Platform Attestation in Consumer Devices

Arto Niemi, Vijay Nayani, Mariam Moustafa, Jan-Erik Ekberg
Huawei Technologies Oy (Finland) Co Ltd.
Helsinki, Finland
firstname.surname@huawei.com

***Abstract*—Platform attestation allows consumer devices to report their security state to relying parties such as cloud services and network gateways. In contrast to more restricted forms of remote attestation, such as key attestation, platform attestation provides more information to the verifier, but is complex to deploy, which has hindered its adoption in the industry. Recently, new approaches such as device health attestation (DHA) have been introduced that simplify the remote attestation process especially from the relying party's perspective. A common denominator in these developments is the use of an external, usually cloud-based verification service that is physically separate from the relying party. The service transforms attestation evidence into a health report – a standard and simplified format that is easier for relying parties to process. In this paper, we survey the state of art in platform attestation in the industry, focusing on Windows DHA, Samsung Knox DHA, Android Play Integrity, Huawei SysIntegrity, and Apple's App integrity and Device Check.**

I. Introduction

Remote attestation – the presentation of cryptographically verifiable evidence regarding the identity and state of software or firmware – is now widely supported in the commercial off-the-shelf devices. The most popular form of remote attestation, especially on smartphones, is *key attestation* [1], [2]. It is used by remote services to verify an application's access to a hardware-protected key that is not extractable from the device and whose usage may require successful user authentication. Key attestation is increasingly complemented with *application attestation* [3] to also verify that an application binary is genuine and not hacked or compromised by malware. In contrast, *platform attestation* [4], [5] has a wider scope, covering the integrity of the hardware and software on top of which apps run – for example, that a known-good operating system version is running and root privileges are available only to trusted system components. The main use case of platform attestation is enterprise *mobile device management* [6], [7], where it is exploited to establish trust in PCs and mobile devices [8] before the devices are allowed to access protected resources, or to connect to internal networks. In smartphones, platform attestation is increasingly combined with app attestation as an *anti-abuse* mechanism to detect misuse of services, such as cheating in online games.

In all attestation methods, evidence is collected and signed by a device-local *attester*, which is more trustworthy than the attestee [9]. In particular, the attestee must not be able to influence evidence collection. This is commonly achieved by protecting the attester using hardware-based isolation. The protection may be vertical such as privilege rings, or horizontal such as a secure processor mode or a trusted execution environment (TEE). Another approach is to allow the attester to finish evidence collection before other, untrusted software is loaded in a boot chain. Trust in the attester and its protections may be achieved by recursive attestation such as measured boot [10] or layered attestation [11], [12]. Once attestation evidence has been generated, it is transmitted to a remote device, where it is appraised by a *verifier*. The verifier's verdict, or *attestation result*, is then used by a *relying party* to make a trust decision [13].

Implementing platform attestation is challenging. A verifier needs to know which device identities and software binaries are trustworthy, and which security features must be enabled on the device to make it secure for a particular use case. Often, this requires an enormous database of trusted reference hashes to be maintained, which makes evidence appraisal complex. The current trend is towards external cloud-based verification services. These hide most of the complexity, or at least delegate it to appraisal policies and their processing engine. The verifier produces a digestible message called *attestation result* that the relying party can use for its decision-making without complex verification machinery. We believe the external verifier paradigm may well be the step that clears the hurdles that have so far prevented platform attestation from being used widely in consumer devices.

In this paper, we attempt to survey the state of the art of platform attestation in the industry. We look for answers to questions such as: How is evidence collection protected from malicious influence? What is the coverage of evidence? How is evidence bound to the attestation context, such as time or the communication channel, — to prevent replay and relay attacks? Who owns and controls the verifier? Or, what are the contents of attestation results — what kind of inferences can the relying party make based on the results? We focus on existing implementations of attestation, deployed in commercial consumer devices — in contrast to prior surveys that cover academic proposals and non-commercial projects. For example, attestation for embedded systems [14], [15] and for TEE and confidential computing [11], [16] have been considered previously.

The rest of the paper is structured as follows. First, we motivate our survey with a short history of platform security in consumer devices and the trends behind the increasing use of platform attestation. Then, we describe the hardware-backed elements that constitute the foundation of platform attestation. We present the terminology framework and a set of questions that shall guide our survey. Then, we analyze, in turn, Windows *Device Health Attestation*, *Knox Attestation*, *Android Play Integrity*, *EMUI SysIntegrity* and *Apple Device Check* and describe how these answer our survey questions. Finally, we present conclusions and topics for further study.

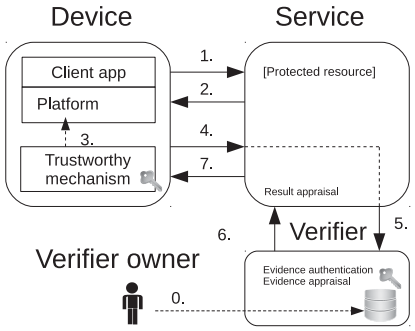


Fig. 1. Platform attestation using the background check model. During a setup phase (step 0), the verifier owner provisions reference values (endorsements) into the verifier's database. Later, (1) a client app requests a protected resource from an online service, which (2) returns an attestation challenge. (3) A trustworthy, hardware-backed mechanism measures the platform and, optionally, application state and generates attestation evidence. (4) The application sends the evidence to the service, who (5) forwards it a verifier. (6) The verifier authenticates the evidence by validating its signature using a pre-provisioned public key or root certificate, appraises the attestation claims and returns an attestation result (health report, or verdict). (7) Based on the result, the service decides whether to grant access or return an error.

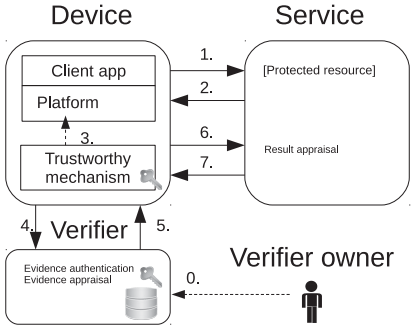


Fig. 2. In the passport model, the device sends (4) the evidence directly to the verifier and receives (5) a result ("the passport"), which it presents (6) to the service when requested. The passport may be generated fresh, based on a challenge from the service (1), or in advance and then cached.

II. Background

End-user devices such as smartphones, desktops and laptops have three characteristics that make them particularly vulnerable to attacks. First, most of these devices allow installation of user-chosen software or access to untrusted webpages, both of which may contain malicious code. Second, these devices provide ample opportunity for attacks requiring physical proximity. Smartphones and laptops, for example, are easily lost, stolen or confiscated, allowing both data extraction and so-called evil maid attacks [17]. In some use cases, such as digital rights management and mobile network subscriber identity, secrets in the device may even need to be protected against the user itself. Finally, supply-chain attacks are a significant risk for devices such as PCs that may be assembled by an untrusted integrator from untrusted components.

Starting from the late 1990s, considerable investments were made to counter these threats. Common operating systems received malware detection suites that now run by default. Mobile phones were universally equipped with trusted execution environments (TEEs) based on secure elements such as embedded secrue elements [18] or processor secure environments such as Arm TrustZone [19]. Most new PCs and laptops stock a Trusted Platform Module (TPM) [20], which operating systems use to protect disk encryption keys and to guard the boot process. More advanced isolation features such as virtualization-based security is also increasingly designed for deployment in consumer devices [21].

In parallel, mechanisms have been devised for *verifying* the existence and status of these security features on a particular platform. The commercial need for such verification is driven by two trends. First, consumers today mainly use their devices to access online services such as websites or cloud-based software; working offline with only locally installed software is becoming rare. Second, company employees are increasingly allowed to use their own devices to access the company network and services.

Remote attestation [9] provides a way to verify the status of security measures enabled on a platform: a trusted component in the device generates cryptographically protected evidence, describing the platform's security status to the remote verifier. Remote attestation was first pioneered in the first decade of the 21th century by the Trusted Computing Group, which specified a TPM-based method for verifying the identities of loaded boot images [20]. This was followed by key attestation for smartphones, first proposed in 2010 for Nokia devices with M-Shield [22], and deployed at scale in Android 7 [23]. Key attestation allows verifying that a credential is protected by the device's TEE. Main use of key attestation was initially to ensure that DRM content decryption keys cannot be extracted from the device, and to bind user identity to a particular device. Since then, coverage of key attestation has been extended to cover more properties of the platform [24].

