Tailored Concepts for Software Integrity Protection in Mobile Networks

Trust Management to Protect Software for Mobile Network Elements

Manfred Schäfer, Wolf-Dietrich Moeller
NSN CTO - Security Research
Nokia Siemens Networks GmbH & Co. KG
Munich, Germany
e-mail: manfred.schaefer@nsn.com, wolf-dietrich.moeller@nsn.com

Abstract—This paper presents research results on SW security for network elements. Our investigations contribute to the ongoing ASMONIA project, which is focusing on collaborative approaches and on protection and warning mechanisms in 4G networks. This work is dedicated to examine specific aspects thereof, concentrating on software integrity protection (SWIP) to securely manage SW products in mobile networks. Based on an analysis of 3GPP standardization requirements and of existing approaches for integrity protection, solution concepts are proposed and discussed to meet the identified needs. These aim at harmonized approaches for a number of different use cases. Particular account is taken of keeping infrastructure efforts as small as possible, both in operator network as well as in manufacturer domain. The proposed solutions are targeting improvements to integrate and establish efficient trust mechanisms into mobile network elements and management systems.

Keywords-Software integrity protection; secure execution environment; code signing; trust management; Evolved Packet System (EPS); autonomous validation;

I. INTRODUCTION

Software (SW) security assurance has many facets, spread over the entire product life cycle. It has to prevent attacks, arising from maliciously modified SW and associated data, determining a product's behavior.

In the following, the term *SW* may include executable code as well as any configuration information, scripts, data, or meta-data that might be protected together with the SW. Roughly we could split SW security issues into two huge areas, namely (1) to specify and to create a SW product so that it matches given security policies and (2) to assure that in a target system only original SW can be used. The former demands a series of secure SW development processes (which are not further discussed here) and organizational efforts, assuring that SW is free of conceptual flaws, vulnerabilities, and back-doors. The latter is to assure that *after SW creation* unwanted modifications (be it by hostile intent or inadvertently) are prevented or at least will be detected. We concentrate on this aspect also including measures to provide trustworthy hardware (HW) and SW co-design solutions.

Depending on contracts for commercial products SW manufacturers are liable for the SW quality and potentially also for damages and incidents arising from (avoidable) security leaks. Apart from negative impacts of incidents on a manufacturers brand and on customer satisfaction there is imperative need for identification and removal of such flaws, for mitigation and for recovery. Altogether this requires trustworthy SW management and protection.

Focusing on products for mobile access and core networks, we give an insight into balanced strategies on SW protection measures that on the one hand are required by mobile network standardization and on the other hand generally ought to be applied to assure product reliability and trustworthiness as well as to protect SW assets. This publication details the aspects addressed in earlier work [1], providing more room for discussion of requirements and of existing and proposed solutions. Starting with a requirements analysis and examining existing approaches in this paper innovative concepts are derived that beneficially enable to apply the same security infrastructure (in manufacturer domain) to different use cases for SW integrity protection (see Section II.F), while efforts in operator domain can be kept on minimal level.

While many of the strategies and principles addressed by our research work may also be applicable to User Equipment (UE) this is not targeted in *this* paper. But, note that our contribution is closely related to and further supported by the German BMBF sponsored ASMONIA [14] project, where a wider context is envisaged. The project is focusing on collaborative protection and warning systems for 4G networks. In addition to the network centric view as presented in this paper, (among other issues) SW integrity protection for mobile user equipment also will be researched by the consortium, targeting the needs, capabilities, implementation, and integration aspects of mobile phones.

II. ANALYSIS OF SW INTEGRITY PROTECTION NEEDS IN MOBILE NETWORKS

We first examine related security requirements in mobile networks as stated by 3GPP standardization and then we derive more general security requirements, based on an analysis of extended aspects for integrity protection.

A. Requirements related to 3GPP Standardization

In evolution of 3GPP (3rd Generation Partnership Project) [11] standards the upcoming security architectures strictly demand local security capabilities.

In particular, in EPS (Evolved Packet System, see Figure 1, for an example) context requirements are stated for *secure execution-environments* (specific for trusted parts of eNodeBs (eNB) [2]) or *trusted-environments* (TrE, specific for Home-eNodeBs (HeNB) [3]). These arise due to the nature of the EPS security architecture (which, e.g., implies terminating of security relations between user equipment and network and storing of session and authentication keys in EPS base stations) and the attack-prone exposition of such base stations also in public areas, outside the security domain(s) of an operator.

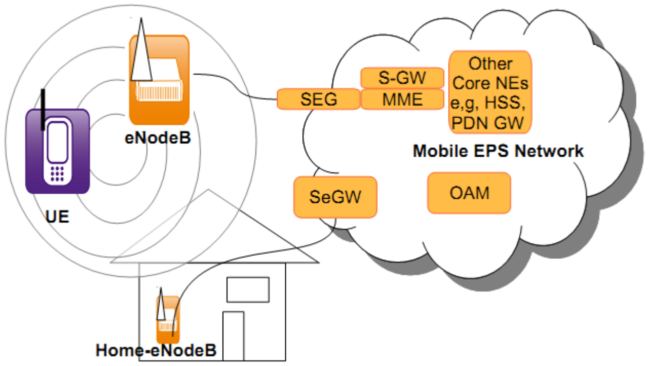

Figure 1. 3GPP EPS Architecture (partial view)

3GPP security requirements include demands for SW integrity checks, e.g., to be applied during secure boot processes whereas any room is left for realization alternatives. As related solutions (if mandatory) have to be implemented in future network products, there is urgent need to identify and to develop efficient methods for trust establishment and management. Essentially, these go back to reliable mechanisms for measuring, for verification, and for enforcement of associated directives for SW installation, loading, and usage.

Specifically, regarding SW integrity [2] demands that '*The eNB shall use authorized data/software*', '*Integrity protection of software transfer towards the eNB shall be ensured'* (clause 5.3.2, for eNB setup and configuration) and regarding the secure environment definition (clause 5.3.5) that '*The secure environment shall support the execution of sensitive parts of the boot process*', '*The secure environment's integrity shall be assured*', and '*Only authorised access shall be granted to the secure environment, i.e. to data stored and used within, and to functions executed within*'. Obviously, authorizing access to an execution environment denotes that any SW, which is brought into it and launched for execution must be targeted to an eNB and must come from an authorized source, which implies proof of origin. Typically, this involves trustworthy boot processes, each time a eNB is started, but also applies to any SW update that has to be made during the life-cycle of such product.

When looking into security requirements for HeNBs, we find in [3] related statements (in clause 5.1.2), explicitly demanding '*The TrE shall be built from an irremovable, HW-based root of trust by way of a secure boot process...*', which '*shall include checks of the integrity of the TrE performed by the root of trust. Only successfully verified components shall be loaded or started.*.' and 's*hall proceed to verify other components of the H(e)NB (e.g., operating system and further programs) that are necessary for trusted operation of the H(e)NB*'.

Moreover, it is required that the HeNB is enabled to act autonomously as it is stated with '*The integrity of a component is verified by comparing the result of a measurement ... to the trusted reference value. If these values agree, the component is successfully verified and can be started*' and thus needs to be securely provisioned with trusted reference values, as e.g., expressed with '*The TrE shall securely store all trusted reference values at all times*' and '*The TrE shall detect un-authorized modifications of the trusted reference values*'. Further, according to clauses 7.1 and 6.1 in [3], a HeNB must support autonomous validation methods '*If the device integrity check according to clause 6.1 failed, the TrE shall not give access to the sensitive functions using the private key needed for H(e)NB device authentication with the SeGW*', preventing that a malicious device (by self-check) anyhow can connect to the mobile network.

As any trust is based on self-validation processes (which implicitly may also apply for the eNB), very high security expectations are seen for any implementation thereof.

B. *Existing methods for SW integrity Protection*

In the following, we examine available approaches to support SW integrity protection and identify weak aspects and open issues from a mobile network point of view.

