Securing Configuration Management and Migration of Virtual Network Functions Using Blockchain

Igor D. Alvarenga*, Gabriel A. F. Rebello* and Otto Carlos M. B. Duarte*

*Universidade Federal do Rio de Janeiro – GTA/COPPE/UFRJ – Brazil

Emails: {alvarenga,gabriel,otto}@gta.ufrj.br

***Abstract*—The integration of network function virtualization (NFV) and service function chaining (SFC) adds intelligence to the core of the network. The programmability of the network core, however, raises new vulnerabilities and increases the number of victims, since a simple modification in the core can affect multiple network users. Thus, the provision of secure virtual network service functions (VNFs) is mandatory to guarantee a correct chaining of network functions. This paper proposes a blockchain-based architecture for secure management, configuration and migration of VNFs, which ensures: (i) immutability, non-repudiation, and auditability of the configuration update history; (ii) integrity and consistency of stored information; and (iii) the anonymity of VNFs, tenants, and configuration information. Furthermore, the proposed architecture guarantees the secure update and migration of configurations at the core of the network. A prototype of the proposed architecture using the Open Platform for NFV (OPNFV) indicates parameter trade-offs and performance bottlenecks.**

***Index Terms*—Network Function Virtualization, Blockchain, Configuration Management, Management and Orchestration.**

I. Introduction

Network function virtualization (NFV) and service function chaining (SFC) appear as alternative software-based technologies to enable commercial off-the-shelf hardware to perform functions previously delegated to proprietary hardware-specialized middleboxes [1]. The software-based approach reduces operational expenditure (OPEX) and capital expenditure (CAPEX) enabling multiple infrastructure providers, such as Internet Service Providers, hereafter called multitenant, that offers end-to-end communication services, and multiple virtual network function vendors, hereafter called multivendor, that offers specific virtual network functions. In an NFV scenario, a user requests, to a network provider tenant, an end-to-end network service with a specific quality of service. Then, this tenant establishes a customized chain of virtual network functions, selecting the appropriate VNF from several multivendor implementations. These VNFs reside in cloud datacenters near the network core.

The use of NFV and SFC technologies exposes the network core to an increased number of vulnerabilities and, therefore, it is fundamental to reduce the possible VNF attack vectors and to provide secure and reliable configuration management [2]. It is worth to note that a threat in the network core affects a great number of traffic flows, which corresponds to much more victims [3], [4]. Therefore, in this scenario, a compromised virtual network function such as a firewall or an intrusion detection system at the network core endangers all traffic forwarded through this VNF [5]. To identify a faulty or compromised VNF configuration, auditability is mandatory. Auditability requires non-repudiation and immutability of previous configuration history. We argue that a blockchain-based configuration repository solves the immutability requirement. The use of blockchain in this context intrinsically provides immutability and traceability of the configuration update history, which will be registered in the form of transactions. Furthermore, there is no simple solution for the immutability requirement using conventional databases [6].

In this paper, we propose a blockchain-based architecture for the secure configuration management of virtualized network functions (VNFs). Besides the immutability and the traceability features provided by blockchain, integrity and consistency of transactions are ensured by a consensus protocol, which validates every transaction before registering it and keeps consistency intact even under byzantine faults. Finally, the adoption of an asymmetric encryption key identification scheme confers anonymity to VNFs and tenants, while allowing sensitive configuration encryption when desired.

The proposed architecture enables secure VNF configuration state migration, defining a trust mechanism between different infrastructure providers (tenants) and VNF vendors, which distrust each other. Furthermore, our architecture intends to cope with compromised systems, which could act maliciously. The proposed VNF configuration management and update mechanism is compliant with the good practices found in the literature [7], [8]. In addition, the proposed architecture was designed to avoid any changes to the NFV and SFC orchestration platforms and it is not restricted to a predicted subset of VNFs. The architecture accounts for signed configuration templates to be shared by VNF vendors as a configuration starting point. Sensitive tenant configuration information is still kept private using encryption. Our proposal eliminates the need for listening services in a VNF, thus allowing the definition of policies which automatically enforce closure of vulnerable ports and reduce VNF exposure to possible attack vectors.

The remainder of this paper is organized as follows. Section II presents architecture considerations and the target

978-1-5386-3416-5/18/$31.00 © 2018 IEEE

attacker model. Section III presents the proposed architecture. Section IV presents the implemented prototype evaluation results of the proposed architecture. Section V discusses the state of the art and related work. Finally, Section VI concludes the paper.

II. Architectural Considerations

A. Assumptions and Requirements

We propose an architecture that focuses on secure configuration of virtual network functions (VNF). We assume that the cloud datacenter provider, in which the virtual network functions reside, is not malicious and that the infrastructure provider, tenant of the cloud datacenter provider, controls and manages the computational structure.

In order to mitigate threats to the VNFs, all VNF listening services are disabled, requiring a proactive VNF configuration model, i.e. the VNF initiates all requests for configuration update. This feature eliminates open ports, avoiding connection from attackers to a VNF. Furthermore, VNF access terminals must be used as a simple information display. Therefore, there is no possibility of direct configuration input and all configuration updates must be retrieved from a trusted configuration repository. This secure repository must accommodate both configuration update requests and configuration templates. Each VNF configuration update request is required to be: (i) confidential, as configuration exposure can reveal vulnerabilities; (ii) anonymous, as VNF or tenant identity exposure can facilitate targeted attacks; (iii) authenticated, as to enable verification of sender configuration permissions; and (iv) traceable, as there is the need keep record of configuration modification history for enabling auditability. Configuration templates do not need to be confidential or anonymous, but when they are not, they must be: (v) accountable, to enable identity verification of a public configuration template proponent; and (vi) permanent, as old configuration data can be used in older systems and also during audits. Cryptographic signature and cypher schemes easily realize requirements (i), (ii), and (iii). On the other hand, requirements (iv), (v), and (vi) imply the immutability of past configuration history, which is not achievable for common distributed databases or versioning systems without trust in third party [6], [9].

