. 54, pp. 1–36, Jul. 2022.

[36] R. Buhren, H. N. Jacob, T. Krachenfels, and J.-P. Seifert, "One glitch to rule them all: Fault injection attacks against AMD's secure encrypted virtualization," in *CCS '21: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2021, pp. 2875–2889.

[37] T. Hardjono and N. Smith, "Decentralized trusted computing base for blockchain infrastructure security," *Frontiers in Blockchain*, vol. 2, pp. 1–15, 2019.

[38] A. SEV-SNP, "Strengthening VM isolation with integrity protection and more," *AMD White Paper, January*, 2020.

[39] G. Arfaoui, P.-A. Fouque, T. Jacques, P. Lafourcade, A. Nedelcu, C. Onete, and L. Robert, "A cryptographic view of deep-attestation, or how to do provably-secure layer-linking," in *Proc. 20th International Conference on Applied Cryptography and Network Security*. Springer, 2022, to appear, https://eprint.iacr.org/2021/1487.

[40] A.-R. Sadeghi, S. Schulz, and C. Wachsmann, "Lightweight remote attestation using physical functions," in *WiSec'11: Proceedings of the fourth ACM conference on Wireless network security*. ACM, 2011, pp. 109–114.

[41] S. Wagner and C. Eckert, "Policy-based implicit attestation for microkernel-based virtualized systems," in *Information Security. ISC 2016*, ser. Lecture Notes in Computer Science, M. Bishop and A. Nascimento, Eds., vol. 9866. Cham: Springer, 2016, pp. 73–89.

[42] A.-R. Sadeghi and C. Stüble, "Property-based attestation for computing platforms: caring about properties, not mechanisms," in *NSPW '04: Proceedings of the 2004 workshop on New security paradigms*. ACM, 2004, pp. 67–77.

[43] V. Haldar, D. Chandra, and M. Franz, "Semantic remote attestation—a virtual machine directed approach to trusted computing," in *USENIX Virtual Machine Research and Technology Symposium*, vol. 2004. San Jose, USA: USENIX Association, 2004.

[44] J. G. Beekman, J. L. Manferdelli, and D. Wagner, "Attestation transparency," in *Asia CCS '16: Proceedings of the 11th ACM on Asia Conference on Computer and Communication Security*. ACM, 2016, pp. 1–13.

[45] J. G. Beekman, "Improving cloud security using secure enclaves," Ph.D. dissertation, University of California at Berkeley, 2016.

[46] A. Laurie, A. Langley, and E. Kasper, "Certificate transparency," RFC 6962, Jun. 2013.

[47] A. Eijdenberg, B. Laurie, and A. Cutter, "Verifiable data structures," https://github.com/google/trillian/blob/master/docs/papers/VerifiableDataStructures.pdf, 2015.

[48] V. Project, "Veracruz attestation," https://github.com/veracruz-project/veracruz/wiki/Veracruz-Attestation, 2021.

[49] M. S. L. Brossard, D. D. Miller, and D. P. Mulligan, "Attestation forwarding," US 2021/0409404 A1, Dec. 2021.

[50] A. Niemi, V. A. B. Bop, and J.-E. Ekberg, "Trusted Sockets Layer: A TLS 1.3 based trusted channel protocol," in *Secure IT Systems: 26th Nordic Conference, NordSec 2021*, ser. Lecture Notes in Computer Science, N. Tuveri, Ed. Cham: Springer International Publishing, 2021, pp. 175–191.

[51] J. H. Østergaard, E. Dushku, and N. Dragoni, "ERAMO: Effective remote attestation through memory offloading," in *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, 2021, pp. 73–80.

[52] L. Moreau, E. Conchon, and D. Sauveron, "CRAFT: A continuous remote attestation framework for IoT," *IEEE Access*, vol. 9, pp. 46 430–46 447, 2021.

[53] C. Bormann and P. Hoffman, "Concise binary object representation (CBOR)," RFC 8949, Dec. 2020.

[54] J. Larmouth, *ASN.1 Complete*. USA: Morgan Kaufmann Academic Press, 2000.

[55] J. C. Viotti and M. Kinderkhedia, "A survey of JSON-compatible binary serialization specifications," *CoRR*, vol. abs/2201.02089, 2022. [Online]. Available: https://arxiv.org/abs/2201.02089

[56] R. Housley, "Cryptographic message syntax (CMS)," RFC 5652, Sep. 2009.

[57] J. Schaad, "CBOR object signing and encryption COSE," RFC 8152, Jul. 2017.

[58] N. Asokan, V. Niemi, and K. Nyberg, "Man-in-the-middle in tunneled authentication protocols," in *Security Protocols*. Springer Heidelberg Berlin, 2005, pp. 28–41.

[59] Y. Gasmi, A.-R. Sadeghi, P. Stewin, M. Unger, and N. Asokan, "Beyond secure channels," in *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing*. New York, USA: ACM Press, Jan. 2007, pp. 30–40.

[60] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *CCS '04: Proceedings of the 11th ACM Conference on Computer and communications security*. ACM, Oct. 2004, pp. 132–145.

[61] I. D. O. Nunes, S. Jakkamsetti, N. Rattanavipanon, and G. Tsudik, "On the TOCTOU problem in remote attestation," in *CCS '21: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2021, pp. 2921–2936.

[62] A. Gupta, "Custom attestation data formats for open enclave," https://github.com/openenclave/openenclave/blob/master/docs/DesignDocs/CustomAttestation.md, Microsoft.

[63] "Asylo: An open and flexible framework for enclave applications," http://asylo.dev, Google.

[64] "Asylo assertion generator enclave," https://asylo.dev/docs/concepts/remote_attestation.html, Google.

[65] "AWS - AWS Nitro enclaves user guide," https://docs.aws.amazon.com/enclaves/latest/user/enclaves-user.pdf, Amazon.