University of Zurich^UZH

Design and Implementation of a Blockchain-based Trusted VNF Package Repository

Manuel Keller
Zürich, Switzerland
Student ID: 13-795-125

Supervisor: Eder John Scheid, Muriel Franco
Date of Submission: May 2, 2019

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

University of Zürich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Network operators are under much pressure to improve their services: On the one side, they need to lower prices for customers, on the other side they have to invest in technologies and at the same time provide their services with great stability. For this reason, operators are turning to Network Function Virtualization (NFV).

In this thesis, first, the current works of using blockchain technology to enhance the security of NFV environments are provided. So far, there have been efforts to create blockchain-secured NFV Management and Orchestration systems as well as to set up trusted computing environments. These projects so far did not include the Virtualized Network Functions repository (VNF repository). However, the blockchain's properties could enhance the security in this area by allowing to verify a package's integrity without relying on a trusted third-party for remote attestation or a secure database. Thus, a design of a trusted VNF repository using blockchain technology is proposed. The smart-contract back end offers a package repository as well as a repository manager and is supplemented by a front end comprised of four distinct systems.

The proposed design is then implemented in the Ethereum network as a proof-of-concept. The smart contract is written in Solidity. The front end is based on the truffle and react framework. The solution design relies on an external NFV environment to deploy, manage and run the network functions.

The resulting implementation succeeds in enhancing the security of the VNF repository without relying on external parties. The system is without access control and thus represents an open market for VNFs that all interested parties can access. The transaction costs associated with the contract are reasonable and within useful boundaries. However, the open design requires well-designed incentives. Otherwise, malicious participants could abuse the system for financial benefit.

This work shows that a blockchain-based trusted repository for VNF packages is feasible and offers advantages over traditional techniques. Even though there are still challenges connected to it, it resolves a weak point in existing NFV systems and shows promise to be integrated in already blockchain-based NFV systems.

Zusammenfassung

Netzbetreiber stehen unter großem Druck, ihre Dienste zu verbessern: Auf der Kundenseite müssen sie die Preise tief halten, auf der anderen Seite sollten sie in neue Technologien investieren und ihre Dienstleistungen mit hoher Stabilität erbringen. Aus diesem Grund setzen Betreiber vermehrt auf Network Function Virtualization (NFV).

In dieser Arbeit werden zunächst die aktuellen Forschungsergebnisse zur Verwendung der Blockchain-Technologie zur Erhöhung der Sicherheit von NFV-Umgebungen vorgestellt. Es gab bereits Bestrebungen, Blockchain-geschützte NFV-Management- und Orchestrierungssysteme zu entwickeln und vertrauenswürdige Computerumgebungen einzurichten. Diese Projekte lassen allerdings das Thema des Repositorys der Virtualized Network Functions (VNF) aus. Die Eigenschaften der Blockchain könnten jedoch auch die Sicherheit in diesem Bereich erhöhen, indem sie es ermöglichen, die Integrität eines Pakets zu überprüfen, ohne sich auf einen vertrauenswürdigen Drittanbieter für remote attestation oder eine zentrale Datenbank verlassen zu müssen. Deshalb wird ein Design eines Trusted VNF-Repositorys auf Basis einer Blockchain vorgeschlagen. Das Repository-Backend basiert dabei auf einem Smart Contract, auf den die vier Systeme des Frontends zugreifen.

Das vorgeschlagene Design wurde im Anschluss im Ethereum-Netzwerk als Proof-of-Concept umgesetzt. Der Smart Contract wurde in Solidity geschrieben, das Frontend basiert auf dem Truffle Framework. Die Implementierung basiert auf einer externen NFV-Umgebung zur Bereitstellung, Verwaltung und Ausführung der Netzwerkfunktionen.

Mit der resultierenden Implementierung gelingt es, die Sicherheit des VNF-Repositorys zu erhöhen, ohne auf externe Parteien angewiesen zu sein. Das System ist ohne Zugangskontrolle und stellt damit einen offenen Markt für VNFs dar, auf den jeder zugreifen kann. Die mit dem Vertrag verbundenen Transaktionskosten sind angemessen und in sinnvollen Grenzen. Das offene Design erfordert jedoch gut gestaltete Anreize. Andernfalls könnten böswillige Teilnehmer das System zum finanziellen Vorteil missbrauchen.

Diese Arbeit zeigt, dass ein blockchain-basiertes Trusted Repository für VNF-Pakete machbar ist und Vorteile gegenüber herkömmlichen Techniken bietet. Auch wenn es immer noch Herausforderungen gibt, löst es einen Schwachpunkt in bestehenden NFV-Systemen und zeigt ein Potenzial für die Integration in bereits Blockchain-basierte NFV-Systeme.

Acknowledgments

I want to thank the Communication Systems Group team and Prof. Dr. Burkhard Stiller for giving me the opportunity to write my master thesis about such a fascinating topic. Special thanks go to Eder Scheid, who not only was my supervisor for this thesis but also in the Communication Systems Seminar, which was the point in time when I got interested in this topic. Also I thank my second supervisor Muriel Franco for his insights into NFV systems through his background with the FENDE project.

Lastly, I thank Claudia Vogel for helping me write this thesis as well as Florian Fuchs for proofreading.

Contents

Chapter 1

Introduction

The concept of Network Functions Virtualization (NFV), introduced in 2012 by the European Telecommunications Standards Institute (ETSI) [19], proposes to decouple network functions such as firewalls, Deep Packet Inspection (DPI), Intrusion Prevention Systems (IPS), and load balancers from their specialized physical hardware. With NFV, these functions are provided in a virtualized way and realized using generic Commercial Off-The-Shelf (COTS) hardware that can be deployed in any location, not just the Service Provider's (SP) premises. This virtualization approach offers the SP several advantages, such as increased scalability, flexibility, security, cost reduction, and a faster product life-cycle [16], because the Virtualized Network Functions (VNFs) are no longer physically bound to a vendor-specific hardware. Thus, VNFs can be developed by third-party developers with a low entry-barrier, fostering competition and the creation of innovative network services.

Blockchain is a recent technology, first described in 2009 in the Bitcoin white paper [17]. A blockchain is a data structure which allows for data to be stored in a distributed ledger. Depending on the implementation and the configuration of a blockchain-based distributed ledger system, different properties can be reached. Blockchains without access control (*i.e.,* public blockchains) operate on an incentive-basis. By design, any user can verify the state of the system. The incentives are designed in such a way that the nodes (users) verify the correctness of any new data that is added to the system. Such a blockchain's most important properties are immutability, and the decentralization of the data [31]. The former ensures that once the data is included in the blockchain it cannot be altered or removed; while the latter provides a high availability of the data, *i.e.,* the data will be available if there is at least one peer holding a copy of the blockchain.

The properties of blockchain form the perfect environment for the execution of Smart Contracts (SC). In the Bitcoin network [17], SCs facilitate the transfer of funds between untrusted entities. In the Ethereum network [8], SCs are written in a Turing-complete programming language, called Solidity [12]. This allows more functionality and helps to enforce a variety of contracts through crytographic principles [4]. SCs deployed in blockchains that provide Turing-complete languages can be used to facilitate trusted exchanges between untrusted entities and the trusted and correct execution of programmed

SC code. These properties can be used in the context of NFV solutions to address the security issues regarding central databases for package integrity verification.

1.1 Motivation

The deployment of NFV solutions faces a major challenge regarding incorporating trust to stakeholders. Research has been conducted in how to address this issue in the NFV computing environment with the introduction of Trusted Platform Modules (TPM) and remote attestation services [23]. Although these systems work well to verify the state of the NFV environment, they rely on a central database to verify package integrity. This thesis proposes to improve VNF package verification by introducing a blockchain-based trusted repository. This can then be used to provide trusted information concerning the VNF packages acquired by stakeholders. In this sense, stakeholders are not bound to rely on central trusted authority, but rather on a distributed and highly available data source.

1.2 Description of Work

In this thesis, a survey on existing NFV marketplaces as well as repository solutions is given. The focus lies on solutions which already implement one or more elements through a blockchain-based distributed ledger. Then, a blockchain-based trusted repository is designed and implemented as a Proof-of-Concept (PoC). The repository is intended to later be integrated with traditional database-based package verification environments; thus acting as a trusted database containing VNF package information. Moreover, the repository allows users to acquire VNFs without the need of a Trusted Third Party (TTP) using an SC deployed on the Ethereum network. The SC automatically transfers any license fees to the vendor once a VNF is acquired, and sends the VNF package link to the buyer before verifying package integrity. Furthermore, to assess the feasibility of implementation, a front end was implemented that allows interaction with the repository.

1.3 Thesis Outline

The remainder of this thesis is structured as follows: Chapter 2 provides the theoretical background of the thesis. It explains the underlying concepts of NFV, details VNF packages, and explains the blockchain technology and SCs. The subsequent chapter 3 provides related works to the thesis by describing existing VNF marketplaces and existing uses of the blockchain technology for management and orchestration in the NFV environment. In Chapter 4, the solution's architecture and design is explained. It is followed by implementation details including the user interface, and its functions. In Chapter 5, the design and implementation are evaluated against the thesis' goals and requirements. For this purpose feasibility, management and security as well as economical aspects are considered. Lastly, Chapter 6 concludes the thesis with a summary and future work.

Chapter 2

Theoretical Background

Technology is changing rapidly and continuously. Thus, it is crucial to establish a common background and overview before delving into the technical details of this thesis. This chapter aims to provide this overview and explains technologies and concepts later used in the solution's design and implementation.

2.1 Network Function Virtualization

Network functions, such as firewalls, load balancers, switches, and routers, generally require specialized physical hardware. Network Function Virtualization (NFV) is a proposal to decouple the functions from their hardware such that they can be supplied via software on generic server hardware. These Virtualized Network Functions (VNF) have great potential to change operations of Service Providers (SP). The European Telecommunications Standards Institute (ETSI) has lead, since 2012, the development and standardization of NFV. It has released the ETSI Management and Operation (ETSI MANO) [19] architectural framework which is considered to be the de-facto industry standard for the implementation of NFV solutions. The main components of the MANO framework (Figure 2.1) are:

- *NFV Orchestrator*: Acts as a coordinator that connects all VNFs into the required structure to offer network services. It manages the resource requirements of VNFs and is responsible for authentication and authorization of network requests.
- *VNF*: A functional block with defined interfaces and functional behaviour that perform a network service.
- *VNF Manager*: Each VNF instance has an associated VNF manager which works in coordination with the NFV Orchestrator and manages the VNF instances in terms of instantiation, configuration and termination.
- *NFV Infrastructure (NFVI)*: The generic server infrastructure on which the VNFs are deployed.

- *Virtualized Infrastructure Manager (VIM)*: Manages the virtualized NFV infrastructure and allocates virtual to physical resources.