We propose a blockchain-based configuration repository as a solution for the immutability property. Furthermore, we propose a transaction model that employs asymmetric keys to provide anonymous authentication [10] of tenants and VNFs, as well as confidentiality of configuration data through encryption. When combined to a blockchain data structure, this signed transaction context intrinsically provides traceability and accountability of the configuration template and update history.

Our blockchain is a linked list of blocks that acts as a log of ordered entries, best known as transactions. The result of a hash function uniquely identifies a block and guarantees integrity of its contents, headers, and preceding block hash. The blockchain data structure achieves immutability of past stored block order and block contents due to the use of cryptographic hash linking. Thus, our blockchain-based proposal guarantees the non-repudiation property by combining the immutability property with signed transactions, which ensures traceability and accountability requirements. To guarantee availability we replicate the blockchain. Moreover, to provide resiliency against faulty systems and collusion attacks, we use a consensus protocol, which validates every transaction before storing it in a block of the blockchain, and keeps consistency intact, even under byzantine faults. The adopted consensus algorithm defines criteria on which node closes a block at each round and how transactions are distributed and validated. Public blockchains, such as the one for Bitcoin currency, usually employ proof-of-work (PoW) based approaches [10], which rely on verifiable expenditure of computational power to solve a cryptographic challenge. This approach is ideal for public networks with an unknown number of participants and complete lack of trust. PoW based approaches, however, imply too much computational power and may take a long time to finally register a transaction [11], [12]. As not all blockchain applications rely on currency, rewarding participants is not always possible. In addition, a number of blockchain applications requires strict transaction registering latency and, then, a different consensus mechanism is needed. Private and federated blockchains (i.e. blockchains owned by a single party or multiple collaborating parties) may relax trustlessness requirements by assuming its set of participants is known [9].

Our architecture assumes and implements a federated consensus model where consensus participants are known and their asymmetric key pair is certified by a certificate issuing party, previously agreed upon by the federation members. The literature on the aforementioned topic has shown that byzantine fault tolerant (BFT) protocols enable low-latency consensus for applications up to a few hundred validating participants [13], [14] and accept erratic or malicious behavior from up to a third of the consensus participants. PBFT consensus protocol uses a list of agreed upon neighbor addresses and public keys, which is provided by certificate issuing party. The cost of neighbor list modification depends on federation governance rules. The implemented consensus protocol is inspired by the PBFT (Practical Byzantine Fault Tolerance) protocol and is modified for the blockchain consensus case, which needs to be robust against collusion attacks performed by up to a third of consensus participants. In addition, the PBFT protocol leverages the consensus participants key pair to sign closed blocks in a way that is easily verifiable by client participants. The PBFT default operation case is depicted in Figure 1. For the blockchain use case, all transactions sets received by all participants during a given interval are forwarded to the current elected leader, which closes, signs and proposes a new block. There is no reply phase in blockchain consensus.

B. Attacker model

This architecture considers a Dolev *et al.* attacker model, that is, the attacker is able to `read`, `send` and `discard` a

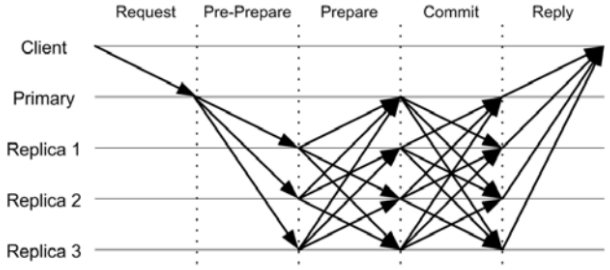


Fig. 1: Sequence of messages/phases, from left to right, for the default case of the PBFT consensus protocol: i) (request) upon a new transaction set sent by a client, ii) (pre-prepare) the primary participant informs all replicas of the new transaction set; ii) (prepare) each participant validates the transactions locally and inform their decision to all other participants using signed messages; iii) (commit) when more than two thirds of participants agree to the new transaction, they send a commit message including all received signatures to all participants. iv) (reply) when more than two thirds of all signed messages are verified, the transaction set is appended to the consensual transaction database.

transaction addressed to the blockchain, or any packet of the network [15]. The attacker can act passively, by connecting to the network and capturing all message exchanges, or actively, by injecting, repeating, filtering or exchanging information. Tenants, VNFs, the blockchain itself and the network can be attacked.

Blockchain attacks are an attempt to prevent a legitimate transaction or block from being embedded in the blockchain. In order for a blockchain attack to succeed, the attacker must control a significant portion of the network to affect the consensus algorithm. This type of attack is mitigated by the adopted consensus mechanism. Attacks that try to modify or corrupt a transaction are not possible because every transaction is accompanied by a correspondent signed hash.