1) TPM based boot control

Existing methods for usual IT systems, such as known with TCG (Trusted Computing Group) standards (PC-trustworthiness with local ownership concept) cannot be converted easily to network elements and to existing 3GPP operator infrastructures. In particular, methods based on TPM (Trusted Platform Module) paradigms [4] have to be considered very carefully. On the one hand a clear, indisputable value of TPMs (or comparable crypto hardware) is that these may provide sufficient protection for storing secrets and for security operations using such secrets. This involves using the built-in crypto algorithms directly and exclusively without requiring external CPU cryptographic operations, e.g., for network element authentication. On the other hand the TPM *attestation* concept and its implementation (TPM as a co-processor) only provide partial security. There are attack-windows before attestation is completed and the TPM is not designed to parry certain physical attacks, e.g., those modifying the CRTM (Core Root of Trust for Measurement) in ROM or manipulating the TPM interface during the boot process. Doing so a skilled local attacker could inject faked PCR (Platform Configuration Register) settings – but at least has to gain access to the TPM command interface in order to control it.

By nature, the attestation approach is lacking autonomy capabilities. Due to missing local reference values for validation, local systems cannot autonomously determine and take decisions on authenticity and integrity of any SW loaded and measured during boot. In addition, particular account needs to be taken to the fact that managing attestation values over an entire SW product life cycle and for many different products is a challenge in its own.

Moreover, when exploiting extended TPM security capabilities - such as sealing - this imposes a lot of SW and trust management efforts and infrastructure invests, which are not easy to handle. For instance, re-sealing (e.g., of parts of the

An interesting aspect in this paperboot images or of internal secrets) to a new state would require individual provisioning per platform (i.e., due to authorization per TPM and tpm-Proof dependency) and could not be deployed independently from a target platform's security settings.

When looking to 3GPP standardization so far there are no discussions and indications of TPM integration into a mobile network environment. Such implementation specific properties and manufacturer restrictions could hardly be justified and would imply technology-dependent solutions. In best case it is imaginable that for a few very specific network elements such impacts could be accepted but in no case as a template for a broader scope.

As a consequence, integration of TPM/attestation based integrity protection may require remarkable proprietary changes and efforts in the infrastructure, which are difficult to motivate and to sell - apart from the fact that establishment of necessary extensions and provisioning of trust management information needs to be solved by convincing technological means. In addition, regarding implementation the required changes in existing HW (embedded platforms, boards, ASICs) have to be balanced with other design, performance, and cost criteria. Often such trade-offs render it quite difficult or even impossible to simply implant commercial off-the-shelf (COTS) TPM chips into a complex and highly specialized HW / SW platform, which is mainly tailored to meet feature-requirements while security efforts may be capped by defined cost margins.

2) MTM based boot control

In 2004, the TCG initiated the Mobile Phone Working Group (MPWG) to meet use cases and requirements of mobile phones. Based on TPM principles MTM (Mobile Trusted Module) specifications have been elaborated and made publicly available [7], [8], and are clearly in scope of mobile phone industries [12]. In contrast to TPM, the MTM is not explicitly meant as a separate chip specification, rather than it leaves room for different implementations, also as firmware or even as protected SW. The MTM concept can be built on a subset of TPM functionality, but comes with own mechanisms for trustworthy boot.

An interesting aspect in this paper is to examine how and what MTM ideas could be transferred to network elements and how these could be extended. Advantageously, the MTM allows remote management of authorized SW updates by introducing new governance schemes relying on several new types of certificates. As a newness, when compared to TPM principles, the certificate based control (to only execute mandatorily signed and verified software) enables a system to autonomously take decisions during the boot process. Due to its supposed attractiveness the MTM concept if further discussed in Section IV.A.

3) SW integrity protection as used for IT systems

Apart from the specifications introduced by the TCG there are several other individual technologies known, developed and widely used by commercial SW publishers as well as by open source communities. In contrast to TCG (which firstly focused on boot-time integrity) earlier approaches mainly concentrated on SW integrity for SW distribution and installation processes. Regarding the applied security management we roughly we can distinguish three different types of approaches: Those relying on cryptographic 'check-sums' (pure hash values as e.g., applied by some open source communities, such as OpenOffice [16]), those using code signatures based on Web-Of-Trust principles (e.g., PGP/GPG based code signing as used with RPM [17]), and those integrating with PKI principles (e.g., as established for JAVA [18] or Symbian Signed [19]).

Concepts based on pure 'check-sums' suffer from the difficulty to obtain valid reference values from trusted sources (no inherent proof of origin) and to reliably store these over a potentially long time – thus, these reference data are always susceptible to man-in-the-middle (MITM) attacks. Moreover, extended security control (e.g., regarding expiry, revocation, self-validation) is rather limited or simply not possible. Note again, that also the TPM paradigm does not natively solve these issues!

Considering mechanisms relying on Web-Of-Trust (WoT) principles, one may complain that WoT based methods do not match very well with demands for vendor driven governance and security control over network elements. As a matter of fact also WoT inherently does not reliably exclude MITM attacks. Everybody could create self-signed signing certificates and keys, as there is no mandatory registration authority established. So, trust always lies in the eyes of a believer. Apart from this deficiency a WoT usually is neither based on enforceable hierarchies nor on expressive and standardized certificates. Moreover, WoT principles do not support effective and reliable revocation schemes as usually there are many trust relations and unclear governance schemes involved (no public policies, no CRLs) and no 'official' mechanisms or entities for enforcement are available.

Of course, for user centric scenarios, individual products, or platforms (e.g., Open Source Linux distributions, based on RPM or similar package management systems) such mechanisms are beneficial. But as we are looking for generic templates for remotely manageable SWIP mechanisms for mobile network equipment, we do not deeply analyze these approaches in this paper. This does not mean that we generally dislike or ignore such concepts, but we have to apply them in the right context and scenarios.

From a vendor's perspective - who should be able to fully control the security capabilities and integrity of its products - potential difficulties may arise from inadequate fundamental security building blocks and in particular from unsuited key and trust management strategies and weak control mechanisms. This clearly argues in favor of PKI based approaches, which are compliant to accepted standards and security best practices and provide well proven governance and control principles (e.g., as defined in X.500 [27] and in particular with X.509 [15]).

While many of the PKI-based known signing concepts (JAVA, Microsoft's Authenticode, Symbian Signed, IBM's Lotus Notes, etc.) apply efficient and partially even comparable mechanisms, they are not directly applicable to the needs of manageable SWIP for mobile network equipment (which usually consists of a number of very different products and technologies). On the one hand such approaches usually are specific for one particular product or technology (e.g., operating system, programming language, controlling sand-box, run-time environment, web- or IT-applications, vendor specific UE-equipment, etc.) and on the other hand they are mostly targeted to support security requirements of distributed developer communities.

In most cases they rely on outsourced PKI entities (certificate and registration authorities) and on verification components, which often allow importing of arbitrary SW publisher certificates and accept umpteen root certification authorities (CA). If a user or administrator decides that these are trustworthy he can change the trust management settings by local administration. Vice versa, preconfigured trust-anchors and credentials could be removed on user decision. Consequently, in addition to local control, such systems chiefly target to enable tracking and juridical inquiry of malicious attackers, which by hostile intent previously have applied for SW publisher keys and certificates.

Some approaches combine code signing with explicit authorization concepts at application level. For instance, this is realized by different types of certificates (e.g., using 'capabilities' as introduced with Symbian Signed), which are associated with classes of API calls with appropriate scope and privileges. Adherence to such assignments can be checked by signing entities (before issuing a valid signature for an object under test), as well as by the devices themselves during verification or execution. To give more examples, mechanisms used by JAVA or IBM's Lotus Notes [20] control application privileges via sand-boxing approaches, but use local administration to set policies and rules for execution.

C. *Run-time Aspects*

When looking beyond the scope of boot-time or installation-time integrity checking additional security improvements are needed to provide attack resilience during long-term operation. These have to be faced as many network elements (in particular threatened eNBs, HeNBs) may be booted or updated only rarely. Then they have to be active for weeks or months, whilst the trust in boot-time checks is the more diminished the longer a system is running. Potentially this is caused by attacks occurring during operation, applying both for local, manual manipulations as well as for remote SW attacks, which cannot be prevented by boot protection alone. Consequently, there is urgency for methods and mechanisms assuring SW integrity at run-time at least for critical security operations. Such critical operations (e.g., as needed for key and credential management, for authentication, or for verification processes) require trusted code, which can only run if *before execution* it is proven to be integer and to stem from an authoritative source.