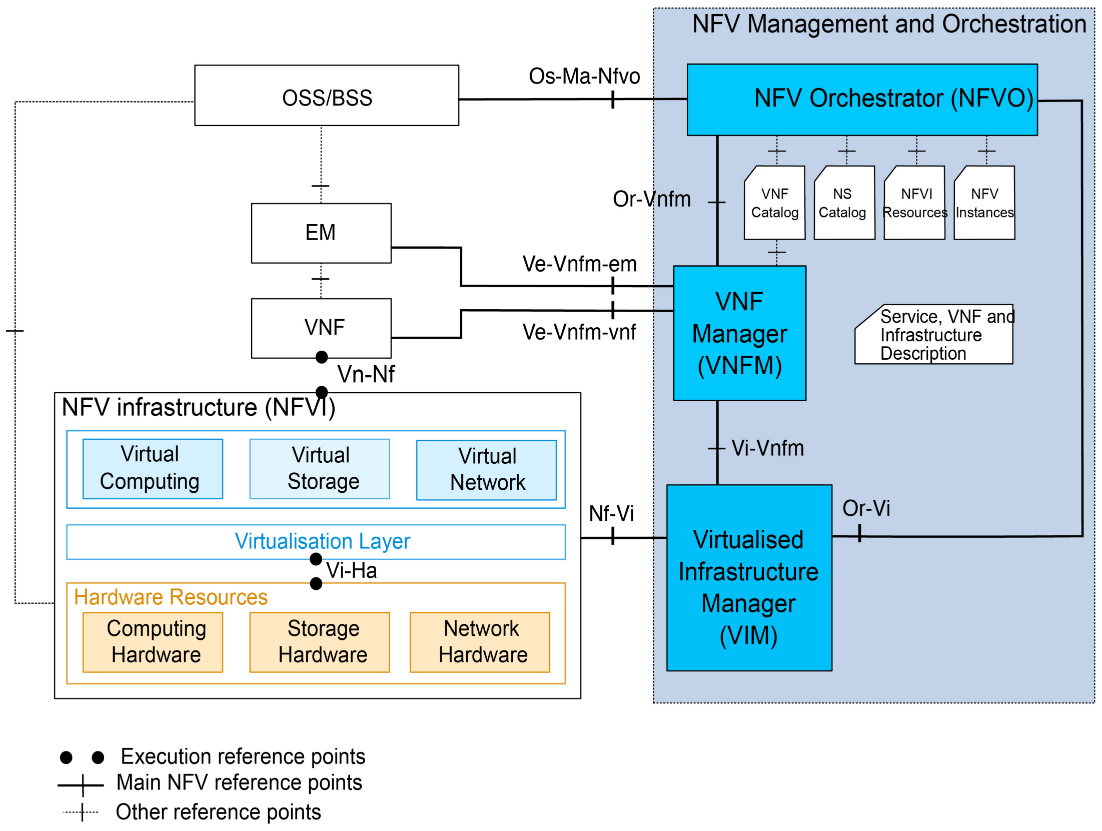


Figure 2.1: ETSI NFV MANO Architectural Framework [19]

The ETSI MANO architectural framework was created by an Industry Specification Group (ISG) NFV consisting of ETSI members. These industry partners have created this standard to improve compatibility between vendors which in turn means more freedom of choice and flexibility. This is a core requirement of the creators of the standard. As seen in the architecture model, depicted in Figure 2.1, the standard does not include specifications about the design of VNF, but rather represents a specification on how the services are incorporated in an NFV environment [19].

2.2 VNF Packages

VNFs are typically shared and deployed in form of packages which can be used on virtualization infrastructure. Such a package contains all relevant code and interface definitions necessary for the operation of a network function and is deployed on the NFVI using one or more Virtual Machines (VM). For configuration, management and orchestration, it is then integrated into an NFV environment (refer to Section 2.1) which links the services to create the desired network function chains [19]. Each package needs to include detailed specifications for operation and deployment in order to work in the context of the NFV environment. This challenge is tackled by ETSI's introduction of a package template called Virtualized Network Function Descriptor (VNFD). This defines the deployment and behaviour through three key components [21]:

- *Topology*: All necessary nodes (*i.e.,* VMs) are specified including their connectivity and relationships. Virtual Deployment Units (VDU) are used to describe capabilities and requirements, such as disk size, memory size, and required CPUs.
- *Deployment aspects*: This section describes aspects such as deployment parameters, instantiation constraints, scaling, among others. Additionally, deployment flavours are used to describe differing requirements and constraints depending on the deployment type. For example, this part would specify that a large scale deployment may need an additional node for supervision.
- *VNF Lifecycle Management (LCM) operations*: Provides a description of management operations with their input parameters.

Based on the VNFD, the industry group OASIS created the Topology and Orchestration Specification for Cloud Applications (TOSCA) data model standard which implements ETSI's specifications. This model is written in YAML [21]. Listing 2.1 presents an excerpt from a TOSCA descriptor.

ETSI's VNFD and its implementations such as TOSCA mean that a VNF's specifications are stored in a single file. Thus, the NFV solutions can rely on those specifications to verify compatibility, set up the required environment and also access lifecycle methods. This leads to better cross-compatibility and reduces the complexity of the NFV environment.

```yaml
topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      properties:
        image: cirros-0.4.0-x86_64-disk
        availability_zone: nova
      capabilities:
        nfv_compute:
          properties:
            disk_size: 10 GB
            mem_size: 2048 MB
            num_cpus: 2
```

Listing 2.1: Excerpt of a TOSCA Descriptor for a Tracker running on CirrOS [22]

2.3 Blockchain and Smart Contracts

Blockchain is the underlying technology of Bitcoin first introduced in 2009 [17]. Since then, it quickly became popular for its novel way of storing transaction data in a distributed ledger. First mostly used for cryptocurrencies and other transaction-based applications, it is now seen as a tool applicable to a variety of problem sets. The blockchain's ledger is comprised of blocks of data. Each block is cryptographically sealed using hashing. As each block contains the hash of the previous block, this chain of blocks guarantees practical integrity: Data in sealed blocks cannot be changed, otherwise all following blocks would become invalid. A block is chosen by a consensus mechanism which is based on economic incentives. If there is an accidental split in the blockchain, it is the longest chain that is chosen for the same economic incentives. In practice, this means that the more follow-up blocks there are to a sealed block, the more secure data stored in it becomes.

SC rely on blockchains to "facilitate, execute and enforce the terms of an agreement between untrusted parties" [1]. In contrast to traditional contracts, SCs do not rely on trust or third-parties such as banks to enforce an agreement. In Bitcoin, the smart contracts are restricted to financial transactions only and are not Turing-complete (*e.g.,* loops are not possible). Other blockchains such as Ethereum [8], introduced in 2015, do have Turing-complete smart contract capabilities. This enables more possibilities but also introduces new security risks. [18, 4].

The Ethereum network is account-based. This means that unlike the Bitcoin network, funds are always in one place, there is no need to keep track of all changes and transactions in the network as the account state is updated whenever a transaction concerning the account occurs. SCs in Ethereum also have such a state. Nodes in the Ethereum network always store the most recent state of each contract, which enables the storage of data. Due to the Turing-complete nature of the network, all computing operations have to be paid to prevent Distributed Denial of Service (DDoS) attacks. These transaction fees are called gas. Only if an SC function is called with sufficient gas it is executed and the new state is stored in the network. The computations themselves are done in the Ethereum Virtual Machine (EVM) in bytecode. There exist several programming languages that can be compiled into this bytecode, the most popular being Solidity [4].

2.4 Solidity

Solidity is the leading programming language for SCs in the Ethereum network. It became a popular language to develop distributed applications (DApps) over the past years with now over 2000 DApps deployed in the Ethereum network [11]. It is designed with influences of C++, JavaScript and Python with the syntax resembling JavaScript and static typing. It was designed intentionally in a simple way to allow easy development of secure contracts. A Solidity file can contain multiple contracts. The language also supports inheritance and polymorphism which is for example used to create new ERC20 tokens[1] that are compatible with all common wallet applications. Each contract can contain the following elements [12]:

- *State Variables:* Such variables are stored in the SC's storage and can be changed by updating the state of the SC.

[1]The ERC20 token standard is a list of rules that simplify the creation of a new token, including functions such as `transfer`

- *Functions:* Functions are executable code that can be called. The visibility and accessibility of the function can be defined, *i.e.,* there exist external, public, internal, and private functions.

- *Function Modifiers:* These can be used to change the behaviour of a function, *e.g.,* checking a condition before execution or control access to a function.

- *Events:* Events can be emitted by the contract and are stored in the transaction log. Events can be captured in a front end via JavaScript callbacks and are a convenient way of further processing data.

- *Struct types:* Structs are a class-like type structure which are composed of variables of other types. Struct variables can be used in the definition of further structs as well.

- *Enum Types:* An enum type has a defined set of possible values. A variable of this type can take one of these values.

The Solidity language simplifies the development of SCs considerably. However, it also comes at a cost: As it is the most used programming language for SCs in the Ethereum network, it is also the focus of hackers and malicious participants. Any newly found vulnerability in the programming language thus impacts the security of all previously deployed SCs.

Chapter 3

Related Work

In this chapter, works that are related to the core topic of this thesis are described. First, an overview of existing marketplaces for VNFs and their repository solutions are presented. Then, previous efforts that combine the blockchain technology and NFV are described. Finally, a discussion regarding the advantages and shortcomings of such works is conducted.

3.1 Marketplaces for Virtualized Network Functions

The standardization approach that industry groups are pushing in the NFV environment context fostered the research and creation of marketplaces for VNF packages. As more and more vendors are developing solutions adapted to this standard, the competition around such topics also grows.

FENDE [5] is a Marketplace and a Federated Ecosystem for the Distribution and Execution of VNFs. It presents the user with all compatible network functions currently listed in its repository. Unlike previous works, it also includes management and orchestration tools which allow users to deploy and manage licensed services in the same ecosystem. As such it is one of the first to combine marketplace and MANO environment in one solution. As the system is configurable for the use with both public and private NFVI hosting, institutions that desire to run network functions on-premise can use it as well. FENDE's architecture (refer to Figure 3.1) is composed of three layers:

- User layer: The part of the system that is responsible for user interactions with three types of users: customers, reviewers, and developers.
- Data layer: Acts as the backend for all user layer functions and offers a communication API. Also, it contains the catalogue of VNFs and the repository that stores package information. The repository manager is responsible for any changes in the repository and keeps local copies of the VNF packages.

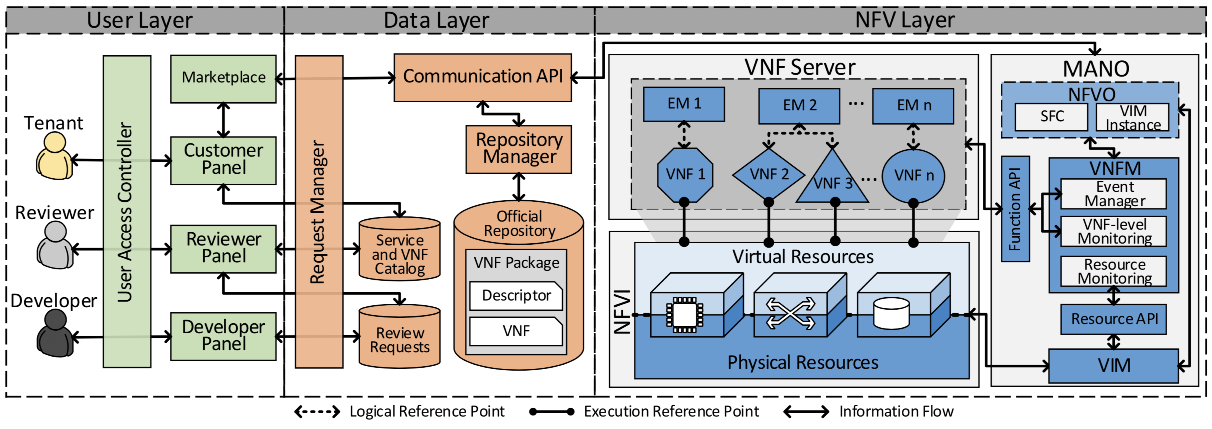


Figure 3.1: FENDE Architecture

- NFV layer: This is the NFV environment responsible for running and managing the VNFs. It is an adapted version of the open-source NFV environment OpenStack.

FENDE uses a traditional database-based repository in the data layer for its VNF functions. It relies on the TOSCA descriptor for deployment and lifecycle management functions [5].

In [6], the authors discuss the challenges that arise in the FENDE ecosystem: *(i)* Business model: The FENDE ecosystem needs to support all business models of VNF vendors. This includes fixed-price and pay-as-you-go pricing as well as newly proposed methods of pricing such as auction-based and custom-built (*e.g.,* a specific VNF is implemented according to specifications negotiated between client and vendor). *(ii)* Auditing: Clients need the possibility to verify that a VNF is providing network functions as promised. *(iii)* VNF recommendation: In a growing market for VNFs, it is important to help the clients choose the right offer according to their requirements. Classification and clustering techniques may need to be explored to solve this challenge. *(iv)* Security: As the VNFs are developed by third-parties and used in different NFV environments, security is a key challenge. Customers need ways of ensuring the integrity of VNF packages and secure ways of sending commands to the management and orchestration modules of the NFV solutions.

T-Nova [31] is an architecture proposal intended for network operators' services. It would enable them to virtualize their own network functions as well as offer them to their clients in an on-demand, per-customer way. This lets them provide network services to their customers without having to deploy specialized hardware on each customer's premises. A marketplace is put in place for customers to acquire and instantiate their required network services on-demand. This greatly improves the speed and efficiency of service rollouts, utilizes hardware in a more efficient manner, introduces improved monitoring capabilities and is much easier to maintain and upgrade. With industry participants that are actively involved in the development of the ETSI NFV standard and support of researchers the aim is to create an integrated ecosystem that is capable of handling the industry's requirements. The T-Nova platform is deployed on network operators premises and access-controlled by the network operator. Thus, the users have to rely on the operator's choice of VNF offers. The repository itself is based on a traditional database, specifically MongoDB [31].