Attacks on tenants or VNFs consist on attempting to obtain either configuration information or personification of the target. Personification attacks are not possible because every transaction sent to the blockchain is signed by its issuer. Attacks that seek to obtain configuration information are mitigated by sensitive information encryption, where the attacker needs to obtain the private key of the targeted recipient. This work does not address the case where a tenant or VNF was compromised through terminal invasion or key hijacking. However, the proposed architecture provides for the absence of any active listening service in a VNF and for the use of a terminal in read-only mode, hence mitigating attack vectors [16]. In addition, the proposed architecture permits auditing of all past transactions at will. Therefore, if an attacker tries to modify the blockchain using stolen key pairs, the attempt will be logged. Upon discovery of an incident, the stolen key pairs can be easily replaced by the tenant, reestablishing security and preventing further damage.

Network attacks represent the attempt to isolate a single tenant, a group of tenants or a group of VNFs from the network, thus preventing the network from performing transactions or reading content from the blockchain. This attack category contemplates classic network attacks which can be mitigated by establishing redundant paths between the distributed blockchain and VNFs or tenants. The proposed architecture assumes all participants are interconnected by a redundant public network, e.g. the Internet. The aforementioned assumption hardens single entity targeting if the attacker is not in its adjacent network. Complete mitigation of network attacks is outside the scope of this work. In the proposed architecture, we focus on blockchain attacks and anticipated transactions. However, by eliminating listening services in VNFs, our architecture eradicates application-layer denial-of-service attacks, which are a common threat in shared cloud environments [5].

III. Proposed System Architecture

A. Proposed Architecture Modules

The proposed architecture is composed of three main modules, as illustrated in Figure 3: i) the blockchain module; ii) the VNF client module; and iii) the tenant client module. The blockchain module is executed by the telecommunication provider at its data centers, in a number of instances sufficient to meet the requests of VNF client modules. A VNF client module runs on each VNF that adopts our configuration management architecture. The tenant client module is run by VNF owners, which are data center tenants using SFC services provided by VNF vendors, and telecommunication providers that wish to offer predefined configuration sets.

The blockchain module, illustrated in Figure 2, is the component responsible for: i) hosting the blockchain; ii) receiving, validating and propagating transactions of client modules; and iii) responding to client module requests. This module connects to other modules of the same type in order to perform these tasks while maintaining consensus on the content of the blockchain, which is replicated at each blockchain module. Prior to the initialization of this module, a list of addresses of neighbor blockchain modules is provided. During module initialization, it requests the latest blockchain version from its neighbors, then updates its local copy with any pending approved blocks. Upon completion , a local relational database

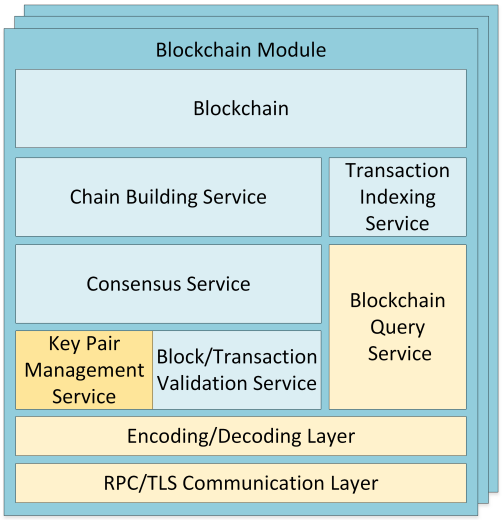


Fig. 2: Blockchain module architecture. The blockchain module hosts a consistent replica of the blockchain and is responsible for reaching consensus with other blockchain modules, as well as answering client requests.

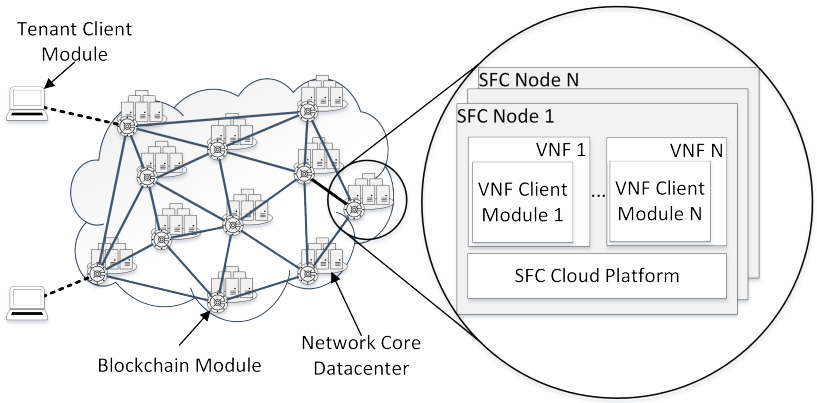


Fig. 3: Blockchain modules are located in network core datacenters and interconnected in a way that allows connection from any other blockchain modules. Both tenant and VNF client modules are able to connect to one or more blockchain modules. No VNF created in the proposed architecture accepts external connections and its configuration state is manage solely by a VNF client module that requests configuration stored in the blockchain by an allowed tenant.

containing indexes is constructed to facilitate the search for content in the blockchain. The indexes allow prompt location of information related to configuration senders and recipients.

Each blockchain module has to verify three conditions for each transaction received: i) whether the transaction format corresponds to the identified type, ii) whether the transaction signature is correct and iii) if there is no transaction duplication. If any of these checks fail, the transaction is discarded. Once the conditions are cleared, the transaction is sent to the neighbor blockchain modules through the consensus mechanism. Each time a round of the consensus algorithm is completed, the new block is appended to the local blockchains and the information relating to the transactions contained in the block is inserted into the local relational database. Once fully initialized, each blockchain module responds to requests for information from a client module based on its local copy of the blockchain, taking into account only consensus-validated information.