However, the complexity of platform attestation has significantly delayed its wide-scale adoption. *Appraisal of evidence*, in particular, has been a major stumbling block. Appraisal typically involves comparing the asserted attestation claims against reference claims provided by a trusted *endorser*. Gathering the reference values for all possibly attestable components can be a daunting task. The exact semantics of each reference value can also be difficult to determine. A smartphone manufacturer may have endorsed a particular OS kernel version, but does this mean the kernel is trustworthy for both consumer and government, or even military use? The problem is that trust is always contextual (Bursell, [25]). Moving evidence verification to a specialized service that is in a better position to handle the complexities of appraisal is one way to address these issues, and this is in practice becoming the dominant paradigm. All the platform attestation methods surveyed in this paper rely on a central verification service provided by either the OS or the device manufacturer.

Current deployments of platform attestation are based on two interaction modes [13]: passport and background check, illustrated in Fig. 1 and Fig. 2, respectively. In the former, the attester sends evidence to a verifier, receiving an attestation result ("passport") that it can later use to prove its trustworthiness to relying parties. In the latter, the attester sends the evidence to the relying party, which forwards it "in the background" to the verifier, from which it receives the result.

III. Implementing the trustworthy mechanism

Platform attestation requires the attesting device to have a trustworthy mechanism that can be relied upon to collect the attestation claims and sign the attestation evidence. The mechanism must be protected from malicious influence. Especially, run-time isolation is needed to ensure that the attestee cannot cheat by influencing the measuring process. In practice, two forms of isolation are used for this purpose in platform attestation: hardware-based isolation and temporal, i.e. "boot-based" isolation, illustrated in Fig. 3.

A. Trusted Execution Environments

A *trusted execution environment* (TEE), also called a *processor secure environment* (PSE), is a secure mode of operation in a general-purpose processor that runs alongside but isolated from the rest of the system, called the *rich execution environment* (REE). The applications protected by a TEE are called trusted applications (TAs). [26] While traditional CPU privilege levels (rings) provide vertical isolation, TEEs provide horizontal isolation [27].

The first TEE that was widely deployed commercially is Arm TrustZone. The TrustZone architecture, popular especially on smartphones, provides a single TEE called the secure world. The secure world runs its own operating system, and peripherals can be configured to be accessible only from the secure world. The operating system, often referred to as *TEE OS*, and its interfaces have been standardized by the *GlobalPlatform* technical standards organization [28]. Recently, the trend has been towards architectures that allow multiple TEEs (often called enclaves) on the same device. Examples include Intel SGX, AMD SEV-SNP and Arm CCA [18]. While enclaves are now offered as a VM protection feature by some cloud providers, they have not yet found much use on consumer devices [29].

B. Secure co-processors

While TEEs are widely used and offer excellent performance by the virtue of being part of the device's main processor, recent years have seen the increasing adoption of physically separate *secure co-processors* in devices [30]. Compared to TEEs, secure co-processors are less vulnerable to side-channel attacks [18, p. 82], and require less extensive interaction with the untrusted OS, making their software interface easier to implement securely [30]. Even if the main CPU and its TEE are compromised, a secure co-processor can still remain trustworthy. Consequently, secure co-processors are increasing used to support platform security and attestation on consumer devices. Examples of such co-processors include *Microsoft Pluton*, Apple's *Secure Enclave Processor* (SEP), Google's *Titan-M* and Samsung's *Knox Vault*.

Microsoft Pluton is a secure co-processor based on the *Security Complex* hardware that first appeared in the SoCs of Microsoft's Xbox One game console in 2013. Jointly designed by Microsoft and AMD, it received the codename Pluton in 2021, when it was introduced into PCs [31]. Currently, Pluton is found in AMD Ryzen 6000 and Qualcomm Snapdragon 8cx Gen 3 series of processors [32]. Pluton implements the TPM 2.0 specification, but also provides other features, such as a hardware key store and the Device Identifier Composition Engine (DICE) for attestation [33], [31].

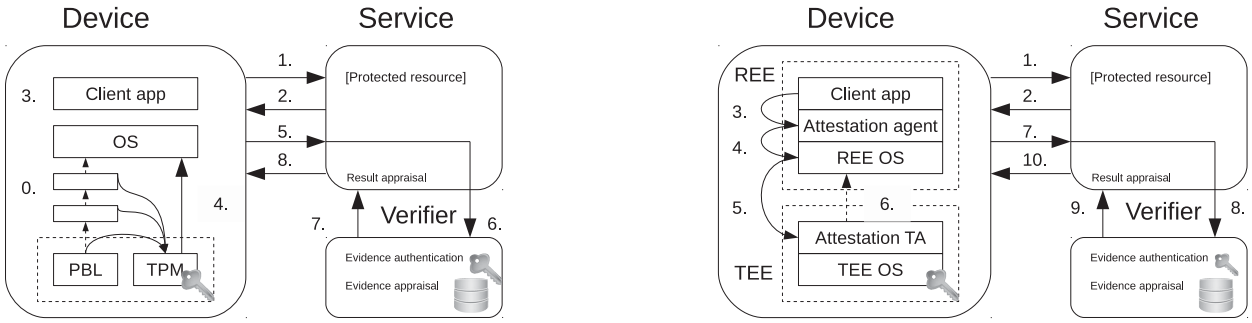
Apple's *Secure Enclave Processor* (SEP), first deployed in the iPhone 5S in 2013, is now commonly found both on Apple's mobile devices, where it is integrated into the main SoC, and on MacOS PCs, where it is part of the T2 security chip [18, p. 84]. The SEP runs its own operating system (SEPOS) that is based on the L4 microkernel. It has its own crypto engine and fuses, and each SEP includes a unique root key called UID. The SEP has little internal memory, but is reserved a region in main memory, which it encrypts and authenticates with AES-XTX and CMAC. [34] The SEP is used both for attestation and for securing the boot process.

The Samsung Knox security framework includes an secure co-processor called *Knox Vault* [5, pp. 15-21]. The processor itself is implemented on the system-on-chip, but uses an external integrated circuit for inline-encrypted non-volatile storage. Knox Vault is used, for example, to protect the boot process, storing the attestation private key and signing attestation evidence.

C. Trusted Platform Modules

The Trusted Platform Module (TPM) [20] is a security component that is found in all recent PCs. It is passive as it can only receive commands from software or firmware, itself having no control or oversight over the platform [35]. The TPM provides a key store service with a small amount of secure non-volatile storage and platform configuration registers (PCRs) for collecting attestation metrics. The PCRs are not directly writable; at start, they contain a constant value (all-0 or all-1) and can only be updated via a *extend* operation that computes the new PCR value as $Hash(oldValue \| newData)$, where $Hash$ is usually SHA-256. The PCRs can be used to record a chain of events (such as boot image loads) as follows. The first event must be extended into the PCR by a trusted component such a read-only primary bootloader (PBL). Further events may then be extended by any component, including non-trustworthy ones. The final PCR value is included in attestation evidence (called a TPM Quote), which is accompanied by an unauthenticated log of the extended events. The verifier then self-hashes the events in the log, validates the quote and compares the self-computed hash against the quoted one. Any non-trusted component may try to force the PCR to a particular value by invoking *extend* with arbitrary data, but succeeding in this requires a pre-image attack against the hash algorithm, which is considered infeasible.