3.2 Trusted NFV Environment

ETSI has released a whitepaper on security and trust guidance [20] in which they analyze and discuss the challenges on those topics. Trust is identified as a key issue regarding the NFV environment which should be validated before and during the runtime of any VNF. As use cases, among others, the following were constituted:

- Establishing a trusted execution platform on which NFV system and VNFs are executed as expected.
- Establishing trusted entities in the NFV environment by verifying all software, services, processes and policies to protect, *e.g.,* from maliciously injected VNF code.
- Establish trusted relationships between the NFV Orchestrator, the NFV Manager and the VIM (refer to Figure 2.1) to manage and orchestrate effectively and safely. Additionally, these entities should not be virtualized using the NFVI to create a barrier between MANO and VNF execution.
- Establish trusted VNF lifecycle management to operate as well as configure services in a trusted way from instantiation, to operation and termination.

Establishing trusted components is especially important, as a compromised VNF may influence the correct execution of related VNFs (*i.e.,* down- or upstream in the network function chain) [20, 23].

In [23], the authors further discuss the incorporation of trust in NFV environments as one of the challenges of NFV. A lack of trust leads to negative implications: NFV solutions are security-relevant for clients, therefore the environment and the VNFs themselves need to be secure and trusted. Two common methods of addressing trust are identified:

- Trusted Module Platform (TPM): A hardware-based root-of-trust module that is being used to create a chain-of-trust to establish a trusted runtime environment for VNF packages.
- Remote attestation server: The platform is verified during the runtime. This enables providers to verify platform trust *e.g.,* before launching new instances of VNFs.

Specifically, in terms of NFVI security, boot time verification using a TPM's launch control policy (LCP) as well as a run time verification by an attestation server is proposed. To determine VNF security before any launch, a VNF package integrity check is conducted via a remote trusted server called TSecO (external trusted security orchestrator). It compares the hashes of the images stored in the NFVI image database with the hashes

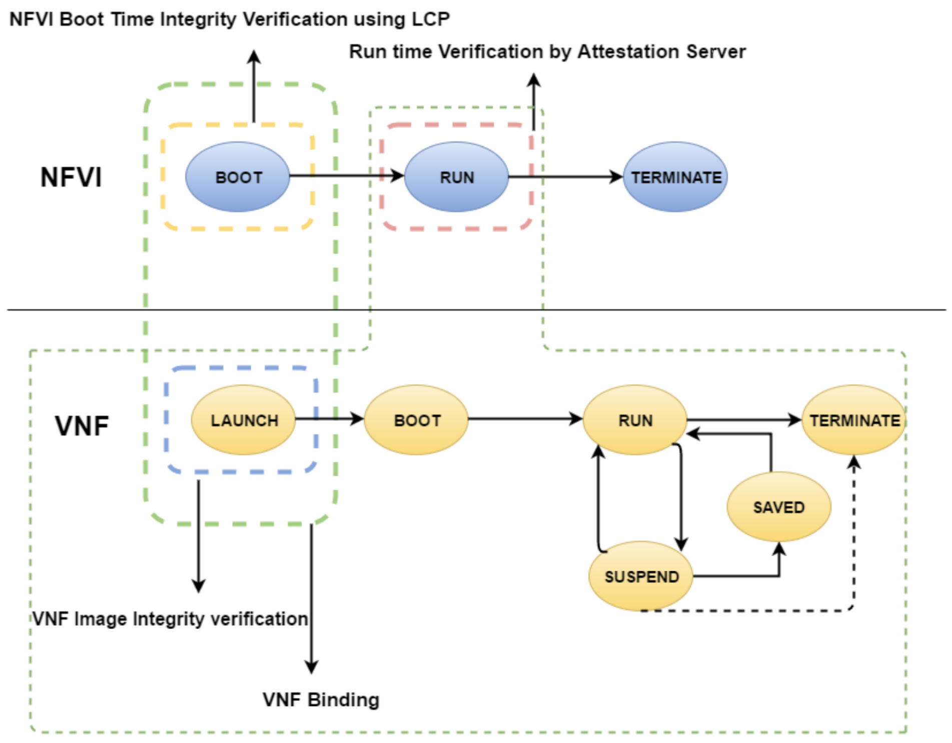

NFVI Boot Time Integrity Verification using LCP
Run time Verification by Attestation Server
NFVI
BOOT
RUN
TERMINATE
VNF
LAUNCH
BOOT
RUN
TERMINATE
SAVED
SUSPEND
VNF Image Integrity verification
VNF Binding

Figure 3.2: Security in VNF Lifecycle States [23]

that have previously been signed by a signing authority and stored by the TSecO. Additionally, VNF binding is proposed to be used during all stages, where VNFs are bound to certain virualized hardware. This aims to guarantee the execution of VNFs in a certain geographical area, *e.g.,* to adhere to privacy laws [23]. Figure 3.2 shows these security measures for each state of VNF execution and NFVI state.

3.3 Blockchain-based NFV Management and Orchestration

As blockchain is independent of any authorization entity and can create trust between untrusted network participants, it is also useful for areas other than financial transactions where trust needs to be established. Also, the practical immutability of public blockchains is a useful tool for areas where auditability is crucial. The combined properties lead to research about using blockchain in the NFV Management and Orchestration environment. Two such approaches are detailed below.

VMOA [7] is an authentication model that establishes a trustful VM environment. Instead of having an internal or external trusted authenticator, the authors propose to establish a VMOA blockchain to offload the authentication responsibility to a distributed ledger. In this system, each orchestration request is sent to a blockchain, authenticated and only then sent to the virtualization server. If successful, the VM manager reports the success to the blockchain. Each step is stored in the blockchain and is thus auditable. The authors propose an implementation of a private blockchain based on the Hyperledger framework [29].

In [26], the authors propose a blockchain-based NFV MANO solution. SINFONIA stands for Secure vIrtual Network Function Orchestrator for Non-repudiation, Integrity, and Auditability and is designed for datacenters in which multiple network services from different clients are deployed. The authors identify that each VNF configuration update must be:

- *Confidential*: Ensures that no vulnerabilities are exposed.
- *Anonymous*: The tenant identity must not be exposed.
- *Authenticated*: Only verified requests must be forwarded.
- *Traceable*: To ensure auditability of past configuration states.
- *Accountable*: To ensure identity verification.
- *Permanent*: To have the configuration history accessible.

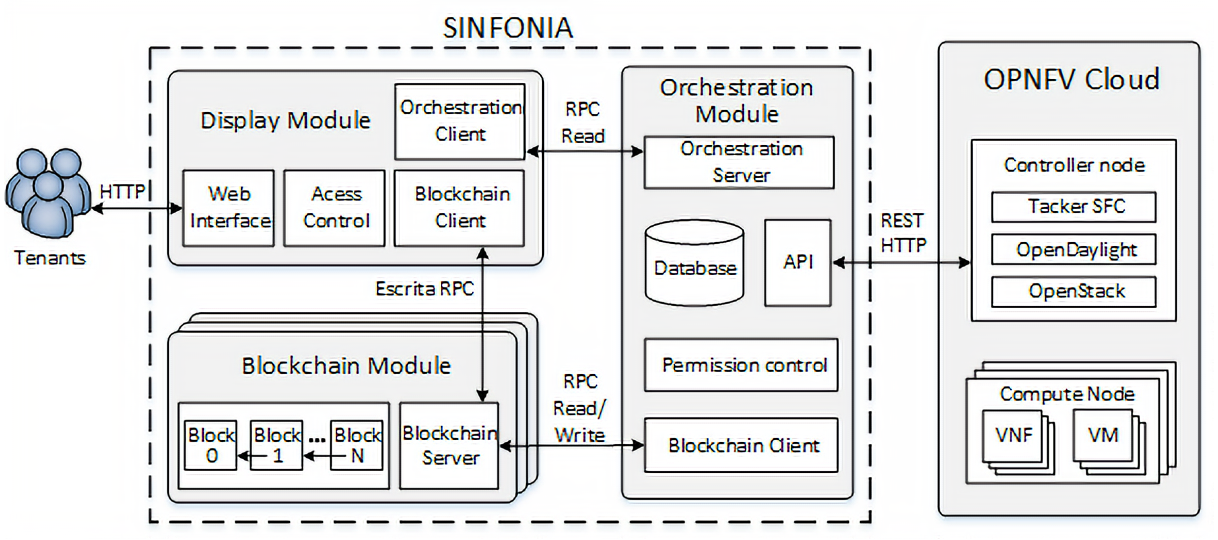


Figure 3.3: SINFONIA Architecture [25]

The authors have designed a blockchain-based NFV architecture which addresses all requirements (see Figure 3.3). The prototype implementation shows that the proposed architecture ensures high availability and eliminates a single point-of-failure [26, 25]. However, it is not clear of where the blockchain nodes are located, and which are the incentives for peers to maintain these blockchain nodes.

3.4 Discussion

As seen in Section 3.1 there have been efforts in developing marketplaces for VNF packages. These lead to the creation of systems where users can access a VNF repository containing various packages that can be deployed quickly and easily. By integrating the marketplace into NFV solutions such as NFV OpenStack, they offer all the base solution's advantages while also offering easy access to a great number of packages which increases competition.

A common challenge of such marketplace-based and traditional NFV solutions is establishing a trusted computing environment. The solutions of the current research presented in Section 3.2 are based on hardware sources of trust or use remote attestation. However, both systems are reliant on a trusted entity: In the case of TPM, the producer of the chip has to be trusted to secure the system. With remote attestation, the external provider of trust has to be trusted. Here, a blockchain-based system may offer advantages over these centralized systems.

With VMOA and SINFONIA (see Section 3.3), the authors addressed the trust challenge by incorporating blockchain technology into the management and orchestration of NFV platforms. This way, they secure the computing environment and the solution's configuration. However, they do not extend to the VNF repository. This leads to a security flaw: Malicious actors can potentially gain access to the central VNF repository to inject malicious code. Even though run in a secure environment, this compromises the security of the whole system. ETSI has identified this challenge and lists the VNF package verification in their Trust and Security Guidance specification as a requirement for a save NFV environment [20]. Extending the blockchain-based system to repositories would be a simple way of guaranteeing VNF integrity, enhancing the solution's security considerably.

As discussed, efforts have been made in providing trust in the NFV management. Still, there is considerable amount of research in this field yet to be done. So far, the research has shown that a trusted NFV environment should extend to the VNF packages and could be blockchain-based. However, current projects have yet to extend to the repositories.

Chapter 4

Blockchain-based Trusted VNF Repository

This chapter describes the design and implementation of a solution that incorporates trust in the VNF repository context by using blockchain technology. This solution tackles the challenge of VNF integrity verification described in Section 3.4 by leveraging the SC's properties of immutability and accessibility to create a trusted VNF repository, where it is possible for a user to verify the integrity of VNF packages running on a local virtualization platform. Moreover, it allows the author to automatically receive any licensing payments once a VNF is acquired. It should be mentioned that this trusted repository can be integrated into existing platforms, such as the ones described in Chapter 3.

4.1 Solution Design

Figure 4.1 depicts the proposed architecture of the solution. It is composed of three main systems: *(i)* Front end responsible for user interaction, *(ii)* Blockchain-based Repository back end handling the requests, and *(iii)* NFV back end executing the VNFs. It is worth mentioning that the third system was not implemented as it can be provided by any third-party NFV solution. An overview of the architecture and the main systems is presented below with more detailed descriptions of the components presented in the following subsections.

As users of the system, two distinct groups were identified: *(i)* Customers want to acquire VNF packages to execute in their NFV environment as well as review their acquisitions. *(ii)* Developers want to offer their packages in the repository and thus need to register as well as maintain their offers.

The first system, representing the front end, is where users and developers interact with the solution. It includes four components:

- *Registration and Upgrade System*: This system is used by developers to register, maintain, and otherwise manage their VNF packages in the repository.