Run-time integrity issues are partly covered by TPM based improvements. For instance, with DRTM (Dynamic Root of Trust for Measurement) mechanisms are known to allow lately measuring and launching SW in a TPM compliant execution environment [4], [5]. The DRTM mechanisms assure that code, which is to be started, is measured properly (e.g., Intel is using authenticated code modules for this purpose [13]) and then executed, but *does not prevent from loading untrusted code*. Such approach requires TPM based attestation (with all the hurdles mentioned above) in combination with dedicated CPUs, which have to support specific instructions and bus cycles. While an external challenger is enabled to prove what has been executed (during run-time) on a DRTM equipped system, the DRTM operation itself is not able to verify any manufacturer code prior to execution. Again we miss an autonomous mechanism enabling a local machine to enforce rejection of manipulated code, preventing execution of any hostile operations at any time. Moreover, the selection of DRTM enabled CPUs may be in conflict with other CPU selection criteria to best match the needs of the specialized embedded architectures of a mobile network element.

The IMA approach [6] is an interesting extension of TPM concepts, introducing TPM protected load time integrity measurements of file-based executables, libraries and data, which are aggregated into a series of TPM signed lists. As with the native TPM principles, IMA relies on attestation paradigms, requiring external entities for validation. Apart from lack of autonomy the major problem of such approach again seems to be the need to maintain a TPM specific infrastructure as well as the efforts to interpret and validate a potentially huge amount of attestation data, which is reported on request. Such data has to be 'known (i.e., must be securely provisioned)' externally or must be re-calculated (where referring to any sequence of loading is not required in the IMA case).

There are other approaches such as Tripwire or Samhain [23] following alternative principles based on file-level integrity checks, which do not rely on a TPM infrastructure. They come with own associated, administrated client/server architectures and self-created, protected databases with 'trusted' hash values for validation. The run-time checks are triggered periodically or based on events, while checking modules are protected at kernel level, which may be sufficient for some attacks scenarios. A particular risk may be the fact that trusted reference values may not (or not only) be created inside a secure developer environment at manufacturer side, but in the operational domain itself, which is not only an organizational, but also a liability issue.

It is worth to mention the DigSig proposal [24] representing a load-time integrity checking solution, which relies on PGP signed ELF binaries (as provided via the Debian BSign utility), but is applicable to Linux systems only (due to dependency on Linux kernel integration and on ELF files and tools). The charm of such approach is the fact that signatures are embedded into ELF binaries, thus no separate data base is required. Moreover, signatures can be created externally, therefore local creation of trustworthy reference values is avoided and the system is enabled to take advantage of the benefits coming along with a code signing approach (e.g., proof of origin, and signature revocation, which is also supported), even though restricted by the WoT paradigm. Such solution is related to previous work [25], also based on signed ELF binaries, relying on comparable principles for run-time integrity protection.

Sand-boxing solutions such as introduced with JAVA cover PKI based SW integrity protection during installation and download scenarios and realize mitigation concepts during run-time, but may be restricted due to an individual programming language environment and due to individual sandbox constraints. Sand-boxing is not only related to integrity checking, but also may constrain program capabilities and performance during execution. This may or may not be a problem, depending on the application, but is limiting general applicability.

For reasons of completeness it should be mentioned that there are also run-time protection methods known, which make use of specific CPU level concepts, such as Intel's System Management Mode (SMM) (e.g., compare [26]). We do not further discuss these in this paper, as they are too dependent from processor capabilities (like the DRTM mechanisms, mentioned above) and thus are not ideal for generic templates for SW integrity protection, we are aiming at. Of course, it is well understood that mechanisms making use of low level HW properties can achieve a higher protection level - but usually at cost of flexibility and portability.

D. *Autonomy and Remediation Aspects*

Autonomous SW integrity protection and trust management mechanisms are highly desirable, enabling a system to take own, reliable decisions e.g., to deny sensitive services, to boot to fail-safe-mode if a new SW release is defective or to generate and transmit (or to store) signed incidence messages in case integrity violations are detected during run-time checks. Autonomy decreases efforts in network and increases security as a system knows about its own trust state, before it connects to a network. Of course, this may be limited to attempted attacks, which can be detected before they are effective and to non-persistent attacks, which can be cleaned, e.g., by re-booting or re-installing, or to attacks, which do not successfully affront and neutralize the integrity protection mechanisms themselves.

Critical are situations where a large number of systems are actually compromised by sudden attacks. For such cases robust remediation mechanisms might be implemented, which are resistant against certain classes of attack, so that they cannot be smarted out in some way - or at least not too easily. Such remediation mechanisms may require reliable, autonomous local mechanisms and even interaction with supporting network entities, assuring that affected systems could be repaired securely from remote. The reasoning behind is that in mobile networks, and in particular with the flat architecture introduced in EPS there are a huge number of systems in field, widely distributed and very often in secluded areas. Any personnel to be sent out for emergency or management services needs time and raises cost and efforts. In some cases, e.g., for HeNBs, it might also be acceptable to involve the hosting party (i.e., the user) into remediation actions, but this depends on the underlying trust model.

In particular, those attacks seem to be very precarious that emerge from remote SW injection attacks occurring during run-time. This is because they could be launched against a large number of systems simultaneously, causing partial outage of large network segments or even complete network breakdown.

Clearly, autonomy and remediation mechanisms require robust implementation, which might by quite expensive and thus, efforts always have to be balanced by cost-efficiency considerations.

E. *Generalization*

The above considerations may be very specific to ‘standardized’ requirements for integrity and trustworthiness of exposed network elements such as eNBs and HeNBs. However, the mechanism applied should also be beneficial to defend against attacks that may target or affect elements located in a (more) secure domain. Particularly, this applies if we want to exclude attacks that could be injected via the SW delivery and installation chain. Therefore, for such broader scope an important strategic goal is to re-use SWIP concepts as well as the involved components and infrastructure at the greatest possible extent, while efforts and changes in operator networks should be minimized. Understandably, it is hardly acceptable to apply (too many) different concepts for different products, if this requires operator invests, be it for organizational or operational measures or be it for technical equipment. The ideal case would be that mechanisms for managed SW integrity protection can smoothly be integrated into existing nodes, protocols and do not require unnecessary changes in standardization.

Consequently we aim at generalized and harmonized approaches for managed SW integrity protection. Such solutions shall provide adequate security and shall be suited to protect many other SW products in a mobile network (also outside the scope of EPS), widely independent from architectural aspects and from complex implementation details.

When thinking of SWIP for products in core network (i.e., those residing in the security domain of a mobile operator) essentially we can concentrate on intended SW update and SW delivery interfaces and processes, as mainly these may offer chances for malicious intervention. On the other hand, physical protection and tricky implementation issues against local attacks may be of less importance there.

Complementing the above analysis, particularly the following requirements are relevant for generalization:

- Ensure that SW (that may be composed of different components and data) has not been altered after creation process. This includes accidentally infected SW as well as *any* intentionally modified code, inserted into the SW update, maintenance or delivery path.
- Identify that SW (and associated data) is coming from a specific, authorized source (Proof of Origin).
- Verify that code is trustworthy and authorized for a specific purpose or target system. This may be expressed implicitly (by SW package) or by explicit verification of meta-data or attributes.
- Allow associating SW with unmodifiable directives and privileges for code, memory and data usage, according to the claims of an authoritative source.
- Support 'static' (before run-time) as well as 'dynamic' (during run-time) protection, preferably based on the same (cryptographic) measures and mechanisms.

F. *Holistic View and Intended Use Cases for SWIP*

Extending the conception of generalization a visionary idea of SW integrity protection is shown in the Figure 2 below. This holistic view reflects how SW may be used in different execution environments and in different operational stages, starting with SW creation and delivery processes and

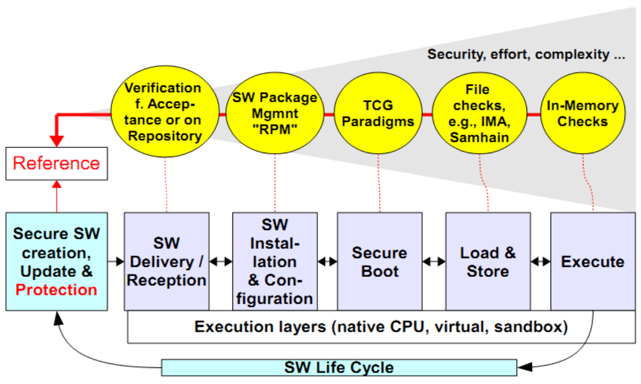
Security, effort, complexity ...
Verification f. Acceptance or on Repository
SW Package Mgmnt "RPM"
TCG Paradigms
File checks, e.g., IMA, Samhain
In-Memory Checks
Reference
Secure SW creation, Update & Protection
SW Delivery / Reception
SW Instal-lation & Con-figuration
Secure Boot
Load & Store
Execute
Execution layers (native CPU, virtual, sandbox)
SW Life Cycle

Figure 2. Holistic view on SWIP: SW in different operational stages

then passing through the possible usage modes and life-cycles.