The TPM can be implemented as firmware or as a discrete chip on the system-on-chip (SoC) or motherboard. To reduce the bill-of-materials, many device vendors have recently moved from discrete chips to firmware-based implementations, especially on consumer PCs [31]. One example of a firmware implementing the TPM specifications is Intel's Platform Trust Technology (PTT), implemented as an application inside the Intel Management Engine (ME). The ME itself can be considered a TEE [18, p. 83]. Another example is AMD's Firmware TPM [36]. Microsoft Pluton also implements TPM 2.0 functionality [32]. In China, where the TPM is banned [37], the locally developed Trusted Cryptography Module (TCM) is used instead. The TCM specifications are not fully available



(a) TPM-based

(b) TEE-based

Fig. 3. Common platform attestation implementations, using TPM-based and TEE-based trustworthy mechanisms. **a)** In the TPM-based implementation, successive bootloaders extend boot image measurements to a PCR in the TPM (step 0). When the client application needs attestation evidence, it invokes the TPM through a driver in the OS. The TPM returns a Quote, containing the final PCR value (step 4). The Quote is signed using the TPMs AIK private key, which usually certified by the OS vendor's server. **b)** In the TEE-based implementation, the client app requests attestation evidence from an attestation agent (step 3). The agent invokes the attestation TA in the TEE via a driver in the REE OS to measure the platform and generate evidence (steps 4 and 5). The evidence is signed with a private key, usually provisioned by the device manufacturer and protected by the TEE's secure storage. The evidence is then returned to the client application (step 6).

in public, but one main difference is that TCM only uses cryptographic algorithms approved in China [35].

D. Boot firmware and bootloaders

On most consumer devices, the chain of trust for attestation starts with the boot process ("late-launch" enclaves such as Intel SGX are an exception). A primary bootloader (PBL) is the first code that runs on the main CPU after device hardware has been initialized by boot firmware. The PBL is almost always immutable. On PCs, the PBL is commonly part of boot firmware that implements the unified extensible firmware interface (UEFI). In a multi-stage boot, the PBL loads the secondary bootloader (SBL), such as GRUB, which then loads the OS bootloader. In some designs, such as Samsung devices based on the Exynos SoC, the boot stages are combined into a single boot image such as S-BOOT [38]. On devices with a TrustZone TEE, the PBL also splits the boot process splits into two parts, covering the secure and normal worlds, with both world's having their own SBL [38].

There are roughly three general approaches to securing the boot process. *Secure boot* checks the signature of each boot image using the vendor's public key and terminates the boot process on verification failure. In contrast, *measured boot* merely records hashes of the booted components to allow later attestation of the boot. *Trusted boot* boot combines the two approaches: it collects the boot image measurements for attestation and verifies image signatures, but does not terminate the boot process on signature verification failure. Instead, trusted boot takes other action such as setting a fuse value that marks the system as untrusted [5, p. 21]. Trusted boot may also perform other checks in addition to signature validation, such as checking whether the image is up-to-date based on measurements stored in a secure location at run-time during an earlier boot.

Smartphones typically rely on secure boot, while PCs often use either secure or measured boot. In UEFI, the PBL is often covered by secure boot [10], i.e. loaded by the boot firmware only if it has been signed using a key that the OEM trusts. On Linux PCs, the information whether secure boot was performed by UEFI-compliant firmware is available, along with other firmware variables, in `/sys/firmware/efi`. On smartphones, secure boot status is often recorded in the TEE's secure storage.

IV. Framework

The end goal of remote attestation is to allow a relying party to establish *trust* in the the attestee. Trust is not an inherent property, but a choice that the trustor makes regarding the trustee. In attestation-based trust establishment, the choice is made based on the evaluation of attestation evidence. A critical question is, then, what information the evidence should contain. Clearly, the evidence must identify the attestee for which the evidence was generated. The evidence must also allow predicting the behaviour of the attestee. Predictable behaviour requires a *specification* to which evidence binds the attestee. A specification can take various forms: it can be as complex as the binary code of the attestee, vendor documentation, a security policy, or something as simple as an endorsement by a trusted evaluator. *Isolation* is required so that the attestee can fulfill its specification without external influence. Finally, attestation aims to evaluate trustworthiness for a particular purpose. The evidence must be bound to a *context*: one may e.g. trust a bootloader to load only authenticated images, but this does not mean the bootloader should be trusted for any other purpose. Evidence generated for one context (such as for the purpose of accessing a particular online resource) should not be used to evaluate trustworthiness in another context. Evidence should also be bound to a temporal context, such as a timestamp or a nonce, to prevent replay attacks. We summarize the above trust requirements as follows:

1) **Authentication**. The component must be identified.
2) **Specification binding**. The component's identity must be bound to a specification.

3) **Context binding**. The component's identity must be bound to the context in which the component's trustworthiness is being evaluated:
   a) **Temporal**: the time at which the evaluation takes place.
   b) **Operational**: identity of the requested resource or operation.
   c) **Communicational**: identity of the communication channel and session over which the resource or operation is being requested.
4) **Isolation**. The component must execute unhindered and protected from the influence of untrusted components and actors.

It is not necessary for the trustor (such as a relying party in attestation) to directly check all of the four requirements. Many of them can be performed by a verifier on behalf of the relying party. At the same time, not all checks can be delegated: eventually it is the relying party's job to make the final verdict based on the information from the verifier. In particular, the verifier typically has no understanding of the operational context mentioned above, for example, whether the attested component is authorized to access a particular resource.

The component's identity need not be unique as long as the *combination* of context and identity is unique. For example, it may be enough to know that the attested component is *some* instance of a particular application (identity), if it is also known that this particular instance is an endpoint of the current communication session (communication context). A common authentication method is the use of public-key cryptography: the public key is a (cryptographic) identity, which may be further bound to a non-cryptographic identities, such as a name, via a certificate. An entity that can demonstrate possession of the corresponding private key is then assumed to have the identity. However, public-key based authentication is only useful under threat models that do not consider co-located attacks, such as with compromised OSs. Insider attackers can easily extract the private key unless it is protected by hardware. In such cases, authentication must be complemented with attestation.

In our discussion of platform attestation, we mostly use the terminology standardized by IETF's remote attestation procedures architecture (RATS) working group [13]. However, we make some terminological choices of our own, described below. We define a *trust anchor* (Bursell [25]), as a static component that allows trustors to assume trust in the system in which the trust anchor is present. Examples of trust anchors include root certificates, public key hashes and symmetric secrets. Trust anchors need not be attested; the mere presence of a trust anchor is enough, assuming that it cannot be modified by untrusted entities. Trust anchors are often provisioned by hardware or firmware vendors, and stored in read-only memory. A *root of trust* (RoT), is defined (GlobalPlatform [26]) as a computing engine, code and possibly data co-located on the same platform. An RoT provides security services to other entities. A critical property of an RoT is that no entity on the device can provide a trustable attestation for the RoT. A key is not an RoT, but an RoT can include a key as a trust anchor. Finally, we use the term *trustworthy mechanism* [9] to refer to the software and hardware (RoT) components on the device that work together to implement reliable measuring and attestation evidence generation so that the process is isolated from the interference of untrusted components such as the attestee.

V. Evaluation Criteria

In this work, we analyze attestation mechanisms and architectures that have been deployed in existing consumer products for the purpose of remote attestation – in either enterprise context (where the devices are part of a *fleet* of devices managed by the company), or in consumer context, where a cloud service might be interested in the *device health* of the terminal device at the other end of a service communication channel. We will evaluate the mechanisms based on the following set of questions, adapted from [11]:

- **Trustworthy mechanism.** What is the implementation of the trustworthy mechanism for evidence collection and signing? What are its trust anchors (secrets, keys and certificates)? How are they provisioned? How are they protected? What are the device's roots of trust for attestation – the components implementing the trustworthy mechanism that are themselves not attested?
- **Coverage**. Does attestation cover boot-time or run-time events, or both? Is application attestation supported in addition to platform attestation? What platform and application properties and state information are attested?
- **Protocol**. What are the interaction patterns involved in attestation? Which entity verifies attestation evidence? Who owns the verifier?
- **Results**. What is the format and the content of the attestation result?
- **Context binding**. How is the attestation evidence and result bound to the temporal, communicational and operational contexts in which the trustworthiness of the attestee is being evaluated?

In the following, we analyse in turn six distinct attestation frameworks: Windows Device Health Attestation, Apple Managed Device Attestation, Knox Device Health Attestation, Android Play Integrity, EMUI SysIntegrity and Apple App Attest.

