D3.1.1
NED specifications (alpha version)

Project number	611458
Project acronym	SECURED
Project title	SECURity at the network EDge
Project duration	36 months (1/10/2013–30/9/2016)
Programme	FP7 (Collaborative Project)

Deliverable type	**R** - Report
Deliverable number	D3.1.1
Version (date)	v1.1 (18/07/2014)
Work package(s)	WP3
Due date	30/06/2014 – M9

Responsible organisation	HPLB
Editor	Adrian L. Shaw
Dissemination level	**PU** - Public

Abstract	This document describes all the elements needed to create a NED capable of hosting multiple personal execution environments (the PSC, Personal Security Controller) to execute one or more PSA (Personal Security Application). The general architecture is described and the essential design requirements for the NED are detailed (for the monolithic version, both on custom and commodity hardware).
Keywords	NED, TVD, PSC, PSA, definition, specification, security, virtualization, component architecture, interfaces

Editor
Adrian L. Shaw (HPLB)

Reviewers
Antonio Pastor (TID)
Antonio Lioy (POLITO) – quality control

Contributors
Adrian L. Shaw (HPLB)
Ludovic Jacquin (HPLB)
Marcelo Yannuzzi (UPC)
Diego Montero (UPC)
Roberto Sassu (POLITO)
Fulvio Risso (POLITO)
Antonio Pastor (TID)
Francesco Ciacca (BSC)
Mario Nemirovsky (BSC)

Acknowledgement
This work was partially supported by the European Commission (EC) through the FP7-ICT programme under project SECURED (grant agreement no. 611458).

Change Log

Version	Date	Note	Author
v1.0	27.06.2014	Base version	A.L.Shaw
v1.1	18.07.2014	Quality control	A.Lioy

Executive Summary

This specification highlights the general architecture for a SECURED Network Edge Device (NED), which is capable of 'off-loading' security functionality away from an end user device and into a secure hosting compartment within the NED. Here we describe the functional components with their associated roles and general interfaces. In addition to the logical architecture, we provide a partial reference implementation, which provides examples to aid vendors and developers in developing a SECURED compatible NED.

This alpha specification is subject to change in future versions, and has not yet undergone a full security review or assessment which will be required as the architecture and specification evolve.

Since it is the ambition of the SECURED project to provide state-of-the-art NED features, such as trustworthiness assessment and live mobility, we provide an early set of the requirements for these features. However, these advanced areas will require substantial research and evaluation before further specification can be made. The appropriate techniques for measurement and for migration of user applications will be defined as part of the full NED specification release (D3.1.3).

Contents


SECURED

1 Introduction

The protection of Internet-connected devices, such as mobile phones and laptops, has typically been achieved through installing appropriate specific tools on each device (e.g. a personal firewall, malware analyser, parental control application). However, this raises several issues: privileged access on the device is often required, tools may use a large amount of computational resources, platform capabilities vary, and appropriate tools may be unavailable on particular systems. Moreover users needs to configure each security control on each of their devices, resulting in an unsolvable maze for non–technically savvy users. This results in ineffective or inconsistent protection for users who use different devices across different networks (e.g. border firewalls are available in corporate Wi-Fi environments, but the user is not protected over a 3G network). In addition to this problem, there is an escalating number of embedded devices connecting to the internet, of which many contain little to no security installed. This includes sensors, controllers and appliances which fall under the broad definition of the Internet of Things (IOT). Furthermore, the inconsistent set of security controls for a large number of devices also creates a policy management and enforcement nightmare in an age where Bring Your Own Device (BYOD) is becoming increasingly popular in the workplace.

The SECURED project aims to provide an innovative architecture to achieve device protection from Internet threats by offloading execution of common security applications away from user devices and into a programmable device at the edge of the network. This *Network Edge Device (NED)* hosts a common set of a security applications, providing a uniform security control point for all connected devices. These secure and trusted NEDs can act in several different scenarios, such as in home gateways, enterprise routers, mobile or public access points.

The architecture allows for different users (e.g. individual users, corporate ICT managers and network providers) to install on-demand and execute *Personal Security Applications (PSA)* within their own trusted and virtualised execution environment. The NED securely hosts and isolates the user environments, as well as isolates the traffic flows of different users. Users are able to configure their PSAs through their own trusted *Personal Security Controller (PSC)* on the NED. Thus, the security architecture of the NED must be carefully designed to securely address the different requirements. An overview of the general threat model for the NED is introduced in Section 2 and the security and functional requirements are detailed in Section 3.

The project anticipates different flavours of a NED, which can cover many different network scenarios. For instance, one flavour is a discrete NED which performs both computation inside a single dedicated network device, which we refer to as a *Monolithic NED*. Alternatively, a complete software-based NED could be hosted on conventional computer hardware, such as a server or a spare personal computer with sufficient computational ability; located in the same network as the protected devices, the user only diverts his traffic to it before it is processed and forwarded to the network. We refer to this as a *Virtual NED (vNED)*. These NED topologies are depicted in Figure 1 and 2. Overall, both these NEDs share the same logical architecture and specification. There are also other models initially foreseen in the project: a split-NED, where network traffic is redirected to the NED based on the Openflow protocol, and remote-NED, which is an ordinary NED but is connected through a traditional VPN through any untrusted network (where the NED is not the direct gateway). These are not discussed in this alpha specification but the architecture has been designed to easily absorb these models in future updates.

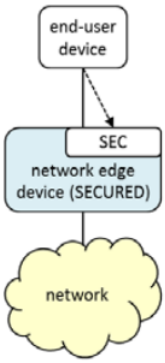

Figure 1: Monolithic NED (network equipment)

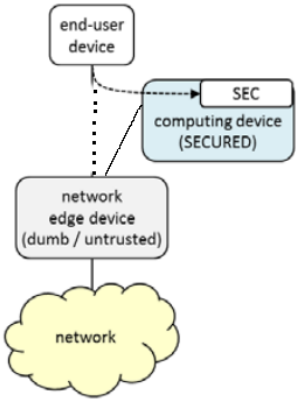

Figure 2: Virtual NED (commodity computer)

In addition to the SECURED NED, there are also external entities, such as the *PSA Manager (PSAM)*, the *PSA Repository (PSAR)* and the *Security Policy Manager (SPM)*: these are centralised services for users to download or manage their PSAs and security policies. Users will be able to register with and login to these SECURED services and be able to choose the required high-level policies to be enforced or alternatively manually select the PSAs of their choice. The definition of these services are not covered as part of this deliverable, but will be specified in future documents. Requirements for building trusted PSAs, the definition of the respective SDK and runtime interfaces are described as part of D5.1.

Section 4 describes the NED software architecture, the functional NED components, and a reference architectural guide on how developers can build a monolithic or virtual NED from currently available software components.

Since the NED hosts the security applications away from the user device, the NED must be both secure and trusted. Section 5 describes early requirements for a Trusted NED from a firmware perspective and how to build trusted isolation and software services that run on the NED. Section C provides the formal definitions for all the NED interfaces, the details for the control and management plane and the basic runtime interfaces needed to support the I/O model of the PSAs.

2 Threat model and security assumptions

In this section, we analyse the possible threats that may affect the behaviour of the NED in a way that the latter does not process the traffic of a SECURED user correctly. While section 1 introduces the general way SECURED addresses the user devices security, this section details the external threat model a NED faces.

2.1 Interfaces

Basically, there are five main interfaces that could be used to conduct an attack:

- **user config interface** – this is the interface through which an user requests SECURED services and negotiates a secure data connection;

- **user data interface** – this interface is used for the transmission of the user traffic from or to the NED through the negotiated secure data connection;
- **user profiles interface** – through this interface the NED acquires (from the internal network or a user-provided repository[1]) either the user policies and the applications needed to implement those policies;
- **NED admin interface** – this interface allows the NED owner or an individual with the necessary credentials to access the lowest software layer, i.e. the operating system and the SECURED services (e.g. the PSCM). Through this interface the NED owner can perform remote management of the NED or obtain a remote console;
- **NED physical access interface** – this is a physical interface (e.g. the case of an access point) that allows an user to gain access to hardware components and, eventually, to a serial console allowing to execute highly privileged commands (e.g. replacing the firmware).

2.2 Attacks definition

In Figure 3, we report the possible attacks that may be conducted by the depicted actors through the five interfaces listed above. These attacks are due to a data exchange between an actor and a NED and we assume that the communication through all the NED interfaces is always protected for confidentiality and integrity (this excludes that a third attacker is acting on behalf of one actor by intercepting or modifying the data in transit). The problem of identifying a NED and determining the channel protection needed for a specific link (e.g. cable, wifi) are addressed by the WP2.

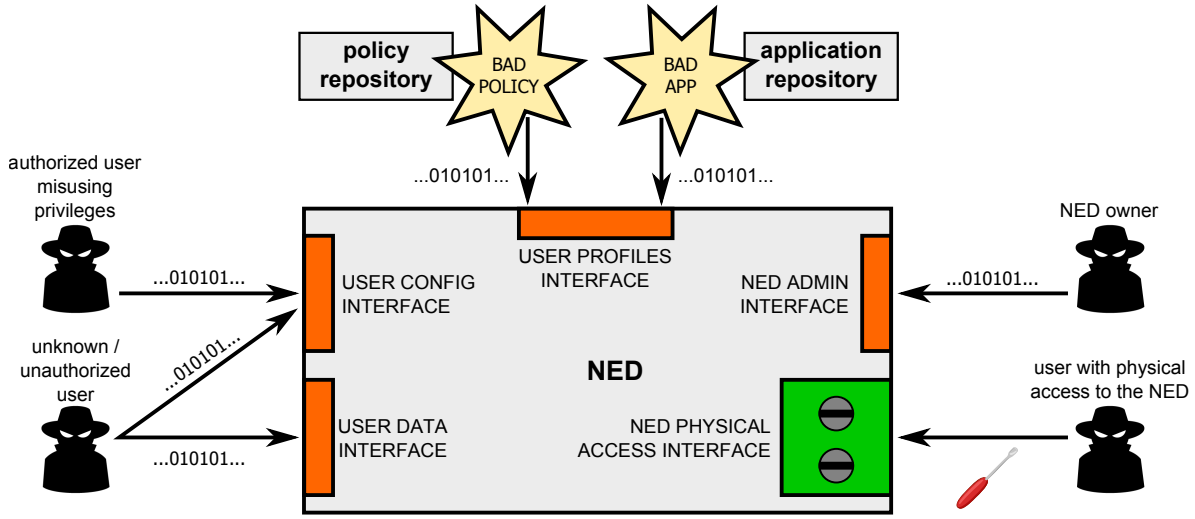

Figure 3: NED threat model.

The main attacks are described in the following.

Attack 1. An unknown/unauthorized user tries to impersonate another user that can legitimately access SECURED services. This attack may lead to accessing the policies and applications of the attacked user or to generate network traffic outside a NED with a falsified identity.

[1] This interface is not meant to be accessed by users.

Attack 2. An authorized user may misuse assigned privileges to alter the network traffic processing of other users connected to the NED. Depending on the scenario, the NED owner, the Internet Service Provider (ISP) or the government have the right of imposing some policy to regular users. In this case, an individual of one of the above categories may request the processing of users network data for a purpose that falls outside the correct behaviour (which may be defined in a contract). For example, a government employer may collect data for business purposes.

Attack 3. Since a user can specify his favourite policies and applications repositories, it may ask the NED to download a malformed data (policy or application). With this attack, a user may try to directly take the control of a NED (for example by exploiting a vulnerability on a SECURED service) or may try to intercept or modify the traffic of other users connected to the NED, if the latter supports multiple tenants.

Attack 4. A malicious NED owner modifies the software running on the NED (the operating system or a SECURED service) to alter the behaviour of the latter. This event has an impact on all users accessing SECURED services as the NED owner has the highest level of privilege on the software in execution at the NED.

Attack 5. A user with physical access to the NED can modify the behaviour of hardware components. Furthermore, it can access a serial console (most devices offer this interface for maintenance reasons) to access the NED software with the same level of privilege of the NED owner.

Appendix A details an extensive review of every attack for each deployment scenario.

2.3 Assumptions

Attacks 1, 2, 3 and 4 are addressed by SECURED with appropriate software techniques that allow a user to identify if a NED is processing the traffic as expected. Instead, SECURED will not address the Attack 5 as it is known that the Trusted Platform Module (the core element used to provide security guarantees to SECURED users) is vulnerable to hardware attacks [1]. If this attack occurs, a SECURED user would not be able to detect a NED misbehaviour.

In our view, this assumption is acceptable by requiring that a SECURED user and the NED owner stipulate a contract that severely punishes an individual breaking the hardware.

As a last remark, in the threat model we did not list possible attacks that may undermine the availability of the SECURED services. NED devices for carrier-grade needs, like those for ISP or Enterprise companies, could include availability protections needed for hardware failures or physical attacks: power, network cards, processors, etc. All this redundancy hardware will be assumed as available and not part of the specification.

3 Security requirements overview

In this section, we detail the NED requirements implied by the overall SECURED architecture presented in Section 1. The three main points are the user–NED link, the authentication procedure and the isolation constraints.

3.1 Client side SECURED application for user devices

The SECURED application is an optional software that runs on the end user device and acts as a way for the user to gain the following additional functionalities:

- secure channel establishment when it is not provided by the underlying network layers (e.g. the VPN to the remote NED);
- support for querying (or even verifying) trustworthiness of a NED and the user's PSC, PSAs and underlying execution environments;
- execution of the authentication procedure;
- feedback about security applications and the policy status.

3.2 Secure channel

One main goal of SECURED is to offload part of the user security at the network border, but this means that the user is exposed to the devices within his local network: this is at its height in the public WiFi hotspot use case. A secure channel must at least provide the following properties:

confidentiality, a third party must not be able to access the data;

integrity, the traffic cannot be tampered with;

mutual authentication, the identity of the NED is verified so that the NED cannot be impersonated and the user must authenticate himself before using the NED.

3.3 User authentication

The user should be able to authenticate himself to the NED, in order for it to securely access and retrieve the user policies and applications. For this to happen, the NED should be both secure and trusted to handle the authetication procedure in a trustworthy manner. Upon success, the authentication system will return a unique token for the user session. A Single-sign On (SSO) mechanism is anticipated to be a part of the bigger architecture of SECURED, where the user only needs to provide his credentials once in order for the NED to interact with the external centralised services. These services and the interlinking authentication system will be described in D5.2.