The major use cases for SWIP include SW verification

- after delivery (at point and time of acceptance); this use case is relevant for scenarios where chiefly SW delivery processes need to be protected from attacks against SW, while it is shipped from the manufacturer to customers or to service personnel. In some cases one-time verification may be sufficient and after acceptance further protection is not needed.
- during installation; this use case requires SWIP (verification) integration into a SW installation process, which can performed locally (self-installation) or by a remote installation server. Protecting downloaded SW (which is installed at run-time) or native SW for virtual environments might also be included here. In dependence of requirements for the installed system it may be beneficial to combine SWIP with directives for the installation process, e.g., on versioning or patching, on revocation or on invalidation of previously installed SW.
- during the boot process; this use case (which typically requires active integration into a system's boot architecture) corresponds to mechanisms addressed earlier, when discussing approaches as introduced by TCG standardization bodies. Essential characteristics comprise the sequential dependency of several SW modules being loaded and the local control taken over the boot process.
- while it is stored in a file system, data base or in flash memory; this use case actually corresponds to a run-time verification scenario on storage level, where SW may be continuously verified, be it periodically or be it triggered by events created through actions, which may affect the stored data. Typically, such scenario may be effective if the status of an installed system must be checked over a long time, and may be seen as a completion to the SW installation use case. Note that the SW verified in the file system may either be in use currently or not, or it may even be stored in a repository.
- while it is executed in cache or CPU memory; this use case again is a run-time scenario. In contrast to the preceding use case, only active SW (i.e., such SW which has been loaded into memory) is under examination. In practice, this use case is the most challenging one and many efforts must be spent for efficient implementation.

In all cases, SWIP aims at checking whether SW (i.e., invariant parts of it, such as executable code or initial data) in each operational stage has been modified, when compared to the originally created reference SW. Depending on expectations on attack resilience, efforts and methodological complexity may be very different.

While partially well known or even standardized individual methods for different aspects of integrity protection are available, in some areas this is still requiring fundamental research. Particularly, SWIP is the more challenging the more we aim to inspect a system during (long) execution time and the deeper we look into a system's CPU memory space. However, at the same time the achievable security and trust-level will remarkably increase when moving from a static view on system integrity towards a dynamic one (i.e., SW module loading and execution). In the context of mobile networks the latter may become of significant importance, regarding indispensable long-lasting trustworthy operation of systems in field (e.g., operating several months per boot).

As conditions of target systems and SW environments are varying, actually a huge number of product specific solutions is required, in particular when confronted with the HW and SW particularities of our systems (e.g., SW installation and update processes, run-time environments and operating systems) where the SW is verified and used.

Consequently, it is not surprising that currently a harmonized, integrative approach is missing, which could cover all the use cases above with a unified or adaptive method. Nevertheless, this would be very beneficial and from the beginning we should aim at unification and adaptability of methods *as far as possible* and this particularly requires identifying those aspects which are widely independent from platform or implementation specific solutions.

The guiding principle of our approach is the cognition that by applying certificate based SW signing schemes (the manufacturer's) infrastructure efforts could be harmonized, while we still have to accept remarkable differences for system specific implementation and secure anchoring of trust and verification mechanisms. Such infrastructure involves, e.g., managing certificates, PKI extensions, signing mechanisms and entities, certificate policy guidelines and rules, key management principles, approval work-flows, secure SW development processes, data structures, conceptual templates, common verification and measurement tools, and so on.

As this all could be provided by the manufacturer and to a large extent could be driven by the products themselves or by (product specific) network management components, impacts on an operator's infrastructure could be kept minimal, e.g., limited to manageable changes in existing mobile network equipment.

III. SWIP PROCESSES AND TARGET SYSTEMS

In the following, we will propose and discuss strategies and concepts to match the requirements and visions as introduced above. Firstly, we consider processes as relevant for SWIP and secondly, on a conceptual level (i.e., without reference to concrete network elements) we examine influences of SWIP on target systems in the network environment.

A. *SWIP Processes*

For SW integrity the following four processes are essential and have to be realized for all the use cases mentioned:

(i) The *protection* process where the SW becomes ‘integrity protected’, e.g., by applying cryptographic methods;
(ii) the *verification* process where it is checked (verified) whether the protection has been broken or not;
(iii) the *enforcement* process where the SW is securely stored, distributed, installed, or executed, following instructions that may be part of the protection paradigms;
(iv) *infrastructure processes*, which are required to enable and support the others listed above.

Ideally, protection (i) is applied as early as possible (i.e., directly after SW is created, tested, and released, e.g., in the build environment). Verification (ii) and enforcement (iii) are done as late as possible (i.e., just before the SW is used or executed) and even better continuously as long as the SW is installed (or is running). It is evident that these processes are closely related to each other and must follow common mechanisms and paradigms that may require information exchange among each other (e.g., keys for encryption or signing mechanism or trusted reference parameters for hash values). Preferably (for a manufacturer dominated approach) the process (i) is executed in a secured domain at vendor side, while the processes (ii) and (iii) are executed in the operator network, but based on manufacturer-provided SW, key material, credentials, and mechanisms implemented within network elements. There may also be other constellations (e.g., where a system itself is responsible to run local protecting processes (i)), but these are not discussed in this paper.

In addition to the above processes, preparatory and operative infrastructure support and management processes (iv) are required, in particular to establish PKI and signing components and to control key material and credentials (in case SWIP is based on certificates) or to provide reliable reference values and trusted sources and secure management and validation capabilities for these (if SWIP is based on pure hash values or attestation principles).

Regarding harmonization certainly the focus lies on infrastructure impacts, but also the above processes (i, ii, iii) would profit from a common methodological framework, as involved tools and data structures to a large extent could be made similar and adaptive.

B. *Target Systems*

We define a target system (TS) as the 'consuming endpoint' (the platform for which the SW is designed and which hosts the execution environment where the SW is running).

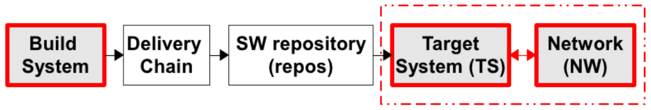

Figure 3. SWIP-aware target system, verified by network

Figure 3 shows a SWIP system where (ii), (iii), and partially (iv) are shared between the TS and an extra, external node, e.g., a verification server residing in the network (NW). Into this category fall systems that

- implement *trusted boot*, following attestation principles and TPM technology (both based on CRTM or on DRTM);
- realize *Integrity Measurement Architecture* (IMA) [6], a load-time extension using TPM attestation principles;
- act as monitoring systems interacting with network, such as Tripwire, Afick, Samhain or also IMA;
- follow principles as applied with TCG's trusted network connect (TNC) [9].

As explained such use cases (when based on external validation) may impose remarkable difficulties – regarding applied security paradigms and trust managements –, which are costly to manage in a mobile network environment. Even if these are not seen as our preferred solutions, some principles could be used, if appropriate.

In Figure 4 the ideal case is shown, where (ii) and (iii) to the greatest possible extent are assigned to the TS. This would be the best solution regarding effort minimization for the network (also regarding (iv) for setup and provisioning). This category includes the following use cases for self-subsistent SWIP-aware TSs, which are enabled to autonomously

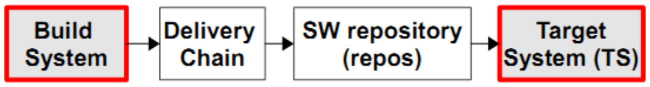

Figure 4. SWIP-aware autonomous target system

- implement *secure boot*, doing verification and enforcement during the start-up process, e.g., as introduced by the Mobile Trusted Module (MTM) specification, issued by TCG [7], [8], see Section IV;
- verify and enforce SW integrity at installation-time, every time before a SW component is installed or stored into a local SW base. Typically, this can be integrated in installation systems, such as packet managers;
- verify and enforce SW integrity, each time a SW component is loaded into system memory and then executed;
- self-monitor and verify SW while a system is running, triggered periodically or by system events (e.g., file access, socket activity, system call). Both, memory-images or files could be checked by such monitoring process;
- … and only occasionally need additional support from SW-repository (or an OAM server) for individual cases, e.g., for autonomous SWIP related SW update processes, for remediation or security management (e.g., remote exchange of secrets, credentials, or of trust anchors).

