(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 December 2016 (22.12.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/202397 A1

(51) International Patent Classification:
H04L 29/06 (2006.01) *H04L 29/12* (2006.01)
H04L 9/08 (2006.01) *H04W 12/04* (2009.01)

(21) International Application Number:
PCT/EP2015/063763

(22) International Filing Date:
18 June 2015 (18.06.2015)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HUAWEI TECHNOLOGIES CO., LTD.** [CN/CN]; Huawei Administration Building Bantian Longgang District, Shenzhen, Guangdong 518129 (CN).

(72) Inventor; and
(71) Applicant *(for US only)*: **RAFIEE, Hosnieh** [IR/DE]; Huawei Technologies Duesseldorf GmbH Riesstr. 25, 80992 Munich (DE).

(74) Agent: **KREUZ, Georg**; c/o Huawei Technologies Duesseldorf GmbH, Riesstr. 8, 80992 Munich (DE).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report (Art. 21(3))*

(54) **Title**: DNS BASED PKI SYSTEM

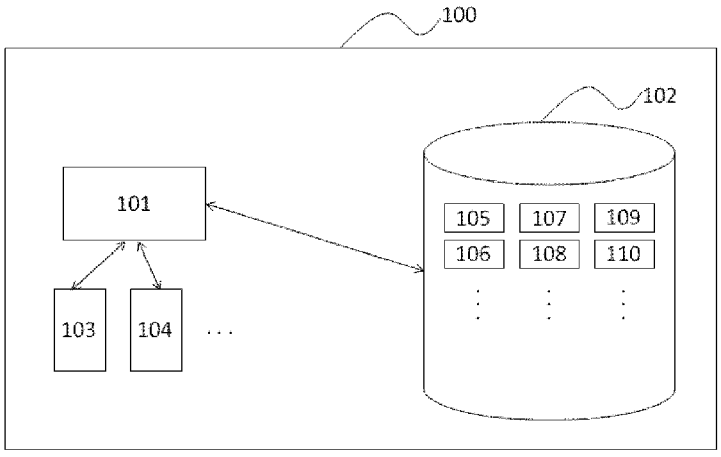

Fig. 3

(57) **Abstract**: A data processing system is provided comprising a DNS server managing a domain name service database, wherein the DNS server is configured to communicate with a plurality of communication entities and wherein the domain name service database is configured to store resource records and to receive requests, characterized in that, upon receiving an update request from a first communication entity, the DNS server is configured to create, change and/or delete at least a first resource record and at least one public cryptographic key and/or a TLSA record for the first communication entity in the domain name service database.

WO 2016/202397 A1

DNS based PKI system

FIELD OF THE INVENTION

The present invention relates to the field of computer networks and network security, especially to a data processing system comprising a DNS server and method for operating a data processing system. In particular, the system comprises means for authentication, authorization and domain based logical isolation, applied e.g. in a generic computer network infrastructure and/or a Software-Defined Networking (SDN) environment.

The invention preferably is applied in a SDN or a network using a Network Function Virtualization (NFV) infrastructure, especially to secure communication between network or SDN components such as e.g. SDN applications and SDN controllers.

BACKGROUND

Authentication and authorization of entities connected via a computer network is critical when implementing network security. Unauthorized network devices might negatively affect the network infrastructure by performing malicious activities and may compromise both the security and the privacy of network entities and users. This in particular applies to SDN components such as SDN applications (also referred to as app or apps) and at least one SDN controller, but also for NFV solutions. SDN and NFV allow easy configuration and management of a large number of nodes in network infrastructure. NFV is a network architecture concept that uses virtualization technologies to virtualize all network functions. A virtualized network

function, or VNF, may consist of one or more virtual machines running different software and processes. For example, a virtualized function (such as virtualized load balancers, firewalls, intrusion detection devices) could be deployed to protect a network without the typical cost and complexity of obtaining and installing physical units. Further, by employing a virtualized function, there is no need for a particular hardware that only supports a particular function.

One authentication technique used in computer networks is the certificates based authentication that uses a certificates generated by X.509 standard and the Transport Layer Security (TLS) protocol. TLS is a cryptographic protocol designed to encrypt and secure the communication of two communicating entities over a computer network. It supports different public key cryptographic algorithms (asymmetric cryptography) to authenticate the identity of the communicating parties. Some example algorithms are RSA, DSA, etc. It uses X.509 certificates to assure the identity of the public key signer. Different algorithms for key agreement are used. One example is Diffe-Hellman to agree on a symmetric key. The symmetric key is encrypted and exchanged via the public key cryptography algorithm. The symmetric key can then be used as a session key to encrypt data being exchanged between the communication entities. TLS can be used with self-signed certificates.

Fig. 1 shows how the TLS session for an encrypted data exchange is established using self-signed certificates in an exemplary SDN solution. In the illustrated process, a SDN component, here an SDN app, generates a public/private key pair and signs it. Then the involved devices establish a TLS session and trust each other. The TLS session

typically expires after a round trip time (RTT) elapsed or an end session request message is sent.

This method is also called "Trust On First Use" (TOFU). The TOFU approach is generally used in computer networks, for example in Secure Shell (SSH) protocol, a network management protocol for secure data communication and remote command execution. When a SSH client wants to establish a SSH session with a SSH server, it establishes a TLS session and trusts the SSH server for the first time. Then, information about the first TLS session is stored in the SSH client configuration file which allows this client to verify the SSH server next time it communicates with it. If the first trust fails and an attacker has a possibility to forge the identity of a SSH server, then the client always trusts the attacker instead of a SSH server. Therefore, TLS alone cannot guarantee a network entity's identity.

Problems with self-signed certificates might arise as outlined in the following. Although being described with reference to SDN systems, they can be also found in common computer networks.

One problem is "Spoofing" which may cause a "Man in the Middle" (MITM) attack. An illegitimate app can send a request to a SDN solution system. The SDN authentication component cannot verify the sender and prove the authenticity of the sender. Therefore, by using a spoofing and/or MITM attack, an attacker can be granted unauthorized access to network resources via a SDN controller.

Even if the certificates of a SDN controller are stored in an app and the certificates of the app are stored in the

SDN controller (the process presently applied in Android systems using self-signed apps), there are drawbacks: One is key management and scalability: an administrator is needed to include any new key of a trusted app or removes any app key in the local storage of a SDN controller. Apps also need to store the keys in their local resources. Another drawback is an increasing risk of misconfiguration: human error is a well-known problem that may result in misconfiguration of a SDN controller and allow an attacker to gain unauthorized access to the resources in a SDN-based infrastructure.

In SDN enabled infrastructures, the network is often provided by an operator and shared by multiple tenants of the operator. This, however, requires proper resource isolation in multi user environments, where several tenants or SDN applications of the tenants share a common infrastructure. Additional systems are needed to identify and isolate apps. A known approach to achieve this goal is to assign identifiers to apps and to store the identifier/assignment information in a database together with information about the network resources belonging to this identifier.

Another solution to provide authentication and authorization is the use of a public Certificate Authority (CA) database, similarly as applied in https, TLS based certificate signed by a CA is used for secure http communication over a computer network. A certificate authority or certification authority is an entity that issues digital certificates. There are public CAs, e.g. VeriSign, who certify a domain so that whenever a user accesses a web resource like https://example.com, the user's browser can ask the public CA to verify the

certificates of this website. However, there are also problems with this solution:

- A risk of a compromised CA database: A private key for a CA database might be compromised which will result in compromising all SDN solutions that use this CA database. This allows an attacker to spoof the certifications of other Apps and to gain access to a SDN-based solution infrastructure via a compromised SDN controller; and
- no isolation: This solution does not allow operators or different tenants to access their own resources and to update their own keys in CA database. There needs to be one administrator for this CA database where all keys are managed.

Another authentication and authorization solution is the use of local CA databases. This solution mitigates the propagation of attacks on SDN solutions that use the same compromised CA database but still cannot eliminate a compromised CA database attack. It might limit the scope of attack to one vendor or one operator's CA database, but an attack can be propagated easily in multi-tenant environments that store their certificates in the same CA. Therefore, the problem will be still similar to what is explained in the public CA databases.

Authentication and authorization of network entities is in particular relevant in mobile environments. Since network devices are becoming increasingly portable, user movements are inevitable. Seamless user authentication, especially during user movements, is important.

User movements can occur in multiple domains of a single operator or in multiple domains of multiple operators (roaming). Several problems are known, that arise from

user authentication. It was tried to address the issue and to solve it especially during roaming (among multiple domains and multiple operators) and user movements. But there is still a need to automate the authentication process as much as possible and to serve a user with the same access control in the visited network as the user has/had in the home network. A home network in this respect is a network the user is a customer of, while a visited network is a network that accepts the user as a guest to access limited resources. Providing seamless authentication by using existing solutions demands much administrative work because manual temporal re-configuration of network devices and security services is required for single movements of a user.

Recent development deals with this problem by employing a SDN controller for automation and improvement of user authentication. For example, a user authentication, authorization and accounting mechanism such as a Remote Authentication Dial-In User Service (RADIUS) client/server is implemented inside a SDN controller. RADIUS is a networking protocol that provides centralized Authentication, Authorization, and Accounting (AAA) management for users who connect and use a network service.

Fig. 2 shows an example of involving a SDN controller in user authentication and authorization according to the prior art. This solution has the drawback that a SDN controller is involved in every single user requests. This might lead to Denial of Service (DoS) attacks on a SDN controller especially when there are millions of users who want to be authenticated and authorized to access to a network.

Therefore, there needs to be a solution that allows to isolate resources especially of communication entities in a computer network, to assign policies to each communication entity and to identify, authenticate and authorize each communication entity easily and quickly without delay (in the sense of performance and computation costs).

In particular communication of SDN components for exchanging customized information (network topology, statistics, etc.) needs to be established securely and effectively.

Moreover, user movement and user authentication during movement (multi-domain authentication & authorization) has to be improved.

The presented solution hence provides a data processing system and method for operating a data processing system according to the independent claims. Further aspects and embodiments are subject to the dependent claims.

SUMMARY

According to a first aspect the invention provides a data processing system, comprising a DNS server managing a domain name service database, wherein the DNS server is configured to communicate with a plurality of communication entities and wherein the domain name service database is configured to store resource records and to receive requests. Upon receiving an update request from a first communication entity, the DNS server is configured to create, change and/or delete at least a first resource record and at least one public cryptographic key and/or a TLSA record for the first communication entity in the

domain name service database. Preferably, the first resource record that is created, deleted or changed matches the update request.

According to a first implementation of the first aspect, the DNS server can be configured to create, change and/or delete the first resource record in the domain name service database in case the update request is cryptographically signed with a private cryptographic key or the update request is encrypted with a session key or a private cryptographic key.

According to a second implementation of the first aspect, the session key can be encrypted by an asymmetric cryptographic algorithm. The private cryptographic key can be a counterpart to at least one public cryptographic key stored in the domain name service database.

According to a third implementation of the first aspect, the DNS server can be configured to create, change and/or delete a second resource record and a second public cryptographic key and/or a TLSA record in the domain name service database upon receiving an update request from a second communication entity, preferably in case the update request is cryptographically signed with the private cryptographic key of the first communication entity.

According to a fourth implementation of the first aspect, the second resource record can define a sub-domain of a domain defined by the first resource record.

According to a fifth implementation of the first aspect, the DNS server can be configured to create and/or change the second resource record and/or the second public cryptographic key and/or the TLSA record in the domain

name service database upon receiving an update request form the second communication entity, preferably in case the update request can be cryptographically signed with a private cryptographic key of the second communication entity, the private cryptographic key (for the second communication entity) being a counterpart to the second public cryptographic key stored and/or a TLSA record in the domain name service database before the DNS server processes an update of the domain name service database according to the update request.

According to a sixth implementation of the first aspect, the first resource record created for the first communication entity can be stored in or a zone.

According to a seventh implementation of the first aspect, the DNS server can be configured to store in the domain name service database at least one policy identification information in association with the first resource record created for the first communication entity. The DNS server can be configured to store, in the domain name service database, a subset of the policy identification information stored for the resource record created for the first communication entity, with the second resource record created for the second communication entity.

According to an eighth implementation of the first aspect, the policy identification information and/or subset of the policy identification information can be created, changed and/or deleted by the update request.

According to a ninth implementation of the first aspect, the update request can be a DDNS update request.

According to a tenth implementation of the first aspect, the data processing system can be a SDN system, further comprising an orchestrator module, a SDN controller and network elements. The second communication entity can be an SDN application operating in a zone defined by the zone record of the first communication entity.

According to an eleventh implementation of the first aspect, the first and/or the second communication entity and the SDN controller can be part of the same SDN system.

According to a twelfth implementation of the first aspect, the first resource record for the first communication entity can be defined by an operator of the SDN system.

