1. INTRODUCTION

This paper continues a trail of ideas (Schmidt, Leicher, & Cha, 2010) on trusted technical systems, emphasising the aspect of communicating trustworthiness between systems. A critical review of traditional notions of trust led us to a synthetic definition of trust in technical systems, which can be effective in applications. We call this the operational interpretation of trust to the relations and interactions between technical systems and between technical systems and human beings:

An entity can be trusted if it predictably and observably behaves in the expected manner for the intended purpose

This is essentially also a synthesis of the meanings that for instance the standardisation organisations Trusted Computing Group (TCG) and the International Standardisation Organisation (ISO) attribute to trust, cf., Pearson (2002b). The operational interpretation, which is actually rooted in physicists’ prevalent understanding of quantum systems (Haag, 1992; Lamb, 1969, 2001), has three salient features:

ABSTRACT

Computing platforms are approaching the era of truly distributed and mobile systems. For such large scale deployments of partly autonomously communicating and connecting network elements, trust issues acquire new qualities. Remote establishment of trust and an enabling architecture to manage distributed network elements remotely become essential. Following the authors’ previous analysis on trust establishment, this paper presents base concepts for platform validation and management, with scalable trust properties and flexible security. The presentation is set in context of machine-to-machine communication and intelligent gateways in mobile networks.

Keywords: IT Security, Remote Management, Trust, Trusted Computing, Validation

DOI: 10.4018/jdtis.2010040101

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
Predictability designates *a priori* knowledge about a system that can be used to a) assess the risk incurred in interacting with that system, and b) allow obtaining knowledge about the system during the interaction by reasoning on observations. 

Observability specifies the means by, and extent to which knowledge about a system can be gained in interactions. It is closely linked to predictability, in that observations, together with predictions, yield further knowledge on a system’s state, properties, and, by that, its future behaviour.

Contextuality designates information delineating the scope of interactions with the system in which predictions hold and observations can be made.

The three properties allow, at least in principle, a mapping between the socio-economic concept of trust and technical concepts. Taken together, they allow an assessment of the trustworthiness of an entity, or reciprocally, the risk it poses to a trustier. The operational understanding of trust is contrasted with traditional notions of enforcement, which strives to reduce risks by behavioural control, rather than *a priori* assessment of a trusted system (see Schmidt, Leicher, & Cha, 2010 for a discussion of nomenclature).

While our mentioned paper explores the conceptual foundations of trusted systems, we now continue and extend those concepts to explore the operational cycle of remotely validated and managed Trusted System (TS) more deeply. We envisage one major, application domain for the concepts presented here: Machine-to-Machine (M2M) Communication – situations in which machines and networks interconnect in a largely autonomous fashion. M2M provides a paradigm for the security threats encountered in future communication networks (Cha, Shah, Schmidt, Leicher, & Meyerstein, 2009).

The core concept we provide in this paper is that of combined platform validation and management (PVM) of a TS by a network. Section 2 introduces the necessary terminology and background from (Schmidt, Leicher, & Cha, 2010). The third, main section details the network architecture of PVM in a generic way. The context is set by a TS accessing a mobile (next generation) communication network. Section 3 also shows the main process flows of PVM between the entities and sketches the preferred ways to solve technical problems such as synchronisation and how to fulfil essential security requirements. The fourth section highlights a central performance question for PVM: How to cater in a scalable way for myriads of smart, connected devices. We explore how this issue could potentially be resolved by a hierarchical organisation of PVM processes, involving gateways to unburden the core network (CN). Section 5 concludes the paper.

## 2. Trust Technology for Platform Validation and Management

A system needs certain security-relevant elements and capabilities so that it can be operationally trusted concepts which emerged in the context of military applications (Department of Defense, 1985). This context can be viewed as paradigmatic for operational trust – they need to securely operate in situations where any kind of external enforcement may fail and a fallback to inherently trustworthy functionality is a core requirement. In this section, we introduce the basic functional building blocks of technology, which have been developed to establish trust into systems, and which are later needed in the development of a platform validation and management architecture. The bulk of the terms used here is directly taken, or slightly abstracted, from the TCG specifications. However, we attempt to provide a unified terminology, taking literature background and current research into account.

### 2.1 Trusted Systems

A hardware security anchor is a part of the system which is protected against unauthorized access by hardware measures, known to be secure enough for the intended purpose to
effectively mitigate attack. It bears the **Root of Trust (RoT)**, which is an abstract system element enabling

Securing the internal system operation, and exposing properties and/or the identity (individually or as a member of a group such as make and model) of the system to external entities in a secure and authentic way.

A system can contain more than one RoT for distinct purposes. Typical realizations of RoTs are asymmetric key pairs together with digital certificates of a trusted third party (TTP) for them. Also, the symmetric secrets of Subscriber Identification Module (SIM) cards in cellular networks may be viewed as RoTs for the closed, trusted system embodied by the SIM card.

Secondly, functional building blocks in a system that are assumed to be trusted, i.e., to behave in a well-defined manner for the intended purpose, form the **Trusted Computing Base (TCB)** of the system. The TCB comprises such components of a system which cannot be examined for their operational trust properties when the system is deployed in the field and during operation, but only by out-of-band processes like compliance and conformance testing, and certification. This kind of certification is usually carried out by an independent evaluator, for instance on behalf of the manufacturer of a certain technical element of the TCB or the TCB as a whole, according to established security evaluation standards such as Common Criteria (2009). For such a certification to be useful, the TCB, respectively, its elements need to be endowed with information identifying them as such certified pieces of technology.

A system equipped with defined security anchor, RoTs, and TCB is called a **Trusted System (TS)**. Below, various capabilities, processes, and architectural elements, summarised under the term **trusted resources (TRs)**, of TS are described. Two kinds of TRs must be generally distinguished: First, TRs which belong to the TCB, and second, TRs which are outside the TCB. Genuine examples for the latter are trusted parts of the operating system, and trusted applications which build on the TCB by using its capabilities. While assertions about the trustworthiness of the TR in the TCB depend on the defined security of the TCB, the trustworthiness of the other TRs can, at most, be derived from that of the TCB. In such a case, the TCB must provide certain internal TRs that allow extension of the trust boundary, i.e., the totality of components of a TS that are considered trustworthy in a given context, to the TRs outside the TCB, for instance authenticated or secure boot as described below. TRs within the TCB often share the same hardware protection with the RoT; for instance, reside on the same tamper-resistant chip. TRs outside the TCB may be realised as logical units in software. Note that the trust boundaries, especially involving TRs that are outside of the TCB, may be ephemeral. They may exist for some time for certain purposes, and then may cease to exist afterwards.

A general model process to extend the trust boundary beyond the TCB is **verification**. Verification is the process to include a new component in the trust boundary, and is realised by a TR implementing the verification process, called **verification entity**, or **verifier**, to distinguish it from the process of **validation** of a TS by an **external** entity, the **validator**. Verification can come in two flavours. First, and as a simple option, the verifier measures a new component at the time of its initialisation. That is, the component, its status and configuration are uniquely identified. The result of this measurement is then stored. As an extension of this, the verifier can compare the measurements with **reference values** and decide whether or not to extend the trust boundary. That is, the verifier makes and enforces a policy decision. From the operational viewpoint, verification corresponds to predictability of the TS, as it can be assumed to be in a certain, pre-defined state after the verification process is completed. Validation, on the other hand, makes this property observable and therefore trustworthy. It means that a **reporting entity** transfers the results of verification to another party. The third, intermediate step performed by the reporting entity is that of **attestation**. Attestation is a consequence of verification and a precondition.
for validation. Attestation vouches for the accuracy of measurement information, such that a relying party – the validator – can use it to decide whether it trusts the remote TS. For this, the measurement information must be bound to the specific TS and then be transmitted in a way that protects its authenticity. Verification, attestation, and validation are core concepts for operational trust which are tied to the lifecycle of a TS. This is detailed below.

As further TRs, a TS may have cryptographic capabilities and secure storage within the TCB. As storage space there is commonly limited, general methods to extend secure storage have been envisaged, e.g., within the TCG standards (TCG, 2007a). The secure storage within the TCB contains a RoT for Storage (RTS) for that, e.g., a cryptographic key. The RTS is then used to protect data outside the TCB, e.g., by encrypting them. A TS often has authorisation functionality incorporated to protect access to TRs. As an example, in the Trusted Platform Module (TPM), authorisation is operated by storing 160 Bit secrets, e.g., password digests, within the hardware protected storage inside the TCB, namely the TPM chip. Similar concepts and possible application scenarios are discussed in the requirements of the Open Mobile Terminal Platform (OMTP), Advanced Trusted Environment TR1 (OMTP, 2009).

2.2 Establishment of Trust

The establishment of operational trust between TS rests on the controlled exchange of information to enable observability and the pre-establishment of predictability. The latter can only be done outside of the TS, and is based on separation of duties (Botha and Eloff, 2001). Separation of duties usually refers to duties on enforcement. But there is a natural relationship to trust. A relying party can delegate the enforcement to the other system only if it is operationally trustworthy.