Considering generalization also SWIP-unaware TS (see Figure 5) are of interest, i.e., those where processes (ii), (iii), and (iv) are completely treated outside a TS. SWIP then is concentrated in network entities (such as operator side repositories *o-repos*, e.g., an OAM or SW management system) and the TS systems security architecture remains unaffected. It is essential that there must be a strict trust relation between the o-repos and the TS, which simply plays a passive role for SWIP.

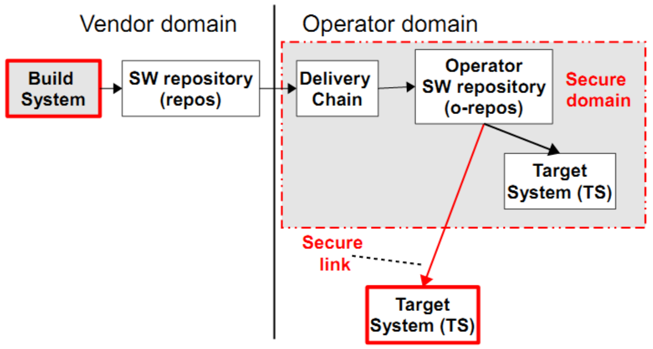

Figure 5. SWIP-unaware target system, supported by network

Such unaware TSs cannot protect themselves and must fully rely on secure domains and on the network entities they are connected to. It is obvious that such solutions cannot be applied in insecure domains, unless some basic security, such as for secure communication of raw data or SW is provided (here the major use case is to protect the regular SW delivery and update processes, managed by NW entities, like *o-repos*).

It is evident that for the latter case, some of the targeted use cases cannot be applied, but this might be acceptable in accordance with the risk assessment and cost considerations of involved network elements.

IV. Certificate based Approaches

As already implied earlier we are convinced that a framework based on certificates and PKI entities would be most suited to fulfill the requirements of managed SWIP for mobile network equipment. In addition to autonomy aspects (as explained in Sections II.D and III.B), we expect positive effects for generalization and harmonization (see Sections II.G and II.F) as well as for vendor dominated governance principles and the expectations we may have on security, regarding the use cases and product-life cycle aspects as introduced.

As an example in the following we discuss an existing approach and make proposals for further improvements, be it from methodological point of view, be it for implementation.

A. The MTM approach

As already mentioned in Section II.B an inspiring idea has been proposed by TCG to assure trustworthiness for mobile phones. Aligned with (basic) TPM paradigms the MTM specification defines certificate based mechanisms for verifying and running trusted software on mobile phones. The new idea behind the MTM specification is to support *secure boot,* allowing local verification (ii) and enforcement (iii) during the boot process, which again may involve several mutually dependent modules (i.e., to be loaded sequentially). MTM introduces so-called RIM (Reference Integrity Metric) certificates containing integrity measures and references to public keys (assigned to so-called RIM_Auths), to verify a complex certificate chain against a (e.g., built-in) root verification key. According to this, the MTM specification enables a system to act autonomously, particularly to identify and to verify downloaded SW, to perform proof of origin and to take decisions in case of detected integrity violations.

As implied, MTMs can be built upon the TPM architecture, but only need a subset of the TPM functionality. As RIM certificates integrate measurement values (as specified with TPM) - in addition to *secure boot* mechanisms - attestation protocols still can be applied, involving external entities if needed.

Regarding SWIP there are many correlations between the requirements for a mobile phone and managed SW integrity protection for NEs within a network infrastructure as demanded above. The certificate based integrity protection principles of the MTM specification can be exploited and beneficially be applied in the context of SWIP strategies and related trust concepts in a mobile network. Such ideas perfectly harmonize with the autonomous and generalized use cases as depicted in Figure 4 and Figure 5, while management support in network infrastructure can be kept at a minimal level (certificates are self-describing and attestation might not stringently be required).

As further explained SWIP based on adapted MTM concepts might very well support both, EPS security needs (as specified with eNB or HeNB), as well as generalization aspects, as explained in Sections II.E-G. In Section IV.B we propose required extensions or adaptations, taking the MTM approach as an exemplary framework. Alternatively, we also could found our concepts on another PKI / certificate based method, but the MTM seems to be a suited start point and might be 'easier' to extend, due to existing ideas on implementation in embedded systems (including TPM mechanisms underneath), to (multi-) vendor centric governance schemes and to solutions for the 'secure boot' use case.

Note that additional local security requirements beyond the scope of SWIP, e.g., related to uniqueness and ‘secure or trusted environment’ (such as secure key management, storage, and usage, and device authentication to prevent HW cloning etc.) must also be fulfilled, but are not described in all details by the solutions below. However, we give some hints on the relevant implementation aspects.

B. Adaptations of the MTM idea

We consider useful adaptations of the initial MTM idea to extend and improve SWIP methods for mobile network elements:

1) Focusing on secure boot

When applying *secure boot,* the additional value of attestation may be rather limited as compared to the organizational efforts and equipment to be invested in network infrastructure. Based on self-validation it must be assured that a system connects to a network only if the boot-time verification was successful. Otherwise, the system shall deny any interaction with the network, except, e.g., for OAM purposes. As a precondition a highly secure root of trust (e.g., non over-writable verification key) must exist. Further secure key-storage (e.g., read and write protection for private authentication keys) and secure usage for such keys in a secure (execution) environment must be guaranteed. The secure boot process is part of the establishment of such a secure environment.

The value of an additional attestation is questionable (if done to reveal a system's trust state during long-time operation), but it has some relevance if we just want to know if a new SW version successfully has been installed. See Section V for alternative approaches, which avoid involving a complete and difficult to manage TPM infrastructure and deployment.

2) Implementation aspects

Just as with TPM any security heavily depends on a secure implementation of a CRTM, in a similar way this applies to the MTM. The initial ‘immutable’ code in the MTM case is called ‘Root of Trust for Verification / Enforcement’ (RTV / RTE). Based on a risk assessment it has to be decided in each case separately which foundation for the security of RTV / RTE and the root verification keys has to be selected. In many cases (e.g., regarding remote SW attacks) it might be sufficient if these data are not over-writable or are only mutable via strong authorization mechanisms that cannot be surmounted via instructions executed by a CPU.

While the specification allows integration of TPM hardware underneath, the MTM concept is also intended for separate firmware or SW implementation. For reasons explained above, this is of particular interest for systems that cannot simply make use of commercial TPM hardware solutions.

However, in all cases 'sufficient protection' has to be provided for using and managing local secrets and credentials, as well as for sensitive processes (e.g., for local measuring and reporting). Apart from TPM or comparable crypto-HW, simpler ASICs or CPU-level integration are effective to achieve higher security levels against pertinacious attacks. See related proposals in Section V.

Evidently, for some scenarios (e.g., where we do not expect highly motivated and perfectly skilled attackers) it might be sufficient to make use of efficient SW integration techniques like kernel-space protection, system level attack mitigation or virtualization for implementation. In practice, accurate shaping of these mechanisms must be based on an individual threat and risk analysis

However, due to the focus of this paper (which concentrates on the conceptual approach) we do not step into details hereto.

3) *Extending certificate concepts and use cases*

When thinking of generalization for SWIP, the following modification of the MTM principles is gaining importance: While RIM certificates are perfectly tailored to implement *secure boot*, they are not designed to support the needs of other SWIP use cases (e.g., SW installation, run-time aspects or SW delivery, as well as bundling with extended authorization concepts). A more flexible and adaptive structure instead of RIM certificates (which actually is not a certificate in PKI sense, but 'standardized' signed data for a specific context) is required, which is adaptive to the needs of a specific SWIP use case or to the particularities of a SW product.

In Figure 6 we introduce a *generic Signed Object (SO)* to substitute RIM certificates. SO preferably might be implemented as XML signed objects to gain profit of the power and flexibility of XML and the associated XML signing framework [10], but alternatively, CMS [21] implementations could be taken as well.