According to a thirteenth implementation of the first aspect, the second communication entity can send an update request to the SDN controller at which the update request is received by the orchestrator module.

According to a fourteenth implementation of the first aspect, the orchestrator module can forward the update request of the second communication entity to the DNS server. The DNS server can be configured to return policy identification information, a public cryptographic key and a TLSA record for the second communication entity. The orchestrator module can be configured to verify whether the returned public key belongs to the private key with which the update request was signed.

According to a fifteenth implementation of the first aspect, the second communication entity can send a resource or query request to access a network element to the SDN controller at which the resource request is received by the orchestrator module.

According to a sixteenth implementation of the first aspect, the orchestrator module can forward the resource request of the second communication entity to the DNS server. The DNS server can be configured to return policy identification information and/or a TLSA record information for the second communication entity. The orchestrator module can be configured to verify whether the returned information belongs to a TLS certificate exchanged during the establishment of a TLS session with the second communication entity.

According to a seventeenth implementation of the first aspect, the data processing system further can comprise a resource policy database and the orchestrator module can query the resource policy database to find policy information based on the policy identification information returned by the DNS server.

According to an eighteenth implementation of the first aspect, the data processing system can include a plurality of second communication entities, each of which can access its own resource record.

According to a nineteenth implementation of the first aspect, each resource record can be unique.

According to a twentieth implementation of the first aspect, the second resource record can define a domain name.

According to a twenty-first implementation of the first aspect, the data processing system can be a Network Function Virtualization system.

According to a second aspect the invention provides a method for operating a data processing system comprising the steps of: managing a domain name service database by a DNS server. The DNS server communicates with a plurality of communication entities. The domain name service database stores resource records and receives requests, characterized by, upon receiving a request from a first communication entity, the DNS server creates, changes and/or deletes at least a first resource record and at least one public cryptographic key and/or a TLSA record for the communication entity in the domain name service database.

According to a first implementation of the second aspect, the DNS server can create, change and/or delete the first resource record in the domain name service database in case the update request is cryptographically signed with a private cryptographic key or the update request is encrypted with a private cryptographic key or a session key.

According to a second implementation of the second aspect, the session key can be encrypted by an asymmetric cryptographic algorithm, and the private cryptographic key can be a counterpart to at least one public cryptographic key stored in the domain name service database.

According to a third implementation of the second aspect, the DNS server can create, change and/or delete a second resource record and a second public cryptographic key and/or a TLSA record in the domain name service database upon receiving an update request from a second communication entity, preferably in case the update request is cryptographically signed with the private cryptographic key of the first communication entity.

According to a fourth implementation of the second aspect, the second resource record can define a sub-domain of a domain defined by the first resource record.

According to a fifth implementation of the second aspect, the DNS server can create, change and/or delete the second resource record and/or the second public cryptographic key and/or the TLSA record in the domain name service database upon receiving an update request form the second communication entity and in case the update request is cryptographically signed with a second private cryptographic key of the second communication entity, the second private cryptographic key being a counterpart to the second public cryptographic key stored and/or a TLSA record in the domain name service database before the DNS server processes an update of the domain name service database according to the update request.

According to a sixth implementation of the second aspect, the first resource record created for the first communication entity can be stored in a zone.

According to a seventh implementation of the second aspect, the DNS server can store in the domain name service database at least one policy identification information in association with the first resource record created for the first communication entity. The DNS server can store, in the domain name service database, a subset of the policy identification information stored for the resource record created for the first communication entity, with the second resource record created for the second communication entity.

According to an eighth implementation of the second aspect, the policy identification information and/or subset of the policy identification information can be created, changed and/or deleted by the update request.

According to a ninth implementation of the second aspect, the update request can be a DDNS update request.

According to a tenth implementation of the second aspect, the data processing system can be an SDN system, further comprising an orchestrator module, a SDN controller and network elements. The second communication entity can be an SDN application operating in a zone defined by the first communication entity.

According to an eleventh implementation of the second aspect, the first and/or the second communication entity and the SDN controller can be part of the same SDN system.

According to a twelfth implementation of the second aspect, the first resource record for the first communication entity can be defined by an operator of the SDN system.

According to a thirteenth implementation of the second aspect, the second communication entity can send an update request to the SDN controller at which the update request is received by the orchestrator module.

According to a fourteenth implementation of the second aspect, the orchestrator module can forward the update request of the second communication entity to the DNS server, wherein the DNS server can return policy identification information, a public cryptographic key and a TLSA record for the second communication entity. The

orchestrator module can verify whether the returned public key belongs to the private key with which the update request was signed.

According to a fifteenth implementation of the second aspect, the second communication entity can send a resource request to access a network element to the SDN controller at which the resource request is received by the orchestrator module.

According to a sixteenth implementation of the second aspect, the orchestrator module can forward the resource request of the second communication entity to the DNS server, wherein the DNS server can return policy identification information and/or a TLSA record information for the second communication entity, and wherein the orchestrator module can verify whether the returned information belongs to a TLS certificate exchanged during the establishment of a TLS session with the second communication entity.

According to a seventeenth implementation of the second aspect, the data processing system further can comprise a resource policy database and the orchestrator module can query the resource policy database to find policy information based on the policy identification information returned by the DNS server.

According to an eighteenth implementation of the second aspect, the data processing system can include a plurality of second communication entities, each of which can access its own resource record.

According to a nineteenth implementation of the second aspect, each resource record can be unique.

According to a twentieth implementation of the second aspect, the second resource record can define a domain name.

According to a twenty-first implementation of the second aspect, the data processing system can be a Network Function Virtualization system.

BRIEF DESCRIPTION OF THE DRAWINGS

The above described aspects and embodiments of the present invention will now be explained in the following also with reference to the figures.

Fig. 1 shows a schematic overview of a process using self-signed certificates.

Fig. 2 shows a schematic overview of an SDN controller involved in user authentication/authorization.

Fig. 3 shows a schematic overview of the presented solution.

Fig. 4 shows a schematic overview of the authentication model according to the presented solution in an exemplary SDN architecture.

Fig. 5 shows a schematic overview of a model for a user authentication in WiFi scenario.

Fig. 6 shows a schematic overview of a key update process according to the presented solution.

Fig. 7 shows a schematic overview of a resource policy database and of zone information.

Fig. 8 shows a schematic overview of a protocol structure for DDNS authentication.

Fig. 9 shows a schematic overview of authentication and authorization according to the presented solution.

Fig. 10 shows a schematic overview of a network topology according to an example of the presented solution.

Fig. 11 shows a schematic overview of a network topology in a virtualized environment according to an example of the presented solution.

Fig. 12 shows a schematic overview of field formats of a DDNS protocol.

Fig. 13 shows a schematic overview of field formats of a DDNS protocol for updating a resource record of type DNSKEY.

Fig. 14 shows a schematic overview of field formats of a DDNS protocol for updating a resource record of type TLSA.

Fig. 15 shows a schematic overview of field formats of a DDNS protocol for updating a resource record of type PP.

Fig. 16 shows a schematic overview of an example off adding two policy indexes.

Fig. 17 shows a schematic overview of a process of user authentication.

DETAILLED DESCRIPTION OF THE EMBODIMENTS

Generally, it has to be noted that all arrangement, devices, modules, components, models, elements, units and means and so forth described in the present application could be implemented by software or hardware elements or any kind of combination thereof. All steps which are performed by the various entities described in the present application as well as the functionality described to be performed the various entities are intended to mean that the respective entity is adapted to or configured to perform the respective steps and functionalities. Even if in the following description of the specific embodiments, a specific functionality or step to be performed by a general entity is not reflected in the description of a specific detailed element of the entity which performs the specific step or functionality, it should be clear for a skilled person that these methods and functionalities can be implemented in respective hardware or software elements, or any kind of combination thereof. Further, the method of the present invention and its various steps are embodied in the functionalities of the various described apparatus elements.

The presented solution primarily focuses on the problems outlined above and proposes a scalable authentication and authorization solution.

A generic Domain Name System (DNS) based PKI model and a means for an automatic update of cryptographic keys is provided based on existing protocols such as DNS-based

Authentication of Named Entities (DANE) and an extended version of Dynamic DNS (DDNS or DynDNS).

The Domain Name System (DNS) is a hierarchical distributed naming system for computers, services, or any resource connected to a network. It associates information with domain names assigned to participating entities. It translates domain names, which can be easily memorized by humans, e.g. to IP addresses.

A public cryptographic key infrastructure (PKI) is a system to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption. The purpose of a PKI is to facilitate the secure electronic transfer of information. In cryptography, a PKI is a technology that binds public and private cryptographic keys with an identity preferably by means of a certificate authority (CA). The binding is established through the registration and issuance process.

Public-key cryptography uses cryptographic protocols and algorithms that define two separate cryptographic keys, one of which is private and one of which is public. The two parts of this key pair are mathematically dependent on each other. A public cryptographic key is e.g. used, to encrypt plaintext or to verify a digital signature. A private cryptographic key is e.g. used to decrypt ciphered text or to create a digital signature. While typically asymmetric cryptography is used (e.g. using asymmetric algorithms like RSA), when it is referred to public and private cryptographic keys herein, also symmetric cryptography (e.g. using algorithms like AES, DES, ...) can be used to generate the cryptographic key pair. Symmetric cryptographic algorithms are algorithms for

cryptography that use the same cryptographic keys for both encryption of plaintext and decryption of ciphered text.

A cryptographic signature is a mathematical scheme for demonstrating the authenticity of a digital message or document. A valid digital signature assures to the recipient that the message was created by a known sender, that the sender cannot deny having sent the message, and that the message was not altered in transit.

A session key is a symmetric key used for encrypting all messages in one communication session.

The presented solution can be applied in all kinds of computer networks but the focus of the following description lies on SDN/NFV specific scenarios to authenticate different SDN/NFV components. The communication entities can hence be typical network equipment and/or network nodes. An advantage of the described approach is isolation of resources of different customers and allowing each customer, such as a tenant, to access its own resources in multi-tenant environment.

Further user authentication is provided as an application (app) in SDN solutions. With SDNauth a protocol is provided that can implement a dynamic way for communication of SDN controllers via an orchestrator layer. The purpose of SDNauth and the orchestrator layer will be discussed in an example below.

Fig. 3 shows a general setup of the invention. In Fig. 3, a data processing system 100 is shown. The data processing system 100 comprises a DNS server 101, a domain name service database 102 and a plurality of communication entities 103, 104. The domain name service database 102

comprises at least a resource record 105, 106 and at least one public cryptographic key 107, 108 and/or a TLSA record 109, 110.

While in Fig. 3 two communication entities 103, 104 are shown, the data processing system can also include more or less communication entities 103, 104. Communication entities 103, 104 can be any kind of physical or logical network device able to communicate with a DNS server 101.

Moreover, in Fig. 3 exemplary two resource records 105, 106, public cryptographic keys 107, 108 and/or TLSA records 109, 110 are shown. However, the domain name service database 102 can also include more or less resource records 105, 106, public cryptographic keys 107, 108 and/or a TLSA records 109, 110. While a cryptographic key may be stored in the domain name service database 102 in association with a single resource record 105, it has to be noted, that also more than one cryptographic key (and/or TLSA record) may be stored with a resource record 105.

Arrows in Fig. 3 connecting the DNS Server 101 with the communication entities 103, 104 illustrate that the DNS Server 101 can communicate with the communication entities 103, 104 and vice versa. The arrow in Fig. 3 connecting the DNS Server 101 with the domain name service database 102 illustrates that the DNS Server 101 can create, change, and/or delete resource records, public cryptographic keys, and/or TLSA records in the domain name service database 102.

The DNS server 101 can be a server that stores resource records 105, 106 for a domain or zone. Resource records 105, 106 can also be called DNS records. A zone or zone

record is typically strode in a zone file, i.e. a configuration file on the DNS server 101. The DNS server 101 responds to requests issued by communication entities with responses by matching entries in the domain name service database 102. The DNS server 101 can also be called "nameserver". A set of DNS requests is implemented that allows creating, updating, fetching and deleting resource and/or zone records 105, 106 from the DNS server 101. Zone records and resource records are often used interchangeably in the following.

Generally, the DNS server stores resource records for a domain and a matching for a query or resource request is based on the type of resource records, which e.g. is A (IPv4 address), AAAA (IPv6 address), PTR (pointer record), CNAME, CERT (Certificate record that stores PKI certificates), DS (authoritative DNS server), TXT (human-readable text), TLSA (TLSA certificate association) etc. (see also https://en.wikipedia.org/wiki/List_of_DNS_record_types). A domain is defined in a zone, which is typically stored in the zone file containing specifics of the zone and/or domain. The zone is an administrative portion of a DNS. Each zone file should have a SOA resource reord. The SOA resource record or Start of Auhortity indicates that this DNS server is the owner of this zone and best source of information for the data within this DNS domain.