Figure 1 shows a generic model for external entities providing organisational assurance to TS. The RoTs of a TS undergo security evaluation during design and development. This is performed by an independent authority which, upon successful evaluation, issues certificates of security to the manufacturer of the security critical components. This process may also comprise other TRs in the TCB, and involve different certification authorities. To ensure the homogeneous quality of evaluation, and the different certification authorities, they are assessed and certified by accreditation authorities, for instance, private entities with state permits, or para-statal entities. Accreditation authorities can also provide bridging information between certification authorities.

Certification authorities or technical entities informed by them, issue credentials to TS which are used by the TRs. These credentials are certificates in the sense that they are verifiable for integrity and provenance. A prime example is the Endorsement Key (EK) certificate issued to the TPM’s main RoT (the EK) by its manufacturer, as well as the Platform Certificate and other components’ certificates. These credentials and secrets derived from them by cryptographic means are used in the interaction with external entities. Secrets and credentials with trust inherited from the TS credentials are essential for operating system and trusted applications to build security associations, that is, channels which provide authentication, confidentiality, and integrity of communication. On top of security associations, applications within the extended trust boundary can build secure communication channels with well defined operational trust properties.

Trust establishment can hardly be effective without a mediator facilitating the various interactions. One key task of a mediation entity is to issue fundamental statements about the trustworthiness of a TS to another TS or relying party. Most importantly, the mediator identifies the TCB (or selected elements, e.g., the trust anchor) as a trusted and certified, component. To this end, the mediation entity needs to know the certificates issued by the certification entities, verify them when it receives it from a TS, and issue an according assurance statement to a relying party. It should be noted that valida-
tion is impossible without a mediator, if not all TS know the credentials of all other TS. Thus, mediation is, in fact, fundamental for validation. The best-known example for a mediator is the Privacy Certification Authority (PCA) defined in TCG standards. The role of a mediator can extend further than protecting the TS privacy in validation processes. For instance, a mediator can also facilitate subsequent security association and secure communication, similarly to a CA in Public Key Infrastructures (PKI).

Next, we exhibit main building blocks for trust establishment, namely verification, validation, and pertinent data structures.

### 2.3 Verification

Verification is, in essence, a recording and controlling of state changes of a TS to the desired granularity. As such, it must be tightly bound to the operational cycle of the platform on which a TS resides, frominitialisation to shutdown. Therefore, practical verification methods are mostly integrated with the boot process and operational cycle of platforms.

One general method for the internal verification of a TS is authenticated boot, and uses capabilities of the TCB to assess the trustworthiness of loaded or started software or hardware components at the time the TS is initialised, e.g., on power on. Authenticated boot is realised by starting certain functions of the RoT and the TCB before other parts of the TS. These parts operate as a RoT for Measurement (RTM). This means that components that are started or loaded later on, are measured, i.e., they, and their status and configuration after start are uniquely identified, e.g., by forming cryptographic digest values over a (binary) representation of hardware component’s embedded code and loaded programs. According to the specific requirements, the measurement values may be stored in secure storage. Together with data necessary to retrace the system state from them, e.g., software names and versions, they form the Stored Measurement Log (SML) of the TS. On PC platforms, authenticated boot may include all components from the BIOS to the Operating System (OS) loader and the OS itself. According to TCG specifications, TS’ state is measured by a reporting process,

---

*Figure 1. Trust between platforms is mediated by organisational and technical methods*
with the TPM as central authority, receiving measurement values and calculating a unique representation of the state using hash values. For this, the TPM has several protected Platform Configuration Registers (PCRs). Beginning with the system initialisation at power-up, for each loaded or started component a measurement value, e.g., a hash value over the BIOS, is reported to the TPM and stored securely in the SML, using the RTM, and the active PCR is updated by an extend procedure, which means that the measurement value is appended to the current PCR value; a digest value is built over this data, and stored in the PCR. In this way, it is said that a transitive chain of trust is built containing all started and loaded components. As a single PCR stores only one value, it can only provide “footprint-like” integrity validation data. This value allows a validator to verify this chain of trust by recalculating this footprint, only in conjunction with the SML.

Secure boot is an extension of authenticated boot. It is of particular importance for devices like set-top boxes or mobile handsets that necessarily have some stand-alone and offline functional requirements. The common characteristic of devices equipped with secure boot is that they are required to operate in a trustworthy set of states when they are not able to communicate assertions on their trustworthiness to the exterior, e.g., before network access. In secure boot, the TS is equipped with a local verifier (a verification entity) and local enforcer supervising the boot process, which establishes the combination of a Policy Enforcement Point (PEP) and Policy Decision Point (PDP) to control the secure boot process. The local verifier compares measurement values of newly loaded or started components with Reference Integrity Measurement (RIM) values which reside in the TCB, or are protected within the TS by a TR, e.g., they are located in protected storage space, and decide whether they are loaded, resp. started, or not. Thus, the system is ensured to boot into a defined, trustworthy state.

It is important to note a dual aspect of RIMs. On the one hand, they serve the local verification in a secure boot process. For that, they are complemented by a RIM provisioning infrastructure (TCG, 2008a, 2008b) that allows, for instance, updates of measured components, by provisioning of new RIMs to the TS. For an external entity to validate a TS after secure boot, it needs to compare the received event structure with stored RIMs and to verify the associated RIM certificates. Thus, RIMs and RIM certificates play an important role not only in verification, but also in validation and management of a TS.

2.4 Validation

The mentioned separation of duties is also present on the process of validating a TS. Namely, based on the result of verification, the trustworthiness of the system may be assessed and, accordingly, policy decisions can be made in the validation. The separation of tasks in this process between TS and validator leads to three variant categories of validation. The validation therefore relies on the chain of trust built from the RoT of the TS. With integrity measurement and reporting, this trust boundary is then extended. Since the RoTs are immutable parts of every TS, the validation reveals compromised parts of the TS. Depending on the validation method, appropriate actions can be taken in such a case. Before we introduce and compare the different methods for validation, we discuss one common base concept needed for any kind of validation. Validation means the ability to technically assess the state of a system for all security-relevant properties.

A validation process of a TS must be supported by a validation identity which is exhibited to the validator. The validation identity must come directly or indirectly from a RoT, namely the Root of Trust for Reporting (RTR). As was noted before, validation is not possible without a mediator. This validation identity
provider has the task to assert that the holder of the validation identity is a TS. Provisioning of a validation identity is an extension of identity provisioning in identity management (IdM) systems. The provider has to perform checks on credentials of the TS, including some or all TRs in the TCB, to assess if the TS is in a trustworthy state for validation. Furthermore, the provisioning of validation identities must be performed in a secure process, e.g., a security protocol on a dedicated secure channel. In case of remote validation, the validation identity may coincide with a global identity of the TS.

Validation using unique persistent validation identities is critical with regard to security. Validation may occur frequently and indiscriminately to many validators for varied purposes. Though the validation identities used may each not be easily associated to a user identity, they generally allow a tracing of the TS’ behaviour. Using the same validation identity for a group or all TS is not an option to resolve this for security reasons. Such a group identity would be a single point of attack/failure, that is, if one TS of the group is compromised, then all others cannot perform validation any more. The other option is to use ephemeral validation identities generated, for instance, once in each boot cycle, with determined frequency, or generated by the RTR for each validation.

Means to validate a system in its operational phase are different from pre-deployment testing and certification, or formal security proofs on a design. They lead into the realm of trusted systems and trusted computing. Validation procedures can come in three essential types. Remote validation assumes no capabilities to verify the integrity of components internally by the TS, before they are loaded and started, i.e., no secure boot. It is realised by the basic TCG attestation model of remote attestation, which just reports measurements to the validator. Autonomous validation is the other extreme, based on a smart card-like, closed system model, which performs a full internal verification at start-up, or is tamper-protected as a whole. Essentially the whole TS is a TCB, and validation consists in a binary signal, e.g., successful network authentication and attachment by the TS. A broad spectrum of other variants lies in between the range marked by these extremes. We call this spectrum (not a single method) semi-autonomous validation (SAV). One concrete example for a SAV, at least on the level of technical specifications, namely, what the TCG’s Mobile Phone Working Group (MPWG) has specified as secure boot. We argued in the previous paper (Schmidt, Leicher, & Cha, 2010) that SAV has many benefits in terms of a) allowing for a conceptual scaling between implicit trust and explicit validation and thus providing more implementation flexibility, and b) enabling complex tasks of platform management, the core subject of the present paper.