VI. Windows TPM attestation and DHA

The relationship between Microsoft, Windows and the TPM module dates back to before TPM, to the so-called Microsoft Palladium project [39]. Therefore it is no surprise that Windows PCs have contained frameworks for TPM platform attestation for more than a decade — in various forms and with varying names. One comprehensive example is the *Platform Configuration Provider Helper-Kit* (PCP-Kit) [40] launched with Windows 8 in 2012. The kit encompassed client components that integrated with the Windows *Crypto Next-Generation* (CNG) interfaces in the Platform Crypto Provider, and a library + tool (PCPTool) for attestation verification in servers.

Trustworthy mechanism. TPM-based platform attestation, relies as, a root of trust for measurement, the hardware or firmware-based TPM implementation. Its trust anchor is the Endorsement Key (EKPriv) provisioned by the TPM manufacturer, often along with a certificate (EKCert). On first boot, the device generates a private key called AIKPriv, and asks Microsoft Cloud CA to issue a certificate (AIKCert) for the public key, using EKPriv and EKCert to prove that it AIKPriv is securely stored on device [4]. The main purpose of the AIK is to improve privacy by providing the platform a per-context identity instead of revealing the TPM's unique identity [41]. Another root of trust is the primary boot loader (PBL), the first code executed by the CPU at boot. It is provisioned by the PC or motherboard manufacturer and trusted implicitly without authentication. The currently executing loader collects claims by measuring (hashing) the next image to be loaded. Evidence is signed using the TPM-bound AIKPriv. Today, some Windows devices use Pluton secure co-processor instead of a TPM as the trust root.

Coverage. During boot, the OS stores core OS measurements as well as integrity measurements of the Early-Launch Anti-Malware (ELAM) functionality, as well as potential anti-malware daemons into TPM PCR registers. The ELAM driver policy is also measured as a metric for what early-boot validations have taken place in this specific device. Further, if ELAM (or the anti-malware) recognizes that an attack has taken place, it adds a PCR record of that fact, changing the PCR metrics as a consequence [40]. Together with an associated audit log (*TCG log*) the PCR values (of firmware, ELAM policy and anti-malware) do provide a comprehensive view of early boot device integrity.

The evidence (a TPM Quote) contains as attestation claims the aggregated hashes of booted. The TCG log is attached to the evidence, and its integrity can be verified by reconstructing the cumulative hash from the log entries and comparing it against hashes in the TPM Quote. [42] In PCP (for Windows 8) there was no integrated protocol or privacy consideration in PCP, although basic recommendations for attestation (server should provide a nonce) were given. During verification, the tool checked all cryptographic bindings between the TPM Quote, CA keys, PCRs and the audit log, but the parsing of the log itself ("the health report"), was left to the business logic in the server. In the latest iterations of the Windows OS, a verification server has been added to further abstract the TPM-based attestation. This architecture is named *Windows Device Health Attestation* (DHA), and is explored next.

Coverage (Windows 10 and 11). With DHA, measured boot covers the Windows boot chain until the so-called early boot drivers loaded by the kernel. Claim collection still occurs only during load-time; no post-boot run-time state is attested. Attestation coverage includes all bootloaders, the kernel loader (winload.exe), the kernel (ntoskrnl.exe) and early boot drivers such ELAM. Note that in Microsoft's terminology, *secure boot* refers to a UEFI 2.2+ feature that covers the boot process until the OS loader. Next, *trusted boot* takes over, and works until the early-launch drivers. Measured boot, on the other hand, refers to the collection of boot image measurements during secure and trusted boot.

Protocol. DHA adds a remote attestation protocol [44] based on a centralized verification service (*DHA-Service*) [4] described in Fig. 4. The main use case is to integrate attestation with Mobile Device Management (MDM), where the state of enterprise's managed devices are checked before they are allowed to access a protected resource. A free cloud-based DHA-Service is provided by Microsoft. Companies with a Windows Server 2016 license can also setup a DHA-Service instance on the company premises or in a private cloud. The DHA-Service converts the TPM-based attestation evidence into a result format called a *Health Report*, which is simpler for relying parties to interpret than hashes and logs.

Windows 11 further adds *Microsoft Azure Attestation* (MAA) which is a unified, Microsoft-provided cloud-based DHA verification service with well-defined and documented APIs. Code for critical operations such as evidence verification may be provided by the user and executed in an SGX enclave, so that Microsoft (as a cloud operator) need not be trusted by the relying party. MAA is not limited to TPM and DHA attestation, but is also able to process evidences of other types such as SGX and AMD SEV-SNP quotes. It additionally provides a rich policy language [45] for evidence appraisal, further improving the developer-friendliness of attestation.

Results. In Windows 10, the DHA service provides to the MDM server attestation results (health reports) in XML format using TLS [43] (see Fig. 5). In Windows 11 more report data was added, the format of the report changed from XML to JSON and the communication protocol changed from TLS to HTTPS. The *GetAttestReport* API implemented in the MAA provider instance returns a signed JWT containing the DHA Report.

Additionally, Windows 11 has also added a TPM-based integrated enterprise attestation solution named *Windows Hello for Business*. Based on available information [46], [47], this mechanisms abstracts TPM and its activation just like PCP-Kit did a decade earlier. It integrates with the Windows Domain Controller as well as with a Kerberos provided for tickets, but Windows Hello functionality only provides key attestation and use for a TPM-hosted key for the single purpose of multi-factor authentication in an enterprise cloud context.

VII. Apple Managed Device Attestation

In 2022, Apple introduced *Managed Device Attestation* (MDA) [48] for devices running the iOS 16, iPadOS 16.1 and tvOS operating systems. Like Windows DHA, the goal of MDA is mainly to support the enterprise use case, and MDA provides MDM servers with information on device identity and integrity via attestation evidence.

Trustworthy mechanism. Hardware backing for attestation on Apple devices is provided by the SEP secure co-processor (see Section III-B). An application-specific private key is generated by the SEP and stored securely. The SEP also measures the platform and generates attestation evidence.

Coverage. The contents of the evidence generated by the SEP are not publicly disclosed, but attestation coverage almost certainly includes identity of the SEP and proof that the device is a genuine Apple device. Whether also runtime state is attested is unclear based on current documentation.

Protocol. In contrast to Windows DHA, Apple's solution is tightly coupled with standard infrastructure (PKI), such

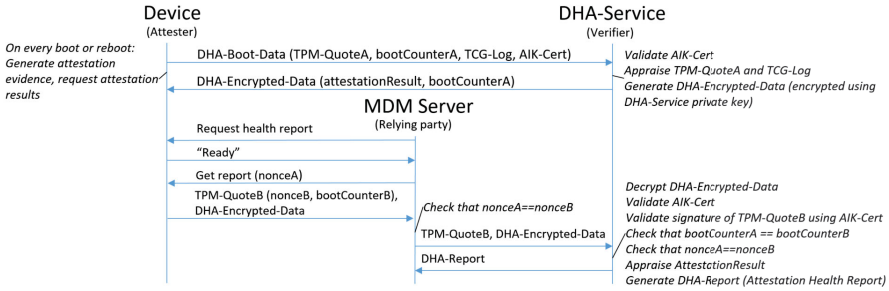


Fig. 4. MS-DHA protocol for Windows Device Health Attestation, symmetric case. The protocol essentially follow the passport model. The passport bound to the boot counter. When the passport is used, a second TPM Quote is generated, also containing a boot counter, thus ensuring that the passport can be deemed valid only if no reboots have occured in the meantime. The asymmetric case similar, except that the Device has no DHA-Encrypted-Data ready and must fetch one from DHA-Service when MDM Server requests a health report.

```
<?xml version="1.0" encoding="utf-8"?>
<HealthCertificateValidationResponse xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ErrorCode="0" ProtocolVersion="0"
    xmlns="http://schemas.microsoft.com/windows/security/healthcertificate/validation/response/v3">
    <HealthCertificateProperties>
        <Issued>2016-10-21T02:12:58.6656577Z</Issued>
        ...
        <BootManagerRevListVersion>0</BootManagerRevListVersion>
        <SecureBootEnabled>false</SecureBootEnabled>
        <CodeIntegrityEnabled>true</CodeIntegrityEnabled>
        ...
        <ELAMDriverLoaded>true</ELAMDriverLoaded>
        ...
        <BootAppSVN>1</BootAppSVN>
        <BootManagerSVN>1</BootManagerSVN>
        ...
        <BootRevListInfo>005D447A7CC6D101200000000B00C...</BootRevListInfo>
        <OSRevListInfo>8073EEA7F8FAD00120000000B00A82...</OSRevListInfo>
    </HealthCertificateProperties>
</HealthCertificateValidationResponse>
```