The VNF client module, shown in Figure 4, is the component responsible for: i) sending transactions originating from the VNF to a blockchain module; ii) carrying out periodic VNF configuration requests for the VNF that hosts

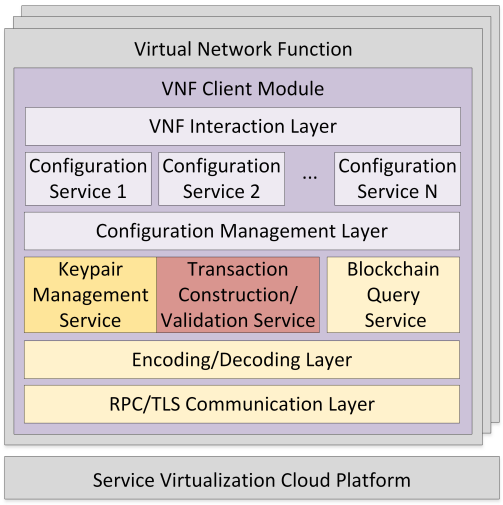


Fig. 4: VNF client module architecture. The VNF client module connects to a blockchain module to send and request transactions regarding the configuration state management of the VNF in which it is installed.

the module; iii) applying the received configuration to the VNF; and iv) managing local VNF key pairs and logical VNF group key pairs. This module connects to one or more blockchain modules as specified in their configuration. Prior to the initialization of the module, the following variables must be configured through the cloud orchestration platform: i) tenant public keys; ii) addresses of blockchain modules; and iii) configuration update time. During the initialization of this module, a pair of asymmetric keys is generated for the VNF. The generated public key is informed through the VNF terminal. This key can be verified through the cloud orchestration platform VNF management console used by the telecommunications provider, and its retrieval can be easily automated by the tenant. Afterwards, the VNF client module will issue a request to a registered blockchain module for the latest configuration assigned to the local VNF. The request is performed after each interval as specified in the module configuration. Three types of responses are possible: configuration not found, a configuration transaction, and a configuration request transaction. A configuration transaction may be accompanied by other referenced configuration transactions.

Upon the retrieval of a set of transactions, the VNF client module will perform the validation of each one. This validation consists on: i) checking whether the received transactions are addressed to a VNF's public key or a VNF logical group belonging to the local VNF; ii) verifying that the sender of the base transaction corresponds to a tenant public key registered in the local VNF client module; iii) verifying that all signatures relating to the information received correspond to the respective sender; iv) verifying that the nonce field of the transaction is different from the last transaction received; and, if the transaction type is of configuration response, v) verifying that the identification of each configuration related to the received transactions corresponds to the one that is listed in the main configuration transaction. If a configuration transaction was received as a response and it was validated correctly, the configuration is applied; otherwise, it is discarded. If a configuration request transaction is received as a response and it has been validated correctly, the VNF client module sends the current configuration state to a blockchain module through

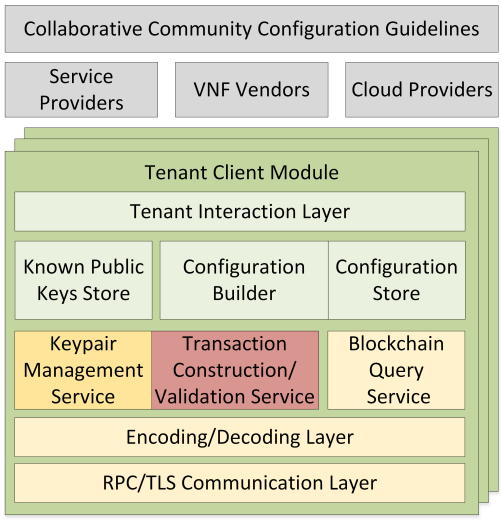


Fig. 5: Tenant client module architecture. The tenant client module is the interface for configuration authors. It allows the tenant who owns the VNFs to assign them confidential configuration information, and also interested parties to publish public configuration templates.

a configuration transaction. The use of the VNF client module requires modifications to the VNF software, so that it is able to install and read configurations and also read pertinent states of the VNF components.

The tenant client module, illustrated in Figure 5, is the component responsible for: i) sending transactions to the blockchain module; and ii) managing tenant key pairs. This module can be run in any station, concomitantly or not with a blockchain module. If the tenant does not have a local blockchain module, the tenant client module must be configured with one or more addresses of blockchain modules. No transaction is ever addressed to the tenant client module, but the tenant client module may request information regarding the blockchain to a known blockchain module.

It was an architectural option to not store the blockchain in each VNF, as this would result in high disk space requirements and setup delays for VNFs. This decision does not imply a significant decrease in security, since messages transmitted between client modules and blockchain modules contain signed and possibly encrypted transactions. A possible attack resulting from this choice is a denial of service of configuration information to a VNF by a malicious blockchain module, negating responses to requests from specific client modules. The configuration of multiple blockchain module addresses in the VNF module is recommended as a way to mitigate this attack.

B. Key Management

The architecture uses four logical groups of asymmetric key pairs: i) VNF key pairs; ii) logical VNF groups key pairs; iii) tenant key pairs; and iv) blockchain module key pairs.

A VNF key pair aims at uniquely identifying a VNF as the sender or recipient of transactions and at signing transactions emitted by a VNF client module. This key pair is generated by the VNF client module during its initialization, then the public key counterpart is displayed on the VNF terminal.