3.4 Compartmentalisation

One of the main feature of the overall SECURED architecture is the potential multi–tenancy aspect of a NED; where users must be cut off from each others as if they were using their own standalone NED. This should be achieved through the use of a compartmentalisation system, where potentially users are each allocated a compartment. Alternatively, one compartment could service multiple end user flows, provided that user flows are sufficiently isolated to address privacy and security concerns.

Should a user compartment fail, the compartmentalisation system must be sufficiently robust such that the failure does not affect the overall service of the NED platform or affect the services of other user compartments. Similarly, each application deployed inside a user comparmtent must be isolated from

the user's other applications in order to prevent any interference in application behaviour. Furthermore, should a failure occur in a user's compartment. Therefore, the NED should enforce user compartment and application separation by providing two isolations layer as depicted in Figure 4. The isolation mechanisms are crucial to the way SECURED addresses some of the threats it is design to protect against. The user should be able to remotely assess the trustworthiness of the compartmentalisation system, which should provide the user a proof that sufficient isolation is being enforced.

Ideally the compartmentalisation layer should only be composed of user-independent components, whilst every user-specific software will be part of one of the user compartments. The second isolation layer is called a *containment layer* and is mainly used to isolate the user's privileged controls from the generally unprivileged applications.

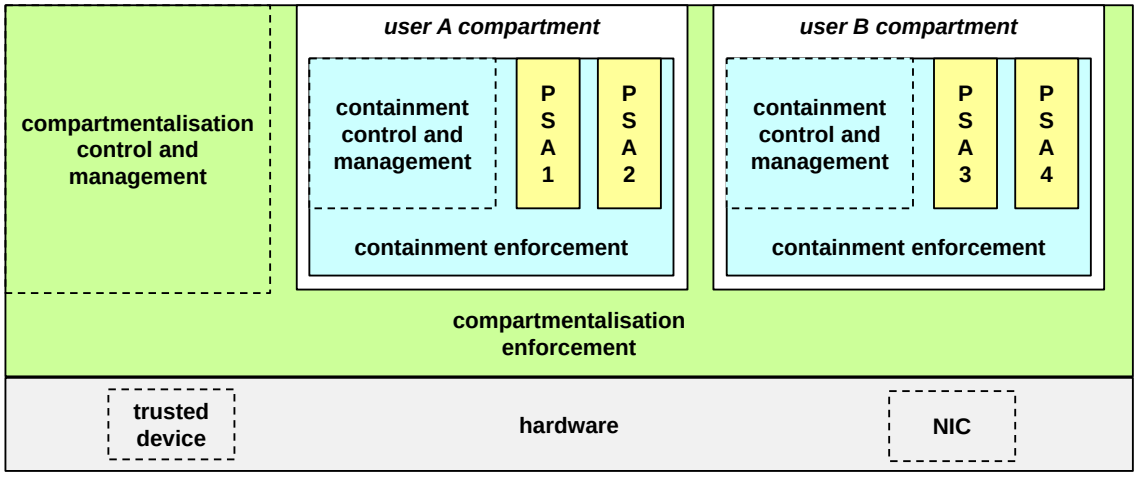

Figure 4: Isolation of users and applications inside a NED.

3.5 Trusted Execution Environment (TEE)

Each user compartment contains an execution environment. This environment must be secure enough and trusted to separate concurrent applications from interfering with each other. It needs a privileged part in order to measure, configure and start applications. The applications are then deployed in an unprivileged area of the TEE.

3.6 Trusted Virtual Domain (TVD)

In addition to a minimal trusted virtual machine template that SECURED shall promote, there is a need to provide support for existing applications that run on other platforms. The architecture must be flexible enough, given enough resources, to support multiple execution environments for providing services to the user which run on specific platforms. This can be achieved through the use of a virtualised infrastructure. Therefore, it will be possible for a user to potentially have multiple SECURED flavours of application environment in the case of hosting computational NEDs.

In order to support multiple TEEs for one user, we can encapsulate them in a user TVD, a logical container of one or more TEEs that belong to a user, all deployed under a private network subnet. This private subnet is unique to the end user and enforces isolation of traffic flows within the NED.

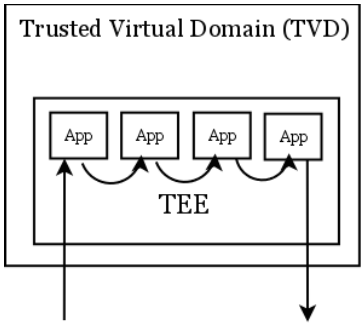

Figure 5: Standard TVD.

Furthermore, it is possible that many different topologies inside a TVD could be made. For instance, instead of running multiple applications inside a TEE within a VM, it could be possible to chain multiple VMs, each of which includes one application service or virtual network function. This is depicted in Figure 6, which is a similar concept to the approach of Network Functions Virtualisation (NFV), with the addition of strictly enforced static chaining. In order to provide isolation guarantees with regards to the user TVD and their traffic flows, we do not consider dynamic flows at this moment in time.

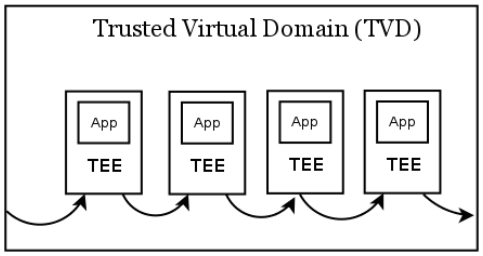

Figure 6: Alternative TVD using miniature VMs, each hosting a Virtual Network Function (VNF).

4 NED architectural model

This section details the overall NED architecture for SECURED and the required components, with particular emphasis on the PSC Manager, the PSC, and the TVD. The general component roles and their required interfaces are described at a functional level, with reference implementation examples given where appropriate. Details of interfaces are listed in Section C. This specification holds a common architecture for NED hardware that covers a) a *Virtual NED*, which is designed to run on commodity computer hardware and b) *Monolithic NED*, which is a network device augmented with substantial computational capabilities. A strong compartmentalisation architecture has been developed based on the requirements made in Section 3.

As far as trust is concerned, the NED architecture is flexible and thus would be able to satisfy several different user needs. Users can trust their network provider and thus not question it for hosting their security applications, or have a partial trust and thus require execution of a remote attestation procedure before executing their applications, or completely dis-trust the provider and hence use either their own local NED or a trusted remote one. The specifications for a Trusted NED are listed in Section 5.3.

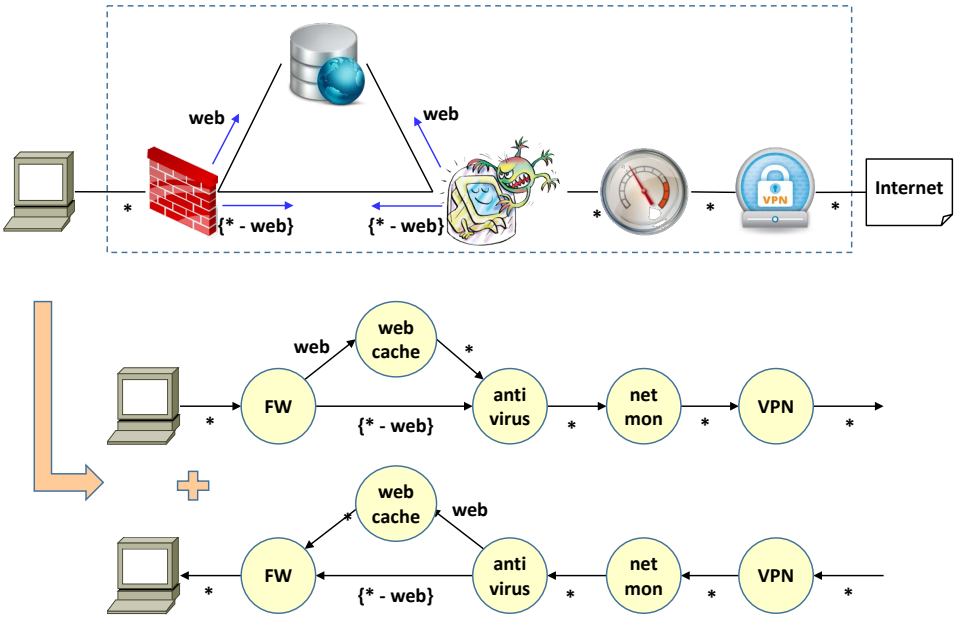

Figure 7: Service graph.

4.1 Service graph creation

The *Service Graph (SG)* is the formalism that describes the service requested by the user and it is, in its simplest form, a set of cascaded PSAs active on the user's traffic. The service graph can be either specified by the user in case of the application-driven service model (e.g. an experienced user able to select the PSA that have to operate on his traffic and their service order) or calculated automatically, e.g. in case of a basic user. In case of the application-driven model, the service graph is stored in the user profile database, while in the latter case we expect the user to specify the service by means of a set of policies, while the selection of the proper applications that are required to satisfy those policies (and, consequently, the responsibility to calculate the service graph) is left to the network operator.

Formally, the service graph is defined as the superposition of two directed graphs in which nodes represent the PSA and arcs represent the flow of the traffic between the two nodes. Arcs can be labeled with a set of packet filtering rules (e.g. OpenFlow Flowmods) that select which traffic has to flow through that arc in that direction. The initial/final arc of the service graph terminates on the Service Access Points, which at first sight can be considered (*i*) the user terminal and (*ii*) the Internet. In fact, service graphs are composable in order to allow the traffic to traverse the service graphs of different actors (e.g. the end user, the service providers, etc); the actual service experienced by the user results from the concatenation of all the service graphs of the involved actors. The composition of the service graphs is carried out by merging the IN/OUT service access point, according to the order given by the priorities of the different actors.

The service graph is intended to describe the *service*, not *how* the service will be provided. As a consequence, no physical information are present such as physical/virtual ports, applications/VNF placement, topology, and other. This implies that the service graph will be translated into a lower-level graph (the *Infrastructure Graph (IG)*, which is implementation-dependent) by the PSCM, which adds the new information needed to instantiate the proper components on the target NED.

An example of a possible service graph is depicted in Figure 7.

4.2 General SECURED components

Whereas section 2, particularly Figure 4, focusses on the users and applications isolation requirements, we now detail the different functional components of a NED, illustrated in Figure **??**. In this model, there are two kinds of components:

i) the user–agnostic components, mainly composed of the NED hardware/firmware, the SECURED compartmentalization layer and the functional components that compose the Personal Security Controller Manager (PSCM);

ii) the user–specific execution environment which is more ephemeral, in the sense that it lasts only during the time of the connection.

4.2.1 Personal Security Controller Manager (PSCM)

The *Personal Security Controller Manager (PSCM)* is the set of components in charge of communicating with the user at the control level (thus the user traffic that will be processed by the PSAs does not pass through the PSCM). Internally, it is mainly connected to the control and management plane network which also connects the NED to the SPM and the repositories (Figure 8).

The PSCM leverages two main sub components: the remote attestation agent and the authentication module, that are used at the early steps of the user connection (namely the NED attestation by the user and the user authentication by the NED). Once these steps are performed and the user is granted access, the PSCM transfers the user session token to the orchestrator, which will continue the user TVD instantiation. Finally, the PSCM reports the initial feedback of the TVD to the user after receiving it from the orchestrator.

The PSCM Remote Attestation agent.
The *Remote Attestation agent* is the first component reached by the user SECURED application. It is in charge of the first step attestation that focuses on the user agnostic part of the NED (see Section 5.1.1 for more details). Specifically, it exposes to the user the measurements stored inside the Trusted Device of the NED. The measurements must provide to the user a proof of the trustworthiness of the PSCM and that the compartmentalisation layer is enforcing user isolation.

The PSCM Authentication Module.
The *authentication module* is the software component which handles user authentication during the second step of the initialisation of his PSC and PSAs.

We foresee four ways to authenticate a user:

- *local authentication*, the module manages a small local database of user credentials and directly authenticates the user;
- *remote SECURED authentication*, the module contacts an external service (managed inside the SECURED infrastructure) providing access to the authentication credentials, either directly or indirectly (e.g. LDAP or RADIUS authentication);
- *remote external authentication*, the authentication is performed by an external third-party authentication system, to permit users to exploit an existing credential (e.g. oAuth or OpenID);

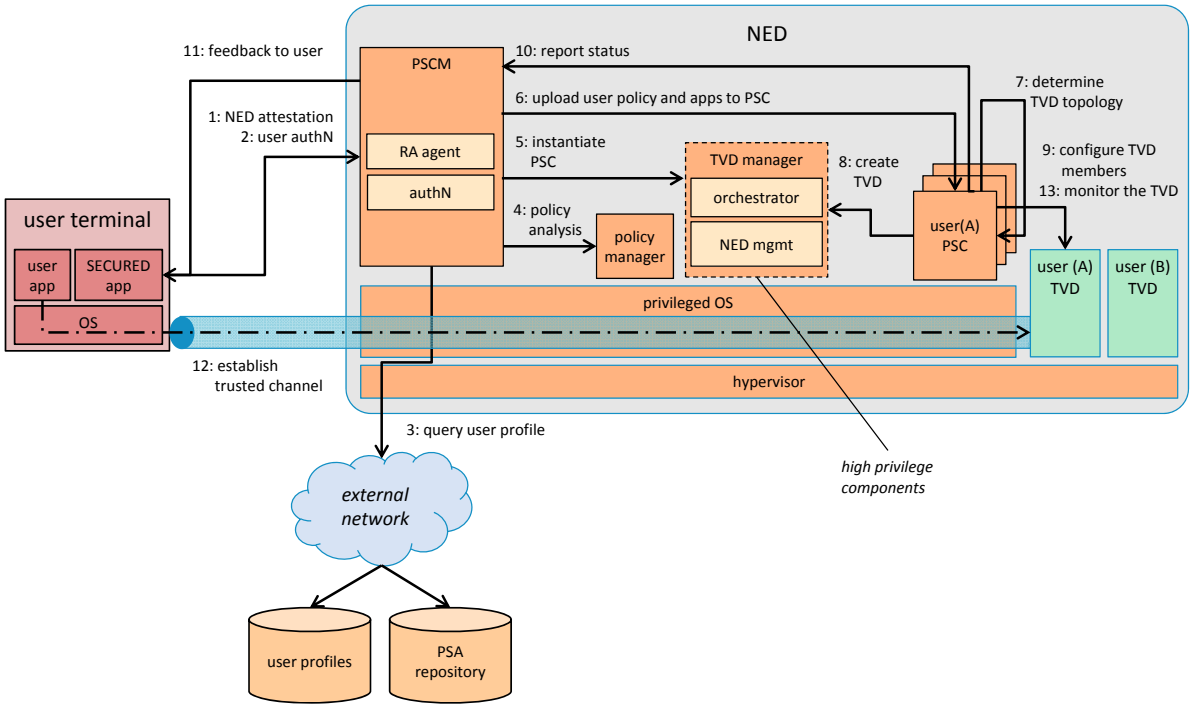

Figure 8: Logical NED architecture.