A zone defining the domain "example.com" can include sub-domains such as "sub.example.com", "sub2.example.com", etc., wherein "example.com" in turn can be a "sub-domain" of the top-level domain (TLD) "com". Sub-domains can be defined by editing DNS zone information. A resource record 105, 106 may be considered the basic data element in the domain name system. Hence, "com" is the parent domain for

"example" and "example.com" is a parent domain for domains "sub1" and "sub2".

There can be different types of requests that can be sent to a DNS server 101: an update request and a query (or resource) request. The DNS server 101 can have at least two types of responses to these requests: an update response and a query response.

The DNS server 101 can further be configured to create, change and/or delete a resource record 105 in the domain name service database 102 in case the update request. Especially, if it is cryptographically signed with a private cryptographic key or if the update request is encrypted with a private cryptographic key or a session key. In particular, a first resource record may be a zone record or a sub-domain. Therefore, the DNS server may store a zone defining a domain, and/or a sub-domain for an already stored zone in the domain name service database when an update request is received. In particular, the resource record 105 defines a sub-domain and/or includes a record type. According to the invention, a cryptographic key 107 and/or a TLSA record 109 can also be stored in association with a resource record and a record type in the domain name service database.

The update request can be encrypted with the session key by a symmetric or asymmetric cryptographic algorithm, and the private cryptographic key can further be a counterpart to at least one public cryptographic key 107, 108 stored in the domain name service database 102.

The DNS server 101 can be configured to create, change and/or delete additional resource records 106, e.g. a sub-domain, a further public cryptographic key 108 and/or a

TLSA record 110 in the domain name service database. This can be performed upon receiving an update request from the same or a further, second communication entity 104. A creation, change or update and/or deletion preferably is only performed in case the update request is cryptographically signed with the private cryptographic key of an already known communication entity 103, 104, e.g. a communication entity for which a resource record exists and for which a (public) cryptographic key and/or TLSA record is stored. The second resource record 106 can define a sub-domain of a domain defined by the first resource record 106.

The DNS server 101 can be configured to store, in the domain name service database 102, at least one policy identification information in association with a resource record and especially with one created for the first communication entity 103. Policy identification information (or policy index/indices) define an access control and can include authorization information, such as what entity is allowed or denied to use a certain resource. However, the policy identification information typically identifies a policy information or rule set that is stored in the system. Thus, if policy identification information is stored with a resource record and returned upon a query request, the communication entity receiving the response can obtain the policy information identified by the policy identification information (or policy index). It is also possible to define a sub-set of policy identification information for a resource record, which comprises all or some of the policy identification information stored for another and preferably a higher-level or "parent" resource record or zone record. The policy identification information and/or subset of the

policy identification information can be created, changed and/or deleted by an update request.

The update requests can be Dynamic DNS (DDNS or DynDNS) requests. DDNS is a DNS based protocol which allows a communication entity to update its resource records 105, 106 (e.g. domain name, PTR (Pointer) record, IP address and additional information about a domain name) on DNS server 101 automatically. Also a cryptographic key and/or a TLSA record with the DDNS extension proposed. DDNS requests are used to update resource records without manual editing. DDNS commonly uses the Transaction Signature (TSIG) mechanism to provide security. TSIG is a computer networking protocol primarily used by the DDNS to provide means of authenticating updates to a DNS database.

As stated above, the data processing system 100 can be an SDN system. The data processing system 100 can further comprise an orchestrator module, an SDN controller and network elements/communication entities. A communication entity 104 is an SDN application can operate in a zone defined by another communication entity 103. The communication entities 103, 104 and the SDN controller can be part of the same SDN system. The resource record 105 for the communication entity 103 can be also defined by an operator of the SDN system. An update request can be received by an SDN controller and/or by an orchestrator module, preferably associated with the SDN controller.

In SDNs, in principle, the control of a network node is decoupled from the physical setup of the network nodes. This allows, on an abstract level, to define the requirement of how and where the network data is sent (this is often referred to as the "control plane"). An underlying, more physical plane exists, that actually

forwards the data to the selected destination according to the requirements set on the control plane (this portion of the setup is typically called "data plane").

Therefore, in an SDN the data forwarding capability is decoupled from the routing, resource and other management functionality, which was previously performed in the network nodes. Network nodes, such as virtual switches (e.g. an open virtual switch (OVS)), that support SDN (i.e. that are SDN-compliant) may be configured to implement the functions according on the data plane, while the control plane functions may be provided by an SDN controller managing and controlling the configuration on the data plane.

An application programming interface (API) may manage the interaction between the data plane and the control plane and allow for the implementation of (non-vendor specific) combinations of networking nodes and SDN controllers within a network. As a result, SDNs in conjunction with the provided API service may provide numerous benefits to network infrastructures which, for example, include increased network virtualization, flexible control and utilization of the network but also customization of networks according to specific requirements and the optimization of the data flow through the network.

Typically, the control plane which comprises the SDN controller interacts with an "application plane" through the API. On the application plane, SDN applications (apps) are provided, which communicate their network requirements and desired network behavior to the control plane. In addition, they may consume an abstracted view of the network for their internal decision making process. An SDN application may also expose another layer of abstracted

network control thus offering one or more higher level interfaces which can be used to communicate with the application.

An orchestrator is a logical abstraction layer of an SDN controller. There are different implementation choices for orchestrators. An option is to implement an orchestrator as a new layer with an SDN controller, and to preferably deploy both on a common virtual machine. Since all of the orchestrators function is logically separated from the SDN controller, several implementation choices are possible. The orchestrator module can facilitate authentication and authorization by communicating with various communication entities and exchanging, e.g., user information or resource policy information. The orchestrator module is also described below with regards to Figs. 4, 5, 6 and 9.

The orchestrator module can forward an update request of a communication entity 104 to the DNS server 101, wherein the DNS server 101 is configured to return policy identification information, a public cryptographic key 107 and/or a TLSA record 109. The orchestrator module verifies whether the returned public key 107 belongs to the private key with which the update request was signed.

A second communication entity 104 can send a resource request to access a network element to the SDN controller at which the resource request can be received by the orchestrator module.

The orchestrator module forwards the resource request of the second communication entity 104 to the DNS server 101. The DNS server 101 returns policy identification information and/or TLSA record 110 information for the second communication entity. The orchestrator module

verifies whether the returned information belongs to a TLS certificate exchanged during the establishment of a TLS session with the second communication entity 104.

The data processing system 100 can further comprise a resource policy database (not shown). The resource policy database can be used to store policy information. The orchestrator module can query the resource policy database to find policy information based on the policy identification information returned by the DNS server 101. If an operator already has a resource policy database in use for other purposes, the policy information can be reused. As mentioned before, the data processing system 100 can also be a NFV system.

In SDN environments, application authorization is provided by introducing parent policy templates that can be assigned to different zone records and sub domains of a zone record or zone file. This can be achieved by using existing protocols, e.g. DANE and DDNS.

DANE is an IETF standard that allows certificates to be bound to DNS names using DNSSEC. It enables additional assurances for a PKI-based model, as well as enabling domain holders to assert certificates for themselves, without reference to third-party certificate authorities. DANE is used as an authentication protocol with combination of DNS server as PKI storage. To associate a TLS server certificate or a public cryptographic key with a domain name, DANE uses a TLSA DNS resource record.

The DNS system can also implement DNS Security Extensions (DNSSEC) for securing information provided by the DNS server, e.g. on Internet Protocol (IP) networks.

Similarly to communication entities or network nodes in a common computer network, the presented solution can be employed by SDN apps to update cryptographic keys in the domain name service database, but also certificates auth value (TLSA) records and/or app names (e.g. sub-domains) in their own or a specific zone record or file securely. "Their" zone record or file means the zone record or file which is affiliated with a tenant having a contract with a network operator and operates the apps. According to the presented solution and as outlined previously, the DDNS protocol is extended. In particular, the standard DDNS protocol can only update resource records of e.g. types A, AAAA and PTR, TXT (text) or CNAME (Canonical Name), but is not capable of updating cryptographic keys (also, because a standard DNS system does not store those keys), policy information, TLSA records or any other security values dynamically.

In the data processing system described herein, existing protocols like DNS, DDNS and DANE are extended and used in new use case scenarios. DNS, DDNS, and DANE are no security mechanisms. However, for updating any values on a DNS server 101, there is the need for a secure authentication mechanism to allow the DDNS protocol to securely update any resource records on a DNS server. The idea of the proposed solution is to use none-security mechanisms for security purposes and to provide a scalable secure authentication model which needs extension of existing protocols.

In Fig. 4 an authentication model is shown in an exemplary SDN architecture 400. Here, a data processing system operated by an operator is used by several tenants, illustrated by "Tenant1" and "Tennant2", each of which can deploys at least one app, "App1", "App2".

After a mutual agreement between an operator "Operator1" and a tenant Tenant1/2, the operator defines a new zone for the tenant Tenant1/2 (e.g. by storing a zone record or zone file for Tenant1) and stores a tenant's public cryptographic key, policy identification information and/or a TLSA record in a DNS server 401.

Tenant1 receives information about a domain name of Operator1's SDN solution, which in particular is a domain of an SDN orchestrator service, and a certificate of Operator1, which includes Operator1's domain name and/or TLSA record. The public cryptographic key, TLSA record and/or name of each app can be automatically updated in the DNS server using the extended DDNS protocol. DDNS is extended because it needs to carry a cryptographic key, but also policy identification information and/or TLSA records. DDNS is also extended with a secure authentication method.

First (see Fig. 4, step 1), an update request of the app App1 of Tenant1 is signed by the apps's parent private cryptographic key, which in this example is Tenant1's private key. This means, that the App1 is a "child" operating in the parent's (Tenant1's) zone. The app from Tenant1 can send the resource request to an SDN controller 402.

If security function services of a security unit of the SDN controller or in the network do not detect any attack (e.g. based on heuristic estimations), the update request is received by the orchestrator service. The orchestrator service implements a mediator and queries the DNS server 401, by using the DANE protocol, to receive more information about the app (See Fig. 4: step 2).

The DNS server 401 responds to a DANE request using the extended DANE protocol and includes policy identification information and the public cryptographic key of the app App1 so that the orchestrator service can verify and authorize the app App1 (see Fig. 4: step 2).

Policy information is defined and stored in a resource policy database 403. Policy information is based on grouping different, e.g. southbound OpenFlow messages and other available possible functions, in the SDN controller Operating System (OS), thereby implementing different levels of security.

The orchestrator service queries the resource policy database 403 to find policy information based on the policy identification information received from the DNS server 401. Then it authorizes the app App1 and grants it access to the SDN controller 402 (e.g. based on the policy information obtained) or executes other processes.

The authentication and high level authorization model can be used for the authentication of any SDN component, e.g. a communication entity or network device (such as a vswitch, a vrouter, a switch, ...) to a SDN controller 402. All communication to a SDN controller 402 from apps App1, App2 takes place via the security unit and orchestrator service so that the identity of senders in all communication from communication entities or network devices is being verified via the orchestrator service (See Fig. 4: step 4).

In the example, each tenant Tenant1, Tenant2 has complete control over its own zone file(s) and its own domain(s).

Each app App1, App2 also accesses its own resource records such as changing its domain name.

In each zone, the DNS server 401 does not allow apps to choose the same names as that of an existing app. Each app needs to have a unique name in its own zone. But the uniqueness of app names is not required in the whole DNS (all zones in DNS server 401). For example, Tenant1 is allowed to have app names like "app1.tenant1" and "app2.tenant1" but is not allowed to have multiple apps with the same names e.g. "app1.tenant1". The DNS server 401 does not allow this in a same zone file.
However, an app can have a same name in another zone file. For example, a Tenant2 can have an app with the name App1 (thus "app1.tenant2"). The absolute domain name of this app is unique in this DNS server 401. As shown in Fig. 4, a DNS server 401 can have different zone files for Tenant1, Tenant2 and Operator1.

The communication of apps App1, App2 with each other or with one or more SDN controller is critical and important. This is because each app might only support some specific features and needs to receive information from other apps so that it can complete a task. For example there are DoS or DDoS attack testing apps and intrusion detection system (IDS) apps in a network. An IDS app might need data from the DoS app so that it can generate a policy and apply these policy rules to switches and other network devices based on the pattern of an attack.

Another example is that the operator uses SDN controllers from different vendors or has the SDN controllers distributed in different data centers. To gather information about the entire topology, an app App1, App2 needs to receive information from all SDN controllers in

all different domains (in Fig. 4 this is illustrated with "external SDN controller" 404 and Step 3). Hence, a communication of the SDN controllers 402, 404 is required. Since the communication of two SDN controllers 402, 404 might be complex, providing many different ways of communication might open new security issues on a SDN controller 402, 404.