A logical VNF group key pair is intended for simultaneous addressing of VNFs by a single transaction, but are not used for signing. A VNF can belong to multiple logical groups, hence storing multiple VNF group key pairs. These key pairs are received as part of an encrypted configuration transaction, to avoid any chance of private keys being leaked as virtual machine configuration data. Although the functionality performed by VNF logical group key pairs can be implemented using a symmetric group key, we chose to use asymmetric key to maintain the uniformity of the transaction recipient field as a public key. This decision makes it possible to maintain the anonymity of VNFs, as an uninformed observer can not distinguish between a transaction intended for a logical VNF group or a specific VNF.

The tenant key pairs are intended to identify the tenant as senders of transactions, as well transaction signing. It is important to remember that a VNF client module will only accept transactions from a known sender.

A blockchain module key pair is intended for the identification of blockchain module instances, as well as the signing of a closed block and the consensus protocol messages. This key pair is certified by a federation-appointed party and is tied to the blockchain node identity. It is important to note that this type o key pair is not used in transactions. Although the proposed architecture assumes software-generated key pairs for simplicity, any module key pair may be stored in smart cards, trusted platform modules or other tamper-proof hardware devices that are compatible with the targeted cloud platform.

As in most blockchain implementations, there is no default public key infrastructure [17] for key pairs used in transactions. Keys can be generated on-the-fly by client modules and can be replaced promptly in VNFs through an encrypted NF configuration transaction. There is no key pair revocation mechanism. This scheme mitigates attacks commonly related to public key infrastructure because the private key is never sent through a communication channel and trust is explicit. Once used, the public keys will remain as transaction identifiers in the blockchain. In this way, it is not possible for a blockchain observer to track transactions between tenants and VNFs, unless the tenant-related key pair information and their VNFs are known in advance. Thus, the key management scheme adopted by the proposed architecture decouples entity identity from their associated key pairs, anonymizing tenant identity and VNF identity [18]. This mitigates targeted attacks based on blockchain public information. However, transaction execution transparency is preserved [19], and transaction history auditing is possible for an authorized observer with shared tenant private information. In some cases, a tenant may be interested in disclosing their public key. Accounted cases stand for predefined VNF configuration, published by VNF manufacturers, or VNF best configuration practices, shared by the tenant community and service providers. These predefined configurations may aid the initial configuration of tenant VNFs. However, it should be noted that the use of predefined configurations implies in partial loss of anonymity

due to exposure of functionality, manufacturer and version information of VNFs.

C. Proposed Transaction Scheme

A transaction represents an atomic action to be stored in the blockchain. Two transaction classes are defined in the proposed architecture: i) the configuration transaction; and ii) the configuration request transaction. Configuration transactions are issued by client modules, both tenants and VNFs, and are intended to install configurations on one or more VNFs or to define a predefined configuration reference. Configuration request transactions are issued only by tenant client modules, and are intended to request the configuration state of a particular VNF.

Every transaction has two sets of attributes: header and content. The header fields of a transaction are the same for any transaction and contain the signed transaction hash to ensure its integrity. Signing is performed with the transaction sender's private key to ensure authenticity. The content fields common to both types of transactions are: i) type, which defines transaction category; ii) CID (Configuration IDentifier), which defines an unique identifier for a configuration; iii) VID (Version IDentifier), which defines the configuration version and is unique for the same CID; iv) compression, which identifies whether the configuration content field is compressed and the employed compression algorithm; v) encryption, which defines whether the content of the configuration field is encrypted; vi) description, an optional field that allows associating a descriptive text with the transaction; vii) sender, which identifies the transaction sender by its public key; viii) recipient, which identifies the transaction recipient by its public key; ix) timestamp, which identifies the date and time of the transaction; and x) nonce, which uniquely identifies the transaction.

A configuration transaction has the following additional fields: i) apply before, which identifies which settings should be applied before the current one; ii) apply after, which identifies which settings should be applied after the current; and iii) configuration, which contains the configuration information stored in the transaction. The 'apply before' and 'apply after' fields are represented by a list of CIDs for the case where the latest version of a configuration is desired, or CIDs with VIDs for the case where a specific configuration version is desired. A configuration transaction can be issued with or without encryption to be used in the configuration field. When the field must be encrypted, the recipient's public key is used to perform encryption. The encrypted use of this field addresses the confidentiality of configuration information. A configuration transaction that aims to define a public default configuration must be done without encryption of the configuration field and without recipient specification.

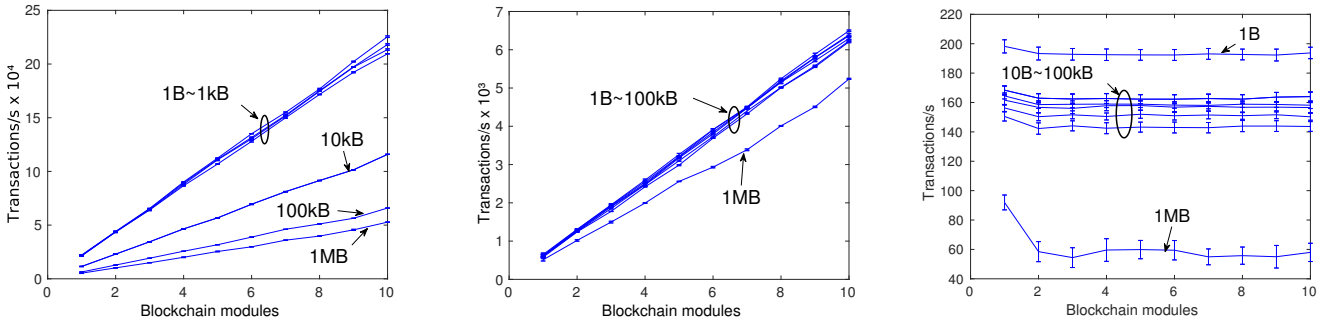
In a configuration request transaction, the semantics of the fields listed above between ii) and vi) is differentiated, indicating the values to be used by the VNF client module in the corresponding configuration transaction. A configuration request transaction of a tenant client module must be answered by a configuration transaction of a VNF client module.