The SAV essential concept is the following. In SAV, the TS’ validity is assessed during verification within itself without depending on external entities, and policy decisions are made during verification. The result of the local verification and required evidence are signalled to the validator, who can make decisions based on the content of the validation messages from the TS. The signalling from TS to validator must be protected to provide authentication, integrity, and confidentiality if desired. A model case for semi-autonomous validation is secure boot, followed by a signalling of the event structure and indication of RIMs to the validator. As opposed to remote validation, the TS is not required to send the full SML to the validator. Furthermore, it is possible to request a re-validation. Specifically, in secure boot, the TS makes decisions at load time of components, while the validator can enforce decisions on the interactions permitted to the TS upon validation, based on the state evidence provided. This design allows, for instance, putting a device into a quarantined network first, which allows it to signal the outcome of the internal validation to the validator. The validator then allows or rejects access to the main network. By scheduling a regular re-validation, or by prescribing events upon whose occurrence re-validation can be initiated, compromised devices can be detected and can be contained in the quarantine network, e.g., to provide remote updates.
2.5 Data Categories for Platform Validation and Management

Basic notions of data transferred between a platform and a validator, and other entities facilitating platform management functions, are reoccurring in the generic descriptions of trust establishment above. These data categories are linked to the processes of verification and validation of a platform. In constructing these data categories, it is important to bear in mind the distinction between the assertions that can be made about TS, and which allow assertion of their trustworthiness in validation, and the assurance provided to these assertions by supplementary data. Following this guiding principle, and according to security-related properties and semantic content, these data can be divided in four main groups. Figure 2 shows the interrelation between their notions.

These four categories of data are created and maintained by a platform and used throughout the processes which embody the central concept of platform validation and management (PVM). The data categories are intrinsic to the validating platform. A different, additional category is the trusted reference data, which is used, to compare validation data to known good values. It may reside at the validator for use in validation of a platform, or inside the platform, where it may be used for secure start-up or be transferred as validation and/or management data to the validator.

2.5.1 Verification Data

Verification data verifiably identifies a platform’s state, or a partial state with a well defined level of assurance. Verification Data is generated internally in a platform in a verification process, for instance during secure start up of the platform, and/or during run-time. The main protection goal for verification data is integrity which needs to be maintained during its generation, i.e., verification, and at least throughout a platform’s operational cycle (e.g., boot cycle). A subordinate goal is freshness. A typical way to define (and raise) the assurance level of verification data is to separate out part of the verification data as protected verification data with a special protection level, e.g., by hardware or cryptographic protection, in which
case we call this storage space a verification data register. Then, verification data requires a protected binding, e.g., cryptographic, to protected verification data. The strength of the binding and the strength of the protection define the assurance level of the verification data. Some assorted realizations of verification data and verification are the following.

The SML contains a sequence of 160-Bit SHA-1 values generated in the authenticated boot process from measurements of loaded components. These hash values uniquely identify the loaded component. The assurance level of SML values depends a) on the security of the RTM which carried out the measurement and b) on the integrity-protection of the SML, for instance by verification data registers, e.g., PCRs.

PCRs of a TPM paradigmatically represent protected verification data registers.

One general method for the internal verification of a TS is authenticated boot, and uses capabilities of the TCB to assess the trustworthiness of loaded or started software or hardware components at the time the TS is initialised, e.g., on power on.Authenticated boot is realised by starting certain functions of the RoT and the TCB before other parts of the TS. These parts operate as a RTM. This means that components that are started or loaded later on, are measured, i.e., they, and their status and configuration after start are uniquely identified, e.g., by forming cryptographic digest values over a (binary) representation of hardware component’s embedded code and loaded programs. According to the specific requirements, the measurement values may be stored in secure storage, e.g., PCRs, which form the protected part of verification data.

Secure boot as an extension of authenticated boot is described in more detail above.

2.5.2 Validation Data

In distinction from verification data, we call validation data the superset of verification data of all data that can be gathered and submitted to the validator, and used to assess the trustworthiness of the state of the platform, in particular by checking it against contained verification data. The process of submission of validation data to the validator, for instance realized as remote attestation according to TCG, and evaluation thereof by the validator, is properly called validation, and it appears in the three fundamental variants described above. Validation data may often comprise verification data such as quoted verification data register (e.g., PCR) values. Validation may, beyond cryptographic verification of verification data, include policy evaluation and triggering of actions by the validator, using for instance
additional management data which may be associated to the validation data.

Similar to verification data, validation data identifies a full or partial system state, but additionally provides more information to make the validation efficient for the validator and determine the level of granularity of the validation result. Typical examples for validation data (beyond verification data) are therefore:

Named system properties, or names and versions of components, their status and parameters.

The platform certificate of a TS, as defined by the TCG and vendor certificates of system components.

Names of TTPs, where further data may be fetched, e.g., certificate chain information.

Component-subcomponent relationships, for instance captured in nested XML data structures, as specified by the TCG Infrastructure Working Group (IWG) (2006) in Trusted Network connect (TNC) (TCG, 2008c) specifications.

A platform validation identity for validation, realized, e.g., by an Attestation Identity Keypair (AIK) key pair.

An important aspect of validation is the binding of validation data to according verification data. This binding, in the spirit of transitive trust chaining, transfers the assurance level of verification data to validation data. Thus, this binding must be verified, for instance by a digital signature, to achieve this. In other words, the notion of validation data is restricted by the requirement of binding. Validation data is only such data which can be verified by the validator, in particular for data integrity, using verification data. Thus, validation data is less arbitrary than general management data defined below.

Remote attestation is the first-best realization of an initial stage of validation, namely the secure transmission of validation data to the validator, signed by the platform. In turn, the best-known example of this binding is the TPM_Quote operation which signs a PCR value with an AIK. Verifying this signature, and recalculating the PCR from the SML and associating the SML measurement values to RIMs of known components which are named in the validation data, the validator can verify that the named components are the one which the platform has measured during authenticated boot, for instance.

2.5.3 Management Data

Management data comprises and is supplementary to validation data. They add expressiveness to the other data specifically for management of a platform based on validation data and results. The binding of management data to validation data is logical, that is, elements of management data link symbolically to associated elements in the validation data, or vice versa. The trustworthiness of management data (for instance by its source) needs to be assessed separately from validation of the platform, in particular if management data comes from a TTP.

Typical examples of management data are:

- Policies which infer management actions from validation results.
- Places, where code updates can be fetched from.
- Incident reporting data.
- User notification data.
- Service credentials provided to the validating platform conditioned on validation results.

2.5.4 Trusted Reference Data

Trusted reference data is used to compare validation data to known good values. Those values, which constitute trusted reference data, are called Trusted Reference Values (TRV). Their best-known examples are RIMs, as specified in the TCG’s MPWG specifications. They can be used genuinely, a) by the platform itself in secure start-up, to ensure that only components whose measurements conform to a TRV are started, or b) by the validator, to compare validation data to known good values and thereby assess the platform state in validation.

Trusted reference data becomes trusted through certain security assertions about it, which are verifiable by the validator or the
agent using the TRV in question. Such verifiable assertions can be realized for instance by digital certificates issued by a TTP, giving rise, in the concrete example, to the so-called RIM certificates. The trust assertions of trusted reference data may also contain additional information for instance about external evaluation (e.g. according to a Common Criteria EAL) of a component or platform. We generically use the term TRV certificates for such assertions.

3. PLATFORM VALIDATION AND MANAGEMENT

We now come to the method and architecture for PVM. To provide a concrete, application-oriented context, we consider PVM in the context of attachment of a device to a next generation mobile network, the architecture of which is given by 3GPP and evolving ETSI standards (3GPP, 2010a, 2010b; ETSI, 2010a, 2010b), in particular the high-level architectures for connection of femtocells, called Home (enhanced) NodeBs, or H(e)NBs, and M2M equipment (M2ME). This context is concrete enough to elucidate important practical problems but still generic enough to let us develop a PVM architecture with high generality, for instance providing some essential network entities, and applicability beyond the set context.

The reader will note that the PVM procedures have some overlap with TCG’s TNC concepts. PVM differs in three essential ways from TNC: a) PVM considers the reaction and remediation following platform validation during network attachment in a unified, dynamical process, while TNC focuses on conditioning network access on validation results, b) PVM provides a more general picture than TNC, based on essential parts of a TS and network entities, rather than a trusted platform and specific validation protocols and data structures, and c) PVM architecture allows for better scaling by a more distributed design approach. Thus, conceptually, TNC can be viewed as a building block for certain, but not all, realizations of PVM.

The other building block for PVM is device management. Concrete methods for remote management are provided by other standards such as Open Mobile Alliance’s OMA-DM (OMA, 2010), targeting mobile devices, and the Broadband Forum’s TR-069/169 standards suite (DSL Forum, 2004; Broadband Forum, 2009), targeting set-top-boxes or similar embedded systems. PVM also borrows conceptually from these standards. So, while we start from TCG’s TNC and OMA-DM concepts, PVM aims to integrate those approaches in a high-level functional architecture. The engineering task of mapping this integrated system back to concrete flows expressed in terms of those, or other technical standards, is not considered in this paper.