- *inherited authentication*, the user canbe authenticated automatically by inheriting his network access credential (if access is performed via a L2 network layer supporting network access authentication, as is the case with 802.1x).

NED Policy Manager

The policy manager is the NED component in charge of modifying the user service graph and policies according to the locality of the NED. For example, it can extend the high–level user service graph to include the company, ISP or government policies, or perform the reconciliation of these policies with the user ones.

NED orchestrator

The orchestrator is the decision–maker component that allocates resources to a user TVD and creates the connections of this TVD (and its TEEs) to the data plane networks. It does not store any user context, but must be able to easily retrieve the NED internal topology (e.g. networks configuration and virtual machines).

It rests upon the NED management to enforce its decisions and uses the control plane network to retrieve the needed user informations from the SPM, thanks to the user session token. The user informations useful to the orchestrator are stored inside the user profile, but the orchestrator will also need to locally load the user's PSC instance, fetch the PSAs and policies, but must not perform any action on them (apart from transferring them to the appropriate component).

NED management

The NED management component is the technology-dependent part of the platform that is privileged

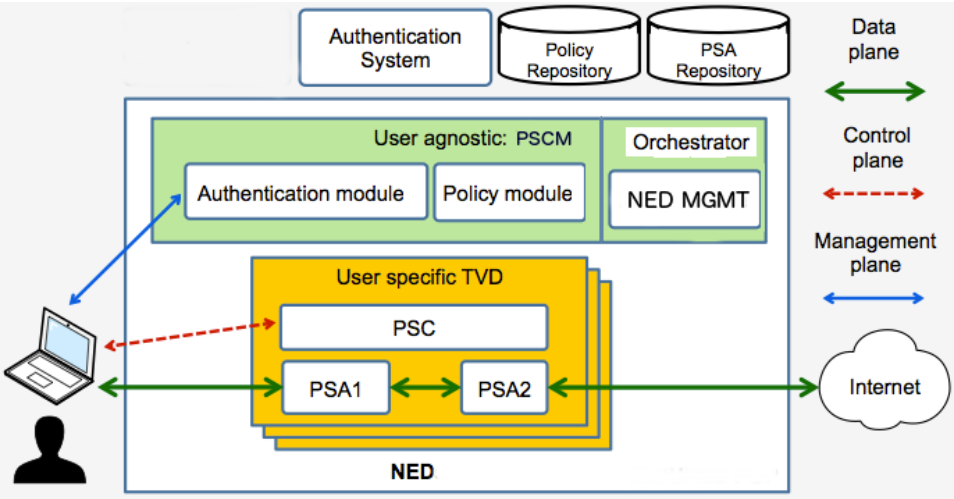

Figure 9: PSC manages PSAs and their TEEs, Orchestrator performs privileged NED operations.

enough to enforce the decisions of the Orchestrator. There are two main functionalities: management of TVD/TEEs and of the configuration of the virtual networks within user TVDs.

4.2.2 Personal Security Controller (PSC)

The *Personal Security Controller (PSC)* is the component which controls and manages the TVD environment, user applications, configurations and network interfaces on behalf of a single user. Each user gets their own instance of a PSC inside his TVD, which captures all the user state and context for their session. Since the PSC is privileged to load applications and enforce configurations, it is by design kept isolated from the applications, and all applications related to the user must communicate to their PSC through a restricted *Control & Management* interface. However, changes to the overall TVD (such as creating a new TEE) require sending a request to the Orchestrator, which is an external privileged entity outside the TVD.

Main roles:

- stores the abstract service graph (PSAs and interconnections) of the user;
- determines the TVD topology (realization of the service graph) from PSAs requirements (it may change depending on NED resources);
- sends the TVD topology to the orchestrator (the latter contains all TVDs topology and how a TVD is connected to another);
- monitors the status of the TVD (e.g. detecting if the TEE has crashed and needs restarting);
- PSC receives the manifests of all the PSAs and assigns each PSA to a TEE.

The PSC is internally divided into two main parts (with respectively two different interfaces, later described in the interfaces section): *Control & Management*.

In the following there is a brief description of the functional requirements of these two components.

TEE management

- control of the execution state of the PSAs (start, stop, restart);
- push the initial configuration of the PSAs;
- receive the low level configurations for the PSAs from the Orchestrator;
- PSAs chaining topology retrieval and enforcement.

TEE control

- monitoring information relay for listing the execution state of the PSAs;
- static attestation of the PSAs;
- PSAs exception handling which can include execution exception (crash, failures) but also flow-related information (malware detected, intrusion detected, illegal access).

For instance, each TEE has its own virtual switch (not optimal for performances, but isolation friendly) - the *management* will configure the switch whereas the *control* will handle any switch failure (eventually using the management commands such as start or reboot).

4.2.3 Personal Security Application (PSA)

PSA is the atomic entity of the SECURED architecture. It is responsible to enforce one or multiple user's security policies; each PSA can be composed by a series of security controls. PSA is exposed to primitive operations oriented to network processing, such as packet filtering and segment/payload reassembling. Each PSA must be accompanied by its security manifest declaring the capabilities it is expected to provide.

A complex security policy can be enforced by a single PSA or by connecting multiple PSAs. The chaining could be between PSAs running inside the same TEE or between PSAs running in different TEEs but belonging to the same TVD. PSAs chaining can be associated to the concept of Service Function Chaining (SFC - IETF). A set of PSAs properly connected enforcing a security service implements the concept of PSA bundle.

Into the SECURED architecture PSAs run in a non-privileged execution space, isolated from the other PSAs and softwares of the TEE by the containment technology; multiple PSAs can run inside the same TEE (belonging to the user's TVD).

PSA Architecture PSA internally should be divided into two planes: Control–Management Plane and Data Plane (Figure 10 and 11).

- ***Control&Management Plane*** is the piece of software providing interfaces towards the Control&Management part of the TEE for configuration, monitoring, signaling and attestation requests from/to the PSC.

- ***Data Plane*** is the part of the PSA which is in charge of actually processing the traffic on/offline, providing interface to the communication layer of the execution environment and allowing packet exchange between different PSAs and also externally. The internal logic of the PSA can be split in different modules.

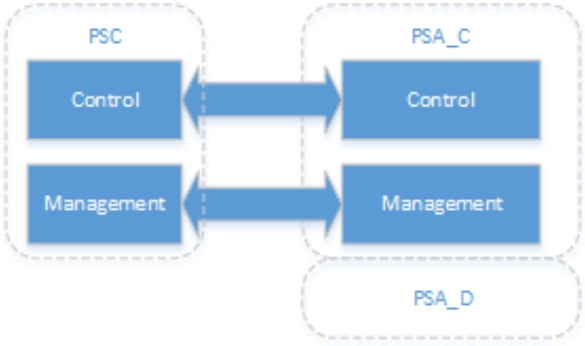

Figure 10: PSA Control Plane

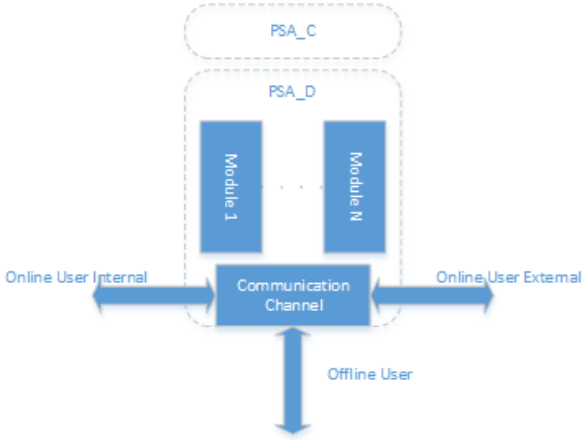

Figure 11: PSA Data Plane

PSA and policies PSAs play an important role into policies refinement and enforcement. The approaches to the policy refinement process are two:

- ***Policy driven*** The user defines his set of security policies through an high level security policy language (HSPL) and optionally can configure some settings through the medium level security policy language (MSPL). This option allows the SECURED infrastructure to decide which PSAs will enforce the policies, even based on some trade-off parameters that can be user defined (cheaper solutions, better performance/throughput, ...).
- ***Application driven*** The user can decide which applications will enforce his security policy. Concurrently he can define through the MSPL the application's configuration. The application driven approach represents a more challenging scenario as it limits the system flexibility, not allowing the SECURED infrastructure to decide which is the best application able to enforce the requested policy in the current environment.

M2L service From an architectural point of view the policy refinement process introduces the need of another software component in charge of translating the MSPL derived to the actual PSAs configurations. This component is the M2L service (Medium to Low level). The M2L service is application specific and needs to be provided by the PSA developer. The SECURED design right now foresees two approaches for the M2L service:

- ***Internal M2L*** The application-specific service is developed as part of the PSA
- ***External M2L*** The application-specific service is developed as a plugin for a software component which is part of the SECURED infrastructure.

Details about the refinement procedures and M2L service are discussed in D5.1 from WP5.

Migration PSAs migration and design implications are treated in depth in chapter 6 of this document.

PSA Manifest The PSA Manifest is used to categorize the PSA and then inform the user about the set of features and functionalities the PSA provides; it provides information about PSA performance, security and mobility capabilities. From an architecture perspective it provides the computational requirements of the PSA. They include: the number of virtual CPUs required, memory footprint of the PSA, bandwidth requirements, execution environment requirements (e.g. kernel version, libraries and other dependencies).

4.3 Instantiation steps

These are the major steps for a user to open a SECURED–protected connection to the Internet using a NED, assuming his profile has already been created as described in section 1: More details (such as the sequence diagram in Figure 18) are provided in Appendix B.

1. **NED First-step attestation**: the user attests the NED identity and the user–independent components (PSCM, Policy Manager, Orchestrator, NED management, etc.) through the RA agent.
2. **User authentication**: the NED authenticates the user using his SECURED credentials; upon successful authentication, PSCM receives session token back.
3. **Request TVD+PSC creation**: the PSCM requests the Orchestrator to instantiate a basic TVD for the user (and passes the session token).
4. **User profile query**: the orchestrator uses the session token to fetch the user profile (for its PSC related section).
5. **Load PSC image**: the PSC image is fetched from local storage by the Orchestrator.
6. **TVD and PSC creation**: the NED management creates the TVD and the PSC is started inside.
7. **Determine profile type**: Determine if the user PSAs are policy–driven or app–driven.
8. **Local policy reconciliation and PSA config transfer**: If needed, then just-in-time policy reconciliation is performed by the local Policy Manager. Afterwards, infrastructure graph, user's session token and PSA configurations are sent to the User's PSC.
9. **Determine TVD topology**: The User's PSC now has the infrastructure graph and PSA configurations to determine the required topology for hosting the user PSAs.
10. **Request TVD expansion**: PSC sends a TVD Expansion Request to the Orchestrator to create a new TEE for the PSAs. This request is issued via the TVD MGMT interface.
11. **Grant TVD expansion**: Orchestrator creates the requested TEE in the user's TVD, based on the information from the user's TVD Expansion Request. TEE certificate is generated and is sent back to PSC as proof of instantiation.
12. **Fetch PSAs**: Upon successful TVD expansion, PSC issues request to the Orchestrator for the PSAs. Orchestrator downloads the user's PSAs using the provided session token. PSAs are given to the PSC.

13. **PSA setup and Second-step attestation**: PSC will set up the TEE appropriately and the TVD network configuration (using the Ctrl/Mgmt interface) and install the PSAs in correct place based on information in its own infrastructure graph and according to the PSA manifests. All PSAs are measured and subsequently started.

14. **Generate status report**: A status report is generated including all PSA measurements and audit logs and is sent to the PSCM using the Ctrl/Mgmt interface

15. **Report sent to user terminal**: User terminal receives status report, verifies status report is OK, disconnects the PSCM connection.

16. **Secure datapath established**: User can now create a secure data channel to the NED, which is relayed by virtual switch to the first PSA in the User's TVD

Additionally, if supported, a NED will feed back the dynamic attestation to the user throughout his applications execution life cycle.

4.4 Reference implementation

This section presents a possible implementation for the monolithic NED using, when appropriate, building blocks that are derived from current available technologies. This choice aims at speeding up the creation of the prototype and to give to the project enough resources to focus on the most challenging scientific aspects of the problem.

Among the existing components, we outline here a design that exploits the services available in OpenStack, a popular software framework targeted to virtualised datacenters, which is able to provide a subset of the capabilities needed for the implementation of a prototype of the SECURED NED.

Please note that this is only one of the possible approaches to implement the NED architecture and it is in no way precluding alternative designs, that – especially for a monolithic NED – could possibly exhibit better performance.

The global architecture is depicted in Figure 12, limited to the components that are most significant for the implementation purposes[2].

4.4.1 NED compartmentalization system

The NED is executed on a trusted platform, featuring an compartmentalization system that provides virtualization capabilities, such as Xen or KVM. Our primary choice is Xen, which features a very compact hypervisor and a privileged execution environment (the so called `Dom0`) that allows to execute the most critical components in the above logical partition of the host. KVM represents a better choice for its easy setup and configuration and it is appropriate as long as we are able to trust and verify the entire host operating system with remote attestation technologies.

The user-oriented portions of our NED will be executed in different execution environments, most notably separate virtual machines running as guests (Section 4.4.4).

[2]Some logical components are not shown for the sake of clarity, although it is evident how they will be integrated with their companion block from the implementation perspective.

4.4.2 NED-wide components: PSCM, policy manager and TVD manager

Those modules represent the global components that have to be instantiated once in the NED and that provide different functions that are specific for our architecture. Therefore, although we foresee the possibility to reuse some components from other projects (e.g. the module in charge of the authentication), in general those modules must be written from scratch.

The TVD mananger (composed of the Orchestrator and NED management components) is in charge of managing the execution environments (Section 4.4.4). This could be done by exploiting the services of the `libvirt` library, which exports a set of functions for starting and stopping user execution environments for different compartmentalisation systems.

4.4.3 TVD

The Trusted Virtual Domain is a logical abstraction that is not mapped to any physical component in the NED and that provides a way to identify all the components that are under the responsibility of a single user. A TVD represents an isolated portion of the NED that is dedicated to a single user and it includes different execution environments and possibly multiple virtual switches (logical switching instances, Section 4.4.7) to implement the expected service chaining among the different PSAs.