The validation of a transaction by the blockchain module must observe if all the rules and field semantics described above are obeyed. An invalid transaction is discarded immediately. The validation of the transactions is performed locally at each blockchain module and is essential for the operation of the consensus algorithm. A node votes for the acceptance of a transaction if and only if the transaction is valid locally and does not conflict with any other transaction already accepted by that node. Such a requirement is important for the employed BFT consensus algorithm [20], [21].

D. Secure Migration of Virtualized Network Functions

Migration of a VNF consists in copying its configuration and states to another machine on the network. The use of conventional migration mechanisms, commonly used to migrate virtual machines, implies exposing the system to its security vulnerabilities [22]. Data can not be encrypted VNF-to-VNF because they need to be understood by the hypervisor [8]. However, VNF migration does not have to be treated as the migration of a traditional virtual machine due to its simpler scope of use. VNFs are special purpose non-persistent virtual machine instances, created on-demand and based on a predefined read-only service image. As such, VNFs do not retrieve past state information from a stored volume and are permanently destroyed when shut down. VNFs do not receive connections either, so there is no client session information associated with a VNF or links to be reestablished. What differentiates one VNF from another VNF based on the same image is solely its configuration state. So the migration of a VNF can be emulated without loss of functionality by creating a new VNF instance on the intended location, transferring its latest configuration state, and finally switching the SFC platform configuration to use the new VNF instance on its service path. The old VNF may be destroyed as soon as it finishes handling pending packets. If any new configuration state is generated at this point, it can be updated in the new VNF instance accordingly.

This work proposes carrying out the migration of the service performed by a VNF through transactions on the proposed blockchain architecture, and that only the initialization and connection of the VNF is to be left in charge of the cloud platform used by the tenant. In this way, after the initialization of another VNF containing a platform VNF client module, the service migration procedure performed consists in: i) a configuration request transaction from the tenant client module addressed to the source VNF client module; ii) the creation o a new VNF instance at the desired destination; iii) a VNF configuration transaction from the tenant client module addressed to the destination VNF client module, including the configuration obtained by the first transaction in the 'apply before' field of the current transaction; iv) switching the SFC path on the NFV platform; and v) repeating steps i) and iii) if any new state is generated at the source VNF. After this procedure, the source VNF can be turned off. Because it does not depend on the cloud platform used, this migration mechanism can be performed between compatible VNFs running on



(a) Read requests/s for a fixed configuration size of 1B, and block data size between 1B and 1MB

(b) Read requests/s for a fixed block data size of 1MB, and configuration size between 1B and 1MB

(c) Write requests/s for a configuration size between 1B and 1MB

Fig. 6: Prototype maximum transaction processing rates

separate cloud platforms, even if the virtualization architecture is different.

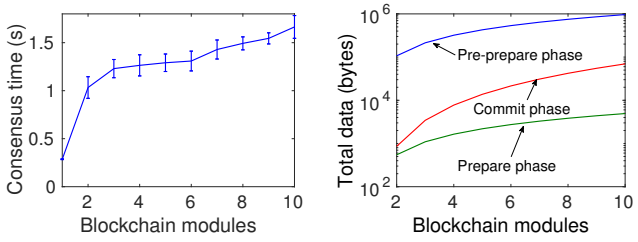
IV. Prototype Evaluation

The authors implemented a prototype of the proposed architecture and a suitable blockchain in the Python 2.7 language, using pyCryptodome library for cryptography related operations. The authors decided to implement a prototype blockchain in order to strictly evaluate the trade-offs of the proposed architecture in a controlled environment with the essential functionality set. Production implementations of this architecture can be constructed, without loss of functionality in Ethereum [23] or Hyperledger[1] based blockchains.

The Rivest–Shamir–Adleman (RSA) public key cryptography system was used, with a 2048-bit key length parameter. This system was used in conjunction Public Key Cryptography Standard #1 Probabilistic Signature Scheme (PKCS#1-PSS), which produces 256-bit signatures based on a RSA key pair. Other suitable asymmetric key and signature schemes could be used to implement the prototype. The prototype of the proposed architecture was evaluated in the Open Platform for NFV (OPNFV)[2]. Although OPNFV was used for the prototype evaluation, it is important to note that the proposed architecture, as well as the prototype, is platform agnostic. The block chain modules were evaluated in Intel Core i7-4770 3.4Ghz based computers with 32GB RAM, the VNF client modules were evaluated in OPNFV VNFs, executing a dummy forwarding service, and the tenant client modules were executed in client notebooks. Two experiments were conducted in order to evaluate what variables may impact transaction processing behavior, consensus delay and total message data exchanged between blockchain modules.