The approach to PVM is top-down, in the sense that the architecture from the outset assumes the availability of most of the central concepts of Trusted Computing technology as defined in the previous section. In particular, PVM relies on secure start-up and TRVs (e.g., RIMs) to build the base for all operations and methods. This does not exclude variants based on less trusted technology. One particular instance regards variants that avoid using RIMs in various steps of the PVM. The reason for choosing the described top-down approach is twofold: a) TS, e.g., based on TCG specifications and networks based on specification background from 3GPP, already describe sufficient technology to provide a solid basis for PVM, and b) many realization variants suggest themselves, e.g., in omitting one or the other part of the trusted infrastructure, or network elements. The generic description of PVM is therefore a toolbox for scaling PVM system design. The main principle of PVM design is maximal separation of duties between the active entities. This approach clearly defines the fields of activity of every entity involved in PVM. The advantages of this method are that: i) Each entity can be optimized for performance separately, ii) PVM entities can operate (with limitations) asynchronously and, as far as possible for the network entities involved, stateless, iii) entities can separately be maintained and
managed, and iv) necessary redundancy and failover conditions are easier to implement. In particular, performance and availability are essential for an effective PVM implementation. In concrete scenarios, there may be events of mass updates of device components or a large number of devices changing validating at the same time, e.g., upon a change of network (selected home, SHO) operator.

The scope of PVM is the validation and management of a device by one operator – usually the SHO. Special variants of PVM may have an impact on roaming access and operator change, but this is not considered here. The establishment of appropriate trust relationships, in which many MNOs co-operate and have to build a trust infrastructure for that is out of scope of this document. Likewise, we do not consider methods for device authentication, except for the special case of binding platform validation to authentication in SAV and subsequent PVM. Also, the internal security functions of the device in its TCB, for instance handling of authentication credentials, are not considered.

3.1 Device-Side Assumptions

Devices come in many flavors and by many names. PVM abstracts from the technical background of Femtocells, or H(e)NBs, and M2M devices, and is applicable to many other networked device satisfying certain preconditions. These preconditions are essentially those of a Trusted System. From a TS, we single out a special functional block inside its trust boundary, called the Trusted Environment (TrE). The TrE in the device is built in a secure start. It provides a secure execution environment and other essential, protected capabilities. More detailed examples of a TrE are found in (Schmidt, Leicher, & Cha, 2010). The TrE may be a managed component (i.e., it is, by itself, not immutable, only the RoT remains immutable). At variance with traditional trusted computing, the TCB constituted by the TrE is not immutable in PVM. It is for this reason that in PVM we need to distinguish between the TrE and its surrounding in the device. Specific, and different, information on both parts is transferred to the infrastructure and used to validate and manage them according to different policies. The TrE is the primary communication partner of the PVM infrastructure and is assumed to perform tasks associated with PVM correctly. The main requirement on the TrE is to support the secure start-up of the device in multiple, manageable stages, based on a set of internally available TRVs. In connection with this essential property a few security and functional requirements immediately follow:

The TrE shall securely store all TRVs at all times.
The TrE shall boot securely according to section 1.3.2.4
The TrE shall support single-component and/or multi-component integrity verification.

In a single-component integrity check, the TrE shall load the full code necessary for the trusted operation of the device as a single-component and before starting this component, the TrE shall perform an integrity check by comparison with an according TRV. Only if the single-component passes its integrity check, the component shall be started.

In a multi-component integrity check, the full code base of the device that is necessary for trusted operation of the device shall be segmented and ordered into several components based upon device functionality. If any component fails its integrity check, that component shall not be started but the TrE may continue checking the integrity of the next component.

The integrity of the device is verified if all components necessary for trusted operation of the device are verified.

We envisage that the secure start-up of remotely validated and managed devices is a structured process performed in more than one stage. A generic sequence of stages is proposed as follows.

Stage 1. The TrE is built from the RoT in a secure start-up. All components loaded or started are verified, and only those which
pass verification, are loaded and started. Control is passed to the TrE to perform stage 2 of secure start-up only if stage 1 is successful.

**Stage 2.** The TrE verifies loads and starts further components which are essential for performing PVM. This comprises for instance communication and protocol stacks, and Radio Access Network (RAN) communication modules. All components loaded and started are verified, and only those which pass verification, are loaded and started. Stage 3 of secure start-up is initiated only if stage 2 is successful.

**Stage 3a.** The TrE verifies loads and starts further components. Only components passing verification are loaded and started. (Finer granularity based on policies is possible in stage 3. For instance, components may be loaded into a sandbox environment, if they fail verification or TRVs are unavailable).

**Stage 3b.** The TrE measures and loads further components.

The differentiation between components in 3a and 3b is assumed to be governed by a locally available policy. The differentiation between stages 3a and 3b is analogous to the one between trusted services and measured services in the secure startup of the MPWG reference architecture (TCG, 2008b). Note also that loading and verification may be combined into one step. A fourth stage could also be added for unverified components in “user space.” Figure 3 depicts an example of multi-stage secure start-up.

The failure of a single or multiple components in stage 2 (communication modules etc.) does not imply that the device won’t be able to communicate. The stages are understood as classes of components which belong to certain categories. As long as the most essential components of stage 2 are loaded, the device will be able to communicate its state and the failed components to the PVM system. This design allows the device to perform PVM (and thus remediation processes) without a restart if some of the components fail internal verification. Another variant could be the use of a fallback code (FBC) base which allows the device to perform PVM in the case that a compromise has been detected during secure start-up. The device would reboot using the FBC and then start into a predefined state allowing device remediation.

During secure start-up, the TrE may record and protect against tampering various data, for example the following:

- **Figure 3. Multi-stage secure start-up**
instance (selection according to realization variants):

- A list of loaded components \( \text{Clist} \)
- Other meta-data associated with loaded components, e.g., start parameters
- Measurement values of some, or all, components
- Verification data identifying the outcome of, measurements, i.e., the platform state
- The validation identity \( \text{Dev_ID} \) of the platform

Depending on the TS’ architecture as a whole, some data may be protected by other secure carriers, such as an UICC, which may for instance bear \( \text{Dev_ID} \) in the form of a standard network authentication credential. In such architecture, a secure channel needs to be provided between the TrE and the data carrier.

3.2 Network-Side Assumptions and PVM Entities

We assume that all entities are operated by the same MNO as a part of the same CN. Thus, we will not explicitly require additional security for the establishment of channels and the actual communication between these CN entities (e.g., mutual authentication, integrity protection of messages, encryption). However, this approach does not preempt solutions in which entities are placed outside of the MNO’s CN or even hosted by another party than the MNO. 3GPP standards provide us with two useful entities to anchor PVM at:

The Security Gateway (SeGW), see 3GPP (2010a, 2010b), which is a role-model for the network entity which builds a border between the MNO’s CN and the outside (device) world. It acts as barrier and Network access control and enforcement instance for the CN. In particular it performs authentication of and de-/encryption of communication with the device, security association management, and session establishment.

The DMS (Device Management Service) is, in taking up device management functions, similar to an enhanced version of the H(e)NB Management Service (HMS) specified by 3GPP (2009). The specific functional enhancements are described below. The DMS acts as the central entity for management of components on the device, including software updates, configuration changes, over-the-air (OTA) management, and failure mode remediation.

The PVE (Platform Validation Entity) is the CN function which actually performs device integrity validation, i.e., is the receiving point for SAV data from the device. The PVE checks if reported values are known and good and issues statements about device integrity to other entities in the CN. In particular it communicates with two following auxiliary functions and associated databases.

The TRV manager (TRVman) is a PKI-like function which performs all tasks of:

- Management and provisioning of TRVs
- Management of according trust certificates, in particular
- Ingestion of foreign TRV trust certificates
- Verification of TRV trust certificates
- Generation of (operator specific) TRV certificates
- Check of certificate validity by, e.g., revocation, time limits and trust relationships

The TRV manager is the unique entity, which is authorized to manage the validation database \( \text{V_DB} \). \( \text{V_DB} \) and TRVman are protected CN components. Write access to the \( \text{V_DB} \) is limited to the TRVman. TRVman is of special importance with regard to security, because it manages the external trust relationships necessary for PVM.

A Configuration Policy manager (CPman) performs all tasks of:

- Management and provisioning of device configurations,
- Management of according policies, in particular
- Ingestion of foreign configurations and policies (e.g., from TTPs)
- Generation of (operator specific) target device configurations and policies

The CPman is the unique entity, which is authorized to manage the configuration policy database C_DB. The CPman is of special importance with regard to security, because it manages the network-internal security policies for PVM.

In the following, we take the convention that it is the device TrE which communicates with the network entities. While other components of the device (e.g., network interfaces) which are needed for this communication are not necessarily an integrated part of the TrE, it should be possible for the TrE to assess the integrity of these components to ensure end-to-end security.

3.3 Architecture and Interfaces

Figure 4 shows the minimum set of entities, their relationships and interfaces for PVM. To give a more complete overview, additional entities such as the authentication, authorization, and accounting (AAA) server and a connected User Equipment (UE) and their interfaces are shown. The latter relates to the use case where the device is an H(e)NB or an M2M gateway, which serves as a communication entry point for many UE (handsets or M2ME).