The user TVD, a logical partition within the host, cannot be mapped on a precise component from the implementation perspective. Instead, the TVD can be considered as the set of components that belong to a specific user, and that correspond to the green portion of Figure 12.

4.4.4 Trusted Execution Environment (TEE)

The *Trusted Execution Environment (TEE)* is, in its base form, a virtual machine that hosts one or more components (PSC, PSA, etc.) under the responsibility of a single user and that can be executed within a single operating system. Execution containers can be implemented in different shapes, among the other:

- Full-fledged VMs: this represents the base form of an execution environment, where a guest operating system runs in a partition of the physical host. In this case, multiple operating systems can be supported, e.g. allowing some PSA to be executed in a Windows guest OS, while other are executed in a Linux guest OS, etc.
- Lightweight VMs: full-fledged virtual machines may be computationally expensive and require huge amount of memory/storage that may prevent to consolidate a massive amount of users on the same physical server. Lightweight virtual machines such as Linux containers (LXC), Docker, or the OSv operating system are more appropriate in that respect, although they introduce other limitations such as the type of PSA that can be supported (e.g. no lightweight solutions are currently available for running a Windows-based PSA).
- Language-specific VMs: this can be represented by solutions such as the Java Virtual Machine, which provides an even more lightweight abstraction at the expense of the compatibility of software written in a specific language. The additional problem of this solution is that it may be needed to extend the host operating system with proper primitives (e.g. Linux `cgroups`) in order to provide the proper computing/storage isolation between the different containers.

The number and type of execution containers is decided by the orchestrator, which analyzes the number and the characteristics of the PSA required to create the user's service graph and (possibly) groups together multiple PSAs that can be executed within the same execution environment (e.g. all the applications that run under Linux). This is shown in Figure 12 by the TVD of User (A), which features two different execution environments in order to create the requested service graph.

In a first phase of the project we will support only full-fledged VMs as execution containers, while we plan to support either Dockers or OSv in a later stage of the project.

4.4.5 PSA implementation categories

A PSA can come in different forms, depending on the characteristics of the application itself. Among the possible options:

- Standard executable: this represents perhaps the most lightweight form for a PSA, which is packaged as standard executable (e.g. an application running in a specific operating system). This can be seen by the case of EE1 in Figure 12, where multiple PSAs can be launched within the same execution environment.
- Full-fledged VM: this represents the easiest way to package a PSA, as each application runs in a different virtual machine. This allow each PSA to define its own execution environment, without any compromise with other PSA. This is shown by the TEE2 in Figure 12; in this case the LSI (Section 4.4.7) may not be even needed. In any case, the PSA, even if running alone in the execution environment, must be coupled with the proper management and control modules that provide the interfaces for handling the execution environment and configuring the PSA.
- Language-specific executable: this can be the case of a Java program, packaged as a JAR file that can be executed within a specific Java Virtual Machine.

The type of PSA and, consequently, the execution environment needed to execute the application, are a property of the PSA itself and are stored in the PSA repository.

In a first phase of the project we will support only full-fledged VMs as PSAs, while we plan to support more sophisticated (and more compact) PSA in a later stage of the project.

4.4.6 PSC and management and control for TEE

Those components are specific for the NED architecture and therefore should be written from scratch. Particularly, the PSC should be executed in a distinct execution environment for safety reasons (it has to control the entire TVD of the user), while the other block of management and control for the execution environment is executed within the TEE itself.

4.4.7 Logical Switch Instance (LSI)

In order to create the proper service chaining for each user, our reference architecture foresees three logical levels of switches, called *Logical Switch Instance (LSI)*:

- **NED switch.** This switch is in charge of providing isolation between different users, i.e. has to guarantee that the traffic of the User (A) will be delivered only to the TVD of that user. Once

reached the TVD, we expect that all the service chaining related to that user will be performed within the TVD itself, with the help of the TVD and TEE switches. Furthermore this switch will be programmed to implement the proper chaining between multiple service graphs, such as the one of the user and the one of his employer.

- **TVD switch.** This switch is in charge of delivering the packets to/from the proper execution environment in order to implement the proper service chaining. For instance, if the service graph of User (A) in Figure 12 requires packets to pass in $PSA_1 \rightarrow PSA_3 \rightarrow PSA_2$, packets received by the NED switch have to be delivered to `TEE1`, then to `TEE2`, then back to `TEE1` and finally sent to the NED switch.
- **TEE switch.** This switch is in charge of delivering the packets to/from the proper PSA in order to implement the piece of the service chaining that is expected to be carried out within the TEE itself.

Logical switching instances can be either implemented with distinct levels of software switches, or emulated by the proper software that exports this programming facility while consolidating all the switching commands within the same software component (e.g. the xDPd project[3]), provided that the resulting software can be properly verified with the remote attestation techniques and guarantees the required level of isolation between users. However, in the general case, all the LSIs could be implemented with distinct softswitches, e.g. using the standard OpenvSwitch for the NED switch (which corresponds to the vSwitch that is usually executed in the hypervisor/privileged OS partition), and instantiating other vSwitch instances for the TVD and TEE switches. An alternative with performance optimisation is shown in Figure 13, however the isolation of user traffic from other NED users must be enforced by this global switch. Since this would contain a lot of logic and state, it may prove to be too challenging to attest for isolation.

Finally, it is worth noting that in specific scenarios neither TVD nor TEE switches would be required and hence a further optimized implementation can emulate their presence without allocating a physical instance of that component. For instance, the service graph of User (B) in Figure 12 features a single PSA, which makes both the TVD and TEE switches (LSI4 and LSI5) unnecessary.

4.4.8 Control and data plane networks

Each user is provided with at least two networks, a data plane network that handles the user's traffic and is managed by all the LSIs, and a control plane network that is needed to allow the communication among the various components (e.g. orchestrator, PSC, etc.) and to permit to download the configuration in the PSA (e.g. a firewall needs to receive the set of policies that have to be applied on the traffic). In addition, an optional management network can be created as well, similarly to the control network.

The control (and, if present, the management) network can be implemented with OpenStack-like primitives, hence configured with the same set of functions that allow an OpenStack Neutron node to support a traditional shared network, with a private addressing for isolation purposes.

4.4.9 External components (user profiles, PSA and policy repositories)

An important extension of the current SECURED architecture consists in implementing our components in a way that is compatible with the Network Functions Virtualization (NFV) architecture, mostly

[3]The xDPd project, available at `http://www.xdpd.org/`.

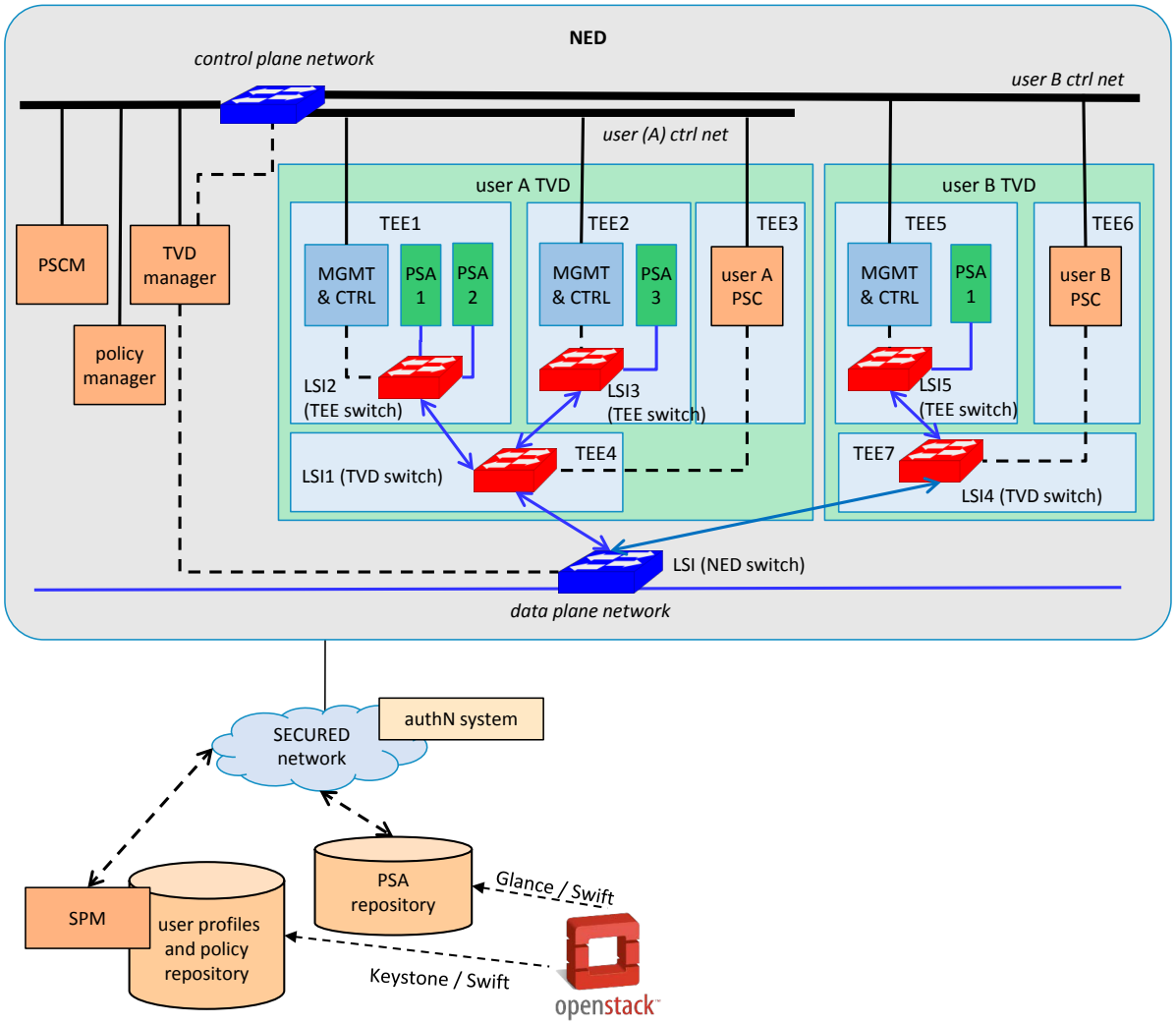

Figure 12: Using OpenStack for implementing the NED architecture.

likely on based on the OpenStack framework. Therefore, whenever possible, the SECURED architecture exploits existing OpenStack components in order to facilitate the future migration to that framework.

In particular, the current architecture for the monolithic NED exploits existing OpenStack services such as Keystone, Glance and Swift to implement most of the services external to the NED. In particular, we exploit:

- **Keystone** to keep the user profile and policy repository;
- **Swift** as general object store services, to keep all the information that cannot be easily stored in Keystone;
- **Glance** as image service, to keep all the images for the VMs we need to instantiate in our architecture, as well as the PSAs.

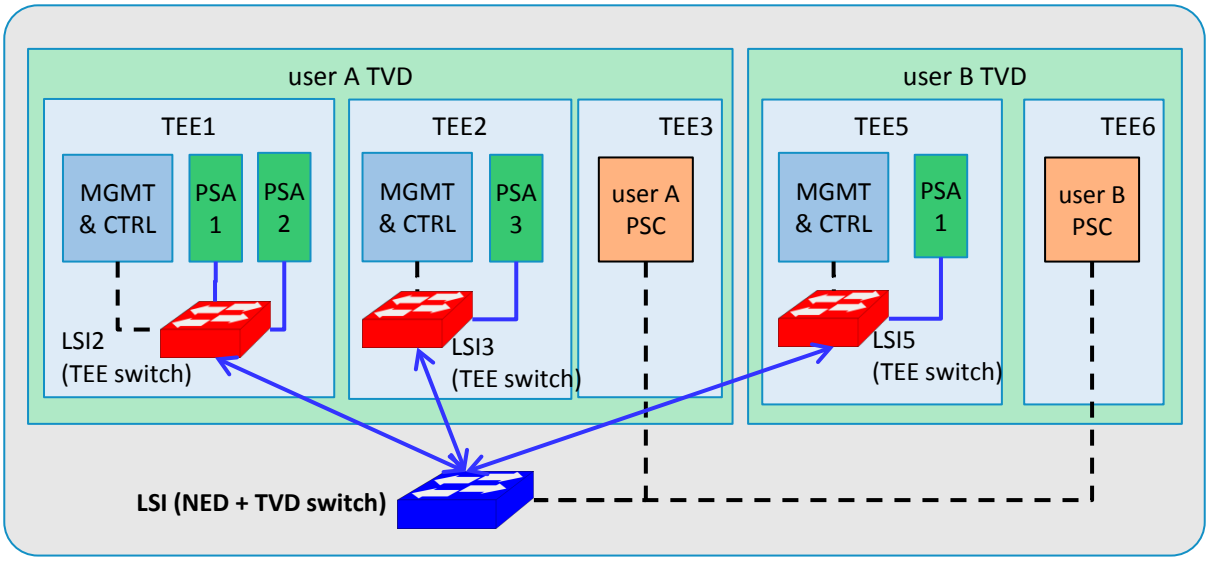

Figure 13: Optimisation removing the TVD switches and relying on a global LSI switch.

5 Remote establishment of trust

One of the main attractive features provided by SECURED is the ability for a user to have a proof that their security applications and policies are correctly (from the user point of view) enforced by a NED. In this section, we first list the user expectations needed for him to trust the NED, then we review the state of the art in Trusted Computing, and finally we specify the remote establishment of trust for the reference implementations.

5.1 User expectations about trust

From a high-level standpoint, the user connects and authenticates to the NED, which then initialises the user's application and policies. Afterwards, the user traffic reaches the Internet via the NED which hosts the user's security applications. The user's expectations of NED behaviour are thus twofold:

- the user traffic will be treated according to the user PSAs/policies (and no oether processing will be performed by teh NED itself, e.g. traffic eavesdropping);
- each PSA (and its corresponding policy) behave as configured by the user.

We refer to the attestation of these two expectations as the "user agnostic attestation" and the "user specific attestation".

5.1.1 First step: user-agnostic attestation

This is the first interaction between a user and a NED: the user wants to attest that he is connected to a genuine NED before he continue with the authentication. In this context, two properties characterise the genuineness of the NED:

(i) that the identity of the NED is correct;

(ii) that it will process the SECURED credentials and set up the user PSAs/policies properly.

In essence, the first step is about attesting the state of the compartmentalisation system, particularly the isolation of the user's future compartment, and that isolation between the user's security applications will be properly applied through the containment system.