The first experiment consisted in measuring the maximum sustained transaction request rate from one to ten blockchain modules. Results show that the read request rate of the prototype scales linearly with the number of blockchain modules, as illustrated in Figures 6a and 6b. Figure 6a demonstrates block sizes up to 1kB do not pose significant decreases in performance. That is explained by the utilization of a disk sector size of 4kB, which allows blocks up to this size to be retrieved in a single read operation. Greater block sizes affect request rates more severely as continuous memory placement is necessary. Figure 6b shows that block size dominates read request time up to 100 kB transactions, and network bandwidth limits the rate for larger transactions. Figure 6c shows that the maximum request write rate is stable and does not scale with blockchain module addition. This is due to the adopted consensus algorithm, which centralizes block creation and thus imposes the consensus leader's writing capacity as a bottleneck for the system. Nevertheless, unprocessed transactions remain in the transaction log to be included in the next consensus round, implying the architecture is tolerant to processing rate peaks up to the buffering capacity of blockchain nodes. Transaction rates for 1B transactions provide an upper bound for write request rates and the disparity between the write and read request rates is a consequence of cryptographic transaction verification. Transaction sizes from 10B to 100kB offer a small linear performance degradation for order size increases. For larger transactions of 1MB, write request rate if affected by process memory constraints.

The second experiment consisted in verifying consensus delay and data exchanged in relation to the number of consensus participants, Figure 7. To this end, a batch of 100 configuration transactions, 400B in size each, was sent to the consensus leader for each experiment setup. Results show that consensus delay slowly increases with the number of nodes, but remains under 2 seconds for ten participants, Figure 7a. In addition, analysis of total data exchanged during consensus, Figure 7b, shows that the pre-prepare and prepare phases of the adopted



(a) Consensus time

(b) Total message data exchanged

Fig. 7: Prototype consensus evaluation for a 400B sample firewall configuration at 100 write transaction requests/s.

[1]https://www.hyperledger.org/

[2]https://www.opnfv.org/

consensus algorithm scale linearly, while commit phase data exchange requirements scales quadratically with the number of consensus participants. The consensus message data overhead in order to propagate configuration data to all replicas is not noticeable up to 188 nodes in the evaluated scenario, when it equals the configuration data volume. However, a noticeable data transmission bottleneck is verified for the consensus leader, responsible for configuration data propagation for all other consensus participants.

V. Related Work

Several works explore the state of the art regarding the application of blockchains in communication network problems [9], [17], [24]. These works focus on the use of blockchain as a replicated incremental data repository where all past transactions are signed and recorded with asymmetric cryptographic keys. Xu *et al.* present blockchain as a mechanism of communication and service coordination through transactions while demonstrating its applicability as a repository of distributed information [6]. Boudguiga *et al*. present a solution for updating IoT devices through blockchain-stored information [25]. The architecture proposed in the present work uses similar mechanisms to configure VNFs. However, our proposal takes into account the needs for confidentiality, anonymity and auditing , not addressed by previous works.

As discussed by Saito *et al.* and Vukolić *et al.*, blockchain replication is a distributed consensus problem due to the need of consistent transaction ordering in a trustless public environment [11], [13]. Solutions to these problems based on the original proposal by Nakamoto [10] have high computational cost and response time. Schwartz *et al.* propose the Ripple protocol for distributed consensus in federated blockchains [26]. This protocol is byzantine fault tolerant (BFT) and achieves robustness against collusion attacks by creating trust zone subsets where collusion is not expected to occur. However, the use of Ripple presents scalability problems regarding the number of nodes designated to reach consensus [27]. Miller *et al.* propose a novel asynchronous BFT protocol that achieves low-latency consensus with higher scalability [14]. Castro and Liskov proposed PBFT, a similar asynchronous BFT protocol suited for state machine replication [20] which provides similar efficiency at the default operation case and can leverage the asymmetric key infrastructure commonly found in blockchain architectures [13].

Previous works [1], [4], [5], [7] investigate the problem of security vulnerabilities due to co-location of multiple tenant VNFs on the same SFC platform. They state that compromising a single VNF at the core of the network endangers entire service function chains and their network users. Firoozjaei *et al.* and Lal *et al.* investigate security threats and propose taxonomies for vulnerabilities in NFV [8], [22]. Besides, these authors discuss the challenge of transferring configurations and states securely during VNF migration. Reynaud *et al.* point out that VNFs are built on software from different vendors, with different vulnerabilities, and mentions attacks that can be carried out against VNFs through privileged terminal access on cloud platforms [28]. Our architecture is capable of performing VNF configuration and migration through blockchain storage while preserving information confidentiality. Besides, the proposed configuration update mechanism eliminates VNF listening services, thus mitigating threats from terminals and from the network.

Several solutions are proposed in the recent literature for the problem of VNF secure configuration. Coughlin *et al.* propose the use of secure hardware through a Trusted Platform Module (TPM) to protect privacy of VNF configuration [29]. While achieving its purpose, this approach relies on specific hardware and centralized remote attestation, impacting up to 50 % of the available bandwidth. Massonet *et al.* propose an architecture for global configuration of security VNFs in federated networks with automated deployment [2]. This architecture, however, is dependent on a centralized controller and on agreement between clouds, resulting in a single point of failure and configuration delays when in situations of great competition. A variation of this architecture is proposed by Massonet *et al.* where the configuration of a security VNF is performed by the cloud platform VNF orchestrator and configuration parameters are encoded in the service function description file in [30]. This approach requires modification of various components of the orchestrator and is restricted to pre-established types of VNF. Additionally, one can not modify the security configuration of the VNF after it is booted by the same mechanism. Pattaranantakul *et al.* propose a similar mechanism to control access to VNFs through service function description files [31]. Reynauld *et al.* find, however, that such approaches render VNF configurations susceptible to cloud platform information access vulnerabilities [28].