In general, the interface I-h between device and the SeGW is unprotected and special measures must be applied to secure this channel for authenticity, integrity and if desired confidentiality. I-h is used to establish the link between the device and the SeGW and thus the CN. It is assumed that the operator has established appropriate means to ensure the security of intra-CN interfaces. I-pve is used by the SeGW to contact the PVE during validation. PVE can

---

Figure 4. PVM core architecture
use I-pve to signal the outcome of validation to the SeGW. I-dms is used for the device configuration related communication between DMS and SeGW. I-pd is used by PVE to communicate with DMS and vice-versa. This interface is of special importance for PVM, since it is used in device software updates and configuration changes. The interfaces I-v and I-d are used by PVE to read TRVs from V_DB and by DMS to read configurations from C_DB. I-r and I-c are used by PVE to communicate to TRVman (e.g. in case of missing TRVs in V_DB) and by DMS to communicate with CPman. TRVman and CPman use the interfaces I-vdb and I-cdb to read, write and manage the respective databases V_DB and C_DB. It may be possible for the device to connect directly to the DMS, e.g., in case of a fallback mode in which the device is not capable to perform the security protocols with the SeGW. In this case, the DMS would act as the point of first contact for the device and then communicate with the PVE to perform a validation or at least to get to know which components have failed during secure start-up.

**PVM PROCESS**

Validation of a TS by the PVM system is a realization of semi-autonomous validation. The basic flow relating the stages of secure start-up of the platform to the PVM processes are fairly simple: In stage 1, the TrE is built from the RoT. In stage 2, the TrE verifies the integrity of a pre-defined portion of the rest of the device. The integrity checked code in this stage consists of e.g. basic OS, basic communications to SeGW and the code which formats SAV reporting messages. The measurement values can be stored in secure storage inside the TrE, or protected outside the TrE (e.g., sealed). If stage 1 and/or Stage 2 checks fail then the TrE blocks authentication from proceeding. If stages 1 and 2 are successful, then stage 3 proceeds. Finally, the validation data is prepared and sent to the SeGW. The data should be authenticated and could, for instance include a list of stage 3 modules that have failed the integrity checks, or a list of all components and their measurement values, depending on PVM realization variants. The SeGW forwards this data to the PVE for a decision. Device authentication may proceed asynchronously but the decision to authorize network connection may be delayed until after the PVE has inspected the validation message and made (or obtained) a network-based policy decision regarding modules which failed the internal verification or validation.

From this basic description, some advantages of PVM become visible. It offers fine granularity and more visibility into device security properties, faster and easier detection of compromised devices, and supports network-initiated remediation for compromised devices which brings more flexibility for Operators in device security management. The validation process in SAV does not require a direct communication between devices and any entity in the CN. Only after a successful validation using SAV, connections to the CN are allowed. No direct threats are posed to PVE and DMS from the device, since they receive only data limited to their tasks and only over secure connections with the SeGW, or established by the SeGW. This ensures that only devices in a proven secure state can communicate to entities inside the CN. In effect, PVM protects the CN completely from rogue devices.

We now proceed to a stepwise description of the PVM processes.

### 3.4.1 Reporting of Validation Data

To perform platform validation, the TrE collects the validation data and communicates it to the SeGW in a validation message, which includes at least:

- **Device Info** (Dev_ID, manufacturer, TrE capabilities and properties including RoT)
- **TrE info** (ID, certification info, manufacturer, build version, model, make, serial-no.). It can be realized for instance as a certificate signed by the vendor of the TrE or a TTP,
and it can be optional if Dev_ID gives a reference to TrE_info.
Verification data (for instance PCR values, or TRV certificates)
Verified binding of validation to verification data (e.g., signature over PCR values)
Ordered list Clist of component indicators (CInd) to components
Optional: parameters for components
Optional: time-stamp (trusted or not), and/or freshness guarantee (e.g., a nonce)

CInd gives a reference to a specific component and may for instance be in URN format (e.g., URN://vendor.path.to/component/certificate). The use of URNs as indicators to components is advantageous because it concurrently allows for this unique identification of a component and the location from where a TRV or a TRV certificate can be fetched. The list of components serves to identify the TRVs for validation, e.g., by pointing to TRV certificates.

The SeGW checks the verified binding, if present. This step ensures the authenticity of the verification data. The verified binding may be realized differently, e.g., by binding the validation message to TrE authentication. The SeGW extracts TrE_info, Clist, and other validation data from the validation message. It can check the format and integrity of the validation messages and parts thereof to mitigate the threat from malformed data attacks.

SeGW can check the timestamp received with local time to detect variations. If the reported time-stamp does not match the local time, the SeGW acts according to the properties of the reported time-stamp. If the device’s time-stamp is a trusted time-stamp and shows a variation, SeGW should trigger a revalidation of the TrE and its trusted time source. In the case of a non-trusted time-stamp, the SeGW could simply add its own time-stamp to the message, to mark the actual time the device submitted the validation message, and leave further decisions to the PVE.

The SeGW now creates a PVM token, T_PVM, which is used as a rolling token and is passed from entity to entity during communication. Every entity should put a timestamp, or sequence, or freshness indicator on the token before sending it to assure freshness and allow asynchronous message flows, i.e., to provide a means to follow the state of the token. For uniqueness and synchronization, the initial T_PVM contains Dev_ID and either the original timestamp from the validation message or the one generated by the SeGW. The SeGW maintains a token database T_DB containing all active T_PVM (not shown in Figure 4, since the token-passing operation is actually only one variant realization of PVM).

3.4.2 Failure Condition Detection and Baseline Reactions

The pivotal point for the assessment of a device’s validity is the recognition of failure conditions by the PVE and their treatment by the PVM system, both based on operator policies, which makes the PVE a PDP for validation. In our strict separation of duty approach, it is the only PDP in the PVM system. It relies on SeGW, and DMS to enforce policies, i.e. to act as PEP. PVM remains, in its general description, agnostic to the question of how policies are generated. In general, decisions on the validation policy can be based not only on the validity of single components but also on the other data contained in Clist. In particular allowed parameter (range), and load order (Clist ordering) can be evaluated.

There are some fundamental classes of failure conditions that may occur in the validation process executed by the PVE:

F1: TrE invalid. By its authenticated Dev_ID and the delivered TrE_info, the PVE identifies the device and/or its TrE as one which is not trustworthy.
F2: Verification data failure, with sub conditions
F2a: Integrity Measurement/Verification Data mis-match, which indicates failure of the secure startup process of the device, and/or presence of false and/or expired TRVs and or TRV certificates on
the device, which then started an invalid component.

**F2b: TRV Missing.** A TRV for a component is missing and needs to be fetched from elsewhere.

**F2c: Expired TRV certificate**

**F3: Clist policy failure, with sub conditions**

**F3a:** Single components are valid, but the configuration fails a policy, for instance on load order, or undesired components, or parameters.

**F3b:** Configuration is unknown, that is, a ‘known good value’ for Clist is not available.

**F4: Pre-Validation Device Authentication Failure, in particular expired device certificate.**

F4 only applies if authentication is bound to validation in a way in which device authentication precedes validation. If no policy failure condition is met, the device is valid. The PVE signals this to the SeGW, which then allows connection to the CN.

We come now to specific reactions of the PVM system in each case. Figure 5 shows a call flow example for PVM to provide the reader with a reference for the following descriptions.

**Detection and Treatment of F1:** The PVE queries V_DB using the received TrE_Info, containing detailed information on the TrE. V_DB stores information on which TrEs can be considered trustworthy, e.g., it is possible to implement policies to trust a certain vendor, model, etc. If the TrE is not trustworthy according to this assessment, the PVE can send a message containing this information to the SeGW. The PVE can then act appropriately on this message. The PVE adds a statement to the T_PVM, e.g., an additional data-field, containing the cause of denied access (e.g., wrong/untrusted manufacturer). The PVE refreshes T_PVM and forwards it to SeGW. The SeGW will then deny network access and device authentication (and block future authentication attempts). This treatment of F1 allows blacklisting of devices according to certain properties, e.g., manufacturer, device version, etc. The PVE could also first attempt to trigger a V_DB update process analogous to the TRV update process described below.

**Detection and Treatment of F2:** The PVE fetches TRVs from the V_DB for all components from the received Clist. The validation database V_DB only stores certified TRVs, and contain also associated TRV certificates. Then procedures branch according to the sub conditions.

**F2a:** The PVE recalculates the correct verification data from the retrieved TRVs and matches it to the verification data received in the validation message, for instance recalculating final PCR values. Based on the verification data (e.g., PCRs) the PVE can also use part of V_DB to caches trusted configurations. That is PVE looks up a table of verification data (a hash table in the case of PCR values) for valid configurations. If a match is found, validation is immediately successful. This can be useful for classes of devices running in the same configuration. Instead of comparing all components against TRVs, a single composite hash value can be compared, lowering computational overhead and speeding up the process of validation.