Once these two properties are proven to the user, he knows that his credentials will only be used by the NED to set up the execution environment for his security applications.

5.1.2 Second step: user-specific attestation

From the security enforcement point of view, the user agnostic attestation focuses on the initialisation of the execution environment for the security applications. This second step aims to prove to the user that his security is enforced accordingly with his choices (i.e. PSAs and policies). The attestation can be performed at the initialisation of the PSAs, before the user traffic is processed by the PSAs, or during the execution of the PSAs.

Static PSA attestation This initial attestation is mandatory and must be performed before the user traffic is redirected through the set of his PSAs. The NED must provide a proof to the user that the instantiation of his PSAs and policies are the one he chooses.

Dynamic PSA attestation Additionally to the PSAs instantiation attestation, a continuous attestation of the PSAs execution may be required by a user to ensure his security. This kind of dynamic attestation is optional currently for a NED as it is an open research topic.

After this characterisation of the user expectations for the behaviour of a NED, the next section focuses on how to accomplish those expectations using the state of the art in Trusted Computing.

5.2 Trusted Computing

In a nutshell, Trusted Computing (TC) aims at answering the following question: "As a user or administrator, how can I have some assurance that a computing system is behaving as it should?"

The major enterprise level TC initiative is the Trusted Computing Group (TCG) [2], which has been established for more than a decade, that primarily focuses on developing TC for commodity computers (servers, desktops, laptops, etc.).

5.2.1 Chain of Trust

The overall scheme proposed by TCG for using Trusted Computing is based on a step-by-step extension of Trust, called a Chain of Trust. It uses a mathematical induction–like mechanism: if I trust the first execution step and each step attests the next executable software for trustworthiness, then a user can trust the system.

Effectively, during the loading of each piece of software, the integrity of each piece of software is measured and stored inside a log that reflects the different boot stages, as illustrated by Figure 14. Later, at the request of a user, the platform can present this log (signed with the unique identity of

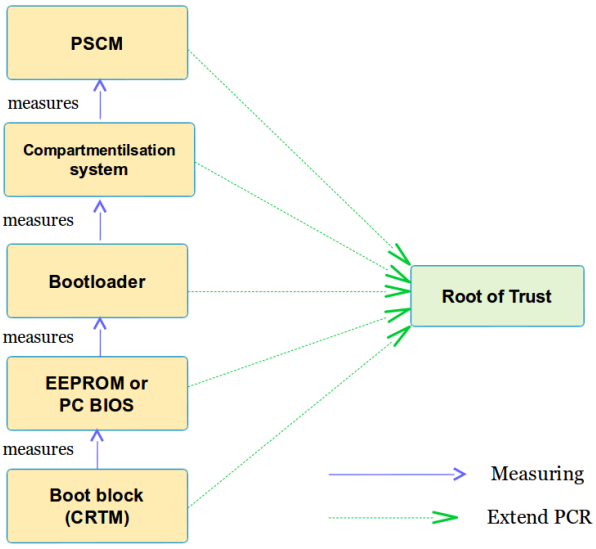

Figure 14: Trusted Boot

the platform) to the user, which can be checked by the user to prove the platform identity and attest the state of the system. The base case for the extension of the Chain of Trust is called the Core Root of Trust. This Core Root of Trust (CRTM) encompasses the minimal combination of hardware and software elements that a remote verifier needs to trust in order to validate the entire Chain of Trust.

5.2.2 Hardware and Software Requirements for Root of Trust (RoT)

From the Chain of Trust extension point of view, any component that needs to be attested is considered an adversary and must not be able to modify its behaviour measurement. It is recommended for this specification the use of hardware mechanisms in combination with software components to create a strong unforgeable identity and provide safer storage of secrets.

Therefore the main components of the Core Root of Trust are: (i) a specialised hardware component to store the log and measurements away from the software access and (ii) a initial isolated component that is able to measure the first non–trusted software, which will be then trusted to measure the next stage software.

Trusted Platform Module (TPM) The TCG created an international standard for the design and usage of a secure cryptoprocessor to address the storage of keys, general secrets, identities and platform integrity measurements. As of writing, the TPM standard has undergone many revisions and reviews for more than ten years, and is generally considered to be a mature technology[3]. It has also been ratified as an ISO standard under ISO/IEC 11889. Several other TCG standards related to embedded devices, networking and others use the TPM as a recommended component to act as a Root of Trust.

Core Root of Trust for Measurements (CRTM) Since the TPM is located outside of the execution path it cannot measure it, thus the need of a second trusted component to do the measurements and send them to the TPM. The initial straightforward solution was to trust the first to–be–executed block of software, using extensive auditing or formal verification for example; but this is still an implicit trust

since nothing guarantees the user that it is the actual executed code. The CRTM can be strengthened by either an out–of–band discrete component as in the HP SureStart [4] technology, or through in–band enforcement as designed by Intel in the "Trusted eXecution Technology" (TXT) [5] for example. The HP product feature verifies the CRTM status, whereas the Intel technology allows the CPU to execute dynamically verified code to do the initial measurements - it is referred to as a Dynamic Root of Trust for Measurement (DRTM).

5.2.3 Trusted Boot

Section 5.2.1 described the definition of a Chain of Trust for measuring each of the loaded software components on a platform. When using a TPM as a root of trust, measurements of the software stack are stored in special on-board Platform Configuration Registers (PCRs) on a discrete TPM. There are normally a small number of PCRs (at least 16) that can be used for storing measurements (e.g. SHA-1 values in TPM 1.2), however it is not possible to directly write to a PCR; instead measurements must be stored using a process called *Extending PCRs*.

The extend operation can update a PCR by producing a global hash of the concatenated values of the previous PCR value with the new measurement value, such as the following:

$$\text{PCR} = \text{SHA-1}\ (\ \text{PCR} \mid \text{measurement}\ )$$

The Extend operation brings a number of benefits. First, it allows for an unlimited number of measurements to be captured in a single PCR, since the size of the value is always the same and it retains a verifiable ordered chain of all the previous measurements. Second, it is computationally infeasible for an attacker to calculate two different hashes that will match the same resulting value of an PCR extend operation.

Whilst PCRs are related to the TPM standard from TCG, equivalent techniques can be used with other forms of root of trust. The overall process of measuring the booted software and storing a chained log of measurements is typically referred to as either *Measured Boot* or *Trusted Boot*. The user must be able to sent a nonced query of the state of a platform to the Root of Trust[4] for a signed set of platform measurements. The user will either validate the signed set of measurements with a trusted third party verifier who will assess whether the software configuration is trusted, or the user can check for themselves against their own set of reference digest values (measurements) that they have obtained a priori, and having already known the public endorsement key of the remote Root of Trust. Optionally, to preserve the privacy of a platform identity, an Attestation Identity Key (AIK) can be generated and used by the Root of Trust for the purpose of attestation instead of the Public Endorsement Key. This must be done in conjunction with a third party Privacy Certificate Authority (CA) who is trusted to not reveal the real identities, but to act as an intermediary.

As well as for providing a signed audit log of boot measurements, the PCR values can also be used as an identity for dynamically decrypting encrypted blobs on the platform (such as encryption keys or configurations that belong to operating system components). Software can choose to submit pieces of data to be encrypted by the Root of Trust (which has its own private asymmetric key and PCR

[4]SECURED recommends a discrete hardware TPM for the Root of Trust. However, alternativessuch as embedded CPU TPMs may also be used. It may also be possible to achieve sufficient isolation by leveraging mechanisms such as ARM TrustZone (http://www.arm.com/products/processors/technologies/trustzone/index.php) for isolated execution and physical memory protection (including Secure Memory for protected RAM, Secure Flash for storage), however this is currently not recommended without futher specification

registers) and only have it decrypted based on a criteria. This criteria can be that the platform booted into a particular state (e.g. a set of PCR values). Once the desired criteria is described and the sensitive data is encrypted by the root of trust, the data has been *sealed* to that platform state. The sealed data will only be decrypted when the platform measurements held in the the root of trust match the particular state.

Measured/Trusted Boot should not be confused with a different mechanism known as *Secure Boot*, as they both are designed to solve different problems. For instance, Secure Boot is a mechanism for a platform owner to lock a platform to only execute particular software. Software components that do not match the configuration digests will not be loaded or executed. This mechanism is particularly useful in preventing bootkits from successfully infecting a platform on reboot. A common standard for implementing Secure Boot is part of the UEFI 2.2 specification [6]. Secure Boot only enforces a particular configuration of software, it does not allow a user to attest or quote for a series of measurements.

As described before, Trusted Boot requires the use of a root of trust for safely storing measurements and secrets. Since the Root of Trust is self-contained and isolated from all the software that is measured, it is able to produce a signed set of platform measurements to a local or remote user. Trusted Boot however does not provide enforcement of a configuration, since the root of trust is a passive component not in the execution path, and is solely used for safe independent storage of secrets and platform measurements. It will respond to attestation requests with the exact measurements that were made during the software boot process. Sealing and unsealing of sensitive data is also a strong advantage of Trusted Boot, since it prevents leakage of secrets in the event of an untrusted software configuration.

5.3 Trusted NEDs

Preceding sections have described the state of the art in Trusted Computing and existing general mechanisms for providing trust in commodity computer systems. Whilst there are multiple different options for implementing a root of trust ranging from various hardware-based mechanisms to software-based mechanisms, this SECURED specification provides a recommendation based on well-established standards and understood mechanisms that also align well with the specifications provided by the Trusted Computing Group.

5.3.1 Requirements for a Trusted NED

1. **Trusted Boot is mandatory**
   All users who interact with a NED must be **a)** be able to identify a NED based on the public key of a Root of Trust and **b)** be able to retrieve a set of measurements of all the base software the NED has booted. This requires that firmware and software (such as the compartmentalisation system and the PSCM) must be measured before loading, with the resulting value being using to extend the appropriate PCR register. The following list describes which PCR registers should be used during a Trusted Boot process for a NED.

   - PCRs 01-07: for use by the CRTM (Initial EEPROM or PC BIOS)
   - PCRs 04-07: for use by the Bootloader stages
   - PCRs 08-14: for use by the booted base system (compartmentalisation system + PSCM)

   A discrete TPM is recommended for providing strong hardware-based root of trust which to provide more resilience to both software and physical attacks. Software-based equivalents for

a Root of Trust, such as a software TPM, may also be used, provided that they are sufficiently isolated from all software in such a way that the measurement process cannot be interfered with.

2. **Remote Attestation Service is mandatory**
   A service must be present for providing signed measurements from the RoT to the end user. The details for the remote attestation protocol have yet to be defined.

3. **Secure Boot is recommended but optional for NEDs**
   Using a mechanism such as Secure Boot helps provide strong prevention of software attacks. Furthermore, in combination with a hardware-based RoT such as a TPM, Secure Boot can provide some resilience to physical attacks (e.g. preventing a class of offline attacks and unauthorised system replacement). Home users who own Monolithic NEDs may choose not to enable this method. For SECURED providers, it is recommended that Secure Boot is employed wherever possible with an appropriate firmware update mechanism, due to the possible threat of software/-firmware modifications in either public places or privately with inside attackers.

Thanks to the availability of TPM in commodity computer hardware (servers, laptops, etc.), SECURED can strengthen the remote establishment of trust with a physical trusted component which can provide a platform with strong identities when combined with a CRTM independent from the software stack (e.g. typical BIOS found in a PC). These identities are: (i) the physical identity of the platform and (ii) the measured software that is executed on the platform. It is recommended that NED implementations use a hardware–enforced root of trust, such as the TPM.

5.4 Reference Remote Attestation Design of the NED

5.4.1 NED attestation

This section explains how a user can perform through his terminal the remote attestation of a NED; the attestation will ensure to him that the NED is able to process his traffic as expected. In Figure 15, we show the steps required to perform the NED attestation. These steps correspond to step 1 of Figure 18.

As shown in Figure 14, initially the NED measures all the hardware and software components involved in the boot process, in order to build the chain of trust. Then, during step 0, the kernel of the privileged OS records the measurements at application level, through a measurement module such as IMA, the Integrity Measurement Architecture,[7]. These measurements will be stored in a Stored Measurements Log (SML) and will be used later during the remote attestation protocol; the integrity of the SML is protected by the TPM.

Since a user terminal may not have enough capabilities to perform the integrity verification of a NED, we envision two possibilities to perform the remote attestation. In the first, the SECURED app performs the entire protocol and determines the integrity status of a NED. In the second, the SECURED app requests the status of a NED to a *Trusted Third Party*, which is in charge of communicating with the NED. This second choice is also preferable as an attacker cannot easily determine the software running at the NED; with this information he may try to exploit vulnerabilities not yet fixed.

If the SECURED app directly performs the remote attestation, it asks (step 1) the NED to generate an integrity report with the format defined by the TCG [8]. The RA agent, placed in the PSCM, retrieves (step 2) the measurements from the SML and asks the TPM to sign the PCRs with an Attestation

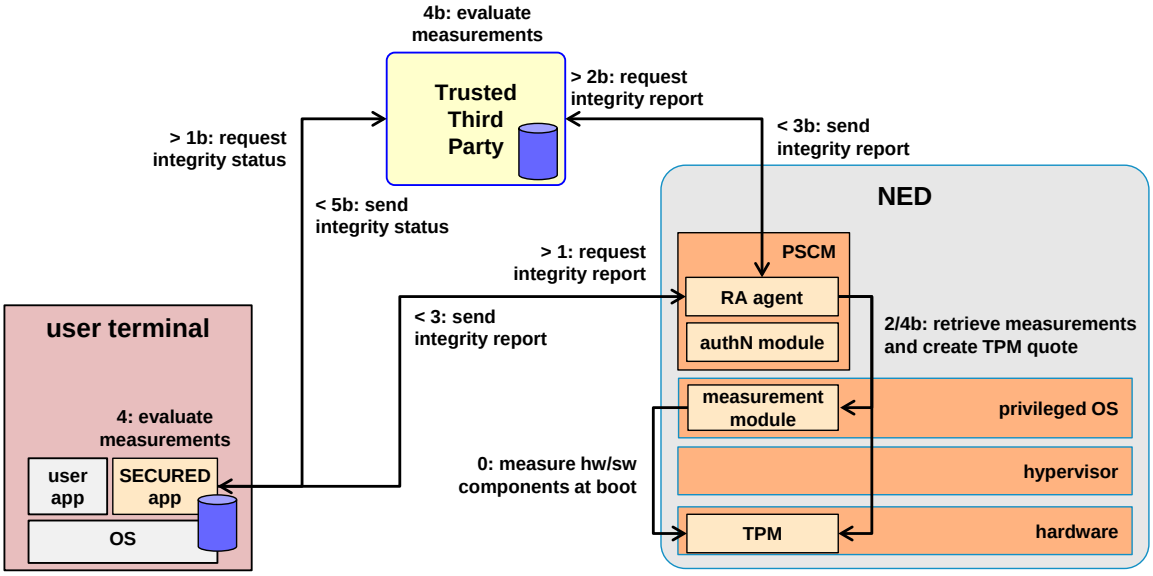

Figure 15: NED attestation.