(a) Excerpt of a Windows 10 health report

```
{
    "iat": 1633665112,
    "iss": "https://contosopolicy.eus.attest.azure.net",
    ...
    "cnf": {
        "jwk": {
            "kty": "RSA",
            "n": "yZGC3-1rFZBt6n6vRHjRjvrOYlH69TftIQWOXiEH…",
            "e": "AQAB"
        }
    },
    "WindowsDefenderElamDriverLoaded": true,
    "bootAppSvn": 1,
    "bootMgrSvn": 1,
    "bootRevListInfo": "gHWqR2F-1wEgAAAACwBxrZXHbaiuTuO0PS…",
    "codeIntegrityEnabled": true,
    ...
    "osRevListInfo": "gHLuW2F-1wEgAAAACwDLyDTUQILjdz_Rf…",
    "secureBootEnabled": true,
}
```

(b) Excerpt of a Windows 11 health report

Fig. 5. Example attestation results (health reports) extracted from Microsoft documentation [43]. Windows 10 uses XML format, while Windows 11 uses JSON Web Token. Both objects contain similar information: (1) The issuance time and date of the report (2) Whether ELAM (Windows Defender) was loaded during initial boot. (3) The security version numbers of the boot application and boot manager that were loaded during the initial boot. (4) Whether only integrity verified code is allowed to execute and the version of the code that is performing those integrity checks. Windows 11 health report also includes additional information such as the "iss", which identifies the entity that generated the JWT, and "cnf" or confirmation, which represents a proof-of-possession of the attested key. A Windows 11 health report can also include SGX and SEV-SNP attestation results.

as CA-issued X.509 certificates [49] and related protocols, enhanced with key attestation. Attestation evidence includes key attestation for an SEP-protected key whose handle is available to the attesting application. The evidence, essentially a certificate signing request, is sent to Apple's Attestation CA, which appraises the evidence and issues an attestation certificate [50]. The device can use this certificate with Apple's enterprise Automatic Certificate Management Environment (ACME) (RFC 8555) server during device enrollment or re-attestation [51], [50]. Transmission freshness and security follows the ACME Device Attestation Challenge mechanism, which is currently a draft RFC [52].

Results. The attestation result is the ACME certificate that is issued based on the evidence sent to the ACME server. The server uses the attestation evidence as part of a trust score based on which the server grants the requested certificate [51]. The MDM validates the attestation result (certificate) by ensuring that the certificate is rooted with Apple's CA. The attestation result can only be updated once a week as generating new attestation is resource consuming to both the Apple device and servers [48].

VIII. Knox Device Health Attestation

In 2013 Samsung introduced *Knox*, a security framework for its Arm-based smartphones. Two attestation schemes are currently deployed in Knox devices: v2 and v3. The main difference is that v2 only provides platform attestation, while v3 provides both platform and app attestation [53]. In the following, we analyse v3.

Trustworthy mechanism. Knox uses a TEE-based trustworthy mechanism. The root of trusts include the TrustZone-protected TEE, which is relied upon to isolate Samsung-approved trusted applications (TAs) from the REE. Three TEE OS implementations are currently in use in Samsung devices: Kinibi, QSEE and TEEGRIS [53] – all of them are GlobalPlatform compliant [18, p. 160]. Another root of trust is the Knox Vault secure co-processor that is used to protect the attestation key. [5, pp. 15–20] All Knox devices have several hardware-protected and factory-provisioned trust anchors. The first is a symmetric Device-Unique Hardware Key (DUHK), generated on the device during manufacturing. The DUHK is used to generate and encrypt an RSA keypair called Device Root Key (DRK) and an ECDSA keypair called Samsung Attestation Key (SAK). Certificates for DRK and SAK are also provisioned during manufacturing. The Samsung Secure

```
{
    "version": 10,
    "nonce": "03f3c19fc71543b0b8acc63e4ab744bf",
    "verdict": "Yes",
    "timestamp": 1560810407037,
    "warrantyFuseState": 0,
    "trustBootState": 0,
    "deviceIdState": 0,
    "app": {
        "pkg": "com.samsung.android.eatest.plat",
        "signature": "CMGFvTzNed6wn81ERInLjY13pyIt0JBN8us58ab+7Zg="
    }
}
```

```
{
    "requestDetails": {
        "requestPackageName": "com.package.name",
        "nonce": "aGVsbG8gd29scmQgdGhlcmU",
        "timestampMillis": 1617893780
    },
    "appIntegrity": {
        "appRecognitionVerdict": "PLAY_RECOGNIZED",
        "packageName": "com.package.name",
        "certificateSha256Digest": [
            "6a6a1474b5cbbb2b1aa57e0bc3"
        ],
        "versionCode": 42
    },
    "deviceIntegrity": {
        "deviceRecognitionVerdict": [
            "MEETS_DEVICE_INTEGRITY"
        ]
    },
    "accountDetails": {
        "appLicensingVerdict": "LICENSED"
    }
}
```

```
{
    "advice": "RESTORE_TO_FACTORY_ROM",
    "apkCertificateDigestSha256": [
        "gwpz0q/WtjtJZZK5lTXl74fI/8QAKGLAhdhlznKkLhQ="
    ],
    "apkDigestSha256": "nBrInk2DLVQrgcQ1DXYnIUIMnjJ+mVo0lwUicF7OzJM=",
    "apkPackageName": "com.huawei.hms.safetydetectsample",
    "appId": "1*******",
    "basicIntegrity": false,
    "detail": [
        "root",
        "unlocked"
    ],
    "nonce": "UjJScmEyNGZWbTV4YTJNZw==",
    "timestampMs": 1612683290520
}
```


(a) Knox Attestation (b) Android Play Integrity (c) EMUI SysIntegrity

Fig. 6. Examples of attestation results in the surveyed TEE-based platform attestation schemes.

Boot Key (SSBK) keypair is used to sign all boot executables that are approved by Samsung. The SSBK public key is stored in one-time-programmable (OTP) fuse memory [5, p. 13]. Furthermore, Knox devices have a so-called warranty fuse, which starts with the value 0, but is set to 1 whenever non-Samsung approved firmware is loaded. Once the bit has been set to 1, it cannot be reverted back to 0. The attestation claims are securely stored in the TEE of the device.

Coverage. Attestation claims include the application identity (name, version and developer key), boot-time measurements and the warranty fuse value. The Samsung's implementation of the Keymaster TA gathers the claims and signs them using the SAK private key. Via the warranty bit, attestation covers not just the current boot-time measurements and the app identity, but also historical information. The warranty it 1 one when a non-approved image was ever loaded during the device's history. In addition, run-time state is attested via the warranty bit: in Knox devices, a runtime kernel protection (RPK) and a TEE-based TrustZone Integrity Measurements Architecture (TIMA) periodically monitor the kernel and some other device components – the warranty bit is set if either of them detects a compromise [53]. The warranty bit value is used in the decryption of some TEE-protected encryption keys, making data encrypted them inaccessible after the bit is set [5, p. 14].

Protocol. To start the attestation process, the relying party first asks for a nonce from Samsung's Attestation Server. It then invokes an API to request attestation from an application running on the device, using the provided nonce. The app invokes the *Knox Attestation Agent*, which forwards the request to Samsung's Keymaster TA. Confusingly, Samsung documentation calls the resulting object attestation results, although according to the RATS architecture [13], it should be called attestation evidence. The evidence is signed using the device's SAK, which is protected and operated by Knox Vault. The evidence is transmitted by the Attestation Agent over a TLS session to the relying party, which forwards it to Samsung Attestation Service. The service checks that the SAK certificate validates against the Samsung root certificate [5, pp. 25–27]. The service returns an attestation results, called verdict.

Results. The attestation result is a JSON-encoded object, which contains information about the attested application, including the package name and signature. In addition, it specifies whether the device has been rooted, whether the device ID passed the integrity checks, and whether the bootloader has been unlocked [54]. Fig. 6a shows an example.