If the calculated verification data does not match the one from the validation message, the secure startup process on the device has been compromised or wrong TRVs are stored in the device, and invalid components have thus been loaded in the secure start up process. The PVE can, in the latter case, compare the measurement values transmitted in the validation message or in answer to a separate request from PVE, to TRVs, to detect the failed components. Depending on the F2a policy, several options for reaction can apply:

**Rejection:** the PVE signals the outcome of the validation to the SeGW. SeGW can then
deny network access or put the device into a quarantine network.

**Update:** PVE sends the validation result to DMS, which starts a management process to replace the components which failed validation, according to the management procedure of Section 3.4.4 below. DMS indicates to SeGW that validation failed and that the device will re-validate later, after component updates. DMS sends correct TRVs, fetched from V_DB, to the device and triggers a reboot. Upon reboot, the device re-authenticates and re-validates using the new TRVs. If the verification data is incorrect again, the device cannot be recovered by remote management procedures. To prevent endless re-boot cycles, the DMS may store the Dev_ID with a timestamp when sending the remote reboot trigger. If the DMS receives a command from PVE to perform an update again, DMS would check if the Dev_ID is already stored. If several storage entries exist, the time-stamps will reveal short reboot cycles, indicating that the device cannot be recovered. Further variants may apply.

**F2b:** In this condition, the PVE would first attempt to fetch TRVs that fit the reported
measurements. This can be done essentially in two ways.

**TRV fetching from TTP:** If TRVs for some components is not stored in \( V_{DB} \), PVE transfers the list of missing TRVs to **TRVman**. **TRVman** then tries to fetch (certified) TRVs from a TTP. **Clist** contains component indicators **CInd**, by which **TRVman** can identify the components and get information on where to find the corresponding TRVs and certificates. **TRVman** performs a TRV ingestion for the new TRVs including verification of the certificates into the \( V_{DB} \), and updates of \( V_{DB} \) storing the **CInd**, TRVs and certificates. **TRVman** signals the \( V_{DB} \) update to **PVE** which can then fetch the previously missing TRVs from \( V_{DB} \).

**TRV fetching from the device:** If the device has indicated in the validation message the capability to provide its own stored TRVs and certificates to the network, the **PVE** may fetch missing ones from the devices. Assuming that the device has used all of them in secure startup, all TRVs are present in the device. In this case the **PVE** forwards the list of missing TRVs, to the **SeGW**. The **SeGW** performs a protocol with the device to retrieve the TRVs and their certificates. The **SeGW** passes them to the **PVE**, which in turn forwards them to **TRVman** (or **SeGW** sends them directly to **TRVman**). In all this, the **T_PVM** is also passed around between entities to mark the active one. **TRVman** then verifies that the received TRVs are issued from a trusted entity and valid. **TRVman** performs a TRV ingestion as described above and hands over to **PVE** to proceed with validation. If TRVs for components are still missing after the retrieval and ingestion process, the **PVE** will not ask the device for again but engage in the TRV fetching from a TTP, if possible. Any TRVs obtained either from the device or TTP may be verified for trustworthiness along the same lines as digital certificates.

The trust model between the PVM components determines the sequence of actions in the TRV ingestion. The **PVE** will not trust the TRVs, but wait for their ingestion to \( V_{DB} \), performed only by **TRVman** after checking trustworthiness of that data. Of course, **PVE** could also concurrently with the **TRVman**’s TRV ingestion operation start to recalculate verification data based on the received TRVs (when it gets it from the **SeGW**), but would have to wait in any case for the **TRVman**’s decision on their trustworthiness.

For trusted computing devices a TRV ingestion process is defined in TCG MPWG Reference Architecture for RIMs. The difference here is that the TRV ingestion procedure is performed by an external entity, while RIM ingestion is a core task of the MTM.

**Detection and Treatment of F3:** The **PVE** fetches a policy on allowed configurations from \( C_{DB} \). \( C_{DB} \) can contain allowed configurations by **Dev_ID**, and also policy actions, for instance desired updates for a device which was disconnected and did not validate for a while. **PVE** evaluates the policies, based on the information in the **Clist**. If the evaluation results in any of the failure condition \( F3a \) or \( F3b \), different actions can apply:

- **Rejection:** **PVE** sends information on the failed configuration policy to **SeGW**, which denies network access and device authentication (and block future authentication attempts).
- **Configuration Search:** If **Clist** is unknown and thus not found in \( C_{DB} \) (failure cond. \( F3b \)), or no policy exists for components in **Clist** (special case of \( F3a \)), **PVE** calls **CPman** to search for configuration policies from TTPs. If **CPman** is able to retrieve new configuration policies, **CPman** updates the \( C_{DB} \) and sends a message to **PVE** with an indicator to the updated configuration policies. It is possible to keep \( C_{DB} \) and \( V_{DB} \) consistent if the update contains new components (\( F3a \)), this is signaled from **CPman**.
to PVE including the new component identifier. The PVE then forwards the necessary information on the new components to TRVman to fetch updated or new TRVs for the components. Here we want to keep the management processes for configuration and TRV management separate so that the components Cman and C_DB and TRVman and V_DB can operate independently.

**Update:** If the policy requires an update to the device, PVE triggers the update process. The update process is detailed in the section on device management below.

The fourth failure condition is a simple authentication failure which is not treated in the PVM system. IT can mean that the TrE credentials are corrupted.

### 3.4.3 Revalidation

Revalidation after (successful) management actions on a device is key for PVM; moreover, devices are such that once it has been authenticated by the network, it will rarely be rebooted barring a loss of power. Revalidation of the device should be a routine part of the execution environment. Periodic revalidation will enable the network to have confidence that the device is working in a defined state with reduced risk of rogue code executing. The revalidation will also enable the authentication procedure to initiate again thereby keeping the key exchange new and re-establishing the secure communications tunnel. There are two triggers for device revalidation, one by the network and the other by the device itself. The principals of revalidation discussed below can be applied to any of the validation techniques discussed previously.

**Device Initiated Revalidation** can occur on a periodic basis. Depending on the frequency of use of the device, the operator can set a periodic revalidation schedule in device setup. At the scheduled time, the device would initiate a reboot sequence that would trigger the validation process to begin again, along with the authentication. At this time, if software updates are required for the device, the corresponding management process could also be initiated. If the device does not re-authenticate/revalidate within in the desired time-frame, the CN can trigger the revalidation. A concern is that operators might not have control over the revalidation process with device-only initiated revalidation. If a large amount of devices runs the same schedule (e.g., first day of month) this would increase the load on the CN infrastructure.

**Network Initiated Revalidation** can occur on a periodic basis as in the device initiated case, but it also can happen at any time the network deems it necessary for security reasons. Revalidation could also be setup by the Operator as part of the policy so that a module in the device is programmed by the Operator to carry out re-validation at the programmed intervals. The PVE would periodically send a revalidation indicator to the SeGW. To keep track of all sent revalidation requests, the PVE stores them with Dev_ID and a timestamp. The PVE then periodically checks if any devices ignored the revalidation request. The SeGW would forward that request to the device via some standard authentication protocol, e.g., the NOTIFY payload in IKEv2 (IETF, 2005). Also, TR-069 (DSL Forum, 2004) provides a reboot signal which could be used for this purpose. The device then initiates a reboot sequence where validation and authentication to the network are re-established. If the device is compromised such that the device ignores the revalidation request, the PVE will detect this during the monitoring of all active revalidation requests. The PVE then signals this to the SeGW which can act properly, e.g., put the device in a quarantine network. The revalidation of a device can also occur by request from other network entities. If a device manufacturer suspects their device has...
been widely compromised, the manufacturer could contact the MNO and request a revalidation; this would be done as a back office process, whereby the MNO shall decide if revalidation will occur or not. The PVE or DMS would initiate the revalidation and re-authentication.

### 3.4.4 Platform Management

The DMS is the main entity responsible for device management. Based on received and stored device information, e.g., vendor, HW/SW configurations, TrE capabilities, etc., the DMS is able to initiate software updates, configuration changes and OTA device management procedures. The management actions are generally determined by the transmitted validation data, validation results from the PVE, and policies in C_DB, in particular desired target configurations.

The DMS first has to establish a secure tunnel with the device’s TrE. The DMS uses the T_PVM to retrieve the Dev_ID, the latest reported validation data and the Clist for the device. The DMS queries the SeGW to establish a secure tunnel to the device’s TrE by sending the T_PVM with an indicator to set the device’s status from ‘active’ to ‘management’. That is, SeGW keeps this token, does not yet allow backhaul connectivity (quarantine), and waits for the DMS to confirm end of management activity.

Depending on the management action by the DMS, the device could be required to revalidate, wherein the PVM system state is maintained by using the existing T_PVM. The DMS sends T_PVM to the SeGW, with device status indicator changed from ‘management’ to ‘revalidate’. The SeGW keeps a list of devices awaiting revalidation, wherein it looks up devices when they request network access. The SeGW will then wait for the device to revalidate for a certain period of time. The outcome of the revalidation is then signaled back (from PVE) to the DMS to confirm successful completion of the management process. To avoid a race condition, the revalidation is triggered by a message from the DMS to the TrE, only after the token passing (otherwise the device could try to revalidate while SeGW did not yet receive the token to prepare for revalidation).