Identity Key (AIK); with this signature[5], the user is provided with the evidence that the measurements received belong to the NED being attested.

After the integrity report has been generated, the RA agent sends it back to the user (step 3). In step 4, the SECURED app first checks if the integrity report is valid (by verifying the quote and the certificate associated to the AIK) and then determines if the NED is behaving as expected, i.e. the NED software has not been compromised and the compartimentalization mechanism is enforcing isolation among the users connected to the NED.

At a first stage, the integrity evaluation will be performed by comparing the digests of measured files (recorded by the measurement module of the NED) against a reference database distributed together with the SECURED app[6]. Given the limitations of this approach (the integrity of an application is checked only at load time), the SECURED app will support advanced verification techniques, e.g. it may inspect the flows of information between the NED components.

If the SECURED app is running in a terminal with low computation resources, it may contact a *Trusted Third Party* which, in turn, attests a NED and returns the result of the integrity evaluation to the user (steps from 1b to 5b).

5.4.2 TVD attestation

The main outcome of the NED attestation is to detect whether or not the NED correctly configures the logical container of PSAs belonging to the connecting user (called Trusted Virtual Domain, TVD) in a way that the user's traffic is processed only by the PSAs within the container. The TVD attestation, instead, evaluates the integrity of the PSAs running within the TVD.

Since the TEEs of a TVD are mainly intended as virtual machines, the integrity measurements are not protected as the same as those performed by the privileged OS, because the hardware TPM is not

[5] It is called *quote* in the TCG specifications.
[6] This methodology is described in [7].

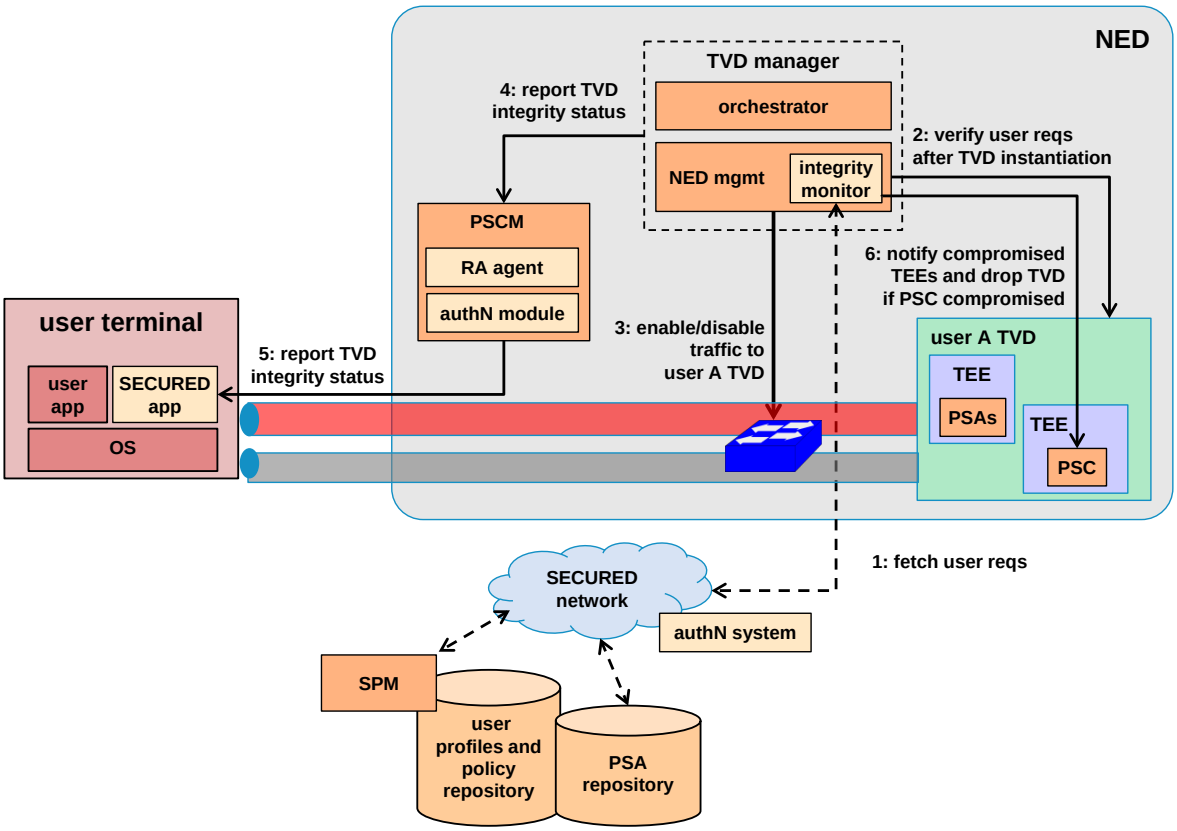

Figure 16: TVD attestation.

available.

In the preliminary study of solutions that address this problem, we found two main alternatives. Berger et al. [9] propose a Virtual TPM (vTPM) consisting of a small front-end driver, added to the kernel of VMs, and a back-end module that emulates a physical TPM. The integrity verification of a VM is done by evaluating the measurements performed by the VM itself and the environment hosting the VM (operating system, hypervisor management software, vTPM manager).

Given the scalability issue of this solution (periodically attesting a high number of VMs may introduce a consistent overhead and noticeable delays), Schiffman et al. [10] defined a new approach for attesting virtual machines. A user attests the software running in the physical host and verifies a new component, called *integrity monitor*, which is in charge of checking the integrity of VMs. Instead of using a Virtual TPM, this solution employs Virtual Machine Introspection (VMI) techniques, which has been applied in multiple different security contexts [11] [12].

In the SECURED context, given that initially TEEs will be implemented as VMs, the second solution seems more appropriate than the former, especially if the attestation is performed by user terminals. Since the final decision has not been already made (further investigation is necessary), we will introduce in the architecture the abstract components necessary to realize the second solution, but we leave the possibility for adopting the first one. The resulting architecture, together with the necessary steps to perform the TVD attestation are depicted in Figure 16. The mapping between these steps and those in Figure 18 is as follows:

- step 1 corresponds to step 4 (query the user profile);
- steps 2-3 are executed before step 14b (report the status);
- step 4 corresponds to step 14b;
- step 5 corresponds to step 15 (feedback to the user);
- step 6 corresponds to step 17a (platform monitoring of the TVD).

Note that steps 2, 3, and 6 are repeated throughout the TVD life cycle.

As shown in the figure, the NED management component contains a module named integrity monitor. Performing the remote attestation of a TVD through the NED management component instead of the user PSC (the PSC is in charge of monitoring the security of the TVD) is necessary because the monitor needs to directly interact with the hypervisor in order to access the memory of VMs. Doing the verification at the PSC would allow the latter to access the memory of VMs belonging to other users (the isolation security property cannot be guaranteed).

For the TVD attestation, there is a configuration phase, not displayed in the figure, where a user defines the criteria for evaluating the VMs integrity (evaluation of binaries and configuration files only – less accurate but faster –, additional evaluation of information flows – more accurate but slower). Those criteria may be stored as part of the user profile. In the step 1, the NED manager retrieves the criteria from the profile.

After all the TVD TEEs have been deployed, the integrity monitor verifies (step 2) the integrity of those TEEs and communicates (step 3) to the blue virtual switch if the user can exchange data with his TEEs or not. The result of the verification is also reported to the PSCM (step 4) and to the user (step 5).

The monitoring process does not stop just after all TEEs of a TVD have been deployed but continues for the whole TVD life cycle. If the integrity of a TEE running the user PSAs gets compromised, the NED management component reports (step 6) this event to the PSC, which may send a notification to the user through the grey channel. If the PSC becomes compromised, the NED management component destroys the whole TVD and the user needs to request a new TVD by contacting the PSCM.

5.4.3 Trusted Channel

The NED and TVD attestation are necessary to determine whether the traffic will be processed by a NED as expected. However, problems may arise if a user wants to establish a secure channel with an integrity verified NED. Indeed, as pointed out in [13], the contacted host may relay the attestation to another genuine platform; in this case, the user believes that is communicating with a trusted endpoint while, instead, contacted a host controlled by an attacker.

To overcome this issue, [13] proposes to include the public key or the certificate for the secure communication as part of the measurements presented by the contacted endpoint during the remote attestation. This way, a malicious platform cannot relay the attestation to another platform as its certificate will not be present in the measurements list of the genuine platform.

In addition, this proposal addresses the problem of loosing the control of the private key (with the latter a malicious endpoint can prove the identity of the genuine endpoint) by defining a long-lived *Platform Property Certificate*. Since this certificate connects the platform's identity to the AIK public key, an attacker cannot use a stolen private key without revealing his identity: he may use the certificate of the

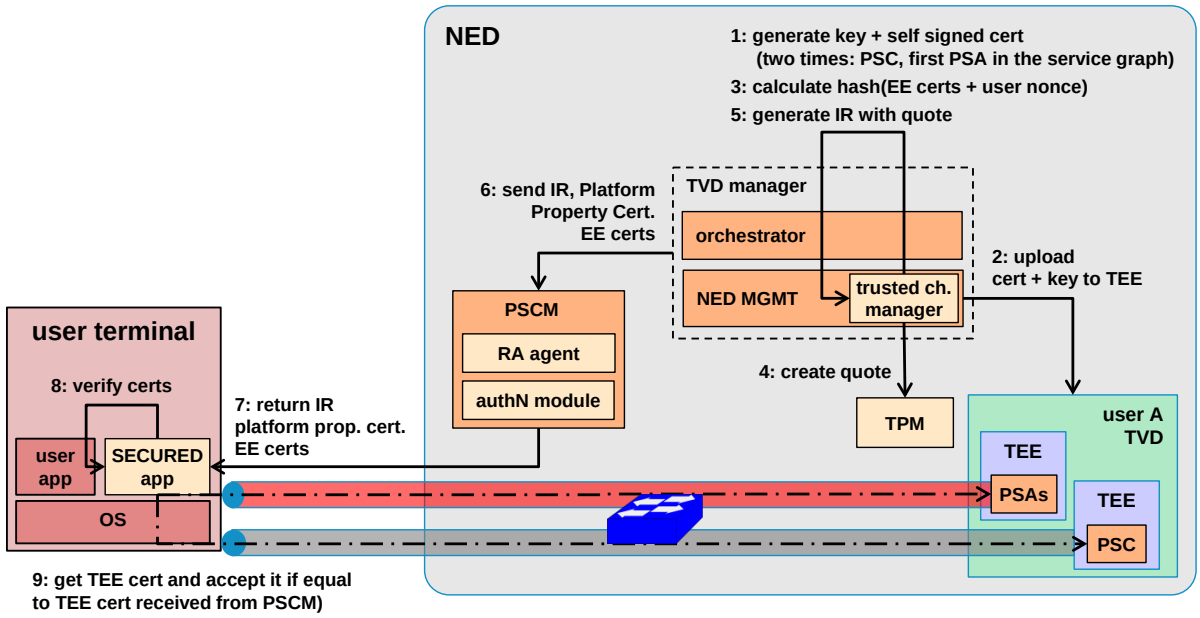

Figure 17: Trusted channel creation.

genuine endpoint but cannot create a quote with the AIK of the other platform; he may generate a quote with his own AIK but the identity associated differs from that of the genuine platform.

Finally, since the platform's identity can be verified from the Platform Property Certificate, the information in the certificate to be presented during the establishment of a secure communication are redundant. Thus, in place of certificates signed by trusted CAs, [13] envisions the use of self-signed certificates (which still need to be included in the measurements list). Since a platform may generate an infinite number of certificates, this proposal can be adopted in a virtualised server scenario where each TEE is given with its own certificate. In SECURED we will adopt this solution and we will further investigate solutions that instead convey remote attestation data during the establishment of a secure communication channel (e.g. [14]).

Figure 17 shows the steps necessary to establish a trusted channel. The mapping between these steps and those in Figure 18 is as follows:

- step 1 (two times) corresponds to step 6 (create TVD) and step 11 (TVD expansion);
- step 2 corresponds to step 12 (fetch and upload apps to TVD);
- steps 3-5 are executed before step 14b (report status);
- step 6 corresponds to step 14b (report status);
- step 7 corresponds to step 15 (feedback to user);
- step 9 corresponds to step 16 (establish trusted channels).

During the creation of a TVD (and later, during the TVD expansion), the trusted channel manager module of the NED management generates (step 1) a self-signed certificate and a key pair to be used for the secure communication between the user terminal and a TVD TEE. The first time the certificate

and key pair will be generated for the PSC, while the second time for the TEE that will run the first PSA of the user's service graph. Certificates and keys are uploaded (step 2) by the NED management to the right TEE during the TEE deployment phase.

Before the result of the TVD setup is returned to the PSCM, the trusted channel manager calculates (step 3) the hash of the generated TEE certificates and the nonce[7] provided by the user during the NED attestation[8], creates a quote with the NED AIK and generates (step 5) an Integrity Report (IR) containing the hash and the quote. Then, the NED management returns (step 6) the generated TEE certificates, the IR and the Platform Property Certificate to the PSCM, which in turn, provides (step 7) them as part of the feedback to the user.

During step 8, the SECURED app verifies through the data sent by the PSCM that the supplied TEE certificates belong to the NED. Lastly, the SECURED app verifies (step 9) during the establishment of the trusted channel with the TEEs that the certificates presented by those entities are the same of those provided by the PSCM.

We recall from Section 5.4.2 that a user is allowed to contact a TEE only if the integrity monitor modules has successfully verified the integrity of that TEE according to the user's criteria. Also, if a TEE gets compromised, the integrity monitor drops the user connection. This means that, differently from [13], a user does not need to periodically receive IRs to verify if a TEE certificate has been revoked because the revocation process is entirely handled by the integrity monitor (whose integrity is verified through the NED attestation).

6 Closing comments

In this document we provided the specifications for the general NED architecture, with particular emphasis on the PSCM frontend and backend components, the PSC design, EE Control & Management interfaces and TVD management services. In addition to the definition of the basic NED component architecture, we describe some early NED-level requirements for remotely establishing trust and mobility.