Therefore, it is essential to limit the scope of attacks. The orchestrator service (See Fig. 4: step 3) is used as a mediator or point of all contact and communication among SDN controllers 402, 404 and apps App1, App2 is facilitated through the orchestrator service. In this case, an SDN controller 402 only needs to know who the responsible orchestrator service is for communicating with another SDN controller 404. An orchestrator service can resolve the address of different SDN controllers e.g. via a local (or public) DNS server.

The following information can be kept in DNS server 401 for registration of an orchestrator service in a zone, e.g. a "operator1" zone:

Orchestrator_ipaddress: The IP address of an orchestrator service.

Orchestrator name: The domain name of orchestrator service, e.g., "orch1.operator1".

Pub key: Orchestrator Public key.

To facilitate communication, each SDN controller may register in DNS server 401. However, an IP address of a SDN controller preferably is not public (for security reasons) and it is the security unit (e.g. of the

orchestrator service) which receives all requests from an app App1, App2 and processes them.

The orchestrator's address is stored in the DNS server 401 and provided to the tenants Tenant1, Tenant2. A structure for this kind of communication is provided by the proposed protocol, in the following referred to as SDNauth, which allows secure information exchange. It is a new protocol, which can also be used for interactions of SDN apps App1, App2 and SDN controllers 402, 404 with each other e.g. to exchange user information.

In another example, a user authenticator can be also an app of the application plane. Fig. 5 illustrates an exemplary model for user authentication in a WiFi scenario.

When a user moves from his home domain to a visited domain belonging to a different operator (see Fig. 5: step 1), the SDN orchestrator service of the visited domain or network can provide communication among apps so that user session information can be exchanged among two different orchestrator services and can be forwarded to a user authenticator app (see Fig. 5: step 2) of the visited domain. The described approach eliminates the need for a user to also have credentials in the visited network and thereby supports seamless authentication for the user.

After the users request is received by the visited domain SDN controller (via existing Extensible Authentication Protocol (EAP)), the SDN controller of the visited network may convert the request to the Remote Authentication Dial-In User Service (RADIUS) protocol, encapsulates the request and adds a new header to the request so that it is

understandable by the orchestrator service "Orch.operator2" of the visited network.

The communication among the SDN controllers and orchestrators follows SDNauth. The SDN controller submits the converted request to the orchestrator service ("Orch.operator2"). Orch.operator2 retrieves the requesting user's domain (e.g. "user@app1.operator1") and queries the DNS server 101 for the location of the user authenticator app which has further information about the user (See Fig. 5: step 3).

A DNS server 501 responds with the location (such as an IP address), policy identification information (top level authorization information) and the TLSA resource record of "app1.operator1", using the extended DANE protocol (See Fig. 5: step 3). Then authentication is processed (e.g. TLS based authentication) and the visited networks controller's request is forwarded to "app1.operator1".

"app1.operator1" decapsulates or decodes the request and forwards it to RADIUS server components where the request is parsed and an authentication, authorization and/or accounting server is queried for user authorization information (See Fig. 5: step 4). This request is again encapsulated according to SDNauth and is forwarded to "Orch.operator2".

"Orch.operator2" processes the authentication of "app1.opeartor1" and then queries the resource policy database to retrieve at least one template for external applications. Then the application is e.g. granted limited access to the SDN controller in operator2's domain and some rules are applied on operator2's on network devices,

e.g. opening a port on a switch so that the user can access the internet.

As it is now described in view of Fig. 6, another example shows how to implement an automatic cryptographic key and TLSA record update in a DNS based PKI system. As an existing PKI cannot provide an automatic cryptographic key update, other mechanisms are in use. In case of using these mechanisms (e.g. MS active directory (ADS)), the mechanism only allows to have automatic update of a cryptographic key for a zone or for domain names but does not allow each sub-domains to update their own cryptographic keys.

Turning to Fig. 6, a process is detailed for updating a (public) cryptographic key in a resource or zone record (or zone file) by using a parent key of the zone and replacing a domain name of an app App1 with the app's own key. In the process, after a contract and agreement between the operator and the tenant Tenant1 is agreed on (which may be achieved electronically), Tenant1 generates a cryptographic key pair and gives its public cryptographic key and TLSA record to Operator1.

Operator1 creates a zone with the name Tenant1 on the DNS server 601, which stores a respective resource record in the domain name service database. Based on the agreement about use of resources, Operator1 assigns at least one parent policy template (parent policy templates can also be called policy information templates) to Tenant1's zone record or file (as schematically illustrated in Fig. 7).

According to the agreement, Tenant1 also receives the domain name "orch1.operator1" and TLSA record of Operator1. The domain name "orch1.operator1" should be

defined in a public DNS server. If it is not possible to use a public DNS server for an orchestrator domain, then Tenant1 should receive the IP address and certificates or public cryptographic key of "orch1.operator1". This eliminates attacks with a spoofed Operator1.

After this manual or automatic step, Tenant1 has control of its own zone and can allow any app in its own zone record to include values in Tenant1's zone information. Tenant1 can assign policy identification information it received from Operator1 to the app according to the need of the app to access resources on the network.

In Fig. 7, a schematic overview of a resource policy database and DNS server zones is shown. Policy information is defined and stored in the resource policy database. The illustrated resource policy database contains policy information grouped in parent policy templates. Parent policy templates (or policy information templates) provide information to define access control and can include authorization information, for example what entity is allowed or denied to use a certain resource.

As illustrated, in the DNS server zones, zone information for each tenant's zone can be stored.

In a tenant's zone file, information on the policy information of the parent policy templates available to the tenant is stored by using policy identification information. Fig. 7 shows policy identification information 1,2,3 assigned to Tenant1 zone, which defines the policy information for apps App1, App2, ..., Appn of Tenant1. Tenant1 can choose from the available policy information and can assign policy identification information to the apps App1, App2, ..., Appn to grant

access to resources on the network according to the needs of the apps or the tenant.

In Fig. 7, e.g., App1.Tenant1 is granted access to resources according to policy information identified by policy identification information 1 (i.e. by index=1), App2.Tenant1 is granted access to resources according to policy identification information 1 and 2 (i.e. by index=1,2) and Appn.Tenant1 is granted access to resources according to policy identification information 1 (i.e. by index=1). Additionally, public cryptographic keys that are associated with the apps are shown in Fig. 7.

It is also possible to assign all parent policy information, but for security reasons, not all communication entities have access to all policy identification information or policy indices and can assign them to their sub-domains. For example, if parent policy indices 1,2,3 is assigned to tenant1 zone, then it only can choose these parent policies to assign new records to its sub-domains, e.g. app1.tenant1. However, there might be also parent policy indices 4, 5, 6. But neither tenant1 nor app1.tenant1 has access to these values and if it wants to assign this value, the DNS server rejects it.

Turning back to Fig. 6, when an application wants to use a SDN controller or access any southbound resources (e.g. a switch, network node, etc.) for the first time, the application needs to be registered in Tenant1's zone file. For doing that, the app generates a cryptographic key pair and asks Tenant1 to sign the app's DDNS request. For authenticating the DDNS requests, there are options such as using TSIG.

TSIG needs a shared secret that should be shared with all members of a zone file. In this case every app can access and modify the information of other apps too. To limit the scope of attacks, e.g., CGA-TSIG can be used as an alternative. CGA-TSIG is a standard draft proposal that uses, e.g., CGA algorithm and proposes a possible way to automate the presently manual process for the authentication of a node with a DNS server during a DNS Update process.

The problem with the use of TSIG is that in this case one App knows the shared secret, it is no longer a secret value and it can control the entire zone file of Tenant1. Hence, the app's access to Tenant1 resources should be restricted.

An app might be provided by a third party and Tenant1 only may have leased the network infrastructure from an operator to serve to third party Apps. An app therefore should not be able to change parameters of other Apps or information in the zone file. TSIG cannot provide this separation.

To avoid this problem and to have completely DNS based logical isolation, other authentication approaches need to be used, e.g. allowing Tenant1 to sign its own DDNS request with its own private key and include the signature in CGA-TSIG field. One of the purposes of CGA-TSIG is to provide a secure authentication during DDNS processes. Since binding the IP address with the public cryptographic key is not important, only signing the values and storing signature in CGA-TSIG data structure would be enough.

Fig. 8 exemplarily shows the use of CGA-TSIG for DDNS authentication purposes. The same method explained in CGA-

TSIG for DDNS authentication in Internet Protocol version 4 (IPv4) can be used. For both Internet Protocol version 6 (IPv6) and IPv4 support, only the use of the approach for IPv4 is enough.

As illustrated in Fig. 8 (a), an application may generate a DNS update request with the illustrated structure and sets zone section to Tenant1 and update section to the resource records it wants to update, includes DNSKEY or SDNKEY resource record (public cryptographic key) and also an application name. A SDNKEY structure is similar to DNSKEY. It is a proposed resource record for keeping public keys for this invention.
Then it includes the additional data by forming the structure as illustrated in Fig. 8 (b). The format as used in Fig. 8 (b) is described in detail in RFC 2845. Only the algorithm name needs to be set to CGA-TSIG. In the "other data" section, the App includes CGA-TSIG data structure. The field shown in Fig. 8 (c) refers to information as outlined in RFC draft-rafiee-intarea-cga-tsig.

The "old signature" field shown in Fig. 8 (d) is used in case that Tenant1 or App1 from Tenant1 wants to replace its key with a new value. It first needs to be proved who the owner of the old value is so that the DNS server 101(Fig 3) allows the tenant to access its own key.

In the following, the parameters used in Fig. 8 (d) are described:

AsyAlgorithm: Asymmetric algorithm. Internet Assigned Numbers Authority (IANA) numeric value for RSA algorithm is 1.2.840.113549.1.1.1. For Elliptic Curve Cryptography (ECC), IANA needs to define a new number.

Type: Name of algorithm. In this invention, it would be 3.

Parameters Len: Length of parameters

Signature Len: Length of public key cryptography signature

Signature: all the DNS update messages excluding CGA-TSIG in additional data section are concatenated and signed by a private cryptographic key:

If Tenant1 wants to replace its own public key in Tenant1's zone after the agreement with operator1, it can sign an update message with its own private cryptographic key and sign the Time Signed with its old private cryptographic key so that DNS server 101 can verify it and allow to replace its key.

If any application from Tenant1 wants to add its values for the first time on Tenant1's zone, this value is signed by Tenant1 private cryptographic key.

If an application wants to update its own value on Tenant1 zone, the private cryptographic key to use is the Apps private cryptographic key.

Old Signature Len: Length of old signature field

Old Signature: Old signature generated by old private cryptographic key. This is only in use when the public cryptographic key values are going to be updated on the DNS server 101. Otherwise the old

signature length will be set to 0 and this value will not be added.

Therefore, all of the existing protocols (with the need for extension) or the new draft standards can be used for the purpose of automatic cryptographic key update and allowing each tenant and each App of tenants to only update their own values on a zone file. This provides tenants control over their own resources and a good PKI model, too.

According to another example of the invention, in Fig. 9 a detailed process of authentication and authorization is illustrated.

An app first sends a "hello client" message to establish TLS communication with operator Operator1. An orchestrator service exchanges TLS version and other information with the app. Then the orchestrator service responds with a "hello server" message and immediately submits its certificates to the app. App "App1.tenant1" verifies Operator1's certificate by generating a TLSA value. "App1.tenant1" then sends its own certificates to Operator1. The orchestrator service uses an existing DANE protocol and queries the DNS server 101 to retrieve more information about "App1.tenant1". The DNS server 101 finds a sub-domain in Tenant1's zone file. The DNS server 101 sends the TLSA record 109, 110 and/or policy identification information of application App1 using the extended DANE protocol. DANE, as known from the prior art, does not carry any information about policy identification information, however, the presented solution extends DANE to carry this information.

The DNS server also signs the TLSA record or submits its signature so that the orchestrator service can verify the DNS server's identity and adds a DNSKEY to the response to the orchestrator service. The orchestrator service verifies the DNS servers 101 signature, verifies "app1.tenant1"'s certificates with the TLSA record obtained from the DNS server and queries the resource policy about the policy identification information or policy indexes (1,2) to retrieve app authorization information.

Then starts the ciphered data exchange between app and SDN controller via one of well-known key exchange algorithms in TLS and the SDN controller encrypts values with the public cryptographic key of "app1.tenant1" and submits it to "app1.tenant1". App1 and the SDN controller start exchanging further data in encrypted form and "app1.tenant1" is authorized to access to the requested resources based on relevant policy information.

When the process is finished, "app1.tenant1" asks to end the TLS session. The TLS session will typically expire after "app1.tenant1" is inactive for a round trip time (RTT) or an end session request message is sent.

Considering communication of SDN components to exchange customized information (network topology, statistics, etc.) in a secure and effective way means of communicating via an orchestrator are provided without the need of a SDN controller to be involved, which is called SDNauth.