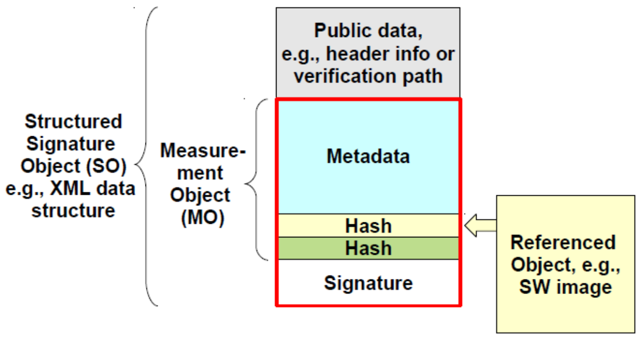
Structured Signature Object (SO) e.g., XML data structure
Measure-ment Object (MO)
Public data, e.g., header info or verification path
Metadata
Hash
Hash
Signature
Referenced Object, e.g., SW image

Figure 6. Generic Signed Objects (SO), describing the protection context

Apart from verification information (e.g., intermediate certificates of signing entities) or other public data, a SO consists of one or more signed Measurement Objects (MO), which essentially contain information, which is measured and gathered by the SW protection process (i). Such information may also contain meta-data (descriptors, circumstantiating the MO) and measured objects, which are representatives of referenced objects (e.g., hash values of one or more SW modules). As shown in Figure 7, such MO meta-data may include

- Object descriptors, specifying the measured objects together with references to associated policies.
- Measurement descriptors, specifying the format and syntax of the MO and of MO elements.
- The Measured Objects (MdO) themselves; either this can be hash values of referenced objects (e.g., a SW module or archive) or even embedded data, such as a small script or configuration information. Also, other existing external MdO or MO information might be referenced, supporting a hierarchical approach (e.g., an archive together with individually protected files stored in this archive).
- Entity descriptors, specifying the responsible entities (e.g., company), together with legal implications (e.g., disclaimers or warranty clauses).
- Crypto descriptors, specifying the applied cryptography, e.g., hashing and signing algorithms.
- Policies, which express directives according to claims of the authoritative (signing) source. Policies may include explicit rules for verification and enforcement processes or they may describe general dependencies between SW modules (including compatibility information or rules for 'sequential loading' as used with RIM certificates). Another scope of policies could be expiry or revocation of individual SW packages (which need not necessarily imply revocation of a signing key and the associated certificate). Polices can be static ('do not load module x together with y') or conditional ('if the target platform CPU is ABC, do not load driver Z'), i.e., may depend on information, time, or the state of the system to which they are applied.
- As such SOs are much more flexible and expressive than RIM certificates, they perfectly match with the requirements and visions as stated in Section II. Depending on the meaning of the descriptors and in particular of the associated, static or conditional policies, very different rules can be stated to influence and to control the processes in the SWIP endpoint (e.g., an OAM server) or within a trusted (i.e., verifying and enforcing) TS itself.

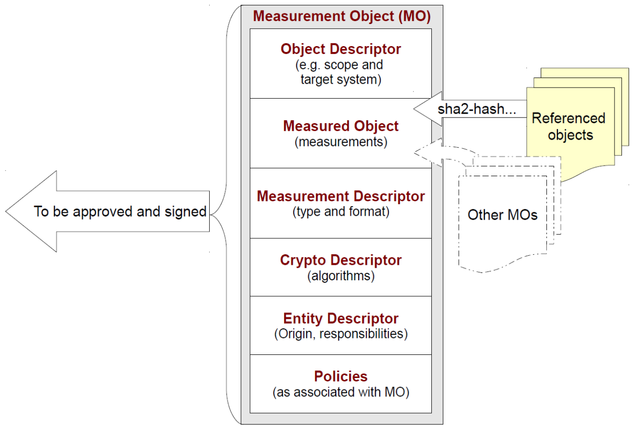

Figure 7. Measurement Objects (MO), specifying measured data

In addition to directives and conditions for SW usage, policies may express directives for the usage of MOs themselves, e.g., by specifying governance rules that have to be applied for a specific object (such as invalidation, deletion, upgrade, required patches, etc.).

The syntax and semantic of such SO may be associated with a company or with a specific use case or product being managed by an individual responsible party. It should be emphasized again that SOs may cover the full meaning of RIM certificates as a specific sub-case.

4) Governance principles

While the native MTM specification is not mandatorily aligned with X.509 and general PKI principles, we recommend to adapt the MTM governance principles to a (potentially vendor controlled) X.509 compliant PKI infrastructure. This only applies to the so-called RIM_Auths and the upper hierarchy up to the root CA. It should be noted that this part of the MTM specification could be easily integrated into X.509 elements and could be adapted to be governed via specific PKI policies, according to the needs of an individual manufacturer. The MTM specifications mention this, but do not specify any details.

V. HW LEVEL TRUST IMPROVEMENTS

In the following Sections, we discuss HW level improvements increasing security and flexibility of trusted systems like the ones alluded above.

A. Authorized SW Update

The first method locally enables authorized updating to new versions of protected SW and data that only after successful verification will be written to non-volatile storage, e.g., Flash EEPROM or hard disk.

In real systems run-time attacks enabled by vulnerable SW (e.g., exploits) are likely to happen. However, solutions for boot time protection cannot not directly provide prevention against (later) run-time attacks, which intend to take control over a system and to run with malicious functionality. Certain exploits could even try to prevent reliable and verifiable SW updates of the system, which for the future could leave the system with an old, flawed SW version. This would hold the device in a vulnerable state where the old, vulnerable SW version is still booted during next secure boot, and still accepted as a valid version, even if the new SW version should already be installed. Therefore an unacceptable security leak may arise.

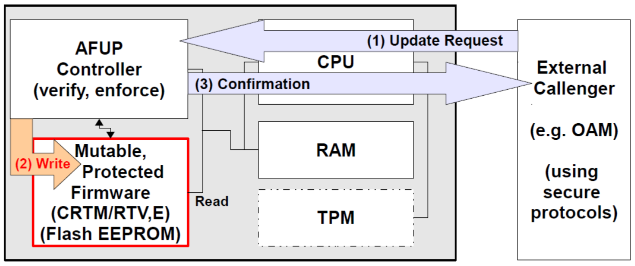

Figure 8. Authorized Flash Update Process (AFUP)

Moreover, it must be prevented that a more sophisticated SW or even local attack could change the content of any persistent trusted (i.e., already verified) code.

For external entities (i.e., regarding secure connection to a network) it is essential that either reliable attestation or one-time proof of a successful secure SW update process can be established.

In the following, we describe a solution that can be established without the need to build up and to maintain an attestation infrastructure and to deal with TPM integration. In addition to authorization and autonomous integrity protection the proposed solution provides a mechanism against specific, persistently implanted or repeated run-time attacks (against required SW updates).

The solution uses Flash EEPROMs protected by an Authorized Flash Update Process (AFUP) depicted in Figure 8, communicating via the system CPU. The control part of this process (the AFUP controller) can be implemented via dedicated hardware (e.g., an ASIC), which by design is the only unit that controls flash programming (at least for critical parts of the flash memory), and could not be affected by defined classes of attacks (e.g., CPU driven SW exploits or even certain physical attacks). For verification it can rely on 'roots of trust' residing in the flash memory.

In its fundamental operation, AFUP uses pre-configured secrets and credentials for a protected communication with an external requester, which initiates the communication by sending an update request (1). The delivered SW (that may also be the CRTM or RTV/RTE SW) is integrity protected (i.e., accompanied by signed objects SO) and is only updated (written into flash memory) upon a successful verification by the AFUP controller (2). On success the AFUP controller sends a confirmation (3) to the external requester which now can be sure that after a next boot the system is updated to an 'invulnerable' SW version and can be trusted again.