From the beginning, the architecture has been designed to be scalable, and may eventually expand into a distributed NED model, where PSAs and flows can be spread across multiple NEDs. This is achieved with little to zero changes to the core architecture, where physical boundaries become virtual. This distributed model aligns with a Network Functions Virtualisation (NFV) architecture, where there would be the possibility of having a master external orchestrator providing load balancing and coordination. However, in a distributed approach which can deploy and orchestrate a large number of computational NEDs and PSAs come a number of security concerns which will need to be researched and evaluated.

Whilst the definition of a *Split-NED* scenario is a simple pre-configured SDN enabled router (e.g. OpenFlow enabled) redirecting traffic to multiple NEDs, it is also expected that in future specifications an advanced Split-NED will be defined which will exhibit a high level of trust and provide more appropriate mechanisms for performance and scalability.

It is envisioned that at some point in the evolution of the architecture, a developer may wish to develop their own programmable controller (PSC) with an SDN-like programmability model that can dynamically re-route traffic flows between PSAs in the TVD. Whilst this feature is not a high priority for the

[7] Including this data in the hash calculation ensures the freshness of the response.

[8] Differently from [13], we do not measure the certificate generated and extend a PCR for scalability reasons. The linking is guaranteed by signing the calculated hash with the NED AIK during the quote operation.

SECURED architecture, it opens the door to new PSA use cases but also creates new security and trust challenges.

This specification pays particular focus to the security of the NED and the isolation of users. However, some choices come at a potentially large performance penalty in packet processing speed. The next version of this document will aim to provide optimisations in the specification, and will further specify the measurement techniques and protocol needed for remote attestation of the NED. In addition to this, remote attestation of the NED will also be evaluated in the context of mobility.

References

[1] J. Winter, and K. Dietrich, "A hijacker's guide to communication interfaces of the Trusted Platform Module", *Computers & Mathematics with Applications*, vol. 65, no. 5, pp. 748–761, 2013, doi:10.1016/j.camwa.2012.06.018

[2] Trusted Computing Group (TCG), https://www.trustedcomputinggroup.org/

[3] "TPM Main Specification Level 2 Version 1.2, Revision 116", https://www.trustedcomputinggroup.org/resources/tpm_main_specification

[4] "HP SureStart", http://h20435.www2.hp.com/t5/367-Addison-Avenue-Blog/New-HPSureStart-The-Cure-for-BIOS-Issues/ba-p/82999

[5] Intel, "The Intel Trusted Execution Technology", http://www.intel.com/technology/security/downloads/TrustedExec_Overview.pdf

[6] "UEFI Specification Version 2.2 (Errata D)", Unified Extensible Firmware Interface Forum, Tech. Rep.

[7] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a TCG-based integrity measurement architecture", *13th USENIX Security Symposium*, August 9–13, 2004, San Diego (CA, USA), pp. 223–238, https://www.usenix.org/legacy/events/sec04/tech/sailer.html

[8] Trusted Computing Group, "Infrastructure Work Group Integrity Report Schema Specification, Version 1.0", https://www.trustedcomputinggroup.org, 2006

[9] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, "vTPM: Virtualizing the Trusted Platform Module," *15th USENIX Security Symposium*, Vancouver (B.C., Canada), July 31 – August 4, 2006, pp. 305–320, https://www.usenix.org/legacy/events/sec06/tech/berger.html

[10] J. Schiffman, H. Vijayakumar, and T. Jaeger, "Verifying system integrity by proxy", *5th Int. Conf. on Trust and Trustworthy Computing*, Vienna (Austria), June 13–15, 2012, pp. 179–201, doi:10.1007/978-3-642-30921-2_11

[11] B. D. Payne, M. Carbone, and W. Lee, "Secure and flexible monitoring of virtual machines", *23rd Annual Computer Security Applications Conference (ACSAC 2007)*, December 10–14, 2007, Miami Beach (FL, USA), pp. 385–397, doi:10.1109/ACSAC.2007.10

[12] A. L. Shaw, B. Bordbar, J. Saxon, K. Harrison, and C. I. Dalton, "Forensic virtual machines: Dynamic defence in the cloud via introspection", *2014 IEEE Int. Conf. on Cloud Engineering (IC2E'14)*, Boston (MA, USA), March 10–14, 2014, pp. 303–310, doi:10.1109/IC2E.2014.59

[13] K. Goldman, R. Perez, and R. Sailer, "Linking remote attestation to secure tunnel endpoints", *1st ACM Workshop on Scalable Trusted Computing (STC'06)*, Alexandria (VA, USA), November 3, 2006, pp. 21–24, doi:10.1145/1179474.1179481

[14] F. Armknecht, Y. Gasmi, A.-R. Sadeghi, P. Stewin, M. Unger, G. Ramunno, and D. Vernizzi, "An efficient implementation of trusted channels based on openSSL", *3rd ACM Workshop on Scalable Trusted Computing (STC'08)*, Fairfax (VA, USA) October 31, 2008, pp. 41-50, DOI doi:10.1145/1456455.1456462

[15] A. Pras and J. Schoenwaelder, "On the difference between Information Models and Data Models", RFC-3444, January 2003

A Analysis of the SECURED scenarios

Depending on the scenario where a NED is deployed, some interfaces or actors may be not available. We discuss the possible occurrences of attacks case by case. The outcome of the analysis is given in Table 1.

scenario / attacks	home (IoT)		enterprise		public hotspot		mobile
	user	ISP	IT admin	ISP	host	ISP	ISP
1: unknown user	N	N	Y (noSEC)	Y (noSEC)	Y (noSEC)	Y (noSEC)	Y (noSEC)
2: authorized user	N	Y (ISP/GOV)	Y (ITADM/ISP/GOV)	Y (ITADM/ISP/GOV)	Y (HOST/ISP/GOV)	Y (ISP/GOV)	Y (ISP/GOV)
3: bad policy/app	N	Y (SEC)	Y (SEC)	Y (SEC)	Y (SEC)	Y (SEC)	Y (SEC)
4: bad NED owner	N	Y (ISP)	Y (ITADM)	Y (ISP)	Y (HOST)	Y (ISP)	Y (ISP)
5: NED physical access	N	Y (SEC)	Y (ITADM)	Y (ITADM)	Y (HOST)	Y (HOST)	Y (ISP)

Table 1: Attacks occurrences.

A.1 Home scenario

In this scenario, the owner of the NED could be a SECURED user, which bought the device and configured it by himself, or the ISP which loaned the device to the user.

Attack 1: this attack should not occur as we assume that the user takes the appropriate countermeasures to avoid undesired accesses to his home network;

Attack 2: when the NED is not managed by an user, the ISP or the government may misuse their privileges to process users traffic for malicious purposes;

Attack 3: this is relevant only if the NED is managed by the ISP as the user may want to circumvent the ISP or the government policies by exploiting a NED vulnerability;

Attack 4: this attack does not apply in this scenario if the NED owner is the SECURED user (as the latter would not intentionally[9] compromise his security), but an ISP may access the NED to gain the network traffic of SECURED users before data is processed by users' applications;

Attack 5: a SECURED user may conduct this attack (if the NED is owned by the ISP) to circumvent the policies of the ISP and the government.

A.2 Enterprise scenario

In this scenario, the owner of the NED may be the IT administrator or the ISP.

Attack 1: this attack can be conducted by company employees with legitimate access the corporate network that may try to impersonate a SECURED user;

Attack 2: the IT administrator, the ISP or the government may misuse their privileges;

Attack 3: an user may conduct this attack to circumvent the policies of the IT administrator, the ISP and the government or to gain information of other employees;

Attack 4: if the NED owner is the IT administrator, the latter may execute administrative commands on the NED to circumvent the ISP and government policies or to gain network traffic of SECURED users before data is processed by users' applications; if the NED owner is the ISP, the latter may try to gain the traffic of the whole corporate network as soon as it is received by a NED (before the processing by SECURED applications);

Attack 5: this attack may be conducted by the IT administrator to alter the NED behaviour in a way that cannot be detected by SECURED users (or the ISP or the government); in addition, the IT administrator may conduct this attack, if he cannot access the NED admin interface, for the purposes listed in Attack 4;

A.3 Public hotspot scenario

In this scenario, the NED owner could by the host of a public place or the ISP. The former may impose his own policies to SECURED users.

Attack 1: the exposure of the NED to this attack is greater than that in the enterprise scenario as the public network could be accessed by many people;

Attack 2: the host of a public place (if he is the NED owner), the ISP or the government may misuse their privileges;

Attack 3: as for the Attack 1, the probability of this attack is higher than in the enterprise scenario due to the greater number of users that may connect to a public network;

Attack 4: the host of a public place (if he is the NED owner) may try to circumvent the ISP and government policies or to intercept the network traffic of SECURED users before it is processed by SECURED applications; the ISP could do the same if it is the owner of the NED;

[9] However, it is possible that he makes mistakes unintentionally, e.g. by setting a wrong value for a configuration parameter.

Attack 5: the host of a public place may conduct this attack[10], if it is not the owner of the NED, for the purposes listed in Attack 4; also, with this attack he may modify the NED behaviour in a way that cannot be detected by SECURED users.

A.4 Mobile scenario

In the mobile scenario, the ISP has the full control of the NED. Additionally, the authentication mechanism could be used to identify the owner of the mobile terminal from which network packets originates (however, it is possible that the user interacting with the NED is not the terminal owner).

Attack 1: this attack can be conducted by an user which has the physical control[11] of the mobile terminal;

Attack 2: the ISP and the government may misuse their privileges as the same as in the previous scenarios;

Attack 3: this attack can be conducted by a SECURED user to circumvent the policies of the ISP or the government or to gain information of other users connected to the NED;

Attack 4: the ISP can conduct this attack to intercept the network traffic of SECURED users before data is processed by users' applications;

Attack 5: an ISP can conduct an attack to the hardware to alter the behaviour of the NED in a way that cannot be detected by SECURED users.

B Initialization sequence diagram

Figure 18 contains the sequence diagram of the NED initialization steps.

C NED interfaces

NED architecture has been designed to clearly separate the front-end from the back-end, guaranteeing isolation between the user space components and sensible data and the external threats. The first point of contact for the user is the PSCM. It will mainly interact with the authentication system and the orchestrator. The latter acts as mediator between the front-end and the back-end getting the authentication token retrieved by the PSCM and executing all the tasks needed to instantiate and manage the user's TVD. The back-end components include: orchestrator, TVD (and subcomponents) and the *Control & Management* of the TEE. When the orchestrator has retrieved the user's profile, it instantiates the TEEs that will host the user's PSAs and PSC; the PSC is then instructed to configure and monitor the TVD. Some components inside the PSC are dedicated to provide a control interface for the user's terminal.

The NED components are divided in subsets that are interfaced through four separate logical subnetworks, depicted in Figure 19:

[10] We assume that he properly protects the access to his devices against users that can physically access the public place.

[11] Although a mobile terminal may be compromised by a remote attacker, this threat is out of the scope of the SECURED project.

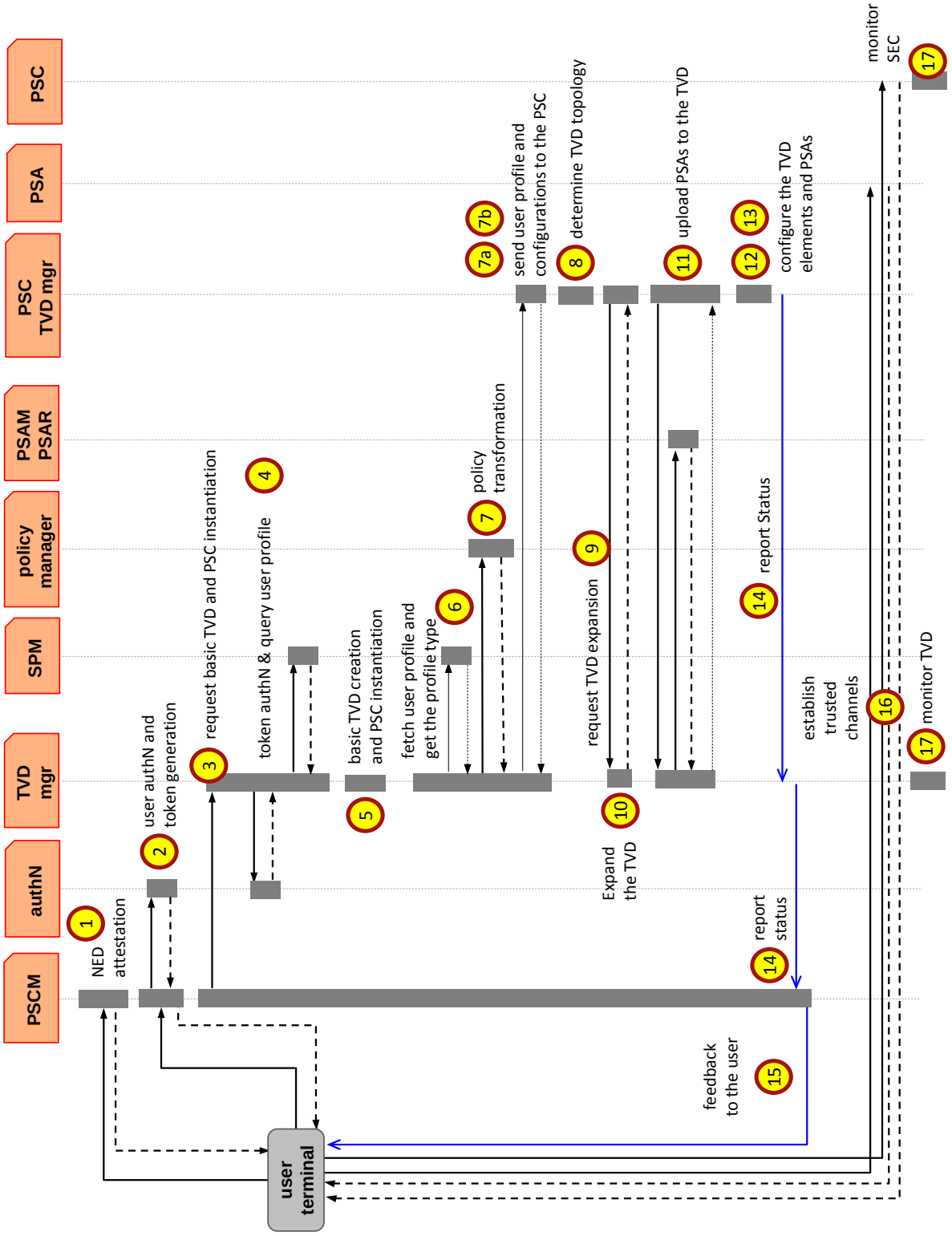

Figure 18: Authentication and initialization steps.