Fig. 10 shows an example for a suggested network topology for prototyping this invention and the role of a TCP/IP-layer 3 device. A TCP/IP-layer 2 switch is used to connect

all devices in the test lab. A DHCP server is used to assign an IP address to nodes in the internal lab.

DNS is one of the key components, which is used in the PKI as previously described. The solution can be implemented with any existing (open source) DNS server that supports DNSSEC. Examples are BIND, OpenDNSSEC PowerDNS. Also a virtual switch Vswitch, a separate orchestrator service, which can be used in the context of an "AAA"-Layer, and an SDN controller (e.g. based on the Open Network Operating System (ONOS)) is depicted. The "AAA"-Layer thereby refers to network functionality that provides authentication, authorization, and accounting means. This can for example be done by a RADIUS server. The TCP/IP-layer 2 switch connects the components, which may be, while depicted as separate computing devices, can be run on one computing device and may at least partially run as virtualized computing devices on one or more virtual machines. The TCP/IP-layer 3 switch or router connects the computing devices to another network, labeled as "white zone", but may also connect directly or indirectly to the Internet. The development machine only is shown for sake of this example.

Turning to Fig. 11, a setup of some components illustrated in Fig. 10 is shown in a virtualized environment. An OF-Switch is an OpenFlow (OF) switch, where OpenFlow is a communication protocol that provides access to configuration components of a switch or router, which process network packets (also called the forwarding plane) and can be used in southbound communications. There are some OpenFlow switches available. One option is the use of Vswitch.

In the DNS server, the new public keys, certificates, and policy identification information are stored.

The DNS server supports DNSSEC and can be used in different ways, such as an authoritative DNS server and a recursive DNS server. A recursive DNS server is a DNS server that provides an answer to the query requests by querying other authoritative DNS server. It first checks its cache for the existence of this query and if it cannot find any records for the request, it will query other DNS servers. For example, to resolve example.com, the recursive DNS server first asks .com about the IP address of a server that is the owner of domain example.com. This server is called an authoritative DNS server that can provide an original answer to the queries, not provided from a cache. Then this server is queried for an A resource record of example.com. The result will be the IPv4 address of example.com. The DNS server then returns this value to the query requester (such as a user's device).

The DNS server can be an authoritative server for one or more zones. For example, example.com is a zone. A zone is an administrative responsibility portion of a domain name space that is delegated to a single manager. Zone example.com can contain third level domains, e.g. www.example.com and has Example Inc. as its domain manager. The DNS server can use different domain name service databases to store zone and domain information. In particular, the domain name service database can be a SQL database, a NoSQL database, a key-value-store, or a text file.

The DNS server implements functions for handling DDNS protocol requests. DDNS needs to be changed so that it

considers secure DDNS updates and also can consider and accept automatic update of DNSKEY, TLSA, TXT (optional) and policy identification information and/or resource records on the DNS server.

Such function can be called "update-my-ip-address". To update policy identification information (also called PP), DNSKEY and/or TLSA resource records on a DNS server, these values should be added to the update section of a DDNS request message. This is illustrated by Fig. 12, which shows resource records that can be updated on a DNS server in the update section of a DDNS update request package.

The DNSKEY is a resource record that is used in DNSSEC for storing cryptographic keys, in particular public cryptographic keys. The DNSKEY resource record can be used to verify a tenant's update requests to update its zone record value or an app's requests to update its name(s) and/or TLSA (certificates authentication value).

Fig. 13 shows the format of the update section of the extended DDNS protocol for a record of type DNSKEY. When an update request is signed or sent by tenant1, who is the owner of tenant1 zone, then bit 7 of a flags field of the RDATA section is set to 1. If the key belongs to an application, then bit 7 of flags field is set to 0.

It is preferable to introduce a new resource record in the DDNS protocol for a public cryptographic key so that it is not confused with DNSSEC. In particular, this can prevent cryptographic keys used in SDNauth, which are called SDNKEY, from being mixed up with cryptographic keys used in DNSSEC, which can be called DNSKEY. DANE is used to store certificates auth values, so called TLSA, on the DNS

server. DANE uses a resource record to store the certificates auth value.

Fig. 14 shows the format of the update section of the extended DDNS protocol or upstate request, and specifically the RDATA (Resource Data) field for updating a TLSA resource record (type is TLSA). IETF RFC 6698 provides further information on the function of the depicted fields.

Fig. 15 illustrates the suggested format for a RDATA section for type "PP". To transport policy information as a response to queries or updating values automatically on a DNS server, the DANE based DNS responses and DDNS can include the new resource record, "policy information" (or parent policy (PP)).

The RDATA section carries the policy identification information or policy indexes. Policy identification information is an index or pointer to policy information stored in a resource policy database and a short description of it.

Storing policy information in DNS has two advantages: It allows the orchestrator to quickly authorize the requester that can be an application or any communication entity, and it allows each tenant to assign its own resource policy to the apps (i.e. applications) from the list of policy information templates defined by operators. Operators do not need to provide access to their resource policy to allow tenants to assign different means of access controls to their applications. This provides abstractions in access control and isolation of resources. The policy identification information resource record is added to all responses from the DNS server 101 sent to the

orchestrator (the SDN component that verifies the requester).

Further parameters of the extended DDNS protocol can be "Length", defining the whole length of the RDATA section, which can be variable because of a short description added to the policy information; "Index", which is a 32-bit reference index number from policy database; "Text, which is a short description of the policy information in ASCII format (e.g. "access to switch x,y,z") and can be stored in the DNS server, returned to an orchestrator and updated or retrieved by a tenant; and "Query Request (QR)", which can be a 3-bit value representing the query requester.

If the requester is an orchestrator (the verifier component), this value is set to 1 (i.e. 001). In this case, the text field will not be added. This is because the text field is only giving more information about the policy. If it is a tenant (e.g. a human) who requests a query to understand what means of access control are available, then it can set it to 000 so that DNS server adds the text field to the policy information in the response to a request.

There can be more policy information assigned to a tenant's zone. For each policy information, one policy information resource record is added to the DNS server. The DNS server can also include them to responses for the request for a tenant's zone information.

Fig. 16 shows an example how to add more policy identification information instances e.g. two policy indexes. Therefore, for three policy information instances, three policy information resource records are added with different indices and different text.

When a tenant updates policy information on a DNS server, the policy information resource record also needs to be added in the update section of DDNS protocol.

Hence, the orchestrator can play two roles dependent on the scenario.

In a first role, the orchestrator can act as DNS proxy. In this case the DNS server is local and only available in an operator's network and not available to public. It cannot respond to any request from outside the operator's network. Therefore, the orchestrator can implement a proxy, so as to receive a request and sends it to the DNS server. Since security of a DNS server is important, the architecture introduced uses an orchestrator as a DNS proxy, too. Tenants do not know the IP address of the DNS server and cannot communicate with it directly but they only know the domain name of the orchestrator to where they can send all their requests. The orchestrator domains (e.g. orch1.operator1) need to have a name in a public DNS server so that they can be accessible for roaming scenarios. If operators do not want to have a public domain name for their orchestrator service, the IP address of the orchestrator needs to be exchanged manually among all operators who wish to allow their users to access internet services in multi-domains and multi-operators environment.

In a second role, the orchestrator can act as SDNauth Parser finder. This is necessary for the communications of SDN controllers and SDN applications and for separation of tasks of different applications. One example is a user authenticator app scenario. Since, an orchestrator might be located in another data center and not in the same

local link as the SDN controller, and EAP protocol used for sending user authentication information from the end system to an access point is a layer 2 protocol, the requests from users are only valid in local link or the same network of the end system. Thus, a SDN controller can support RADIUS client functionalities, receive EAP requests, generate a RADIUS message and encapsulate it in another REST protocol in an SDNauth format (which is described below). An Orchestrator finds the responsible App for this request based on the domain of this request and via querying the DNS server. Then the SDNauth request is forwarded to the authenticator application. The authenticator application knows where an authentication, authorization and accounting server is located and can verify this user and retrieves the user's authorization information.

A more generic scenario facilitated by SDNauth is communication of two SDN controllers: This is a general scenario applicable for any scenario with multiple SDN controllers from different vendors or from single vendors that are distributed in different data centers. When an operator has different SDN controllers on different data centers, an orchestrator can exchange data among them and can give information about the domains an operator is managing.

ONOS, for example, is an open source SDN controller that can be used and extended to fulfill the requirements. To use ONOS in a system according as presented, the following extensions are relevant:

EAP is only a requirement for user authentication scenario via an access point. ONOS needs to be extended to support EAP protocol so that it allows the recipient of any user

authentication information sent from an Access Point (AP) to a SDN controller. There is one exception that would not need the support of EAP by the SDN controller. That is, extending OpenFlow to also support user information. In this case an access point needs to support OpenFlow standards as well so that the end system directly produces OpenFlow messages, includes user authentication information and submits them to the SDN controller. Since ONOS already supports OpenFlow protocol, the functions related to parsing OpenFlow messages need to be modified to support the extension so that it can retrieve user authentication information from OpenFlow messages. Furthermore a SDNauth parser is needed for communication of SDN controllers. SDNauth can provide a dynamic way for the communication of SDN controllers via the orchestrator layer.

In the following, SDNauth is described in more detail as a communication protocol. REST (Representational State Transfer) is widely adopted for communication among SDN solutions. However, for sending or receiving information, there needs to be a specific structure so that the receiver of a message can parse the information and knows what to do with it.

The idea of SDNauth is to only use the orchestrator as a coordinator of different components, similar to its name (such as applications to controllers). Therefore, all requests from a SDN controller are encapsulated into a different message, that is exemplary show in this example:

```
<?xml version=”1.0” encoding =”UTF-8”?>
<tag>radius</tag>
<domain>tenant1</domain>
<data>the encapsulated protocol information</data>
```

Thus, the orchestrator only parses this request and submits it to the responsible App in the related domain. It finds the responsible App information by querying the domain name (for example, app1.tenant1).

To allow an administrator of each domain to know the purpose of each app, information about the App can be stored in a TXT resource record of the domain. For example, a radius tag can be stored in TXT resource record of app1.tenant1 domain. For example, for a user authentication scenario during user movements and for a seamless authentication, the DNS servers 101 of each operator need to store the domain name of the authenticator App. Thereby, the orchestrator can easily query the DNS server 101 and find the right information for the responsible App that can parse the SDNauth message of a particular user. When the orchestrator finds this App, it forwards this SDNauth request to the app.

Fig. 17 illustrates this process. For example, when a user of operator1 chooses an access point in its visited network that belongs to operator2, the user's device uses the EAP protocol to submit the session re-association request. This request is forwarded to a SDN controller via a OpenFlow Switch. The SDN controller converts this request to RADIUS protocol and then encapsulates it into SDNauth REST structure. Then the SDN controller submits the request to its orchestrator, which is orch1.operator2. orch1.operator2 queries the DNS server in its own network for A or AAAA resource record of the domain name app1.tenant1. DNS server responds to this query by sending back the IP address of app1.tenant1. Orch1.operator2 forwards SDNauth message to app1.tenant1. After app1.tenant1 authenticates the orch1.operator2, it

processes this request and authenticates the user. Then it submits the user authorization information and any accounting information back to the orch1.operator2. Orch1.operator2 processes the authentication of app1.tenant1 and retrieves the PP for this App from the DNS server and then queries the resource policy for actual authorization information. Then this information is returned to the SDN controller. The SDN controller applies the rules on the switches and devices in the network so that the user can start using the network and has the same authorization as it had in its home network.

Policy information can be regarded as a means of access control. A policy information database includes all detailed policy information, e.g. about how a switch can be accessed and how it behaves, who can add what rules on the switch, etc. Usually all vendors have a default implementation of this database running in their data center, that can be re-used for storing policy information. The most important thing that should be considered with the resource policy database is that there need to be different levels of policies and all policies are categorized in different group. This can be done by a tree structure format where the root level in this tree structure is the policy information template. The indexes of the policy information template are then kept in the DNS server to provide means of identifying policy information.

However, the DNS based PKI model presented can be used in general computer networking use cases, especially when providing automatic TLSA and key update for a PKI system. The DNS based PKI model can in particular be applied in a SDN/NFV based architecture.

The presented solution hence shows, amongst others, the following features and advantages over the prior art:

- Key management without the need for a local or public CA is provided.
  Further, identification of network entities based on their domains and resource isolation based on the domains is provided by assigning parent resource policies to domains and sub-domains. Parent resource policies can also be referred to as policy information. Hence, DNS is used in a new way to separate and isolate resources in a network, e.g. to separate customers or tenants. In particular, a zone is what makes this isolation of resource records.
- Tenants of the network of an operator can access and control their own domain with the use of existing resources (using the DNS protocol).