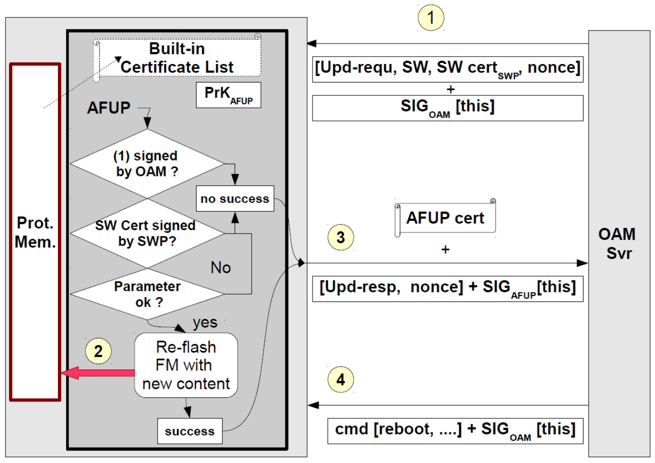
1
Built-in Certificate List
PrK_AFUP
AFUP
[Upd-requ, SW, SW cert_SWP, nonce]
+
SIG_OAM [this]
(1) signed by OAM ?
no success
SW Cert signed by SWP?
Prot. Mem.
AFUP cert
OAM Svr
3
+
No
Parameter ok ?
[Upd-resp, nonce] + SIG_AFUP[this]
yes
2
Re-flash FM with new content
4
success
cmd [reboot, ...] + SIG_OAM [this]

Figure 9. Example for AFUP communication protocol

In Figure 9, an implementation example is shown detailing the AFUP communication relying on a certificate based security protocol. According to this the AFUP controller is personalized with a (well protected) private key and a built-in, write protected certificate list (this list could be stored, e.g., in the protected memory), which initially have to be implanted by a secured process, e.g., during manufacturing. The certificate list might contain the manufacturer's root CA (Certificate Authority) certificate under which certificates for a SW provider are issued, denoted here as SW Cert$_{SWP}$.

In accordance with the scheme shown in Figure 8, the communication is started by a SW update request (1), sent by an OAM server, which typically is located in the operator network. This request may contain the new SW itself or may also provide a link to a location where it can be fetched from. The SW itself is protected by a signature, issued by the SW provider SWP and the associated certificate, which -in this example- is part of the update message .

To prevent from replay attacks and to assure a trusted source the OAM server adds a nonce and signs the message, which can be verified by a root certificate, which is stored in the AFUP's certificate list depending on the key material the OAM server is provisioned with. In the simplest case this could be the manufacturer's root CA certificate, too. But also an operator root CA certificate is imaginable.

Depending on the result of the verification and update process (2) enforced by the AFUP an update response message (3) is signed by the AFUP using its built-in private key PrK_{AFUP}. This also includes the nonce and additional parameters to assure freshness and to support this process by other, optional means (e.g., logging and confirming exact actions that have been taken by the AFUP, reporting of failure events, or even inserting time stamps if these can be provided).

Thus, the OAM server knows the exact state of the AFUP as well of the SW version stored in the network element and can continue with further service actions (which may or may not be transmitted over protected protocols, depending on the security relevance of such action), for instance by initiating a reboot process, as indicated by (4).

Note that such mechanism may involve (and support) additional security and key management processes, which imply, e.g., a secure time base (or at least monotone time counters) for expiry control or for revocation or secure processes to exchange the root CA certificates or private keys, in case this is needed. Also encryption of the SW transfer can be used if confidentiality of the SW is required. Realization and implementation of such issues is a matter of a refined security security specification, which is not further discussed here.

If not done during the reliable boot phase, *initiation of the AFUP* depends on the HW-SW function split and the CPU involvement for message transport, which at run-time may be influenced by SW attacks (potentially causing denial of service).

To prevent such influence the security design might rely on a more sophisticated realization of the AFUP process, in combination with autonomous basic communication capabilities. This would enable *reliable* enforcement of SW updates *at any time*, even in case the network element is compromised by dangerous attacks (e.g., remote SW attacks that however, cannot be directed against the AFUP mechanism if isolated by well designed logic). Accordingly, the AFUP supports remote remediation measures, which cannot be circumvented by such attacks.

B. *Protected CPU / Flash-Memory integration*

The solution presented above needs separate logic for the AFUP mechanism and in its simpler shape (without autonomous communication) it is mainly targeted to assure SW integrity through a (re-)boot process. As an alternative, we can also think of a more flexible realization, where the AFUP is realized by protected firmware being processed by the system CPU.

In the following, we present initial ideas for realization: For security reasons a suited CPU or CPU core is integrated together with an isolated Flash EEPROM, e.g., using Multi-chip modules or dedicated ASICs, as shown in Figure 10. The flash memory might only be accessible in a privileged CPU mode P1 (e.g., controlled by an MMU or by some logic).

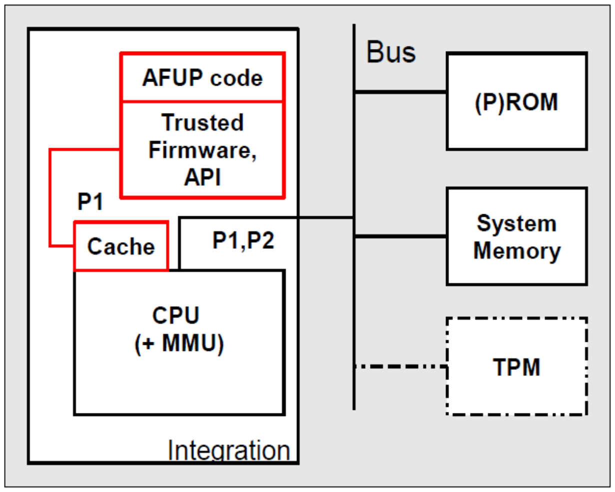

Figure 10. Flexible, protected CPU/Flash memory integration

Trusted functions can only be invoked via a protected API (e.g., by SW-interrupt), assuring that the CPU runs in P1 mode with specific security settings (e.g., indivisible operations, cleared CPU registers etc.). In P1 mode the CPU executes the AFUP process in accordance with the methods and protocols introduced in Figures 8 and 9. Neither external, nor remotely injected SW, nor a local attacker could read or modify any content of the protected flash memory, unless the integrated CPU-Memory device is physically analyzed, requiring extremely high efforts.

In addition to supporting secure boot and the AFUP mechanism this approach could also be used to allow secure run-time integrity protection. To that purpose, trusted API functions could be designed to run checks over parts of the memory content (declared to be invariant), during system operation. Moreover, 'executable' parts of the memory content could be checked and reloaded - periodically or based on events -, in order to wipe out potential hostile modifications that could have been injected during long time operation.

By expanding the above idea on the AFUP functionality for boot-time and run-time checking (of loaded SW) the functionality could also be extended to securely launch any security code (such as a crypto-algorithms) - or even small parts of sensitive general purpose code - at run-time, after successful validation of integrity and authorization. This would be an improvement over the DRTM idea, only allowing for *trusted measurements* on launched code. Such 'authorized SW' could (at run-time) be installed into the trusted memory and externally made available via an extended or updated API.

In addition, such SW could be associated with policies for usage and memory control (e.g., implemented as signed MMU instructions, which could not be changed by 'normal' user-land SW). This would enable a SW security designer to instantiate individual shielded areas of memory, for instance to read- or write protect memory areas being private to a certain SW module (e.g., to contain derived session keys or even secrets, which could be imported in encrypted form).

VI. PKI and Infrastructure Aspects

The presented SWIP concept essentially can be built on (proprietary) Signed Objects SO and on X.509 certificates assigned to signing entities. A X.509 compliant PKI hierarchy might be established, beneficially in manufacturer environment, together with manufacturer specific governance schemes for SWIP. The following observations may substantiate such reasoning:

PKI governance is executed essentially by applying policies associated with the PKI infrastructure, with regard to key and credential management, as well as by organizational control over the involved entities. In accordance with the principles mentioned in Section II.B and E, the conditions for SW signing necessarily must be aligned with the needs for products in mobile network, where each manufacturer individually is responsible for. One impact is the long-term usage (which may be 20 years and more), requiring, e.g., root CA certificates with long expiry periods and related security parameters and capabilities of involved keys. Despite long validity periods there must be an overlapping scheme of valid root CA certificates, which also implies secure exchange of these for products in field for a very long time. Typically this requires issuing of cross certificates as a base for (automated) secure exchange processes, be it via CMP [22] or be it by local means, and sufficient attack protection of the verifying endpoint storing the trust anchor.

Control over the root CA certificates in verifier components is a closely related issue. It must be assured that exclusively such root CA certificates (as well as all intermediate certificates) are accepted, which are compliant with the manufacturer's certificate policies. Such requirements are difficult to fulfill with 'public CAs' (but not impossible, depending on contractual conditions), which typically are designed to meet the requirement of distributed developer scenarios for products with shorter life cycles than those in network environment. Moreover, each product individually may set different conditions for validity (of the SO), for revocation and invalidation, and for SW management and versioning, as well as for the exact mechanisms and rules for verification and enforcement.