IX. Android Play Integrity

Google's Android is the market-leading smartphone operating system. Since smartphone applications are often used in security-sensitive contexts such as online banking or two-factor user authentication, there is a need for the applications and their backends to verify the integrity of the platform and the application code. In particular, applications and their backends often wish to check whether the device is rooted or not, i.e. whether it is running custom system software. Traditional methods, such as checking for the presence of the `su` binary, do not work if the device is under the control of an attacker due to a compromised operating system. Also, such checks must be implemented in the application code, which is hard for developers to get right. To make integrity checks easier for developers, Google provides the Play Integrity API (previously called SafetyNet Attestation API, which is now deprecated) [3]. Although it is possible to use Play Integrity on devices without hardware-backed security, using a software-only trustworthy mechanism, this does not provide a satisfactory level of security. Here, we assume that Play Integrity is used on a device with a TEE.

Trustworthy mechanism. In Play Integrity attestation, the trustworthy mechanism consists of two parts. The hardware-backed part generates the key attestation evidence – a chain of X.509 certificates ending a trusted root certificate – and is implemented using the TEE, secure storage and the Keymaster TA. The evidence signing key is required to be shared by enough devices so that it cannot be used as a device identifier [55, Sec 9.11]. The second part is REE-based, consisting of lightweight Java-based checks such as inspecting Android system properties, complemented with an obfuscated, closed-source monitor called DroidGuard. The inner workings of DroidGuard have been reverse engineered by Romain Thomas [56]. The current findings indicate that DroidGuard is a virtual machine running obfuscated interpreted code. Separately for each attestation, the latest version of DroidGuard is downloaded from Google's server, and provisioned with unique

code [56, p. 5]. DroidGuard attempts to detect system-level tampering such as the presence of rooting software like Magisk and KingRoot, mostly via filesystem based checks [56, p. 13].

Coverage. The key attestation evidence generated by the Keymaster TA contains claims on both the platform and application, including, for example, bootloader status, OS version and the application name and hash of the application's signature [24, p. 317]. Most of the claims in the definition of the key attestation data type are optional [57] so coverage varies per device vendor. However, the Android Compatibility Definition requires the secure boot status to be stored in a tamper-proof location and the private component of the attested key to be protected by a TEE [55], so it seems safe to assume that these properties are always attested. The key attestation claims are gathered by the Keymaster TA, and combined with the DroidGuard-provided REE-based claims to form the final attestation evidence.

Protocol. The Play Integrity attestation is started by the application's backend service, which must provide a challenge to the application. The challenge consists of a nonce, and optional suffix such as a session or request identifier that allows binding the attestation to a particular context. The application invokes the Play Integrity API with the challenge and the developer key. DroidGuard is then started, and the Keymaster TA is invoked to collect the measurements and combine them into attestation evidence. The Play Integrity API sends the combined evidence to the Google Play server, which validates the evidence and returns an attestation result.

Results. The result, called integrity verdict, is an encrypted and integrity-protected JSON Web Token (JWT). As example of a decrypted result is shown in Fig. 6b. The most important fields are `appRecognitionVerdict`, which encodes whether the app and its certificate match the versions available on Google Play, and `deviceRecognitionVerdict`, which indicates whether the device passed integrity checks and whether the device meets Android compatibility requirements. How exactly Google's verifier performs these evaluations based on the evidence is not disclosed.

X. Huawei SysIntegrity

Huawei's SysIntegrity [58], included in its EMUI operating system since version 3.0 as part of the Safety Detect solution, closely resembles Play Integrity, both in terms of the API and the attestation process.

Trustworthy mechanism. Hardware-backing is provided by a TrustZone-based TEE, *iTrustee*, previously *TrustedCore* [59]. Claims are collected both during boot and dynamically at runtime and stored in the TEE. Integrity measurements are provided by the EMUI Integrity Measurement Architecture (EIMA) [60, p. 19–20]. A TA in iTrustee generates and signs the evidence [61, p. 42]. It is not documented whether REE-based processes are also involved in claim collection.

Coverage. The attestation claims include various boot and runtime integrity measurements and identifiers, such as device ID, version numbers of system components, and the Huawei ID of the user is included [62].

Protocol. To start the attestation process, an application invokes the SysIntegrity API in the Safety Detect SDK. The calling application shall provide a 16–66 octet nonce, which the documentation recommends to be derived from the data the application sends to its backend server. In addition, the request contains an App ID and a signature algorithm identifier. Once collected and signed, the SDK securely transmits the evidence over a TLS channel to Huawei's HMS server. The server appraises the evidence, generates and signs an attestation result, and returns it to the SDK. The application can request the results from the SDK via an API, and send them to the application's backend server as a proof of integrity.

Results. The attestation result is JSON-encoded and signed using the JWS format. The JWS signature can be verified using an embedded X.509 certificate chain, rooted at a Huawei Root CA certificate. The results are illustrated in Fig. 6c. The `nonce` field in the attestation result shall match the nonce in the original application challenge. The boolean field named `basicIntegrity` is set to true if if system integrity has not been violated (the device rooted or unlocked). The array `detail` can then be consulted for further info. Here, once can find values such as `unlocked` (indicating an unlocked bootloader), `Root` (indicating the device is rooted), `Emulator` (indicates thath the EMUI OS is running in an emulator) and `Attack` (which indicates that the device has been attacked). As with Play Integrity and Knox attestation, the inner workings of the verifier are not disclosed.

XI. Apple App Attest

Apple's AppAttest [63] supported from iOS 14 onwards, provides developers with a mechanism to validate the integrity of their iOS application when in use, and communicating with a service provider.

Trustworthy mechanism. The SEP provides a hardware-backed root of trust, and is used for secure boot and key attestation. The operating system is involved in attestation and assumed be the trustworthy [64].

Coverage. According to Apple's documentation, the following properties are attested: (1) Integrity of the iOS application code, (2) whether the application is running on a genuine Apple device and (3) authenticity of the data exchanged between iOS app and the service. However, the exacts of attestation evidence that is transmitted to Apple's verifier is not disclosed.

Protocol. The installed application uses the AppAttest framework to generate a key pair in the SEP that binds the application with an user account on the device, associates the key with a keyID, and sends key attestation evidence to an Apple server. This application credential certificate follows the web authentication specification [65]. The key will remain valid for all future application updates but will not survive activities such as re-installation, device migration or restoration of device from backups [63]. When the application subsequently communicates with the service, the iOS app requests a one time challenge from the server, combines this challenge with whatever data the application wants to attest to into a hash, and calls the AppAttest service with the hash and Key ID. The AppAttest service returns an assertion token signed by the application's key and containing the app's Key ID, the provided authentication data and counter values that the server can use for verification.

TABLE I. ATTESTATION COVERAGE IN THE SURVEYED PLATFORM ATTESTATION SCHEMES

	App		OS		Boot		Identity		Context binding		
Scheme	**State**	**Code**	**State**	**Code**	**Images**	**History**	**Device**	**User**	**Time**	**Channel**	**Operation**
Windows DHA	N	N	note 1	Y	Y	N	N	N	Y	N	N
Apple MDA	N	N	N	N	N	N	Y	N	Y	N	Y
Knox Attestation	N	Y	note 2	Y	Y	Y	Y	N	Y	N	Y
Play Integrity	note 3	Y	note 3	Y	Y	N	N	N	Y	N	Y
SysIntegrity	N	Y	Y	Y	Y	N	Y	Y	Y	N	note 4
App Attest	note 5	Y	N	N	N	N	N	note 6	N	N	N
Device check	note 7	N	N	N	N	N	N	N	N	N	N

Note 1: In Windows DHA, part of the OS runtime state is implicitly attested via proof that Windows Defender has been loaded.
Note 2: Runtime kernel measurements are collected in Knox by RKP, but it is unclear if they are attested.
Note 3: Via software-based REE monitoring; runtime claims are not TEE-backed.
Note 4: A context string is recommended, but not required.
Note 5: Unclear if application's runtime state is attested. Note 6: App ID is recorded.
Note 7: App state can be recorded in the DeviceCheck bits.