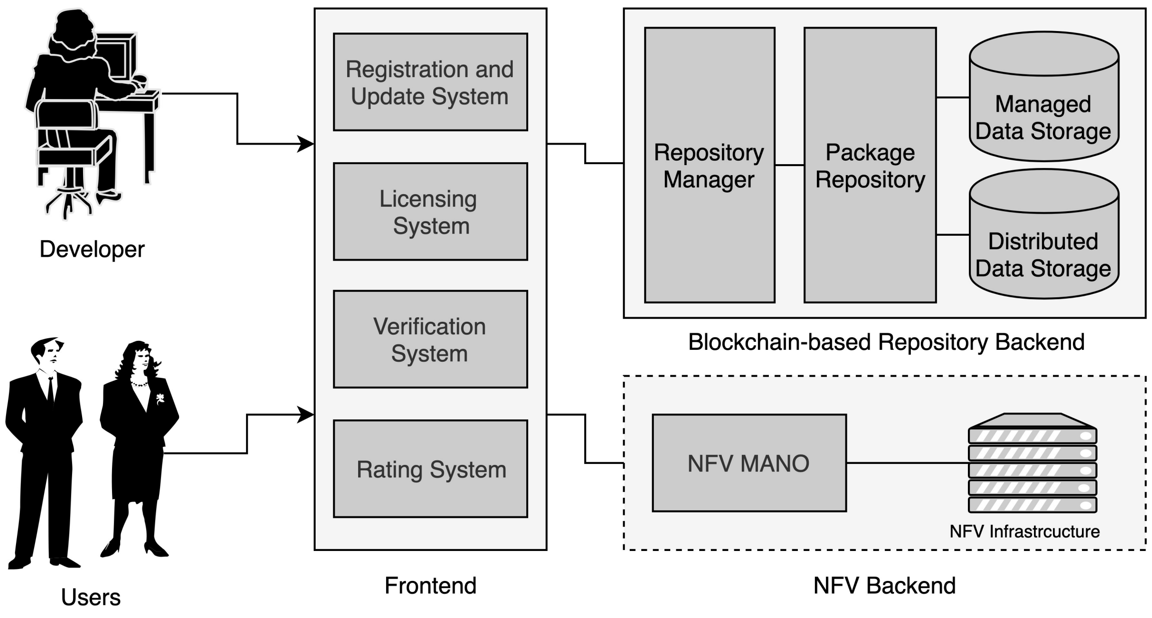


Figure 4.1: Proposed Architecture

- *Licensing System*: Using this system, customers can acquire VNF package licenses.
- *Rating System*: Verified licensees can rate their VNF packages to help others choose packages and increase the trust in the offers.
- *Verification System*: This system is in place to enable buyers to verify the integrity of a VNF package. Also, it enables the verification of a buyer's license for rights management purposes.

The second system is the blockchain-based repository back end and comprises the following components:

- *Repository Manager*: The manager accesses the package repository and offers the users functions to query and maintain the repository. It controls access and authenticates the user such that, depending on the function, only authorized users can perform operations.
- *Package Repository*: This is the database where package details are stored.
- *Data Storage*: The VNF packages can be either stored using *(i)* Managed data storage where a traditional server is used to host the data or *(ii)* Distributed data storage, such as the InterPlanetary File System (IPFS) [2], where a Peer-to-peer (P2P) network is used to increase availability and integrity of files.

The third system represents an existing third-party NFV solution that will set up, configure and execute the VNFs after they have been acquired. This can be any NFV solution (*e.g.,* the NVF solutions presented in Chapter 3) that is capable of connecting to the repository using an Application Programming Interface (API).

Attribute	Description	Use	Example
Name	Name of image	Marketplace title	NextGeneration Firewall
Description	Description of functionality, selling points, advantages	Marketplace description	The next Generation Firewall is the next big thing in NFV technology
Description iFrame	iFrame for further customized description in marketplace	Marketplace description	https://www.cisco.com/marketplace/ NGFW_iFrame.html
Image link	Customized marketplace cover art location	Marketplace description	https://www.cisco.com/marketplace/NGFW.jpg
Vendor	Vendor name	Marketplace description	Cisco
Family	Package family name	Marketplace description	VNFX 2000
VNF category	Categorization of VNF package	Marketplace categorization	Firewall
Supported Platform	Supported platforms to run package	Compatibility, marketplace filtering	{[Openstack: version: ">1.2", testedOn: ["1.2", "1.3"], GT-FENDE: version: ">1.2", testedOn: ["1.2", "1.3"]]
TOSCA descriptor	TOSCA JSON describing nodes, services, relationships of package	Compatibility and deployment	{"vnfd": { "name": "vnfd2", "description": "Click on OSv Template", "service$_{t}ypes$" : [. . .
Licencing options	Licensing options selectable by buyer	Licensing	[Creative Common BY NC ND: cost: "0.00", duration: "-1", Commercial License: cost: "150.00", duration:"30"]
Vendor address	Licensing payments destination address	Licensing	0x3273c04FCd088365730B2a83448100862aF535DE
Repository link	Link to package repository (package location)	Retrieval of VNF image	https://get.cisco.com/VNFX2000/ NextGenerationFW.qcow2
Release date	Release date of package	Metadata	22.01.18
Build name	build name / number of package	Marketplace, update history	NGFW-qcow2-release1.5.2.2-definitive-22012018-Ce3Bd7
Version number	Version number of package	Marketplace, ratings	1.5.2.2
Size	File size of VNF package	Marketplace description, validation	1253.43Mb
Hash	Keccak-256 hash of package	Package integrity validation	0x57bb94d96eab149447ff109ff941fe8f9117f2cae80e3f0dfcfe4

Table 4.1: VNF Package Definition

4.1.1 Registration and Update System

The registration and update system is used by developers to submit new VNF packages to the repository. Also, it is where developers can maintain their existing packages, *e.g.,* to update the package to a new version or to update information that is stored in the repository. Table 4.1 shows all relevant attributes and metadata of VNF packages that can be stored in the repository. The table contains each attribute's name, short descriptions, the intended usage in the system, as well as an example. This component allows vendors to change any of these attributes and the possibility to remove the offering from the repository.

4.1.2 Licensing System

This system is responsible for handling the customer request to acquire a VNF package. It collects the licensing fees from the customer and enables the customer to access the package after the funds have successfully been transferred. Figure 4.2 shows the process of acquiring a package: First, the user requests a license of a VNF package through the front end. With the request, the customer sends the licensing fee as well as any transaction fees that occur. The Smart Contract checks if sufficient funds were included in the request and if so, transfers the licensing fee to the vendor. Then, it reads the package data from the repository. Afterwards, a licensing event with all necessary information to retrieve the VNF and to execute it in the NFV environment is emitted. The front end captures this event and proceeds to retrieve the package data from the external data storage. The package can now be deployed and used in the NFV environment.

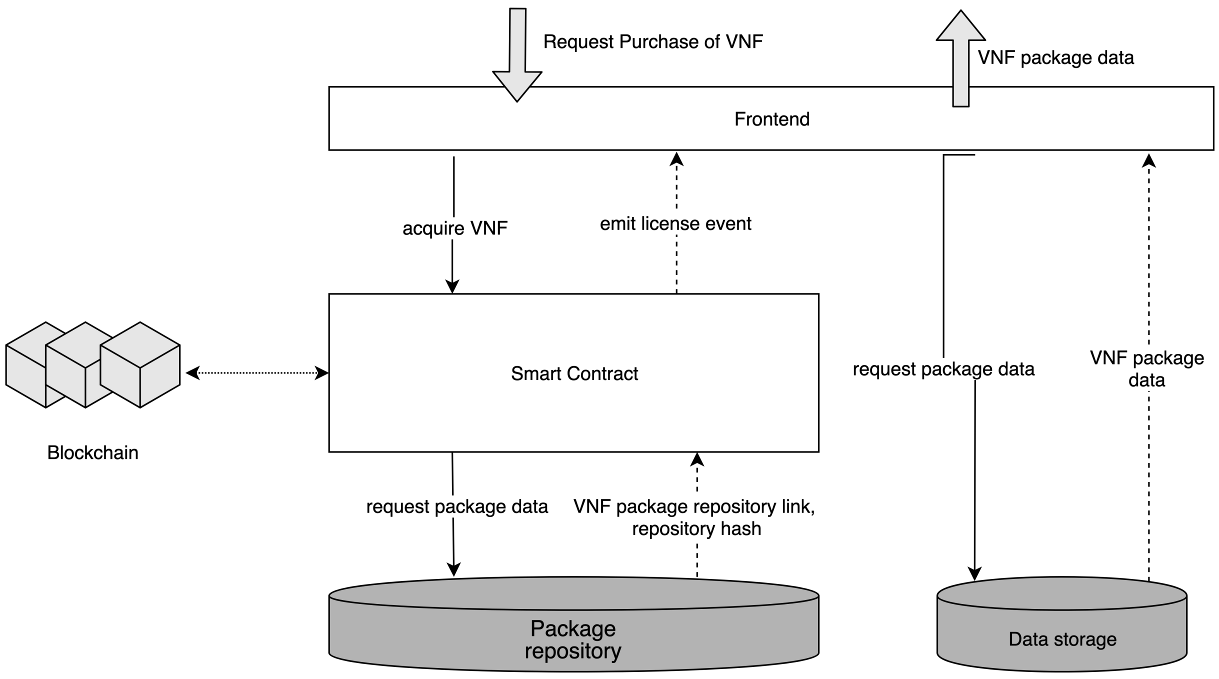


Figure 4.2: Data Flow of Acquiring a VNF Package

4.1.3 Verification System

Verifying the integrity of the VNF package before deployment and execution is crucial as it ensures that no tampering with the code or file corruption has happened after a package has been retrieved from the repository. For this reason, a verification system is included in the solution. It allows image integrity verification by comparing the hash of the downloaded package with the hash previously generated when the package was added to the repository. Such a verification can happen in two stages of the deployment:

- Before deployment: When a new VNF package is acquired, the system retrieves the package and verifies it against the information stored in the trusted repository.
- During runtime: The system offers capabilities to re-retrieve the hash and to re-verify the package integrity. This is useful before lifecycle operations such as up-scaling to more instances. Also, it allows to re-retrieve the VNFD to verify the correctness of configuration and lifecycle operations.

4.1.4 Rating System

The package repository has no access control, so any interested party can register new VNF packages. Also, there is no curation of the repository's offerings. This leads to a trust issue, as malicious parties may register packages that do not adhere to their specifications. Thus, the customers need another way to assess the quality of an offering. For this reason, a rating system was included: It allows licensees to give a rating which will be presented to future customers. In Table 4.2, all relevant attributes of such ratings are listed. It contains name, description, usage and an example for each attribute. It is worth noting that these attributes offer the possibility to write a in-depth rating text. However, for easier classification, there is a summary as well as a rating score which reflect the overall satisfaction with the package.

4.1.5 Blockchain-based Repository Manager

The repository manager is the only component that is accessing the repository and as such, is responsible for creating, managing and maintaining repository entries. It precedes the repository and acts as an intermediate party between user and repository, receiving all requests to the blockchain-based repository back end and accessing the repository as necessary. Therefore, this component offers an API for all functions needed in the front end . When a function is called, the repository manager authenticates the user and if authorized, executes the function call and returns the result.

The repository manager is based on a smart contract. If one of the functions is called by the front end, it is thus executed on a blockchain VM (refer to Section 2.3). This means the code is running in a trusted environment and is capable of handling secure data such as authentication of the request. Additionally, it can be used to perform access control and data validation before it is stored in the repository.

Attribute	Description	Use	Example
Package ID	Identifier of rated package	Show target VNF package, rating filtering	3fsdf4e-34rgs
Package version	Rated package version	Show target VNF package, rating filtering	1.3.1.0
Rating score	Rating score on scale	Rating summary	9 of 10
Summary	Summary of rating	Rating title	Works as intended
Description	Rating full text	Rating details	[...]
Advantages	List of advantages	Rating details	performance; quick deployment
Disadvantages	List of disadvantages	Rating details	few customization options
Platform	Platform used by rater	Rating details	OpenStack 1.3.5
Rater	Name of rater	Rating metadata	Paul Simon

Table 4.2: VNF Package Rating Definition

4.1.6 Blockchain-based Package Repository

The package repository stores all relevant VNF package details (refer to the VNF package definition in Table 4.1) together with acquired licenses, ratings and verification information. The repository is only accessible through the repository manager as described in Subsection 4.1.5. As it is based on an SC, all information stored in the repository will subsequently be stored in the underlying blockchain network. This incurs cost; thus growing with increasing amount of data. In practice, this means that storing large amounts of data should be omitted and the repository data should be downsized as much as possible. In the package repository context, the most amount of data is used on the VNF package. To decrease cost, only a link to the VNF package location is to be stored in the blockchain. The package data itself should be hosted on external data storage, even though this introduces a new problem in that the external data storage cannot be trusted. However, the verification system included in the solution allows to verify the integrity of the packages such that this challenge is tackled.