The solution allows each customer or tenant in multi-tenant environments to access its own zone information and to update the cryptographic keys used for its own communication entities, e.g. apps. Access control information for a zone is stored in the same place as domain information and a public cryptographic key. Thereby an advantage over existing PKI models is provided, which do not allow each customer to control their own resources and have to involve an administrator to include new cryptographic keys for communication entities e.g. for new Apps.

Also an extended DDNS method is provided that can be used for automatic update of cryptographic keys and TLSA information in the DNS zone files. In addition, communication entities are distinguishable easily and quickly by their domain names that map to a zone record. This solves the problem of the prior art, where in

existing PKI models, if operator 1s cryptographic key is compromised, all customers are influenced by this problem.

The solution further solves the problem of the prior art how to isolate the resources of communication entities, how to assign policy information to each communication entity and how to identify each communication entity easily and quickly without delay. This is achieved by using domains and zones. Each communication entity carries its own domain which is a way to identify this communication entity. Logical isolation is also addressed by using extended DANE for authentication and high level authorization. The Automatic cryptographic key update in a DNS based PKI Model is performed by introducing extensions to the DDNS protocol.

User movement and user authentication during movement (multi-domain authentication & authorization) is improved by the integration of existing protocols with SDNauth. Therefore, a public cryptographic key is used for authentication purposes during DDNS process and for updating TLSA records, domain names and other resource records. A TLSA record is used during TLS based authentication process to associate a domain with its certificates.

The invention has been described in conjunction with various embodiments herein. However, other variations to the enclosed embodiments can be understood and effected by those skilled in the art and practicing the claimed invention, from a study of the drawings, the disclosure and the appended claims. In the claims, the word "comprising" does not exclude other elements or steps, and the indefinite article "a" or "an" does not exclude a plurality. A single processor or other unit may fulfill

the functions of several items recited in the claims. The mere fact that certain measures are recited in mutually different dependent claims does not indicate that a combination of these measures cannot be used to advantage. A computer program may be stored/distributed on a suitable medium, such as an optical storage medium or a solid-state medium supplied together with or as part of other hardware, but may also be distributed in other forms, such as via the internet or other wired or wireless telecommunication systems.

Claims

1. Data processing system (100) comprising:
a DNS server (101, 401, 501, 601) managing a domain name service database (102), wherein the DNS server (101, 401, 501, 601) is configured to communicate with a plurality of communication entities (103, 104) and wherein the domain name service database (102) is configured to store resource records (105, 106) and to receive requests,
characterized in that,
upon receiving an update request from a first communication entity (103, 104), the DNS server (101, 401, 501, 601) is configured to create, change and/or delete at least a first resource record (105, 106) and at least one public cryptographic key (107, 108) and/or a TLSA record (109, 110) for the first communication entity (103, 104) in the domain name service database (102).

2. Data processing system according to claim 1, wherein the DNS server (101, 401, 501, 601) is configured to create, change and/or delete the first resource record (105, 106) in the domain name service database (102), in case the update request is cryptographically signed with a private cryptographic key or the update request is encrypted with a session key or a private cryptographic key.

3. Data processing system according to claim 1 or 2, wherein the session key is encrypted by an asymmetric cryptographic algorithm, and wherein the private cryptographic key is a counterpart to at least one public cryptographic key (107, 108) stored in the domain name service database (102).

4. Data processing system according to any one of the preceding claims, wherein the DNS server (101, 401, 501, 601) is configured to create, change and/or delete a second resource record (105, 106) and a second public cryptographic key (107, 108) and/or a TLSA record (109, 110)in the domain name service database (102) upon receiving an update request from a second communication entity (103, 104), and in case the update request is cryptographically signed with the private cryptographic key of the first communication entity (103, 104).

5. Data processing system according to any one of the preceding claims, wherein the DNS server (101, 401, 501, 601) is configured to create, change and/or delete the second resource record (105, 106) and/or the second public cryptographic key (107, 108) and/or the TLSA record (109, 110) in the domain name service database (102) upon receiving an update request from the second communication entity (103, 104) and in case the update request is cryptographically signed with a second private cryptographic key of the second communication entity (103, 104), the second private cryptographic key being a counterpart to the second public cryptographic key (107, 108) and/or a TLSA record (109, 110) stored in the domain name service database (102) before the DNS server (101, 401, 501, 601) processes an update of the domain name service database (102) according to the update request.

6. Data processing system according to any one of the preceding claims, wherein the DNS server (101, 401, 501, 601) is configured to store in the domain name service database (102) at least one policy

identification information in association with the first resource record (105, 106) created for the first communication entity (103, 104), and wherein the DNS server (101, 401, 501, 601) is configured to store, in the domain name service database (102), a subset of the policy identification information stored for the resource record (105, 106) created for the first communication entity (103, 104), with the second resource record (105, 106) created for the second communication entity (103, 104).

7. Data processing system according to claim 6, wherein the policy identification information and/or subset of the policy identification information is created, changed and/or deleted by the update request.

8. Data processing system according to any one of the preceding claims, wherein the data processing system is an SDN system, further comprising an orchestrator module, a SDN controller and network elements, and wherein the second communication entity (103, 104) is an SDN application operating in a zone defined by the first communication entity (103, 104).

9. Data processing system according to claim 8, wherein the second communication entity (103, 104) sends an update request to the SDN controller at which the update request is received by the orchestrator module.

10. Data processing system according to claim 8, wherein the orchestrator module forwards the update request of the second communication entity (103, 104) to the DNS server (101, 401, 501, 601), wherein the DNS server (101, 401, 501, 601) is configured to return policy identification information, a public

cryptographic key (107, 108) and a TLSA record (109, 110) for the second communication entity (103, 104), and wherein the orchestrator module is configured to verify whether the returned public cryptographic key (107, 108) belongs to the private key with which the update request was signed.

11. Data processing system according to claim 8, wherein the second communication entity (103, 104) sends a resource request to access a network element to the SDN controller at which the resource request is received by the orchestrator module.

12. Data processing system according to claim 8, wherein the orchestrator module forwards the resource request of the second communication entity (103, 104) to the DNS server (101, 401, 501, 601), wherein the DNS server (101, 401, 501, 601) is configured to return policy identification information and/or a TLSA record (109, 110) information for the second communication entity (103, 104), and wherein the orchestrator module is configured to verify whether the returned information belongs to a TLS certificate exchanged during the establishment of a TLS session with the second communication entity (103, 104).

13. Data processing system according to claim 8, wherein the data processing system further comprises a resource policy database and wherein the orchestrator module queries the resource policy database to find policy information based on the policy identification information returned by the DNS server (101, 401, 501, 601).

14. Data processing system according to any one of the preceding claims, wherein each resource record (105, 106) is unique.

15. Method for operating a data processing system comprising the steps of:
managing a domain name service database (102) by a DNS server (101, 401, 501, 601), wherein the DNS server (101, 401, 501, 601) communicates with a plurality of communication entities (103, 104) and wherein the domain name service database (102) stores resource records (105, 106) and receives requests, characterized by,
upon receiving a request from a first communication entity (103, 104), the DNS server (101, 401, 501, 601) creates, changes and/or deletes at least a first resource record (105, 106) and at least one public cryptographic key (107, 108) and/or a TLSA record (109, 110) for the communication entity (103, 104) in the domain name service database (102).

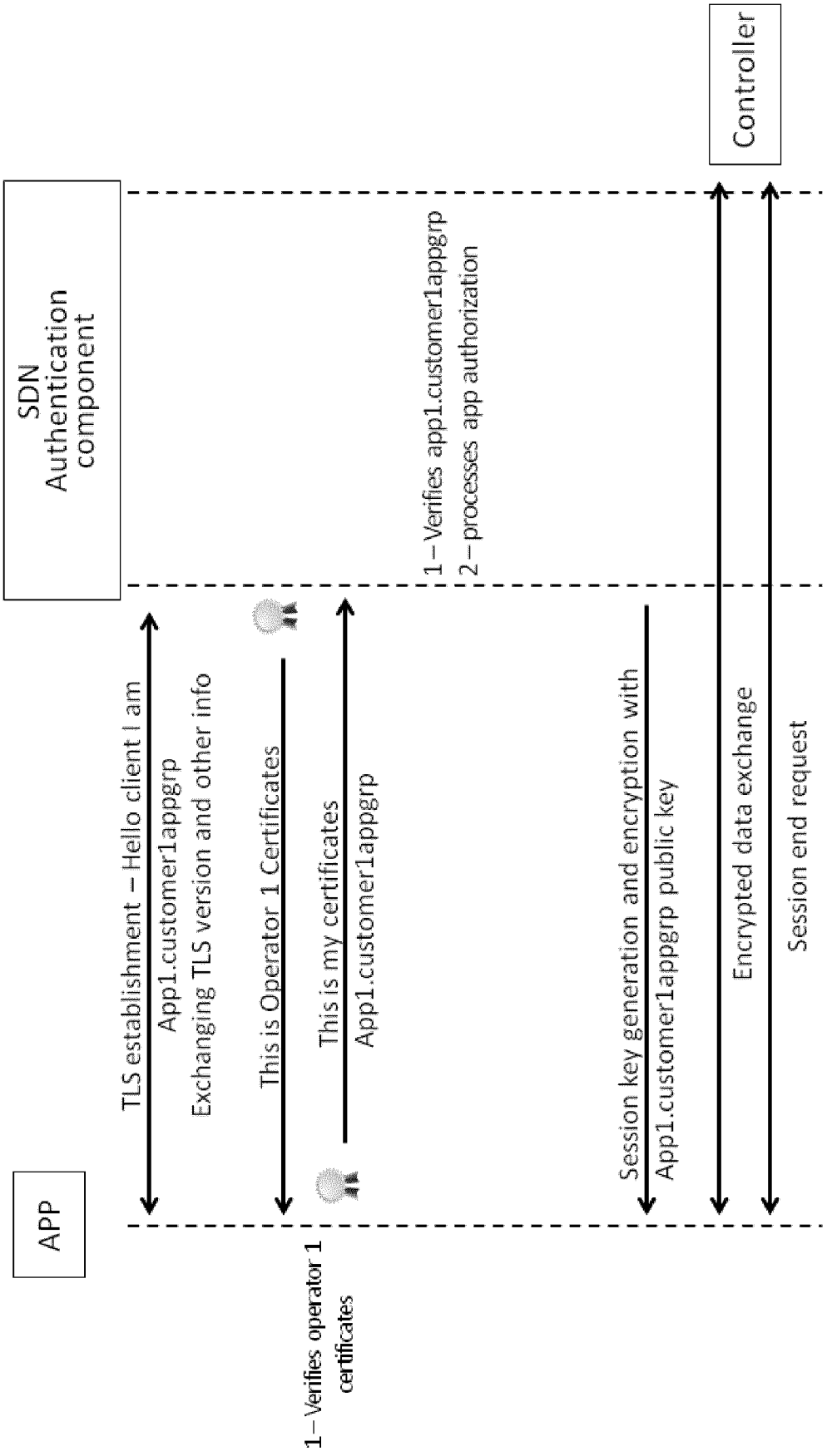
APP
SDN Authentication component
Controller
TLS establishment – Hello client I am App1.customer1appgrp Exchanging TLS version and other info
This is Operator 1 Certificates
This is my certificates App1.customer1appgrp
1 – Verifies app1.customer1appgrp
2 – processes app authorization
Session key generation and encryption with App1.customer1appgrp public key
Encrypted data exchange
Session end request
1 – Verifies operator 1 certificates

Fig. 1

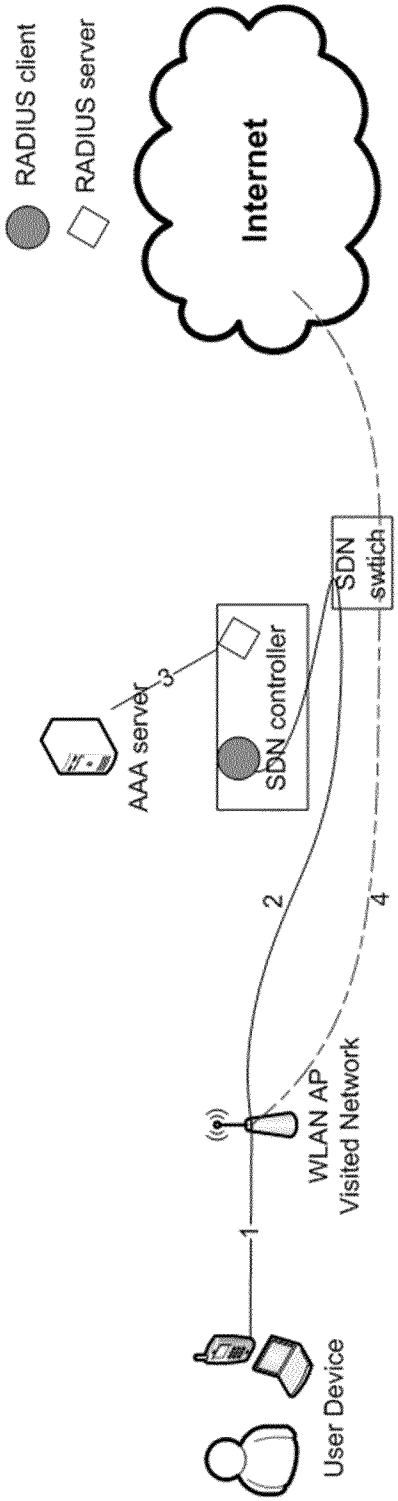
RADIUS client
RADIUS server
Internet
SDN switch
SDN controller
AAA server
3
2
4
1
WLAN AP
Visited Network
User Device