Results. The attestation token contains both *authenticator data* and an *attestation statement* in a proprietary Apple format. *Authenticator data* consists of application identification metrics such as application ID digest, a counter denoting times the attested key has been used to sign assertions and if the attested key belongs to development or production environment. The attestation statement includes an application credential certificate along with a chain of intermediate ones that can be validated against Apple App Attest Root Certificate from Apple PKI. After successful validation, the backend stores the verified public key from credCert (application credential certificate) and associate it with the user for the specific device. This key is used validate any assertions that follow later. The attestation statement also contains receipts, which the backend server can later submit to an Apple server to request a fraud assessment metric, for example, to examine the number of attested keys in use for a specific device.

Apple introduced the DeviceCheck [66] feature in iOS 11 to help developers detect on-device frauds. Activities such as game cheating and illegitimate access to premium content often cuts into the revenue of developers. The DeviceCheck framework provides interfaces both on device and server side that developers can use to verify whether their iOS applications are used legitimately. To accomplish this, Apple allows two bits of information per device and application, along with a timestamp, to be stored in Apple's back-end servers, from where they are provided to application developers on request. It is up to developers to decide on how to use these bits. The bits can be used, for example, to mark various app life-cycle states, or record app re-installation counts or promotional offer use. The bits and timestamp maintain user privacy while giving developers a tool to monitor application use "in the wild".

DeviceCheck and AppAttest are complementary frameworks that work independently: the former aims to mitigate fraud and the latter to safeguard application integrity. Apple recommends developers to integrate both frameworks in their business logic [66]. According to Apple [64], a compromised iOS application running on genuine Apple hardware cannot create valid assertions. No guarantees are given if the attacker has compromised the operating system. User privacy is a design goal: the produced attestation objects do not contain device identifiers or information that would allow client applications to track the users via device tracking [67].

TABLE II. SUMMARY OF THE SURVEYED PLATFORM ATTESTATION SCHEMES

Method	**Trustworthy mechanism**	**Trust anchors**	**Interaction**	**Result format**
Windows DHA	TPM or Pluton, PBL	EK, AIK	Passport	XML/ JWT
Apple MDA	SEP	SEP UID key	Passport	X.509 certificate
Knox Attestation	TEE, Knox Vault, PBL	DUHK, SAK, SSBK pub	Background	JSON
Play Integrity	TEE, PBL	Device and TEE keys	Passport	JSON
SysIntegrity	TEE (iTrustee), PBL	HUK	Passport	JSON + JWS
App Attest	SEP	SEP UID key	Passport	custom

XII. SUMMARY

The coverage of attestation evidence of the surveyed schemes is summarized in Table I. We find that boot time events are well covered by most schemes. System state attestation is however less supported. For example, Windows DHA focuses on attesting boot time events, and only implicitly covers the run-time, by attesting the launch of an anti-malware solution and that no root-level malware is present which could affect its operation. Play Integrity includes run-time measurements, but these are performed by an REE-based process and thus not backed by the hardware-based trustworthiness mechanism. Revealing user or device identity to the verifier can be considered detrimental for user privacy. Fortunately, verifiers in the surveyed schemes are operated by established, well-known OEM and OS vendors. Evidence is also transmitted to the verifier either in encrypted form or over a secure channel such as a TLS session, protecting it from eavesdroppers.

We also note that there is still a lot of room for improvement in context binding of attestation evidence. On one hand, evidence must be bound to a temporal context, such as a timestamp or a nonce, to prevent attackers using the evidence in a replay attack. This is well accounted for in the surveyed

schemes. On the other hand, channel binding is required to prevent relay attacks, i.e., where an attacker presents evidence from a valid platform to attest a compromised device over a separate channel [68], [69]. This binding is in practice absent from the surveyed mechanisms. In addition to time and channel binding, we also stress the need to bind attestation evidence to the particular resource or operation being requested – trust should never be regarded as absolute. In the current schemes, evidence generation is a black box, and attacks where evidence meant for evaluating trust in e.g., a low-security context but used as evidence for a high-secure operation should be prevented.

Referring to Table II, we see that the surveyed schemes can be categorized into three groups: TPM, TEE and enclave based ones. We find that most of the surveyed schemes follow the passport model of interaction, with only Knox using the background check model. Which model should be preferred is not security-critical and the choice can be made mostly on implementation related grounds. However, a passport is effectively a cached result, and with this some form of replay protection should be used. The passport model may leak less privacy-sensitive claims to the relying party, since such claims can be stripped out by the verifier.

XIII. Conclusions

The survey of deployed platform attestation solutions for consumer devices shows that verification is indeed increasingly outsourced to a service operated by the device OEM or OS vendor, for convenience but also presumably to hide some of the brittleness of detailed OS measurements, metrics and state information that likely changes over time when OSs evolve and new versions appear. Privacy for the communication between attestor and attestee is today considered, and collected evidence is vetted against the likelihood that it might end up indentifying a single user, his device use or both. Beyond the selection of data and protocol formats, the device health solutions of the Apple, Android and Harmony ecosystems are surprisingly similar to each other, only Windows DHA stands out being based on TPM and measured boot. Interesting new developments are the Apple DeviceCheck and AppAttest frameworks which tweak attestation in new ways – here we are not any more only attesting the platform, but rather use attestation as a tool to support the application framework, and more specifically to satisfy existing needs of application developers. Maybe, when device health attestation becomes a commonplace requirement for a cloud service to serve a consumer customer and his device, we will see more of these business-oriented systems that leverage platform attestation as a core function, but apply it for a specific purpose in the user domain.

References

[1] P. Vachon, "The identity in everyone's pocket," *Communications of the ACM*, vol. 64, pp. 46–55, Jan. 2021.

[2] A. Shakevsky, E. Ronen, and A. Wool, "Trust dies in darkness: Shedding light on Samsung's TrustZone Keymaster design," in *Proceedings of the 31st USENIX Security Symposium*, Aug. 2022.

[3] M. Ibrahim, A. Imran, and A. Bianchi, "SafetyNOT: On the usage of the SafetyNet attestation API in Android," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications and Services*, ser. MobiSys '21. New York, NY, USA: ACM, Jun. 2021, pp. 150–162.

[4] "Windows security / Control the health of Windows 10-based devices," https://web.archive.org/web/20230205141009/https://learn.microsoft.com/en-us/windows/security/threat-protection/protect-high-value-assets-by-controlling-the-health-of-windows-10-based-devices, 2022.

[5] "Samsung Knox white paper v2.1," https://web.archive.org/web/20230208155147/https://image-us.samsung.com/SamsungUS/samsungbusiness/solutions/topics/iot/071421/Knox-Whitepaper-v1.5-20210709.pdf, 2021.

[6] M. M. Yamin and B. Katt, "Mobile device management (MDM) technologies, issues and challenges," in *Proceedings of the 3rd International Conference on Cryptography, Security and Privacy*, ser. ICCSP '19. New York, NY, USA: ACM, Jan. 2019, pp. 143–147.

[7] H. Batool and A. Masood, "Enterprise mobile device management requirements and features," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, Jul. 2020, pp. 109–114.

[8] A. Lima, B. Sousa, T. Cruz, and P. Simões, "Security for mobile device assets: A survey," in *Proceedings of the 12th International Conference on Cyber Warfare and Security (ICCWS 2017)*. Academic Conferences Ltd, Mar. 2017.

[9] 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.

[10] J. Frazelle, "Securing the boot process," *Communications of the ACM*, vol. 63, pp. 38–42, Mar. 2020.

[11] A. Niemi, S. Sovio, and J.-E. Ekberg, "Towards interoperable enclave attestation: Learnings from decades of academic work," in *2022 31st Conference of Open Innovations Association (FRUCT)*. IEEE, Apr. 2022, pp. 189–200.

[12] 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 *Proceedings of the 20th International Conference on Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 399–418.

[13] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, "Remote attestation procedures (RATS) architecture," RFC 9334, Jan. 2023.

[14] 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.

[15] 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.

[16] M. Sardar, T. Fossati, and S. Frost, "SoK: Attestation in confidential computing," *ResearchGate pre-print*, Jan. 2023.

[17] T. Müller and F. C. Freiling, "A systematic assessment of the security of full disk encryption," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 5, pp. 491–503, 2015.