As this solution only represents a PoC, not all attributes of the VNF package definition (refer to Table 4.1) were included. Instead, they were reduced to include only the most relevant attributes that can be divided into four areas of usage:

- Marketplace descriptions: name, description, version,
- Marketplace Categorization: service type, resources, requirements, ratings
- License payments: author, price
- Package verification: repository link, repository hash

4.1.7 Data Storage

The SC-based package repository does not offer storage for the VNF package image. Instead, it is stored externally and can follow two different approaches: *(i)* Managed data storage, or *(ii)* Distributed data storage.

Managed Data Storage

Managed data storage was chosen as an option as it is the most straightforward approach to storing data. Another advantage is that it could perform access control if necessary, based on licensing information in the SC. In this approach, the files are stored on a centralized server. In the context of VNF packages, this means that the packages are stored in a source code repository, *e.g.,* GitHub or GitLab. Alternatively, the VNF code can be stored on a dedicated server controlled by the package author.

This approach to data storage represents an untrusted data source, as a central management of the server can tamper with the link's target files unnoticed. Thus, if this method is used, it is crucial to verify the integrity of the package before it is used in a production environment.

Distributed Data Storage

In a distributed data storage environment, the package data is registered in a P2P file sharing network and accessible through a network protocol. For example, IPFS could be used [2]. In this protocol, data can be registered in the network and is then accessible through the data's hash, which is used as key. To keep a local copy of a file, they have to be pinned. If one or more users have pinned a file (*i.e.,* possess a local copy), the files can be queried through the hash (*i.e.,* file key). Figure 4.3 shows both the process of file registration as well as retrieval.

Any change in the file would result in a new hash, thus in a new key. In this sense, the user querying for a file can be certain that if accessible, the file is in its original state. Thus, this form of data storage represents a trusted data source.

4.1.8 NFV Back End

As mentioned in Section 4.1, the NFV back end can be provided by any NFV solution and was not implemented in this work. This system is responsible for downloading the VNF images acquired from the blockchain-based repository, verifying them and subsequently deploying them on an NFV stack.

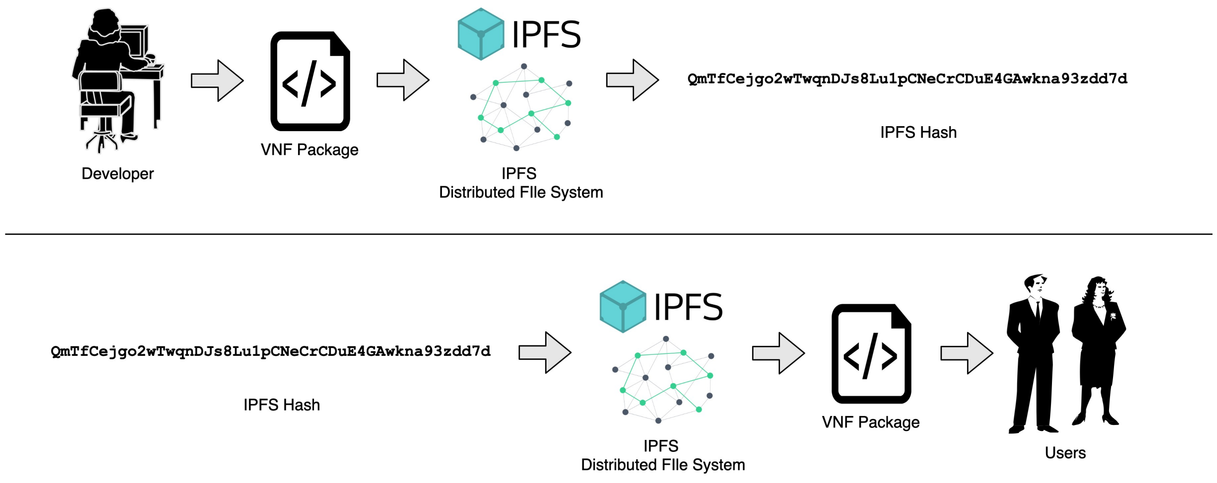


Figure 4.3: IPFS Data Registration (top) and IPFS Data Retrieval (bottom)

4.2 Implementation

To assess the solution feasibility, a PoC that does not include all functionality mentioned in Section 4.1 was implemented. This work focused on the blockchain-based repository back end, described in the following Section 4.2.1. This part was implemented as an SC on the Ethereum network, written in Solidity. This SC includes both the package repository as well as the repository manager and provides links to the external data storage. Afterwards, a minimum Graphical User Interface (GUI) was implemented to provide access to the back-end API's functions. This is detailed in the Subsection 4.2.2.

4.2.1 Blockchain-based Repository Backend

The solution was implemented as a SC on the Ethereum network using the Solidity programming language. This means that the SC's code will be executed in a trusted environment. Refer to Section 2.3 for more details on Ethereum and Solidity. The Ethereum network offers the following advantages over other blockchain networks:

- Established platform: Ethereum has the second highest active addresses after Bitcoin[1]. Thus, there is great developer support from the community and from the Ethereum foundation.
- Developer tools: Development of Ethereum-based Smart Contracts has been simplified by tools such as the Truffle framework, a development environment for Smart Contract and Ganache which emulates an Ethereum network on the local machine [9].
- SC security: As an increasing number of SCs have been created, audits have revealed many common security flaws in the programming code of SCs. Tools, such as Securify [14], have been developed to identify these.

[1]Active addresses in the last 24h: Bitcoin: 677'099; Ethereum: 412'737 as of 2 April 2018 [3]

Package Repository

A VNF package in the repository is defined as a custom type (`struct`) (see Listing 4.1 for the implementation). The first attributes `name`, `description`, `image_link` and `service_type` are used in the marketplace. These allow the vendor to promote the package and describe its functions. Attribute `image_link` is used as a cover art for the package. The `service_type` allows for filtering and categorization of the packages. Attributes `version`, `requirements` and `resources` are additional information available on demand. They allow to check specifications of the package and inform potential buyers of the system requirements and resources needed. The `price` is the package's licensing fee. The `author` represents the vendor's Ethereum address to which licensing fees should be transferred. The `repository_link` and the `repository_hash` are transmitted to the licensee after an acquisition of a package, so that the package can be retrieved and checked for integrity. The `ratings` array consists of all ratings submitted for a package.

These VNF packages have to be stored in the SC storage. To do this, an VNF object array, that represents the package repository, has been included (See line 16 of Listing 4.1).

```
1   struct VNF {
2     string name;
3     string description;
4     string image_link;
5     string service_type;
6     string version;
7     string requirements;
8     string resources;
9     uint price;
10    address payable author;
11    string repository_link;
12    bytes32 repository_hash;
13    int[] ratings;
14  }
15
16  VNF[] VNF_repository;
```

Listing 4.1: VNF Struct and Repository Array in the SC

Repository Manager

The repository manager acts as an intermediary between the repository and the users and offers public functions for accessing the repository data. It is included in the same SC as the package repository and includes functionality used by all four systems of the architecture discussed in Section 4.1 an. Following is an overview of all functions offered and their purpose in the solution. The subsequent paragraphs depict the implementations in more details.

- Data retrieval functions: `retrieve_numberOf_VNF`, `retrieve_VNF`, `retrieve_VNF_requirements`;

- Registration and update system: `register_VNF`, `update_VNF`, `delete_VNF`;
- Licensing system: `buy_VNF`;
- Rating system: `rate_VNF`;
- Verification system: `has_VNF_license`, `get_licensed_VNF`;

The list of VNF packages registered in the repository can be retrieved using the three functions. As a struct array cannot be returned by a function, first, the number of VNF packages has to be retrieved using the function `retrieve_numberOf_VNF` (See Listing 4.2, line 1). This represents the length of the repository and can be used as an iterator to fetch details of each VNF using the two functions `retrieve_VNF` and `retrieve_VNF_requirements` (See Listing 4.2, line 5 and 21 respectively). The details are given through two functions because of a further limitation in Soldidity: If too many local variables (return parameters included) are introduced, a "Stack is too deep" error is returned. To circumvent the problem, only 8 return parameters should be used. These three functions are declared `view`, which means that they cannot alter the state of the SC and do not require any gas to run. Therefore, all repository data can be retrieved without transaction fees.

```
1  function retrieve_numberOf_VNF() public view returns (uint count) {
2    count = VNF_repository.length;
3  }
4
5  function retrieve_VNF(uint index) public view returns
6  (string memory name, string memory description,
7   string memory image_link, string memory service_type,
8   uint price, address author, string memory version,
9   uint[] memory ratings) {
10
11     name = VNF_repository[index].name;
12     description = VNF_repository[index].description;
13     image_link = VNF_repository[index].image_link;
14     service_type = VNF_repository[index].service_type;
15     price = VNF_repository[index].price;
16     author = VNF_repository[index].author;
17     version = VNF_repository[index].version;
18     ratings = VNF_repository[index].ratings;
19 }
20
21 function retrieve_VNF_requirements(uint index) public view returns
22 (string memory requirements, string memory resources) {
23     requirements = VNF_repository[index].requirements;
24     resources = VNF_repository[index].resources;
25 }
```

Listing 4.2: Data Retrieval Functions

The `register_VNF` function allows developers to register new VNF packages to the repository. Listing 4.3 shows the implementation: The function takes VNF package attributes as arguments. As the sender is the author of the VNF package, the sender's address is used as author address. The package details are used to create a VNF package object, which is then pushed to the repository array.

```
function register_VNF(string memory name,
                      string memory description,
                      string memory image_link,
                      string memory service_type,
                      string memory repository_link,
                      uint price, string memory version,
                      string memory requirements,
                      string memory resources,
                      bytes32 repository_hash) public {

  address payable author = msg.sender;
  VNF memory vnf = VNF(name, description, image_link,
                       service_type, repository_link,
                       price, author, version, requirements,
                       resources, repository_hash, new uint[](0));

  VNF_repository.push(vnf);
}
```

Listing 4.3: `register_VNF` Function

To change the package information of a VNF in the repository, the user calls the `update_VNF` function (see Listing 4.4): Instead of creating a new VNF package entry it takes an existing entry using its index and overwrites all attributes. The caller of the function is required to be the author of the package. This means that only the authors are allowed to change information of their packages.

```
function update_VNF(uint index, string memory name,
                    string memory description,
                    string memory image_link,
                    string memory service_type,
                    string memory repository_link,
                    uint price, string memory version,
                    string memory requirements,
                    string memory resources,
                    bytes32 repository_hash) public {

  require(msg.sender == VNF_repository[index].author,
    "Only author can update VNF package");

  VNF_repository[index].name = name;
  VNF_repository[index].description = description;
  VNF_repository[index].image_link = image_link;
  VNF_repository[index].service_type = service_type;
  VNF_repository[index].repository_link = repository_link;
  VNF_repository[index].price = price;
  VNF_repository[index].version = version;
  VNF_repository[index].requirements = requirements;
  VNF_repository[index].resources = resources;
  VNF_repository[index].repository_hash = repository_hash;
}
```

Listing 4.4: `update_VNF` Function

The function `delete_VNF` (see Listing 4.5) receives a VNF package using the index and removes it from the repository array. To do this, it overwrites the index of the package to be removed with the last package of the array. The duplicated package is then removed. Finally, the length of the repository array gets reduced by 1. This object shift represents the cheapest way of deleting an object from an array in Solidity.

```
function delete_VNF(uint index) public {
  VNF_repository[index] = VNF_repository[VNF_repository.length-1];
  delete VNF_repository[VNF_repository.length-1];
  VNF_repository.length--;
}
```

Listing 4.5: `delete_VNF` Function

If a user decides to acquire a VNF, the front end calls the `buy_VNF` (see Listing 4.6). A call to this function contains the index of the package to be acquired as argument. The Smart Contract checks that sufficient funds were included in the transaction. If so, the funds are transferred to the author of the package. Then, a license event is emitted including the link to the package's repository and the repository hash to verify the integrity. This is the data needed by the front end to start to download, verify and deploy the VNF package.

```
function buy_VNF(uint index) public payable {
  require(msg.value == VNF_repository[index].price,
      "price and value do not match");

  address payable receiver = VNF_repository[index].author;
  uint amount = VNF_repository[index].price;
  receiver.transfer(amount);
  VNF_repository[index].licensees[msg.sender] = true;

  emit License(msg.sender, index,
               VNF_repository[index].repository_link,
               VNF_repository[index].repository_hash);
}
```