Fig. 2

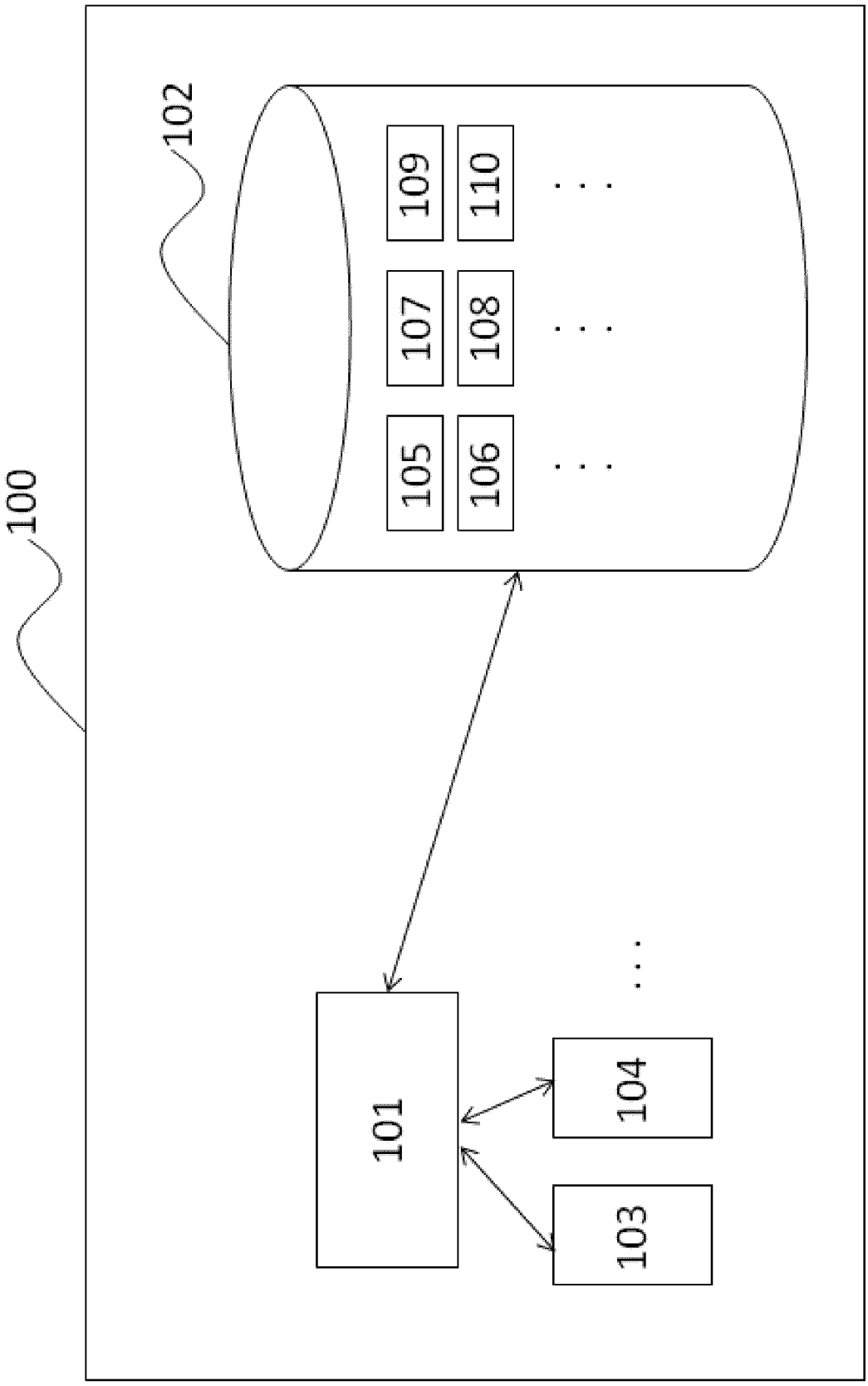

Fig. 3

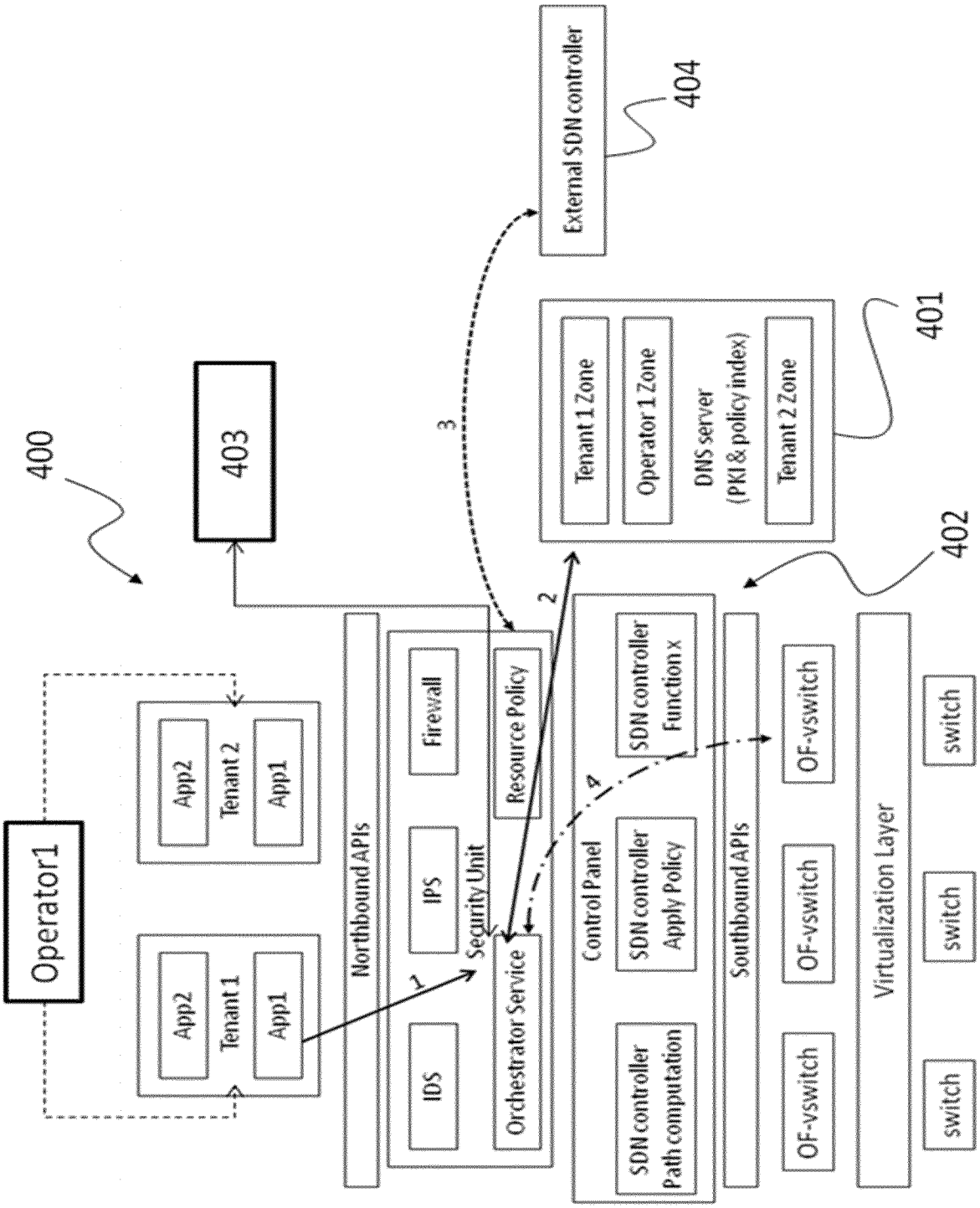
External SDN controller
404
401
Tenant 1 Zone
Operator 1 Zone
DNS server
(PKI & policy index)
Tenant 2 Zone
402
403
400
3
2
Operator1
App2
Tenant 2
App1
App2
Tenant 1
App1
Northbound APIs
Firewall
IPS
IDS
Security Unit
Resource Policy
Orchestrator Service
1
Control Panel
SDN controller Function x
SDN controller Apply Policy
SDN controller Path computation
Southbound APIs
OF-vswitch
OF-vswitch
OF-vswitch
Virtualization Layer
switch
switch
switch

Fig. 4

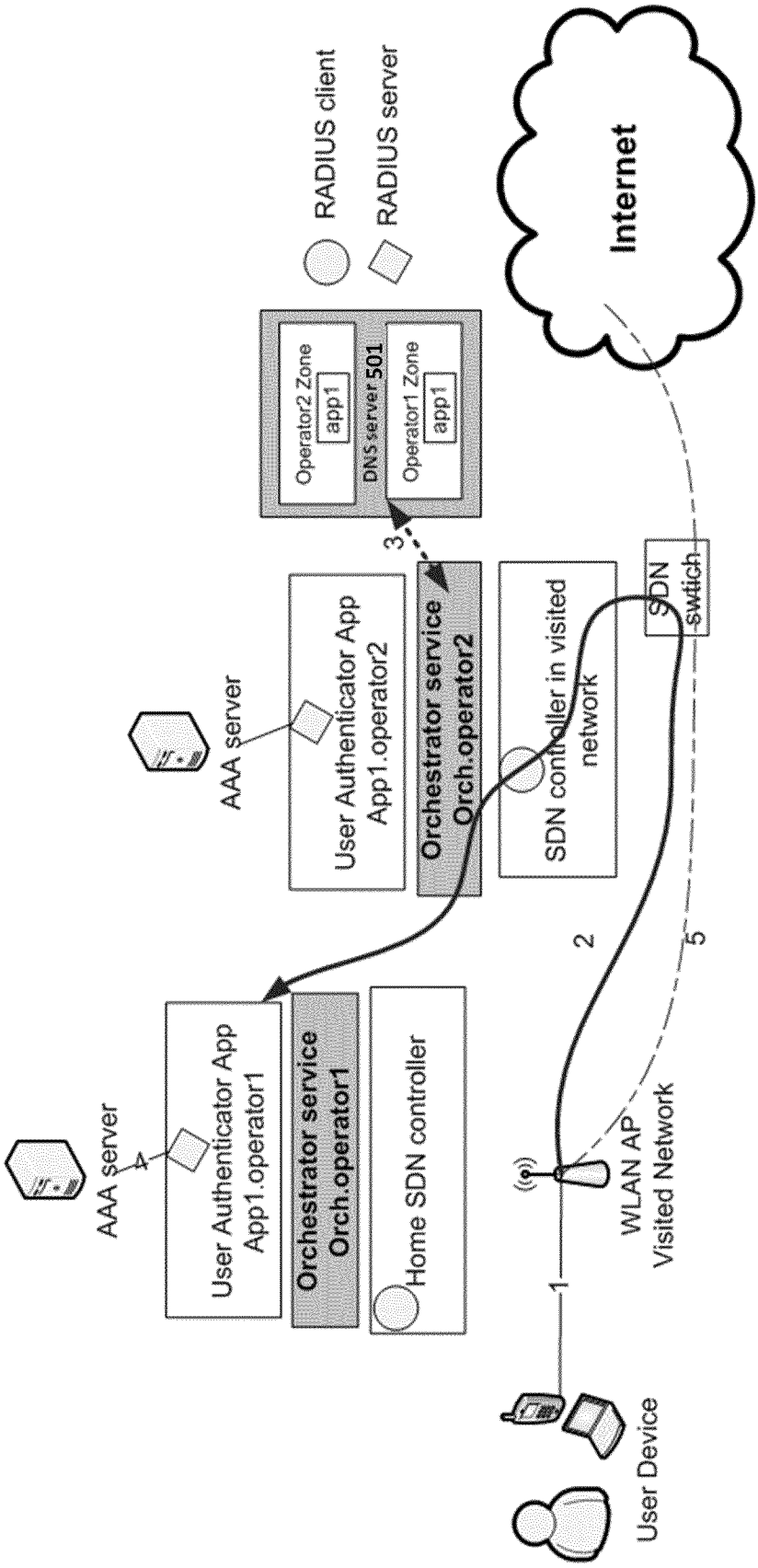
RADIUS client
RADIUS server
Internet
Operator2 Zone
app1
DNS server 501
Operator1 Zone
app1
3
AAA server
User Authenticator App
App1.operator2
Orchestrator service
Orch.operator2
SDN controller in visited network
SDN switch
2
5
AAA server
User Authenticator App
App1.operator1
Orchestrator service
Orch.operator1
Home SDN controller
1
WLAN AP
Visited Network
User Device