[18] L. Gunn, N. Asokan, J.-E. Ekberg, H. Liljestrand, V. Nayani, and T. Nyman, "Hardware platform security for mobile devices," *Foundations and Trends in Privacy and Security*, vol. 3, pp. 214–394, Jun. 2022.

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

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

[21] V. Ushakov, S. Sovio, Q. Qi, V. Nayani, P. Ginzboorg, and J.-E. Ekberg, "Trusted hart for mobile RISC-V security," in *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, dec 2022, pp. 1587–1596.

[22] K. Kostiainen, A. Dmitrienko, J.-E. Ekberg, A.-R. Sadeghi, and N. Asokan, "Key attestation from trusted execution environments," in *Trust and Trustworthy Computing*, ser. Lecture Notes in Computer Science, vol 6101. Berlin, Heidelberg: Springer, 2010, pp. 30–46.

[23] "Android 7.0 for developers - key attestation," https://developer.android.com/about/versions/nougat/android-7.0?hl=en#key_attestation, 2016.

[24] 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.

[25] M. Bursell, *Trust in Computer Systems and the Cloud*. Hoboken, New Jersey, USA: John Wiley & Sons, 2022.

[26] GlobalPaltform, "Root of trust definitions and requirements – public release v1.1.1," GlobalPlatform, Tech. Rep. GP_REQ_025, 2022.

[27] J. Szefer, *Principles of Secure Processor Architecture Design*. Morgan & Claypool Publishers, 2019.

[28] 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.

[29] K. Vrancken and F. Piessens, "Do we need consumer-side enclaved execution?" in *Proceedings of the 5th Workshop on System Software for Trusted Execution (SysTEX'22 Workshop)*. New York, USA: ACM, 2022.

[30] K. Kostiainen, A. Dhar, and S. Capkun, "Dedicated security chips in the age of secure enclaves," *IEEE Security & Privacy*, vol. 18, pp. 38–46, Sep. 2020.

[31] M. Mattioli, "PCs take a page from Xbox with Pluton," *IEEE Micro*, vol. 41, pp. 125–128, 2021.

[32] "Microsoft Pluton security processor," https://web.archive.org/web/20230224120614/https://learn.microsoft.com/en-us/windows/security/information-protection/pluton/microsoft-pluton-security-processor, 2022.

[33] *DICE Attestation Architecture*, Trusted Computing Group, Mar. 2021, version 1.0, revision 0.23.

[34] T. Mandt, M. Solnik, and D. Wang, "Demystifying the Secure Enclave Processor," in *Black Hat USA*, 2016.

[35] J. D. Osborn and D. C. Challener, "Trusted platform module evolution," *John Hopkins APL Technical Digest*, vol. 32, pp. 536–543, 2013.

[36] "White paper: AMD Ryzen PRO 5000 series mobile processor security features," https://www.amd.com/system/files/documents/amd-security-white-paper.pdf, 2021.

[37] J. Ye, "How a banned encryption chip is stopping China from running Windows 11, for now," *South China Morning Post*, Oct. 2021.

[38] A. R. Bertels, R. E. Bell, and B. K. Eames, "Emulating the android boot process," Sandia National Laboratories, Albuquerque, New Mexico, USA, Tech. Rep. SAND2022-13571, Oct. 2022.

[39] A. Carroll, M. Juarez, J. Polk, and T. Leininger, "Microsoft palladium: A business overview," *Microsoft Content Security Business Unit*, vol. 5, 2002.

[40] "Using the Windows 8 platform crypto provider and associated TPM functionality," https://tinyurl.com/3794ndph, Microsoft.

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

[42] "Compatibility cookbook for Windows / Measured boot," https://web.archive.org/web/20230205111217/https://learn.microsoft.com/en-us/windows/win32/w8cookbook/measured-boot, 2021.

[43] "HealthAttestation CSP," https://learn.microsoft.com/en-us/windows/client-management/mdm/healthattestation-csp, Microsoft, 2023.

[44] "MS-DHA: Device health attestation protocol," https://web.archive.org/web/20221207003013/https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-DHA/[MS-DHA.pdf].

[45] "Microsoft Azure Attestation / Attestation policy / Claim rule grammar," https://web.archive.org/web/20230208153416/https://learn.microsoft.com/en-us/azure/attestation/claim-rule-grammar, 2022.

[46] "Windows Hello for Business overview," https://learn.microsoft.com/en-us/windows/security/identity-protection/hello-for-business/hello-overview, Microsoft.

[47] "Windows Hello for Business cloud trust and kdc proxy," https://cloudbrothers.info/en/windows-business-cloud-trust-kdc-proxy/, Cloudbrothers.

[48] "Managed device attestation for Apple devices," https://support.apple.com/guide/deployment/managed-device-attestation-dep28afbde6a/web, 2022.

[49] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," RFC 5280, May 2008.

[50] H. Slatman, "Managed device attestation: ACME as the bottom turtle in mobile device management," *Smallstep blog*, 2022.

[51] "Device management profile / ACMECertificate," https://developer.apple.com/documentation/devicemanagement/acmecertificate, 2022.

[52] B. Weeks, "Automated certificate management environment (ACME) device attestation extension," https://datatracker.ietf.org/doc/draft-acme-device-attest/00/, Dec. 2022.

[53] A. Aldoseri, T. Clothia, J. Moreira, and D. Oswald, "Symbolic modelling of remote attestation protocols for device and app integrity on Android," in *Asia CCS '23: Proceedings of the 2023 ACM on Asia Conference on Computer and Communication Security*. ACM, 2023, to appear.

[54] "Knox attestation API reference (v3.0)," https://docs.samsungknox.com/devref/knox-attestation/index.htm#tag/Attestation/paths/~1attestations/get, 2022.

[55] "Android 13 compatibility definition," https://source.android.com/docs/compatibility/13/android-13, Google Inc., 2023.

[56] R. Thomas, "DroidGuard: A deep dive into SafetyNet," in *SSTIC22: Symposium sur la sécurité des technologies de l'information et des communications*. Cesson-Sévigné, France: Association STIC, Jun. 2022.

[57] "AOSP / Docs / Security / Key and ID attestation," https://source.android.com/docs/security/features/keystore/attestation, 2023.

[58] Huawei, "Security - Safety Detect - guides - SysIntegrity API," https://developer.huawei.com/consumer/en/doc/development/Security-Guides/dysintegritydevelopment-0000001050156331, 2022.

[59] M. Busch, J. Westphal, and T. Müller, "Unearthing the TrustedCore: a critical review on Huawei's trusted execution environment," in *14th USENIX Workshop on Offensive Technologies*. Boston, MA, USA: USENIX Association, Aug. 2020.

[60] "EMUI 11.0 security technical white paper," https://consumer-img.huawei.com/content/dam/huawei-cbg-site/common/campaign/privacy/whitepaper/emui_11.0_security_technical_white_paper_v1.0.pdf, Huawei, 2020.

[61] "Huawei mobile services (HMS) security technical white paper v.2.0," https://consumer-img.huawei.com/content/dam/huawei-cbg-site/common/campaign/privacy/whitepaper/huawei-mobile-services-(hms)-security-technical-white-paper-v2.0.pdf, Huawei, 2021.

[62] "Security / Safety Detect / Guides / SDK privacy and security statement," https://developer.huawei.com/consumer/en/doc/development/Security-Guides/sdk-data-security-0000001050156339, Huawei, 2023.

[63] "Establishing your app's integrity, Apple developer documentation," https://developer.apple.com/documentation/devicecheck/establishing_your_app_s_integrity, Apple Inc, 2022.

[64] "Assessing fraud risk, apple developer documentation," https://developer.apple.com/documentation/devicecheck/assessing_fraud_risk, Apple Inc, 2022.

[65] "Web authentication: An API for accessing public key credentials, level 2, W3C documentation," https://www.w3.org/TR/2021/REC-webauthn-2-20210408/, W3C, 8 April 2021.

[66] "DeviceCheck, Apple developer documentation," https://developer.apple.com/documentation/devicecheck, Apple Inc, 2022.

[67] "Mitigate fraud with App Attest and DeviceCheck, transcript, WWDC21," https://developer.apple.com/videos/play/wwdc2021/10244, Apple, 2021.

[68] 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.

[69] 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.