Listing 4.6: `buy_VNF` Function

Users can rate a package using a scale from 1 to 10 by calling the function `rate_VNF` (See Listing 4.7 for the implementation). First, the function checks if the rating is between 1 and 10. If this requirement is fulfilled, it pushes the rating to a VNF package's rating list.

```
function rate_VNF(uint index, uint rating) public {
  require(rating <= 10, "Rating scale is 1-10");
  require(rating >= 1, "Rating scale is 1-10");
  require(VNF_repository[index].licensees[msg.sender] == true,
    "Rating only possible for licensees");

  VNF_repository[index].ratings.push(rating);
}
```

Listing 4.7: `rate_VNF` Function

To check if a user has already acquired a license for a VNF package, the function `has_VNF_license` (See Listing 4.8) can be used. It it declared as view and as such free of transaction cost. When called, the function checks the `licensees` mapping for the caller's address. If a license was previously acquired, this has been set to true. Otherwise (*i.e.,* when the entry is inexistent because no license was acquired), the return value is false.

```
function has_VNF_license(uint index) public view returns
(bool license) {
   license = VNF_repository[index].licensees[msg.sender];
}
```

Listing 4.8: `has_VNF_license` Function

For the re-verification of a VNF package's integrity, the API offers the function `get_licensed_VNF` (See Listing 4.9). First it checks that the caller has acquired a license for the package. If so, it returns the values previously emitted by the license event: `repository_link`, as well as `repository_hash`. These can be used to retrieve the package data and validate its integrity.

```
function get_licensed_VNF(uint index) public view returns
(string memory repository_link, bytes32 repository_hash) {
   require(VNF_repository[index].licensees[msg.sender] == true,
     "No license acquired for specified VNF");
   repository_link = VNF_repository[index].repository_link;
   repository_hash = VNF_repository[index].repository_hash;
}
```

Listing 4.9: `get_licensed_VNF` Function

4.2.2 Graphical User Interface

This subsection is organized in two sections. The first section explains the technologies and frameworks used to create the GUI. Afterwards, the GUI is presented with all functionality explained in detail.

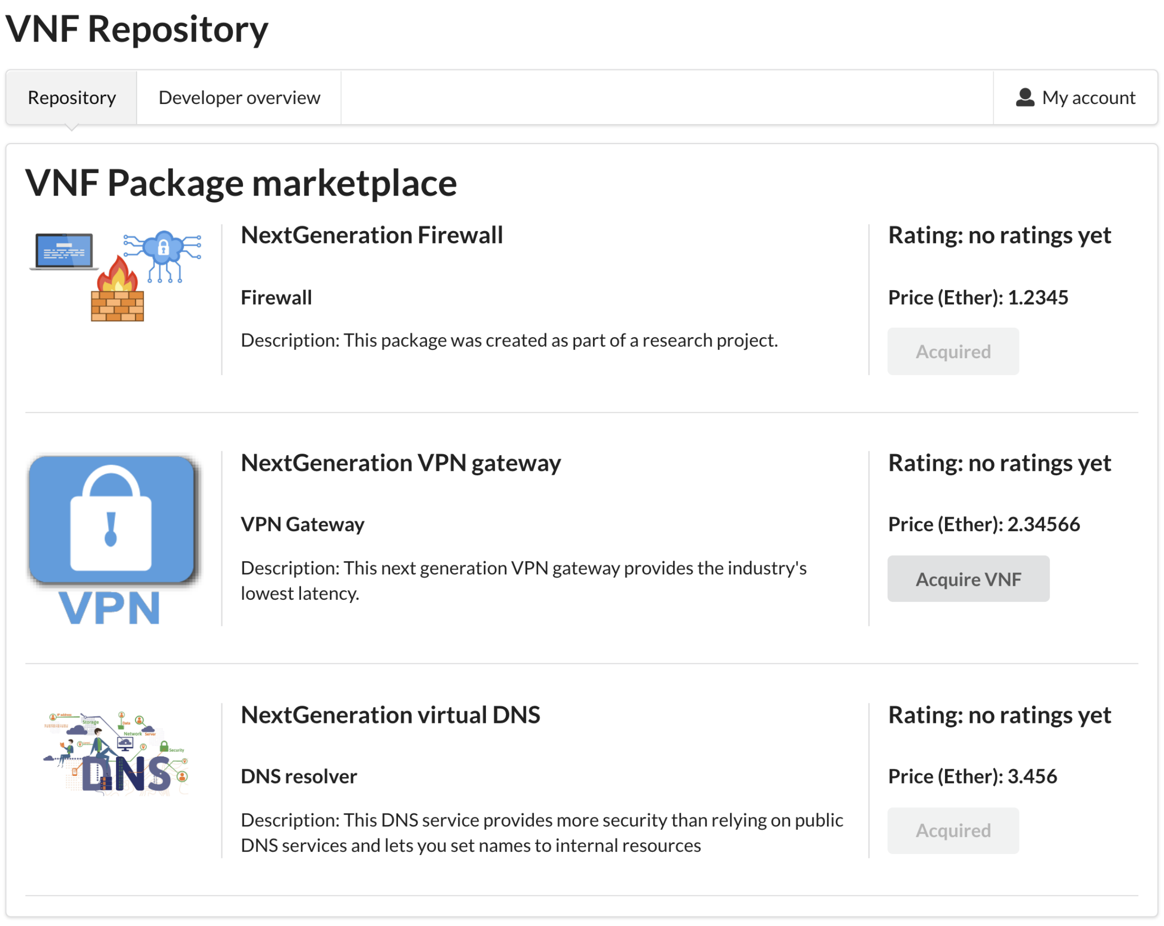
Technology Stack

The user interface was developed using the react framework [24], a JavaScript framework that divides the interface into reusable pieces called components, eliminating the need for code duplication. For this project it is used in combination with JSX, a JavaScript syntax extension for easier readability which, together with react, allows the developer to create interactive user interfaces in a declarative way. Components are split into two categories: *(i)* stateless and *(ii)* stateful. Both can take inputs (called *props*), although only the stateful components maintain an internal state. Furthermore, lifecycle methods are available to allow preparation and cleanup of any component, which is useful for freeing up unused resources or retrieving additional information from an API. To accelerate the implementation of a clean, simple and user-friendly User Interface (UI), the Semantic UI react [28] library of user interface components was integrated.

To allow interaction with the SC, a PoC GUI was implemented as a Distributed App (DApp). It was developed using the Truffle development framework [9]. It aims to simplify the development by providing SC interactions for the application. It takes over compilation, automatically links the SC to the application, and offers simple deployment and migration capabilities. The DApp is using the Web3.js JavaScript library [30] which offers a connection to the Ethereum network and acts as gateway to the smart-contract back end functions. The DApp requires the user to be logged in to the MetaMask browser extension [15] or a web3-compatible browser and has to be granted privileges to access the user's account and addresses. When a function of the SC requires a transaction (thus requires gas or ether to be executed), MetaMask automatically authenticates the user and ask for confirmation. For cryptography functions, the CryptoJS library was used [10].

Functionality

The GUI is divided into three parts: *(i)* The Repository tab which is used by customers seeking to acquire VNF packages or interact on the marketplace. This tab also represents the landing page of the DApp. *(ii)* The Developer tab which is used by developers to register and maintain their packages in the VNF repository. *(iii)* The Account tab, where a user can access licenses and rate packages. Below, details and functionality of these tabs are presented.



NextGeneration Firewall

Firewall

Description: This package was created as part of a research project.

Rating: no ratings yet

Price (Ether): 1.2345

Acquired

NextGeneration VPN gateway

VPN Gateway

Description: This next generation VPN gateway provides the industry's lowest latency.

Rating: no ratings yet

Price (Ether): 2.34566

Acquire VNF

NextGeneration virtual DNS

DNS resolver

Description: This DNS service provides more security than relying on public DNS services and lets you set names to internal resources

Rating: no ratings yet

Price (Ether): 3.456

Acquired

Figure 4.4: VNF Package Repository and Marketplace GUI

Figure 4.4 shows the **Repository tab interface**. It consists of a table presenting all VNFs in the repository. Vertically, it is divided into three columns. The first one shows a cover art image of the package and allows developers to attract a user's attention by showcasing key functionality, icons, interfaces or other. The second column presents the VNF's title and description. Lastly, the third column shows information about the package's price and offers a way of acquiring it.

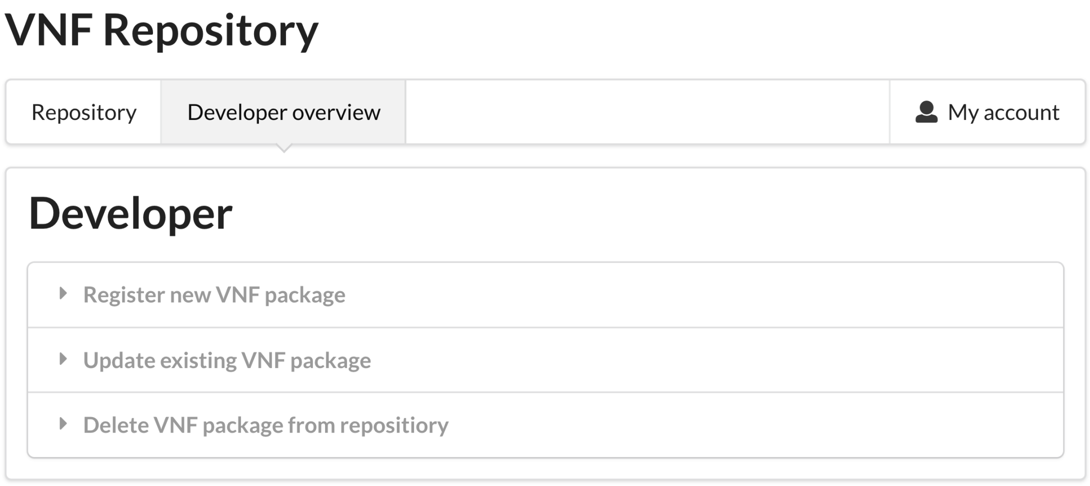


Figure 4.5: Developer Tab GUI

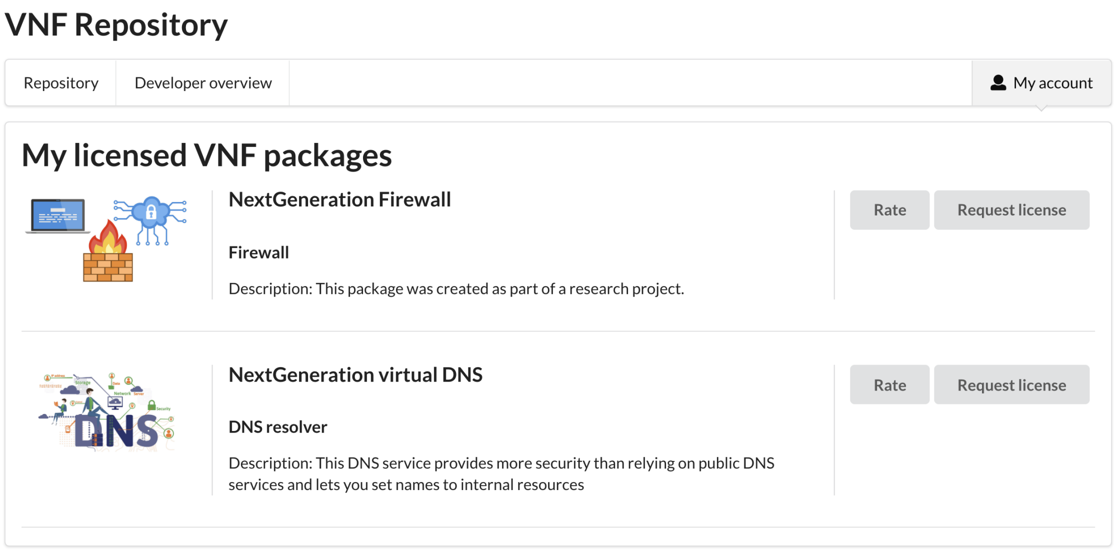
Figure 4.5 shows the **Developer tab interface**. Here, developers of VNF packages can access functions to manage existing packages or registering new ones. The following functions are available:

Registering new VNF packages can be done using the first option of the accordion *Register new VNF package*. It requires the developer to input the following package attributes: name, description, image link, service type, version, requirements, resources, price, and repository link. When this is submitted, the repository hash is calculated by downloading the package from the repository link and creating a SHA256 hash of the file. All attributes are then sent to the back end (refer to function `register_VNF` in Section 4.2.1).