In addition to requirements for daily use, it also has to be assured that for exceptional cases (such as 'loss of key material' due to defects or in case of security incidents) disaster and recovery plans are in place and in emergency situations these can be realized very quickly. Even if such incidents (hopefully) are very unlikely to happen, customers may require related features.

Within the manufacturer's development infrastructure protected signing entities have to be established assuring proper usage of associated private keys to sign the SOs for the different products, in accordance with a secure approval work-flow. Such approval work-flow is required to avoid misuse of signing processes for other purposes than those intended by the manufacturer for an individual product. This not only involves personal responsibilities, but also security control such as by appropriate authentication and authorization principles.

Altogether, and in particular with regard to harmonization and generalization (i.e., the different use cases that should be covered) it seems to be the only economic (and perhaps technical) way that manufacturers themselves fully control the environmental conditions and policies for the SWIP infrastructure.

Following such principles the entire SWIP approach is self-contained and may be remotely managed without requiring new specific network infrastructure nodes, neither for modified MTM concepts for secure boot, nor for generalized use cases, such as SW installation or secure SW delivery.

Instead, processes running in TS, OAM or SW management systems might be adapted appropriately. We expect that this could be done in a manufacturer specific way, without the need to standardize commonly agreed solutions.

Note that with the presented approach also protection for SW coming from third parties could be integrated, applying suited extensions for *protection*, *verification*, *enforcement or infrastructure* processes, e.g., by a OEM sided sub-CA, by a manufacturer signed policy that allows a second root , by re-signing SW, or by cross signing of root CAs.

VII. Conclusion and future work

The authors feel that above concepts open a promising way to cover many use cases for SWIP with a harmonized, certificate based approach. It is suited both to cover requirements coming from 3GPP standardization, as well as those that in general increase SW security and reliability for SW products in mobile networks.

One essential benefit is that the same PKI and signing infrastructure could be re-used for many different use cases (e.g., secure boot, SW installation, or integrity monitoring), mainly determined by shaping content, syntax and semantic of SOs and by secure anchoring of adapted verification and enforcement components.

While key points are identified and promising ideas on HW level improvements are tangible (beyond the scope of the native AFUP functionality, as introduced in Section V.B), further research is required, in particular, to solve security issues emerging from cost effective implementation and from long-term operation of network elements.

In practice, trade-offs have to be balanced between achievable security level and efforts for additional HW or CPU modifications, which should be portable among different platforms and CPU types. In our future research work in ASMONIA these issues will be examined, also including virtualization principles. This will go in line with further detailing methods and mechanisms for smooth integration of SWIP management concepts into mobile network elements and security infrastructure.

VIII. Acknowledgments

Parts of the work presented in this article has been supported by the ASMONIA research project, partially funded by the *German Federal Ministry of Education and Research* (BMBF).

The authors acknowledge the incitations and assistance through the ASMONIA consortium and also like to thank their colleagues at Nokia Siemens Networks for the valuable ideas, discussions, and comments contributing to this work.

REFERENCES

[1] M. Schäfer and W.-D. Moeller, "Strategies for Managed Software Integrity Protection - Managing SW Protection and Trust in Mobile Networks", Proceedings SECURWARE 2010, Fourth International Conference on Emerging Security Information, Systems and Technologies; Venice, Italy, July 2010.

[2] 3GPP TS 33.401, 3GPP System Architecture Evolution (SAE), Security architecture; http://www.3gpp.org/ftp/Specs/html-info/33401.htm, last accessed: January 2011.

[3] 3GPP TS 33.320, Security of Home Node B (HNB) / Home evolved Node B (HeNB); http://www.3gpp.org/ftp/Specs/html-info/33320.htm, last accessed: May 2011.

[4] Trusted Computing Group (TCG), TPM Main Specification, Parts 1-3, Specification Version 1.2, Level 2, Revisions 103, July 2007.

[5] B. Kauer, "OSLO: Improving the security of Trusted Computing"; 16th USENIX security symposium, proceedings, pp. 6-10, August 2007; http://os.inf.tu-dresden.de/papers_ps/kauer07-oslo.pdf, last accessed: May 2011.

[6] R. Sailer, X. Zhang, T.Jaeger, and L. van Doorn, "Design and Implementation of a TCG-based Integrity Measurement Architecture". Proceedings of 13th Usenix Security Symposium, pp. 223-238, San Diego, California, August, 2004.

[7] Trusted Computing Group (TCG), Mobile Reference Architecture, specification version 1.0, revision 1, June 2007.

[8] Trusted Computing Group (TCG), Mobile Trusted Module (MTM) Specification, version 1.0, revision 6, June 2008.

[9] Trusted Network Connect; http://www.trustedcomputinggroup.org/developers/trusted_network_connect, last accessed: May 2011.

[10] W3C Recommendation, "XML Signature Syntax and Processing (Second Edition)", June 2008; http://www.w3.org/TR/xmldsig-core, last accessed: May 2011.

[11] 3GPPP (3rd Generation Partnership Project), http://www.3gpp.org/, ast accessed: May 2011.

[12] J. E. Ekberg and M. Kylänpää, "Mobile Trusted Module (MTM) - an introduction", Nokia Research Center Helsinki, Finland, NRC-TR-2007-015, 2007.

[13] Intel® Trusted Execution Technology (Intel® TXT), Software Development Guide, Measured Launched Environment Developer's Guide, December 2009 (in particular, see Section 1.2.1 therein).

[14] ASMONIA, "Attack analysis and Security concepts for MObile Network infrastructures, supported by collaborative Information exchAnge", BMBF sponsored project; since September 2010, http://www.asmonia.de/, last accessed: May 2011.

[15] ITU-T Recommendation X.509, "Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks", 2008-11.

[16] Online article, "Using Md5 checksums", http://www.openoffice.org/dev_docs/using_md5sums.html, last accessed: May 2011.

[17] Online article, "Maximum RPM: Taking the Red Hat Package Manager to the Limit", Chapter 17. Adding PGP Signatures to a Package, http://www.rpm.org/max-rpm/s1-rpm-pgp-signing-packages.html, last accessed: May 2011.

[18] Entrust Certificate Services, "Java Code Signing",User Guide http://www.entrust.net/ssl-resources/pdf/ECS_Java_Code_Signing_Guide.pdf , November 2010, last accessed: May 2011.

[19] Online Article, "SymbianSigned", http://wiki.forum.nokia.com/index.php/Category:Symbian_Signed, last accessed: May 2011.

[20] K. E. Sanders, SANS Institute, InfoSec Reading Room, Understanding Lotus Notes Security; Execution Control List (ECL) Settings, http://www.sans.org/reading_room/whitepapers/commerical/understanding-lotus-notes-security-execution-control-list-eclsettings_785, last accessed: May 2011.

[21] Cryptographic Message Syntax (CMS), IETF document, Network Working Group, September 2009; http://tools.ietf.org/html/rfc5652, last accessed: May 2011.

[22] Certificate Management Protocol (CMP), IETF document, Network Working Group, September 2005; http://tools.ietf.org/html/rfc4210, last accessed: May 2011.

[23] R. Wichmann, "The Samhain Host Integrity Monitoring System", Samhain User Manual, 2002-2009; http://www.la-samhna.de/samhain/MANUAL-2_3.pdf, last accessed: May 2011 .

[24] A. Apvrille, D. Gordon, S. Hallyn, M. Pourzandi, and V. Roy, "DigSig Novelties", Libre Software Meeting 2005 – Security Topic, slides, July 4-9 2005.

[25] L. Catuogno and I. Visconti, "An Architecture for Kernel-Level Verification of Executables at Run Time", The Computer Journal, Oxford Press, Vol. 47, no. 5, pp. 511-526, September 2004; also: http://www.dia.unisa.it/~luicat/publications/tcj04.pdf, last accessed: May 2011.

[26] T. Schluessler, H. Khosravi, P. Rajagopal, R. Sahita, G. Nagabhushan, and U. Savagaokar, "OS Independent Run-Time System Integrity Services", Corporate Technology Group, Intel Corporation, 2005, see http://www.thefengs.com/wuchang/work/courses/cs592_spring2007/SystemIntegrityServices.pdf, last accessed: May 2011.

[27] ITU-T Recommendation X.500, "Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services ", 2008-11.