- control and management plane infrastructure networks;
- TVD control plane network;
- user control plane network;
- user data plane network.

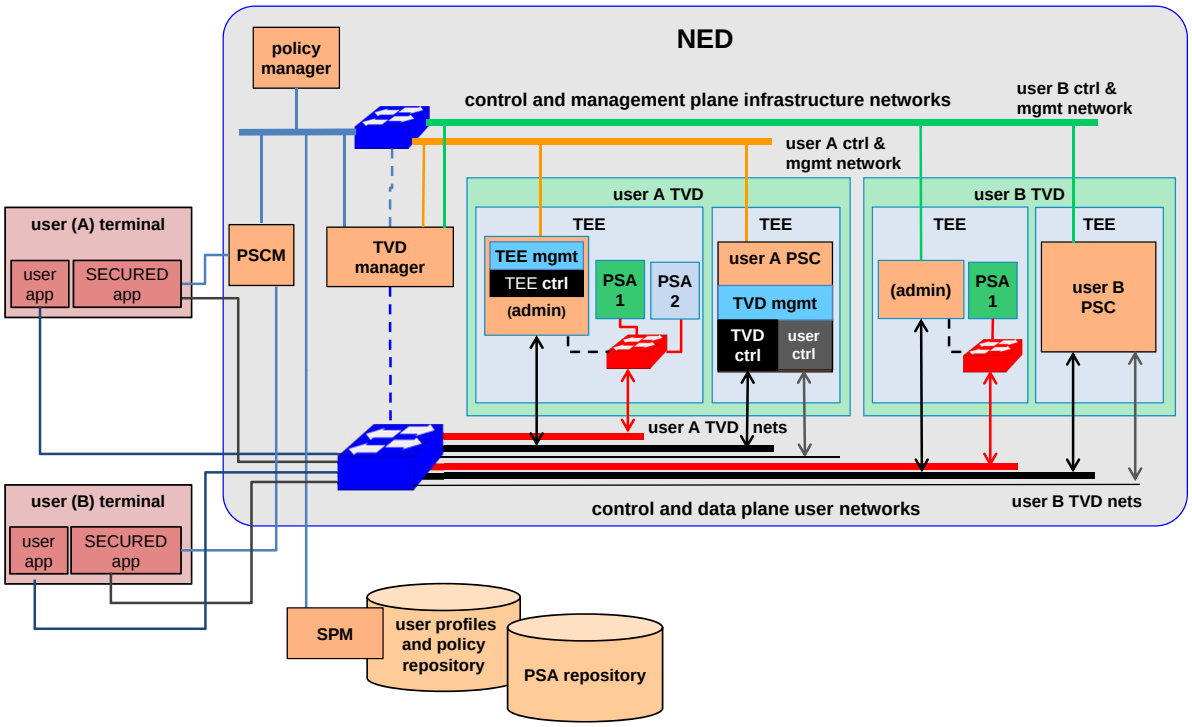

Figure 19: NED architecture highlighting the components' interfaces and the different subnetworks.

In the following subsections a brief summary listing all the main interfaces between the architecture components is shown. Dedicated sections for important interfaces follow.

C.1 Control and Management Plane Interfaces

C.1.1 TVD manager

Table 2 briefly describes the interfaces between the TVD manager (composed of the Orchestrator and NED management) and the other NED components.

C.1.2 PSA

The PSA has the following interfaces:

PSA – TEE ctrl
With this interface the PSC (through the TEE components) is able to collect the PSAs running status and handle any generated exception.

interface between the TVD manager and . . .	*interface description*
Authentication System	On this interface the Orchestrator/NED MGMT passes the user session token to the Authentication System to retrieve the user profile.
Policy Manager	Used by the Orchestrator to translate the MSPL retrieved from the SPM (in case of a policy-driven user profile).
PSAR/PSAM	Used to fetch the PSAs from the repository once the TVD is ready and configured to host them (subject to modification according to progress in WP4 and WP5).
PSC TVD manager	Used to communicate with the PSC TVD MGMT by sending the user profile, low level PSAs' configuration and the fetched PSAs; other control operations happen on this interface such as the PSC asking to extend the TVD, NED MGMT informing the PSC that an EE has been compromised, . . . (see subsection Orchestrator - PSC C.4).
PSCM	Allows the PSCM to request the instantiation of the basic user TVD and PSC; through this interface the Orchestrator receives the user session token and provide the final status report of the instantiation to the front end so that it can be reported to the user terminal (more details in C.3).
SPM	Allows the interaction between the NED and the SPM to retrieve the user's profile, possibly the MSPL policies, PSA information related and M2L plugins.

Table 2: TVD manager interfaces.

PSA – TEE mgmt
With this interface PSC and PSAs (through the TEE components) exchange commands (start, stop, ...), configurations messages and status management (e.g. for migration).

Longer descriptions of the PSA interfaces are given in section C.7.

C.1.3 PSC

PSC has mainly four interfaces:

PSC TVD ctrl – TEE ctrl
Allows the PSC to monitor the PSAs chain status and control the privileged part of the EEs.

PSC TVD mgr – TEE mgr
Used to instruct the privileged part of the TEE to configure the PSAs and the chain (vSwitches and other low level configurations)

PSC USER ctrl – User Terminal
Allows the SECURED App running in the user terminal to monitor the TVD status (PSA running state, enforcement, violations, ...) and trigger some events (e.g. handover). It acts as tunnel endpoint for the control plane channel established between the user and the NED.

PSC TVD mgr – Orchestrator
Previously defined (in the last subsection).

C.1.4 PSCM

The PSCM interfaces are:

PSCM – authN system
Allows the PSCM to autheticate the user and retrieve the authetication token to pass to the Orchestrator.

PSCM - User Terminal
This is the first point of contact between the user and the SECURED infrastructure; it is a secure channel through which the user terminal passes the authentication credentials and waits for the infrastructure feedback (first RA measurements, static PSA attestation, dynamic PSAs attestation, instantiation report, ...).

PSCM – TVD manager
Previously defined.

C.1.5 SECURED user terminal

The user terminal has two interfaces:

User Terminal – PSC user ctrl
Previously defined.

User Terminal – PSCM
Previously defined.

C.2 Data plane interfaces

The data plane interface is unique and is between the user terminal and the EE that runs the first PSA of his chain. Other components can be found on path (e.g. the vSwitch that switches the data plane traffic to the different TVDs) but they do not act as tunnel terminator.

C.3 Key interface: PSCM - TVD manager

This interface represents the only point of contact between the front-end and the back-end of the infrastructure. Particular attention has to be paid in designing and developing it. Once the PSCM has authenticated the user against the Authentication System it is provided with a user session token; this token is then passed to the Orchestrator allowing it to interact with the security policies and PSAs repositories. By using the token, this can avoid the PSCM from directly handling the user profile and service graph, as the PSCM is exposed to potential attacks from the outside world. On the other hand the PSCM neither deals with the user's profile (which is retrieved by the Orchestrator) thus avoiding another security threat (as the PSCM is exposed to attacks from the outside).

C.4 Key interface: TVD manager - PSC

The interface between the Orchestrator and the PSC is a key point for the initialization procedure and for later monitoring of the users' TVDs. The main messages exchanged on this interface accomplish the following tasks:

- user profile retrieval;
- PSAs fetch;
- TVD expansion request;
- PSAs' low level configuration and chaining retrieval;
- PSC status monitoring; this can be accomplished in two ways: polling approach (the Orchestrator pings the user's PSC to verify if it is alive and running) and heartbeat approach (PSC sends heartbeat to the Orchestrator to confirm it is alive);
- TEE management; e.g. PSC requests to the Orchestrator a TEE reboot, shutdown, or migration.

C.5 Policy/configuration setup

One of the goal of SECURED is to allow multi-tenant configurations, for example enforcing the security controls that the user requires plus the controls required by his company (in the NED's company). To enable multiple actors policy enforcement a context aware reconciliation of the policies is needed; it is done in the SPM premises before the user's profile is downloaded to the NED.

The user's profile type influences the translation process; it could be:

- *App-driven profile* In this case a final low-level configuration file will be retrieved from the SPM. It can be passed to the PSC directly, without the need of any intermediate translation.

- *Policy-driven profile* In this case the consolidated MSPL will be downloaded from the SPM. An intermediate translation is needed and it is performed by the Policy Manager.

The interfaces involved in the process are:

- TVD manager- Authentication System
- TVD manager- SPM
- TVD manager- Policy Manager
- TVD manager- PSC
- TVD manager- PSAM/PSAR
- TVD manager- TEE Manager

A possible reconciliation scenario involving a monolithic NED is:

- User's device attests the NED and is authenticated; an authentication token is passed to the Orchestrator.
- The NED contacts the SPM (*Orchestrator/NED MGMT - SPM interface*) and retrieve the user's profile (step 4 of the sequence diagram in Fig. 18). In the meanwhile the ID of the NED is registered in the SPM premises.
- The SPM based on the NED ID (providing information about location and administrative domain) can execute the policy reconciliation process for the user (N.B. the reconciliation result could be cached or even computed offline).
- Once the profile type is fetched from the SPM the Policy Manager is invoked (for policy-driven profiles) through the *Orchestrator/NED MGMT - Policy Manager* interface. PSA-specific M2L translation plugins are also dowloaded at this step.
- Based on the service graph the PSAs are downloaded from the PSAR (*Orchestrator/NED MGMT - PSAM/PSAR* interface).
- Low level configurations are then passed to the PSC which will enforce them once the TVD is up and running (*Orchestrator/NED MGMT - PSC* interface).
- The PSAs configuration process is instructed by the PSC to the EE MGMT module thus enforcing the final configuration to the PSAs. The configuration files can be auto-generated PSA-specific scripts.

C.6 Full component interaction matrices

Figure 20 contains the matrix of interfaces for the Control plane.

The corresponding matrix for the Data plane is not shown because it's trivial: there are only interfaces between the user terminal and the PSAs, and among the PSAs themselves.

		User Terminal	PSCM	SPM	PSAM / PSAR	Policy Manager	Auth System	Orchestrator	PSC			EE		PSAs
								NED MGMT	PSC TVD MGMT	PSC TVD CTRL	USER CTRL	EEMGMT	EE CTRL	
User Terminal		×	√	×	×	×	×	×	×	×	√	×	×	×
PSCM		√	√ (R)	×	×	×	√	√	×	×	×	×	×	×
SPM		×	×	×	√	×	×	√	×	×	×	×	×	×
PSAM / PSAR		×	×	√	×	×	×	√	×	×	×	×	×	×
Policy Manager		×	×	×	×	×	×	√	×	×	×	×	×	×
Auth System		×	√	×	×	×	×	√	×	×	×	×	×	×
Orchestrator	NED MGMT	×	√	√	√	√	√	× (R)	√	×	×	×	×	×
PSC	PSC TVD MGMT	×	×	×	×	×	×	√	×	×	×	√	×	×
	PSC SEC MGMT	×	×	×	×	×	×	×	×	×	×	×	√	×
	USER CTRL	√	×	×	×	×	×	×	×	×	×	×	×	×
EE	EE MGMT	×	×	×	×	×	×	×	√	×	×	×	×	√
	EE CTRL	×	×	×	×	×	×	×	×	√	×	×	×	√
PSAs		×	×	×	×	×	×	×	×	×	×	√	√	×

Figure 20: Matrix of interfaces for the Control Plane

C.7 The standard SECURED PSA I/O model

The objective of this section is to define a high level information model that describes which are the list of PSA interfaces that has relation with NED entities, specifically with PSC. The figures 10 and 11 illustrate the internal PSA architecture and are a useful context for this section.

It is interesting to highlight one of the main project requirements – namely isolation – in two main aspects:

- Isolation between PSCs of different users. Each user will have their execution environment containing his own applications (aka PSA). Some exceptions are in discussion like a Shared PSA which can be used by to service a number of users. This will be researched and develop along the lifetime of the project.
- Isolation between privileged entities and process from not-privileged user applications (PSA).

This isolation model forces the limits of PSA interfaces: PSA can't talk directly with other user's PSA.

There is a clear dynamic nature of PSC and PSA entities, because of dynamic instantiation of this elements in the NED when a SECURED service is required by the user. If we join this concept with the clear technological implementation dependency of the PSAs around their PSC, then an information model [15] will be the best approach in the interface definition. Each PSC implementation will implement the interfaces in a data model based on their own execution environment conditions.

Based on these principles these are the main PSA interfaces considered:

1. **PSA managment API**

All PSAs interact through this interface against the PSC management module. The common operating model is that PSC initiates a request to the PSA and the replies to it.

This interface defines actions to control the PSA behaviour. These are some of the main actions:

- actions related to control of the PSA (pause or freeze, restart one of the modules or restart all the PSA, end the PSA execution);
- actions related to the configuration process based on user policies translated to low level commands. These actions usually are performed in the first instantiation of the PSA. An advanced model would include an option that allow to reconfigure some modules or the full PSA.

2. **PSA control API**

   Again each PSA interacts throughout this interface against the Control module owned by PSC, and only with its PSC.

   The operating model in that case is different because depending on what action is executed, the initiator can be PSA or PSC.

   - Actions to monitor PSA status. Center in collect PSA or PSA modules status: running, stopped, valid, invalid,...
   - Exception handling. This action will manage any alert generated by the PSA itself caused by runtime exceptions, security alerts, ... All information or alerts that affects the secure execution of the PSA will be sent throughout this interface.

3. **PSA internal interface (Int-PSA) and external interface (Ext-PSA)**

   Main traffic associated to the user will be accessed by this interface. Int-PSA manages traffic originated at the user terminal (or previous PSA in chaining models). Ext-PSA manages traffic designed to be sent to a remote network (or next PSA in chaining models). By definition these interfaces will need strong requirements of capacity in traffic management, that must be supported by the NED. Probably additional capacities like rate-limit or packet marking prioritization will be recommended to offer enough QoS in Enterprise or Network operators scenarios with the objective of accomplish data transfer rate and minimal time delays.

4. **PSA offline interface**

   It will include any other additional traffic not directly related with the user traffic. This interface can be used to access external resources (e.g. Internet, ISP CDNs, user cloud services, etc) that the PSA may need. It will also manage traffic not associated with SECURED, but with the PSA itself (e.g. virus database update). In order to guarantee security and integrity of the PSA this interface will be implemented with secure protocols.