If a package already exists in the repository but is to be updated with new information, developers use the second option of the accordion *Update existing VNF package*. The developer needs to submit the same package attributes as when registering a new package as well as the index of the package. With this information, the back end is called (refer to function `update_VNF` in Section 4.2.1).

If a developer wishes to remove a package from the repository, the last function in the accordion *Delete VNF package from repository* is used. Here, the developer specifies the index of the package to be removed, which is then sent to the back end (refer to function `delete_VNF` in Subsection 4.2.1). Only the author of a package can trigger the removal of a package.

In the **Account tab interface** (refer to Figure 4.6), all acquired packages are listed. The overview shows the same information as the repository tab. However, the third column here is reserved for the two interaction possibilities: *(i)* The *Rate* button opens a dialog in which the user can rate the package on a scale from 1 to 10 (refer to Figure 4.7). *(ii)* The *Request license* button opens a dialog in which the package's repository link and repository hash are presented. This can be used for (re-) verification of the VNF package.



VNF Repository

Repository | Developer overview | My account

My licensed VNF packages

NextGeneration Firewall

Firewall

Description: This package was created as part of a research project.

Rate | Request license

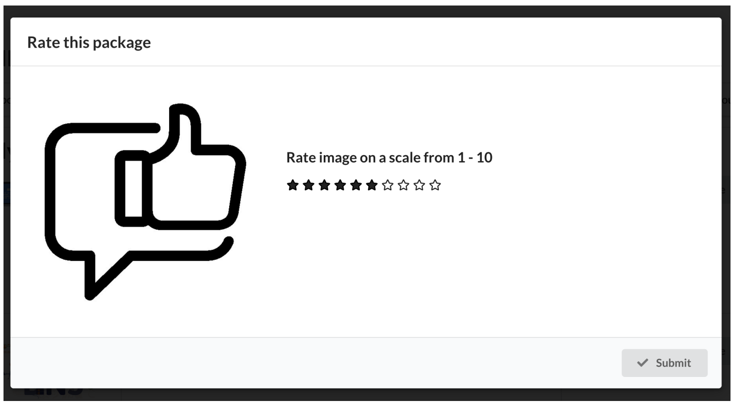
NextGeneration virtual DNS

DNS resolver

Description: This DNS service provides more security than relying on public DNS services and lets you set names to internal resources

Rate | Request license

Figure 4.6: Account Tab GUI



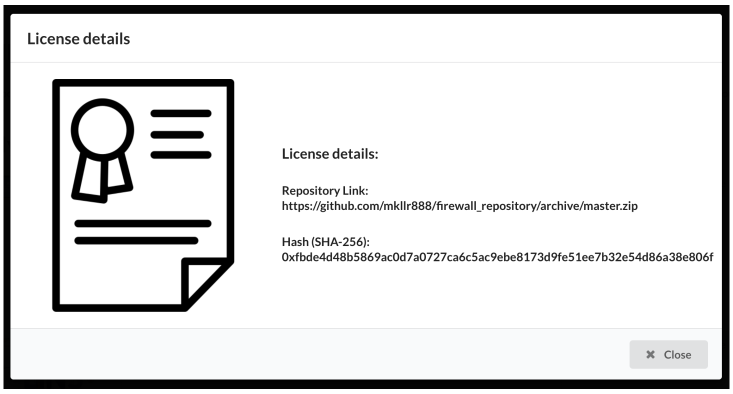
Rate this package

Rate image on a scale from 1 - 10

★★★★★★☆☆☆☆

Submit

Figure 4.7: Rating Dialog


License details

License details:

Repository Link:
https://github.com/mkllr888/firewall_repository/archive/master.zip

Hash (SHA-256):
0xfbde4d48b5869ac0d7a0727ca6c5ac9ebe8173d9fe51ee7b32e54d86a38e806f

Figure 4.8: License Dialog

Chapter 5

Evaluation and Discussion

The trusted blockchain-based VNF repository solution presented in Chapter 4 aims to address a particular gap in the security of NFV environments. In this chapter, the blockchain-based approach to improve the security of the VNF repository is evaluated in terms of management and security, as well as economic aspects. Furthermore, a cost analysis of the SC is conducted. Lastly, in the discussion, the advantages and disadvantages are presented and a feasibility analysis is conducted.

5.1 Management and Security

As described in [23], determining VNF package integrity is a key challenge in the setup of a trusted NFV environment. The blockchain-based trusted VNF repository addresses this challenge successfully without having to rely on an external trusted security orchestrator (TSecO). This mitigates a central point of failure. The proposed solution is based on an SC without access control and management. As such, any interested party can use the functions provided, given that they pay the gas fee needed to update the SC's state. This means that the SC runs fully distributed and without the need for management. As there is no maintenance cost, no contract fees have to be collected to keep the SC running. On the other hand, there is a potential for spam and fake entries that do not deliver on their promised functions or infringe on trademarks and intellectual property.

Adding access control and verification of vendors would increase the trust in the repository's contents. To do so, the SC repository would need to be managed either by a central authority, by a consortium or in a distributed, open manner. This management could then verify authenticity of vendors before any package can be registered. In addition, it could curate the repository's offerings by checking the VNF packages for malicious code and verifying that the functionality complies with the package's specifications. The same management would also be responsible for pushing updates to improve user experience as well as code maintenance. The downside of management and curation is that it introduces new challenges: A centralized management of the smart contract as well as authorization of participants may be unreliable and against the intent of the repository's distributed nature. The alternative, offloading the management to a consortium might not mitigate the problem of malicious participants and still create conflicts of interest. Thus, the current design without access control and an uncurated repository may hold challenges, but it best reflects the solution's core concept.

Security is a highly relevant topic in all areas of computing. Particularly when money is involved, it must be assumed that malicious participants will try to find vulnerabilities for their benefit. The solution's SC does not hold any funds itself, shielding it from direct attacks. However, any code potentially contains bugs, so it may be possible to exploit a vulnerability, which would *e.g.,* allow attackers to redirect licensing payments to other accounts. Before deployment in a production environment it is therefore necessary to inspect all SC code for any potential vulnerability. It is well known that there is and will never be an algorithm that can definitively determine that code is free of vulnerabilities. That is why such a security audit has to be done by hand by a trusted auditor that is experienced in SC security.

The repository offers functionality to verify the integrity of a VNF package. This verification is done in the front end after acquiring and downloading the package. A better way would be to enable verification by the smart contract before acquisition. However, due to the high cost of memory inside the SC, this is infeasible. Yet, a common solution to this problem are oracles, which perform calculations off-chain for a fee. However, this means that the oracle has to be trusted to execute code correctly.

The risk of malicious packages and malicious vendors in the repository is tackled with the use of ratings. These give users an idea of what to expect from the package and can build up trust. However fake reviews and wrong vendor descriptions are a challenge that needs to be tackled.

5.2 Economical Aspect

To see adoption of a blockchain-based trusted repository in production NFV environments, such a solution needs to support multiple business models for licensing. Companies should be able to chose their model (*e.g.,* fixed-price or pay-as-you-go) without restrictions [6]. The current solution only offers a fixed-price business model. Of course, the SC code can be extended to cover additional business models. However, by themselves, SCs cannot offer recurring payments such as a monthly subscription service. This is due to a technical restriction: SCs cannot initiate state changes themselves (*e.g.,* pay licensing fee after a month), but instead can only do computations when called externally. The use of such business models thus needs to be implemented in the VNF package instead of the contract code in the sense of a Digital Rights Management (DRM) mechanism that verifies that the monthly fee was paid.

So far, the rating system is designed in a way as to only allow users to rate acquired packages. This should help to mitigate fake reviews. However, as ratings require a SC state change, the rater has to pay transaction fees (refer to Section 5.3) which might discourage them.

5.3 Cost Analysis

As discussed in Section 2.3, the deployment of an SC to the Ethereum network and interactions with it consume gas that has to be included in the transactions. To analyze the feasibility of deploying and using the functions, the following Table 5.1 depict the cost in ETH as well as fiat currencies. Gas costs of operations on the Ethereum Virtual Machine are specified in the Ethereum yellow paper [13]. Based on those operation cost, the gas cost for the SC's deployment and interactions can be calculated. To estimate the actual fees, two methods were used: First, the Remix project's online IDE [27] was used in addition to a local deployment using the truffle frameworks Ganache application [9], with which an Ethereum blockchain of ten nodes was created. The gas price in the Ethereum network varies depending on network load and required speed of confirmation. For this analysis, the gas price was fixed to 10 Gwei (10000000000 wei) which represents the price used as default in MetaMask[1]. Table 5.1 shows the resulting cost in gas and US Dollar ($) for all SC functions that require gas.

For the creation of a Smart Contract on the Ethereum network (*i.e.,* to deploy the SC), the EVM requires a constant fee of 32000 gas in addition to the standard 21000 gas fee of any transaction. In addition to that, the Ethereum Virtual Machine requires 200 gas per Byte of contract code and 20000 gas per word that has to be stored. All other transactions require the standard 21000 gas fee plus additional gas for every operation performed (*e.g.,* 3 gas for any addition or subtraction) [13].

SC Function	Gas Consumed	Price (USD)
SC Creation	2'926'168	5.16 $
`register_VNF`	372'119	0.64 $
`update_VNF`	369'635	0.64 $
`delete_VNF`	195'108	0.33 $
`buy_VNF`	56'137	0.10 $
`rate_VNF`	48'445	0.09 $

1 ETH = 171.40 USD price as of 18 April 2019

Gas consumed represents avg. of results of Remix IDE and Ganache blockchain

Table 5.1: Gas Estimation and Prices

5.4 Discussion

The proposed design of a blockchain-based trusted repository for VNF packages offers enhanced security over a traditional database repository setup. By leveraging an SC's blockchain technology, the system is able to forgo external TSecO or remote attestation which present a vulnerability in current NFV environments. It allows the creation of an open marketplace without access control, which can help increase competition.

[1]gas price as of April 2019

Section 5.1 presents several disadvantages to such an open solution. The repository is unmanaged and uncurated. Thus, it is open to spam and malicious offerings. In the approach herein presented, ratings are included to lessen the problem. However, these also cannot be fully trusted due to fake reviews. Switching from an open design to a curated repository with verified participants thus may prove beneficial. Yet, this change will decrease the distributed nature of the repository and thus comes at a cost in terms of the advantages mentioned. This challenge should be further researched.

Some disadvantages are connected to the repository's underlying Smart Contract design: As data storage is expensive, packages cannot be stored in it, only their hashes. This also means that the front end is responsible for the process of integrity verification, as the smart contract can only provide hashes. Also, state changes of SCs require transaction fees to be paid. However, these are minor amounts for both vendors and buyers of VNF packages and could be further reduced by researching the most efficient code structures.

In economic aspects, there are a few challenges to explore before a wider adoption could be achieved. NFV has allowed the development of new business models which have to be supported, such as VNF-as-a-Service offerings. In the current form of a PoC, such business models are not yet implemented. However, a possible integration into a capable NFV environment that already supports these models may reduce the implementation cost.

At the moment, the solution is independent of existing blockchain-based approaches to management and orchestration. These approaches, as discussed in Section 3.3, offer advantages by setting up a trusted execution environment but at this stage do not include Blockchain-based VNF package integrity verification. Integrating this work's solution would increase the security and offer new advantages as mentioned in this chapter.

Chapter 6

Conclusion

The work of this thesis aimed to research the use of blockchain technology in the area of NFV and VNF repositories. The first part of the thesis provides background and related works. This is to present an overview over the topics and to classify existing works to determine the current state of research. The second part depicted the design and implementation of a blockchain-based repository for VNF packages.

In the current research environment, there have already been attempts to create blockchain-based NFV management and orchestration solutions with prototype solutions available. Even though these included approaches to create a trusted computing environment, they did not leverage the blockchain's advantages in the area of the VNF repository and instead used a traditional database. As a result, the solution designed in the second part of the thesis focuses on the aspect of the VNF repository and how blockchain can enhance security and functionality. Based on the architecture's design, a PoC was implemented. It employs a smart-contract back end to store the repository's information in the blockchain and offers an API to access the VNF package repository. Users can register new VNFs, manage or delete existing ones, retrieve the contents of the repository, as well as acquire (*i.e.,* license) packages and give ratings. For all these functions, the implementation details and design decisions were stated. In addition, a GUI front end was developed to make the repository accessible.