The necessity for revalidation arises in the system model for the device as described in Section 3.1. New components downloaded from the DMS are inserted into the device configuration precisely after the next secure start up process. Therefore it is necessary to trigger revalidation as the concluding step of platform management. Since the device has to reboot, and if the platform validation is furthermore bound to the authentication of the platform, revalidation requires cutting the existing connection for platform validation and management. The SeGW should, in this case, maintain a state for the revalidation as described in the last paragraph. Using the T_PVM continuously over many rounds of revalidation is useful to detect recurring update failures and other patterns of irregular behavior.

If the DMS installs new components to the device, it must be ensured that the TRVs for the software are included in the same management message from DMS to TrE. The TRV for any new software is sent to the PVE which stores it, via TRVman, into the V_DB. The DMS updates the configuration policy database C_DB accordingly, using CPman. The TRV for the new component must be made available in the V_DB before the device engages in revalidation, for the PVE to validate the new configuration. In the case of a configuration change, e.g. if DMS changes parameters for a given component, the C_DB must be updated by DMS via CPman.

Many more options in management apply, for instance:

- Disabling components, i.e., force revalidation without this component in Clist. This may apply if a valid update for a component is not available.
- Force change of load order
- Force change of parameters
One should be aware that it is, in PVM, differently from trusted computing, generally not necessary to send TRV certificates to the device. The verification and management is, in the presented architecture, a task of the operator network, located in the TRVman. The device can trust received TRVs and Clinds in the management process, because it trusts the network. TCG MPWG, on the other hand, has defined RIM ingestion by a trusted device as a de-centralized process, in which the device also verifies obtained certificates for RIMs, before installing them, protected by the MTM. One should further note that both variants are not mutually exclusive. The DMS can send the RIMCs along with the other data and a TCG MPWG compliant device may install them according to the TCG specifications.

**SPLIT VALIDATION**

A major challenge for PVM is scalability. In M2M scenarios the number of connected devices may be orders of magnitude larger than the common number of handsets connected to any mobile network. The common approach to this problem is to introduce a hierarchical network architecture involving (M2M) gateways (ETSI, 2010a, b), distributing load, and unburdening the CN. Gateways may serve access layer as well as application layer tasks in such an architecture. They must also perform security critical tasks on behalf of the CN, in particular device and subscriber authentication, network access control and, of course, device validation. In all that, a gateway has a dual role as an authenticator/validator of subordinate (M2ME) devices, and as a device which authenticates and validates to the network. This section proposes a high-level concept to leverage the mentioned hierarchy for validation in what we call **split-validation**.

Split-Validation allows distributing validation tasks between two (or more) networked entities. The idea is focused on the validation procedure itself and not bound to a specific architecture. We assume a client-server model where a client C wants to access a service S where S only provides the service to C if C is in a trustworthy state. We further assume that C is able to convey validation data v to S. In this scenario, S needs tools to derive the state of C from v, which is achieved by TRVs r to which S can compare v. C will be connected to S via a gateway device G (e.g., M2M gateway for M2ME, or an H(e)NB as gateway for UE). If G is able to perform initial parts of validation of connected Cs, i.e., G is equipped with r for connected devices, the load for validation tasks can be distributed between G and S, which is referred to as **Split-Validation**. For Split-Validation to work, it must be possible for G to validate single connected Cs, and S must trust G to perform this validation securely.

To enable split-validation, the hierarchy C-G-S, should best be reflected by a hierarchical structure of the validation data as well, which relates the bulk validation data structure carried by G to the substructures of connected Cs. Here we do not go into details of that, but mention that hierarchical structuring of descriptions of internal device structures are already specified by OMA (2007), and tree formed XML structures, relating components to sub-components, are specified by the TCG IWG explicitly for validation purposes (TCG, 2006). The latter are even bound to verification data, namely PCR values included in the XML tree structure. Thus also verified binding of hierarchical validation data is in reach of current technology. We would envisage the use of Merkle (1980, 1989) hash trees as a candidate for realization since they are known to provide a very efficient security structure for large datasets. This shall be treated elsewhere. For now, we assume that G is able to generate own structured validation data, G is able to perform structured updates, including removal and addition of substructures, and that G is equipped with a measurement agent (MA), which is able to integrate validation data of Cs into the validation data structure of G, as substructures. These substructures are assumed to be verifiable by other parties independently, e.g., they could be protected by G’s or a TTP’s signature, and provided assurance by the verified binding of G’s total validation data.
The steps in which split-validation can be carried out are the following:

**Collection of measurements** (Figure 6). The MA in G collects measurements from the connected Cs and integrates them into the validation data of G as new substructures. The collection of measurements can either be performed during a validation of the Cs to G (e.g., by performing SAV) or after authentication in an additional protocol step. In general, this step should be bound to a device authentication of the Cs, to prevent use of replayed, or forged, validation data. However, this authentication is not required to represent the authentication used by the Cs to access S (e.g., MNO network access credentials), since this authentication could also be performed in an additional step, when G establishes the connection to S for this specific C.

**Certification of substructures** (Figure 7). After collecting the relevant validation data for each C, G contacts a TTP which issues certificates for the substructures to G. The TTP is a delegated authenticator for the validation data provided by Cs. Typically, there could be different TTPs for instance for Cs from different manufacturers/vendors. TTP may either validate the full data substructure submitted by G. To improve efficiency, G may also learn, over many validation runs of devices, substructures known to certain TTPs. G may then, just send an assertion of the form ‘device C connected with validation substructure X’ to the TTP.
Such an assertion may consist in a small piece of verification data only, for instance a single PCR value. Upon receipt, \( G \) incorporates the certificates into its own validation data structure. As a variant, the TTP can also be a part of \( G \), e.g., implemented as an application which runs inside the TrE of \( G \).

**Validation of composite structure** (Figure 8). \( G \) sends the composed validation data structure including the certificates replacing some or all of the substructures for the connected devices to \( S \). We decompose \( S \) for modeling purposes into two subunits, a Validation Entity \( VE \) (e.g., the PVE) and a Service Entity \( SE \). In a variant, where \( G \) is not able to authenticate the Cs with the credentials needed to access \( S \), \( SE \) would also perform authentication, e.g., \( S \) could perform the 3G authentication, while \( G \) only authenticated the TrE in the devices to verify authenticity of the received validation data and device/TrE identity of the connected Cs. After successful validation, the service can be provided to \( C \).

In Split Validation, \( G \) can preserve the privacy of connected Cs towards \( S \), since \( S \) will only receive certificates, which replace substructures of connected devices. \( S \) has to trust the TTP and the MA in \( G \), where the trust in MA can be derived from the validation data received from \( G \), since the validation data contains measurements of MA. That is, split-validation enables the validation and certification of device properties, in the spirit of property-based attestation (Poritz, Schunter, Herreweghen, & Waidner, 2004; Sadeghi & Stüble, 2004; Chen, et al., 2006).

It may be noted that the TTPs could become a performance bottleneck in split validation, since they concentrate validation tasks for the single, connected Cs. On the one hand, this issue may be partly resolved by organization, since TTPs are distributed between device manufacturers/vendors, or outsourced as an M2M service with sufficient computing power. These TTPs would also be able to establish various optimization techniques internally. In conjunction with PVM, they might even be able to control validation cycles. On the other hand, generically some tasks may be split between TTP and \( G \), if \( G \) is intelligent enough, as mentioned above.

One issue in split validation is to determine which Cs can be validated by the TTP (and hence replace their substructures at \( G \)) and which portion of the Cs has to be validated by \( S \), i.e., which reference values are available at the TTP which \( G \) can use to certify substructures of connected Cs. In general, solutions to

**Figure 8. Split-validation step 3: service connection**
this *discovery problem* can be divided in three main classes: TTP based, gateway based and shared discovery.

The TTP based approach is suitable for Gs lacking the computation power to perform complex analysis to find suitable substructures for certification. In this approach, the TTP receives the complete tree from G and returns the results in the form of a set of certificates for all substructures which it can verify. The MA at G performs ingestion, i.e., replaces the substructures with the received certificates. G does not have to perform discovery again, since TTP passes indicators to the location of the substructures in the message with the certificates.

In the gateway based discovery, G has means to decide which Cs can be certified by the TTP and the TTP will only receive the necessary data. This approach allows to minimize the amount of communication between G and TTP, since all transferred data belongs to certifiable substructures. However, G needs to be pre-equipped with suitable metrics, which allow G to discover the right substructures, and the TTPs which can certify them. This method requires G to search its own validation tree for suitable substructures, putting more computational effort on G.