VI. Conclusion

This paper proposes a blockchain architecture agnostic to specific hardware or cloud platform architectures for securely managing the configuration of virtualized network functions (VNFs). Besides eliminating single point of failure and providing high availability of configuration information, the proposed architecture ensures the confidentiality and anonymity of transactions without compromising auditing capability by an authorized entity. In addition, a migration scheme is proposed that preserves the confidentiality of VNF information. The proposed architecture is resilient to collusion attacks from up to a third of the blockchain modules, and configuration information cannot be compromised even during a successful collusion attack. Prototype results show that the optimal block size should be close to disk sector size, and transactions up to 100kB can be written at sustainable rates. Prototype consensus performance is within bounds of a federated consensus participant model, with an acceptable configuration delay of two seconds. In a future work, we will extend the configuration update functionality to other SFC platform elements.

Acknowledgment

This research is supported by CNPq, CAPES, FAPERJ, and FAPESP (2015/24514-9, 2015/24485-9, and 2014/50937-1).

REFERENCES

[1] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *J. Netw. Comput. Appl.*, vol. 75, no. C, pp. 138–155, Nov. 2016.

[2] P. Massonet, S. Dupont, A. Michot, A. Levin, and M. Villari, "Enforcement of global security policies in federated cloud networks with virtual network functions," in *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, Oct 2016, pp. 81–84.

[3] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu, "Research directions in network service chaining," in *2013 IEEE SDN for Future Networks and Services*, Nov 2013, pp. 1–7.

[4] C. Bouras, A. Kollia, and A. Papazois, "SDN & NFV in 5G: Advancements and challenges," in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, March 2017, pp. 107–111.

[5] R. Mijumbi, J. Serrat, J. l. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, January 2016.

[6] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen, "The blockchain as a software connector," in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, April 2016, pp. 182–191.

[7] M. Pattaranantakul, R. He, A. Meddahi, and Z. Zhang, "Secmano: Towards network functions virtualization (nfv) based security management and orchestration," in *2016 IEEE Trustcom/BigDataSE/ISPA*, Aug 2016, pp. 598–605.

[8] M. D. Firoozjaei, J. P. Jeong, H. Ko, and H. Kim, "Security challenges with network functions virtualization," *Future Generation Computer Systems*, vol. 67, pp. 315 – 324, 2017.

[9] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, http://bitcoin.org/bitcoin.pdf.

[11] K. Saito and H. Yamada, "What's so different about blockchain? blockchain is a probabilistic state machine," in *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, June 2016, pp. 168–175.

[12] L. Tseng, "Bitcoin's consistency property," in *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, Jan 2017, pp. 219–220.

[13] M. Vukolić, *The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication*. Cham: Springer International Publishing, 2016, pp. 112–125.

[14] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols." in *ACM Conference on Computer and Communications Security*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 31–42.

[15] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, Mar 1983.

[16] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker, "Off by default!" 2016.

[17] U. Mukhopadhyay, A. Skjellum, O. Hambolu, J. Oakley, L. Yu, and R. Brooks, "A brief survey of cryptocurrency systems," in *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, Dec 2016, pp. 745–752.

[18] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 839–858.

[19] Y. Zhang and J. Wen, "An iot electric business model based on the protocol of bitcoin," in *2015 18th International Conference on Intelligence in Next Generation Networks*, Feb 2015, pp. 184–191.

[20] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186.

[21] B. Liskov, *From Viewstamped Replication to Byzantine Fault Tolerance*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 121–149.

[22] S. Lal, T. Taleb, and A. Dutta, "NFV: Security threats and best practices," *IEEE Communications Magazine*, vol. PP, no. 99, pp. 2–8, 2017.

[23] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014. [Online]. Available: http://bitcoinaffiliatelist.com/wp-content/uploads/ethereum.pdf

[24] N. Bozic, G. Pujolle, and S. Secci, "A tutorial on blockchain and applications to secure network control-planes," in *3rd Smart Cloud Networks Systems*, Dec 2016, pp. 1–8.

[25] A. Boudguiga, N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey, "Towards better availability and accountability for IoT updates by means of a blockchain," in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, April 2017, pp. 50–58.

[26] D. Schwartz, N. Youngs, and A. Britto, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, 2014, https://ripple.com/files/ripple_consensus_whitepaper.pdf.

[27] C. Cachin and M. Vukolic, "Blockchain consensus protocols in the wild," 2017.

[28] F. Reynaud, F. X. Aguessy, O. Bettan, M. Bouet, and V. Conan, "Attacks against network functions virtualization and software-defined networking: State-of-the-art," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016, pp. 471–476.

[29] M. Coughlin, E. Keller, and E. Wustrow, "Trusted click: Overcoming security issues of NFV in the cloud," in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, ser. SDN-NFVSec '17. New York, NY, USA: ACM, 2017, pp. 31–36.

[30] P. Massonet, S. Dupont, A. Michot, A. Levin, and M. Villari, "An architecture for securing federated cloud networks with service function chaining," in *2016 IEEE Symposium on Computers and Communication (ISCC)*, June 2016, pp. 38–43.

[31] M. Pattaranantakul, Y. Tseng, R. He, Z. Zhang, and A. Meddahi, "A first step towards security extension for nfv orchestrator," in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, ser. SDN-NFVSec '17. New York, NY, USA: ACM, 2017, pp. 25–30.