The result were evaluated against the requirements. The blockchain-based trusted VNF package repository system succeeds in giving users the possibility to verify package integrity without relying on external trusted operators. The solution also achieves the requirements that developers have in registering and maintaining packages as well as user requirements for acquisition of packages, integrity verification and post-acquisition tools, such as ratings and requests for renewed verification.

The evaluation shows the security advantages of the solution. However, it also showcases the limitations of the access-control free design. The repository may be insufficiently protected against spam entries and other methods of malicious participants. The underlying smart contract technology of the solution also has its weaknesses: As storage is expensive, only VNF package metadata can be kept inside the repository. The package itself is stored in an untrusted, external data storage and thus has to be checked before deployment. Also, at the moment the integrity verification is done after acquisition of a VNF package.

In short, the proposed architecture and its PoC implementation show that a blockchain-based trusted VNF package repository is feasible and offers security advantages over traditional methods. However, there are still challenges connected to the concept which need to be addressed before it is ready for deployment to production environments. In order to leverage the advantages of Blockchain technology in the area of NFV, a promising step forward would be to integrate the concept into an existing Blockchain-based MANO framework.

6.1 Future Work

As this work only presents a prototype implementation of the proposed architecture, the next step would be to finish the concept and further develop the functionality. To reduce the implementation work needed, the best step forward is to integrate the solution into an existing, blockchain-based NFV MANO framework as presented in Section 3.3. Alternatively, it could also be incorporated into the FENDE project (refer to Section 3.1). This way, work on the GUI could be omitted and a second feasibility analysis as part of a production-ready environment could be conducted.

At the moment, the VNF package data is stored in an untrusted, external data storage. To realize a fully blockchain-based repository, a distributed file storage like IPFS [2] could be introduced. IPFS would in practice guarantee a package's integrity before the acquisition. Also, the smart contract could check availability of the data before the licensing fees are transferred to prevent fraud [2].

The rating system is designed to only allow licensees to rate a package. However, the current form is very basic and allows only for a rating of 0 to 10. This should be improved to reflect the complexity of VNF packages. Also, users might not have an incentive to submit a rating at all, as they have to pay a transaction fee, even though it's a minor amount. Lastly, vendors might use fake reviews to boost their sales as the only cost associated with such behaviour is paying the transaction fee for acquiring a package and submitting a rating. Thus, the rating system's incentives should be analyzed and if necessary redesigned to only allow legitimate ratings.

Bibliography

[1] M. Alharby and A. van Moorsel. Blockchain-based Smart Contracts: A Systematic Mapping Study. *CoRR*, abs/1710.06372, 2017.

[2] J. Benet. IPFS - content addressed, versioned, P2P file system. *CoRR*, abs/1407.3561, 2014.

[3] BitInfoCharts. BitInfoCharts, 2019. Available at `https://bitinfocharts.com/` Accessed 2 April, 2019.

[4] T. Bocek and B. Stiller. Smart Contracts – Blockchains in the Wings. In *Digital Marketplaces Unleashed*, pages 169–184. Springer, 2018.

[5] L. Bondan, M. Franco, A. E. Schaeffer-Filho, L. Granville, L. Marcuzzo, C. A. D. S. Schneider, C. R. Paula dos Santos, G. Venâncio, and E. Duarte Jr. FENDE: Marketplace and Federated Ecosystem for the Distribution and Execution of VNFs. pages 135–137, 08 2018.

[6] L. Bondan, M. F. Franco, L. Marcuzzo, G. Venancio, R. L. Santos, R. J. Pfitscher, E. J. Scheid, B. Stiller, F. De Turck, E. P. Duarte, A. E. Schaeffer-Filho, C. R. P. d. Santos, and L. Z. Granville. Fende: Marketplace-based distribution, execution, and life cycle management of vnfs. *IEEE Communications Magazine*, 57(1):13–19, January 2019.

[7] N. Bozic, G. Pujolle, and S. Secci. Securing Virtual Mchine Orchestration with Blockchains. In *2017 1st Cyber Security in Networking Conference (CSNet)*, pages 1–8, Oct 2017.

[8] V. Buterin. Ethereum White Paper. Available at `https://github.com/ethereum/wiki/wiki/White-Paper` Accessed 23 April, 2019.

[9] ConsenSys Inc. Truffle Suite - Sweet Tools for Smart Contracts, 2019. Available at `https://securify.chainsecurity.com/` Accessed 2 April, 2019.

[10] Crypto-js. brix/crypto-js - JavaScript library of crypto standards, 2019. Available at `https://github.com/brix/crypto-js` Accessed 11 April, 2019.

[11] Duo Inc. DApp Statistiken. Available at `https://www.stateofthedapps.com/de/stats` Accessed 28 April, 2019.

[12] Ethereum Foundation. Solidity - Solidity 0.58.0 Documentation. Available at https://solidity.readthedocs.io/ Accessed 28 April, 2019.

[13] e. a. Gavin Wood, Nick Savers. Ethereum Yellow Paper, 2019. Available at https://github.com/ethereum/yellowpaper Accessed 12 April, 2019.

[14] ICE center, ETH Zurich, and ChainSecurity AG. Securify - Security Scanner for Ethereum Smart Contracts, 2019. Available at https://securify.chainsecurity.com/ Accessed 2 April, 2019.

[15] MetaMask. MetaMask - Brings Ethereum to your browser, 2019. Available at https://metamask.io Accessed 10 April, 2019.

[16] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262, Firstquarter 2016.

[17] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009. Available at https://bitcoin.org/bitcoin.pdf Accessed 22 March, 2019.

[18] NCC Group. Decentralized Application Security Project. Available at https://dasp.co/ Accessed 23 April, 2019.

[19] Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG). ETSI GS NFV-MAN 001 - V1.1.1 - Network Functions Virtualisation (NFV); Management and Orchestration, 2014. Available at https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf Accessed 1 April, 2019.

[20] Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG). ETSI GS NFV-SEC 003 - V1.1.1 - Network Functions Virtualisation (NFV); NFV Security; Security and Trust Guidance, 2014. Available at https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf Accessed 1 April, 2019.

[21] O. Open. TOSCA Simple Profile for Network Functions Virtualization (NFV) Version 1.0, 2017. Available at http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/csd04/tosca-nfv-v1.0-csd04.html#_Toc482896036 Accessed 25 March, 2019.

[22] OpenStack. OpenStack Docs: VNF Descriptor Template Guide, 2019. Available at https://docs.openstack.org/tacker/latest/contributor/vnfd_template_description.html Accessed 15 April, 2019.

[23] S. Ravidas, S. Lal, I. Oliver, and L. Hippelainen. Incorporating trust in NFV: Addressing the challenges. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 87–91, March 2017.

[24] ReactJS. React - A JavaScript library for building user interfaces, 2019. Available at https://reactjs.org/ Accessed 11 April, 2019.

[25] G. Rebello. sinfonia - GitHub, 2017. Available at https://github.com/gfrebello/sinfonia/wiki Accessed 1 April, 2019.

[26] G. A. F. Rebello, I. D. Alvarenga, and G. de Teleinformatica e Automacao. SINFONIA: Gerenciamento Seguro de Funcoes Virtualizadas de Rede atraves de Corrente de Blocos. In *Anais do I Workshop em Blockchain: Teoria, Tecnologias e Aplicacoes (WBlockchain - SBRC 2018)*, volume 1, Porto Alegre, RS, Brasil, 2018. SBC.

[27] Remix project. Remix - Solidity IDE, 2019. Available at `https://remix.ethereum.org/` Accessed 12 April, 2019.

[28] Semantic UI. Semantic UI React, 2019. Available at `https://react.semantic-ui.com/` Accessed 30 April, 2019.

[29] The Linux Foundation. Hyperledger, 2019. Available at `https://www.hyperledger.org/` Accessed 1 April, 2019.

[30] Web3.js. web3.js - Ethereum JavaScript API, 2019. Available at `https://github.com/ethereum/web3.js` Accessed 2 April, 2019.

[31] G. Xilouris, E. Trouva, F. Lobillo, J. M. Soares, J. Carapinha, M. J. McGrath, G. Gardikis, P. Paglierani, E. Pallis, L. Zuccaro, Y. Rebahi, and A. Kourtis. T-NOVA: A Marketplace for Virtualized Network Functions. In *2014 European Conference on Networks and Communications (EuCNC)*, pages 1–5, June 2014.

Abbreviations

API	Application Programming Interface
DDoS	Distributed Denial of Service
ETSI	European Telecommunications Standards Institute
ETH	Ether (Currency of Ethereum network)
EVM	Ethereum Virtual Machine
GUI	Graphical User Interface
MANO	Management and Orchestration
NFV	Network Function Virtualization
P2P	Peer-to-peer
PoC	Proof-of-Concept
SC	Smart Contract
SP	Service Provider
TPP	Trusted Third Party
VNF	Virtualized Network Function
VNFD	Virtualized Network Function Descriptor
VDU	Virtual Deployment Unit

Glossary

Access Control Restricting access to certain areas (such as access to specific functions in an application) to a defined group of authorized entities.

Authentication The act of identifying an entity and validating its authenticity.

Authorization Authorization is the decision whether an entity is allowed to perform a particular action or not, *e.g.,* whether a user is allowed to attach to a network or not.

Attestation Confirmation / verification of something such as the state of the system.

Blockchain A shared record of transaction in a distributed ledger system.

Environment (Computing) Entire set of conditions under which programs are run.

Integrity Integrity in terms of data integrity is achieved when data is complete, unaltered from the original.

Hash Output of a one-way function.

License Permission to use something, *e.g.,* in terms of software licensing: The right to access, use software.

Management & Orchestration (NFV) Administration, configuration and surveillance of components such as virtualized services, virtual machines. It includes resource management and life-cycle operations.

Peer-to-peer network A distributed network where participants can query other participants for objects, mostly used for file sharing.

Repository A storage location for files, especially software packages.

List of Figures

List of Tables

Listings

Appendix A

Installation Guidelines

Browser Setup

To access the DApp, the MetaMask [15] browser extension needs to be installed. It can be retrieved from the following URL and is compatible with all common browsers:

`https://metamask.io/`

In MetaMask, the user should be logged in to an account that is in possession of at least 100’000 WEI for the transaction fees in addition to the currency needed for acquiring the VNF packages.

For testing, the use of Ganache [9] is recommended. This application sets up a test environment of the Ethereum network and offers a quick entry into an account with 100 ETH ready to use. The Ganache application can be retrieved from the following URL and is available for all major operating systems:

`https://truffleframework.com/ganache`

When Ganache is launched, MetaMask can access the accounts with the URL and mnemonic provided in the Ganache application.

Smart Contract Setup and Deployment

The Smart Contract was developed with the truffle framework. For this framework to run, the following is needed:

Prerequisites

- Terminal with administrator access

- yarn package manager
- npm package manager

Setup Instructions

First, the application source code has to be copied onto the local disc. Alternatively, it can be retrieved from GitHub with the following link:

`https://github.com/mkllr888/trusted-VNF-repository`

Via terminal, the following command should be executed in the folder `trusted-VNF-repository` to install all necessary dependencies for the front end:

```
$ yarn install
$ cd client
$ yarn install
```

The second step is to configure the deployment of the smart contract. For this, the truffle framework is needed and can be installed using:

```
$ npm install -g truffle
```

Now the desired network to which the smart contract should be deployed is specified in the file `trusted-VNF-repository/truffle-config.js`. For Ganache in the standard configuration, the Ganache appplication needs to be launched and this URL should point to `http://127.0.0.1:7545` (Check Ganache application)

Deployment of Smart Contract

To deploy the SC, the following commands should be executed.

```
$ # compile the smart contract before deployment
$ truffle compile
$ # deploy to network
$ truffle deploy
```

If successful, the GUI can no be launched to access the DApp.

Launching the GUI Client

The GUI is located in the folder `trusted-VNF-repository/client`. As it is linked to the SC, it is only possible to launch the GUI correctly after the instructions in section A have been followed. To launch the GUI, the following command should be executed in that folder:

```
$ # start GUI
$ yarn run start
```

Usage of GUI

After launching, the GUI should be accessible in the localhost on port 3000:

```
http://localhost:3000
```

When the URL is called, MetaMask will ask for permission to access the accounts. The application is now ready to be used.

Appendix B

Contents of the CD

- Thesis PDF
- Trusted VNF Package Repository Smart Contract Source Code
- GUI Source Code
- LaTeX Source Code
- Intermediate Presentation