In a shared discovery model, the TTP and G make a joint effort to determine which devices can be certified. The result of such a protocol is a list of certifiable substructures. It can include the exchange of additional (meta-) data, such as device classes, capabilities, device profiles, dynamic parameters, such as load and bandwidth. The communication load is slightly more than in the gateway based discovery, but likely to be less than in the TTP based discovery. Ideally, G is able to store the outcome of the negotiation phase, which allows G to subsequently perform a gateway based decision. In the shared discovery model, it is possible for G to be pre-provisioned with a set of certifiable sub trees which can then be updated during a shared discovery phase.

Let us finally look at some concrete use cases of split-validation in connection with PVM. In a corporate deployment scenario (Figure 9), where the H(e)NB is able to validate only a subset of the components of each UE (the UEs may be private employee phones allowed to be used with the corporate Femtocell), e.g., corporate software, which shall not be exposed to the MNO (SeGW and PVE). The PVE then performs the network side validation of the underlying platform components of the UE.

In the case of a rogue device connecting to the H(e)NB, the H(e)NB would include the integrity tree of the UE for reporting purposes. The PVE is then able to command the H(e)NB to block access for the device or engage in remediation steps using PVM.

In M2M scenarios, a broad variety of different devices might be connected via a single gateway, and hence the gateway might not be able to validate all devices (or all device types). Performing split-validation allows the network to offload some effort to the M2M gateway (GW). The M2M GW can group connected devices based on their type, device class, and device properties and then provide group certificates for the device validation substructures. Another variant is a Peer-to-Peer (P2P) approach, in which multiple M2M GWs are organized in clusters (Figure 10), which allows them to communicate over better links (e.g., more bandwidth, less latency, etc.). Each M2M GW has its own backhaul link to the network. However, local communication is can be cheaper (e.g., via WiFi, Bluetooth, etc), meaning that offloading traffic to the local network provides a benefit. All M2M GWs on this Local Area Exchange Network (LAEN) should be mutually authenticated, allowing them to trust messages coming from other M2M GWs.

If an M2M GW is unable to certify a substructure of a C (e.g., no TTP can be found or the device is unknown to the M2M GW), the M2M GW distributes the substructure on the LAEN, with a request message to acquire a suitable certificate (step 2 in Figure 10). If another M2M GW is able to certify the substructure, either on its own or via a TTP, which might not be reachable by or not known to the originating M2M GW), the M2M GW returns the certificate to the requesting GW (step 2 in
Figure 10). The M2M GW can then integrate the certificate in its own validation tree. The M2M GW can also store the certificate for future validation of connecting devices of the same device class. In this way, split-validation may evolve into a peer-to-peer based validation architecture, which could also extend to device management as well.

CONCLUSION

We have presented high-level concepts for the remote trust validation and management of arbitrary networked devices. By fully leveraging methods of Trusted Computing and the architecture of Trusted Systems, security and flexibility can be provided to devices and networks. Platform Validation and Management is herein shown to blend in existing (mobile) communication network architectures, and to link back to existing technical standards.
M2M communication scenarios are paradigm examples for the applicability of PVM. We envisage that PVM could become a standard method for securely handling the lifecycle of the myriads of devices operating in future networks.

REFERENCES


OMA. (2007). *Open Mobile Alliance; OMA Device Management Tree and Description Serialization; OMA-TS-DM_TNDS-V1_2-20070209-A*.


ENDNOTES

1 This deviates from the literature, where mostly verifier is a receiver of some information which can be computationally matched to yield a binary answer to a question related to the security of a system. In particular in semi-autonomous validation, which we argue is the practically most important case, this function is internalized in the TS. This justifies the introduction of the term validator to denote the external entity which ultimately assesses the operational trust in a TS based on the verifier’s information. “Attestation” in turn is, as we explain, the process of securely (protecting data authenticity) communicating with the validator. Remote Attestation is just one embodiment thereof.
## LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3GPP</td>
<td>3rd Generation Partnership Project, collaboration between groups of telecommunications associations to standardize the third-generation (3G) mobile phone system</td>
</tr>
<tr>
<td>AAA</td>
<td>Authentication, Authorization, and Accounting (server)</td>
</tr>
<tr>
<td>AIK</td>
<td>Attestation Identity Keypair, used by a TPM as signature key in validation, provides the validation identity</td>
</tr>
<tr>
<td>BIOS</td>
<td>Basic Input Output System</td>
</tr>
<tr>
<td>CN</td>
<td>Core Network (of an MNO)</td>
</tr>
<tr>
<td>DMS</td>
<td>Device Management Service</td>
</tr>
<tr>
<td>EAL</td>
<td>Evaluation Assurance Level, a numerical grade assigned to an IT system following the completion of a Common Criteria security evaluation</td>
</tr>
<tr>
<td>EK</td>
<td>Endorsement Key, the TPM’s main RoT</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute, an independent, non-profit, standardization organization in the telecommunications industry</td>
</tr>
<tr>
<td>FBC</td>
<td>Fallback Code Base, a second code base in a device which allows the device to perform PVM in the case that a compromise has been detected during secure start-up.</td>
</tr>
<tr>
<td>H(e)NB</td>
<td>Home (evolved) Node-B, a 3GPP femtocell</td>
</tr>
<tr>
<td>HMS</td>
<td>H(e)NB Management Service</td>
</tr>
<tr>
<td>IdM</td>
<td>Identity Management</td>
</tr>
<tr>
<td>IKEv2</td>
<td>Internet Key Exchange, the protocol used to set up a security association (SA) in the IPsec protocol suite</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standardisation Organisation</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine-to-Machine</td>
</tr>
<tr>
<td>M2ME</td>
<td>M2M equipment</td>
</tr>
<tr>
<td>MNO</td>
<td>Mobile Network Operator</td>
</tr>
<tr>
<td>OMTP</td>
<td>Open Mobile Terminal Platform</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>OTA</td>
<td>Over-The-Air</td>
</tr>
<tr>
<td>PCA</td>
<td>Privacy Certification Authority</td>
</tr>
<tr>
<td>PCR</td>
<td>Platform Configuration Registers, protected registers inside a TPM, able to store 160bit values</td>
</tr>
<tr>
<td>PDP</td>
<td>Policy Decision Point</td>
</tr>
<tr>
<td>PEP</td>
<td>Policy Enforcement Point</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>PVE</td>
<td>Platform Validation Entity, CN function which actually performs device integrity validation</td>
</tr>
<tr>
<td>PVM</td>
<td>Platform Validation and Management</td>
</tr>
<tr>
<td>RAN</td>
<td>Radio Access Network, ground-based infrastructure required for delivery of third-generation (3G) wireless communications services</td>
</tr>
</tbody>
</table>

*continued on following page*
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIM</td>
<td>Reference Integrity Measurement, used as reference for measurements taken during secure boot</td>
</tr>
<tr>
<td>RoT</td>
<td>Root of Trust</td>
</tr>
<tr>
<td>RTM</td>
<td>Root of Trust for Measurement, protects measurements taken in verification during authenticated boot.</td>
</tr>
<tr>
<td>RTR</td>
<td>Root of Trust for Reporting</td>
</tr>
<tr>
<td>RTS</td>
<td>Root of Trust for Storage, enabler for secure storage</td>
</tr>
<tr>
<td>SAV</td>
<td>Semi-autonomous Validation, describing a spectrum of methods for system validation, which combine methods from remote and autonomous validation of a TS</td>
</tr>
<tr>
<td>SeGW</td>
<td>Security Gateway</td>
</tr>
<tr>
<td>SHA</td>
<td>Secure Hash Algorithm</td>
</tr>
<tr>
<td>SHO</td>
<td>Selected Home Operator</td>
</tr>
<tr>
<td>SIM</td>
<td>Subscriber Identification Module</td>
</tr>
<tr>
<td>SML</td>
<td>Stored Measurement Log, stores the measurement values and data necessary to retrace the system state, e.g. software names and versions</td>
</tr>
<tr>
<td>TCB</td>
<td>Trusted Computing Base, functional building blocks in a system that are assumed to be trusted</td>
</tr>
<tr>
<td>TCG</td>
<td>Trusted Computing Group</td>
</tr>
<tr>
<td>TNC</td>
<td>Trusted Network Connect, an architecture and a suite of protocols defined by the TCG for secure and trusted network attachment of devices.</td>
</tr>
<tr>
<td>TPM</td>
<td>Trusted Platform Module</td>
</tr>
<tr>
<td>TR</td>
<td>Trusted Resource, capabilities, processes and architectural elements, which can be either part of the TCB or live outside of it.</td>
</tr>
<tr>
<td>TrE</td>
<td>Trusted Environment</td>
</tr>
<tr>
<td>TRV</td>
<td>Trusted Reference Values, trusted reference data which is used to compare validation data to known good values</td>
</tr>
<tr>
<td>TS</td>
<td>Trusted System, a system equipped with security anchor, RoTs and TCB</td>
</tr>
<tr>
<td>TTP</td>
<td>Trusted Third Party</td>
</tr>
<tr>
<td>UE</td>
<td>User Equipment</td>
</tr>
<tr>
<td>URN</td>
<td>Uniform Resource Name</td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Network</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language, set of rules for encoding documents electronically</td>
</tr>
</tbody>
</table>