Fig. 5

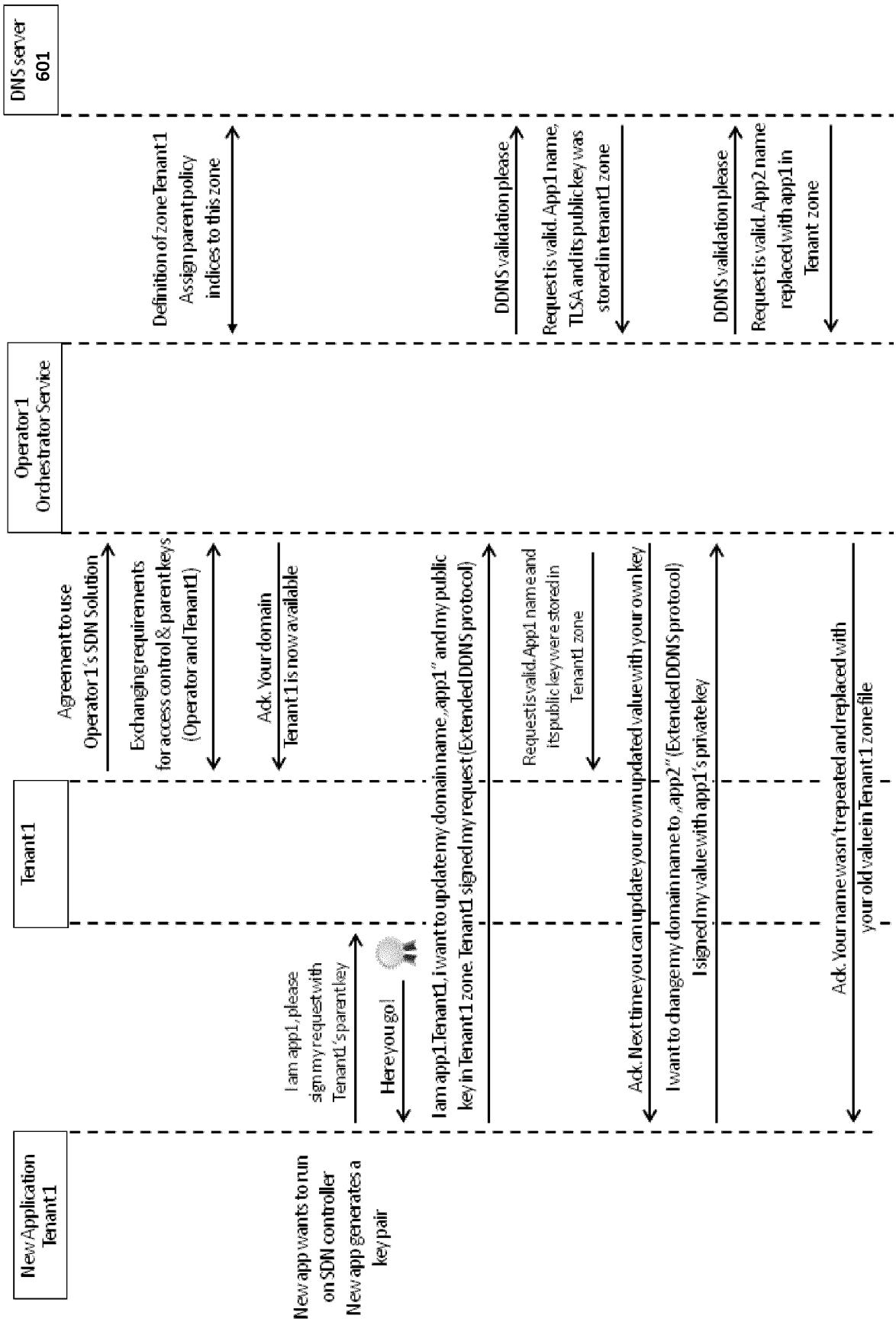
New Application Tenant1
Tenant1
Operator1 Orchestrator Service
DNS server 601
New app wants to run on SDN controller
New app generates a key pair
I am app1, please sign my request with Tenant1's parent key
Here you go!
Agreement to use Operator1's SDN Solution
Exchanging requirements for access control & parent keys (Operator and Tenant1)
Ack. Your domain Tenant1 is now available
Definition of zone Tenant1 Assign parent policy indices to this zone
I am app1.Tenant1, i want to update my domain name „app1" and my public key in Tenant1 zone. Tenant1 signed my request (Extended DDNS protocol)
DDNS validation please
Request is valid. App1 name and its public key were stored in Tenant1 zone
Request is valid. App1 name, TLSA and its public key was stored in tenant1 zone
Ack. Next time you can update your own updated value with your own key
I want to change my domain name to „app2" (Extended DDNS protocol) I signed my value with app1's private key
DDNS validation please
Request is valid. App2 name replaced with app1 in Tenant zone
Ack. Your name wasn't repeated and replaced with your old value in Tenant1 zone file

Fig. 6

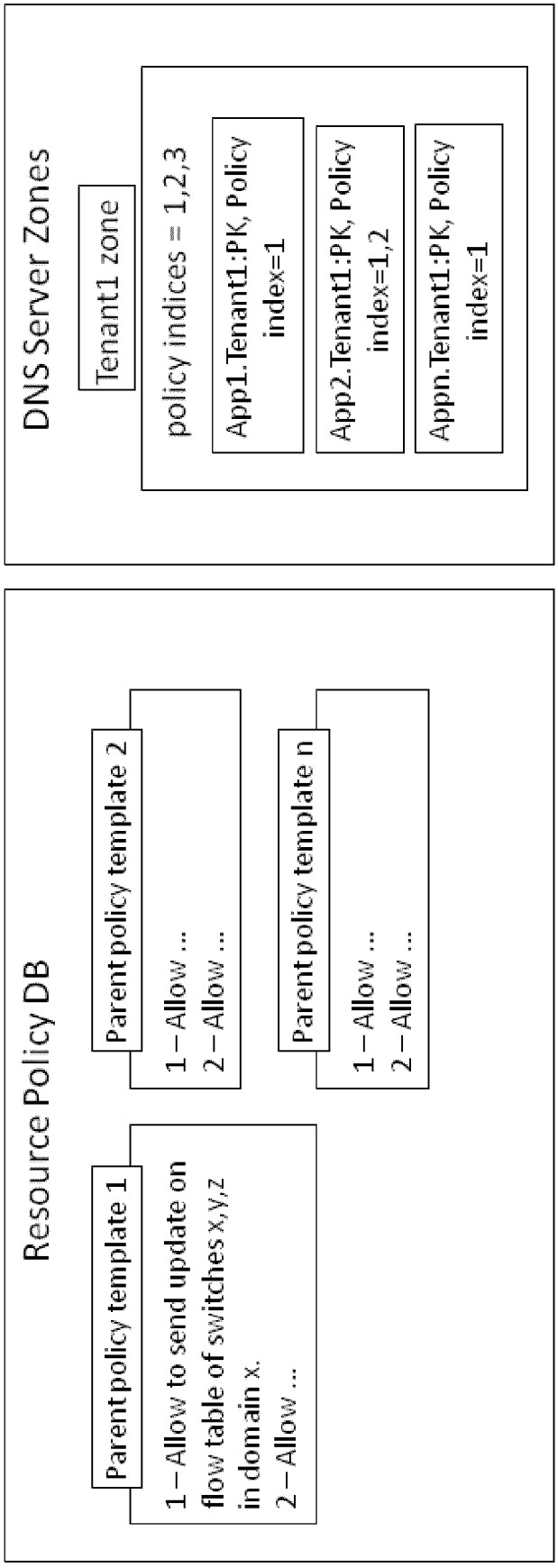
DNS Server Zones
Tenant1 zone
policy indices = 1,2,3
App1.Tenant1:PK, Policy index=1
App2.Tenant1:PK, Policy index=1,2
Appn.Tenant1:PK, Policy index=1
Resource Policy DB
Parent policy template 1
1 – Allow to send update on flow table of switches x,y,z in domain x.
2 – Allow ...
Parent policy template 2
1 – Allow ...
2 – Allow ...
Parent policy template n
1 – Allow ...
2 – Allow ...

Fig. 7

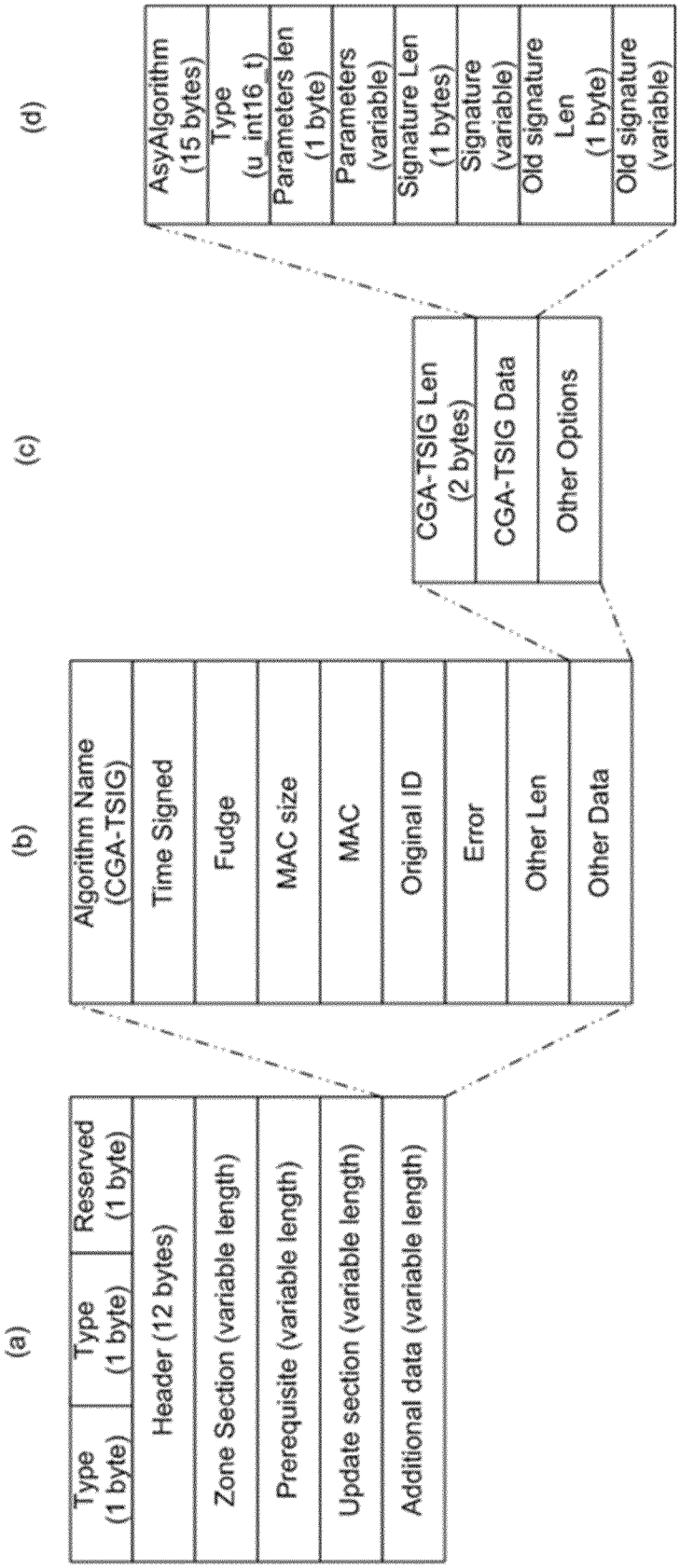
(a)
Type (1 byte)
Type (1 byte)
Reserved (1 byte)
Header (12 bytes)
Zone Section (variable length)
Prerequisite (variable length)
Update section (variable length)
Additional data (variable length)
(b)
Algorithm Name (CGA-TSIG)
Time Signed
Fudge
MAC size
MAC
Original ID
Error
Other Len
Other Data
(c)
CGA-TSIG Len (2 bytes)
CGA-TSIG Data
Other Options
(d)
AsyAlgorithm (15 bytes)
Type (u_int16_t)
Parameters len (1 byte)
Parameters (variable)
Signature Len (1 bytes)
Signature (variable)
Old signature Len (1 byte)
Old signature (variable)

Fig. 8

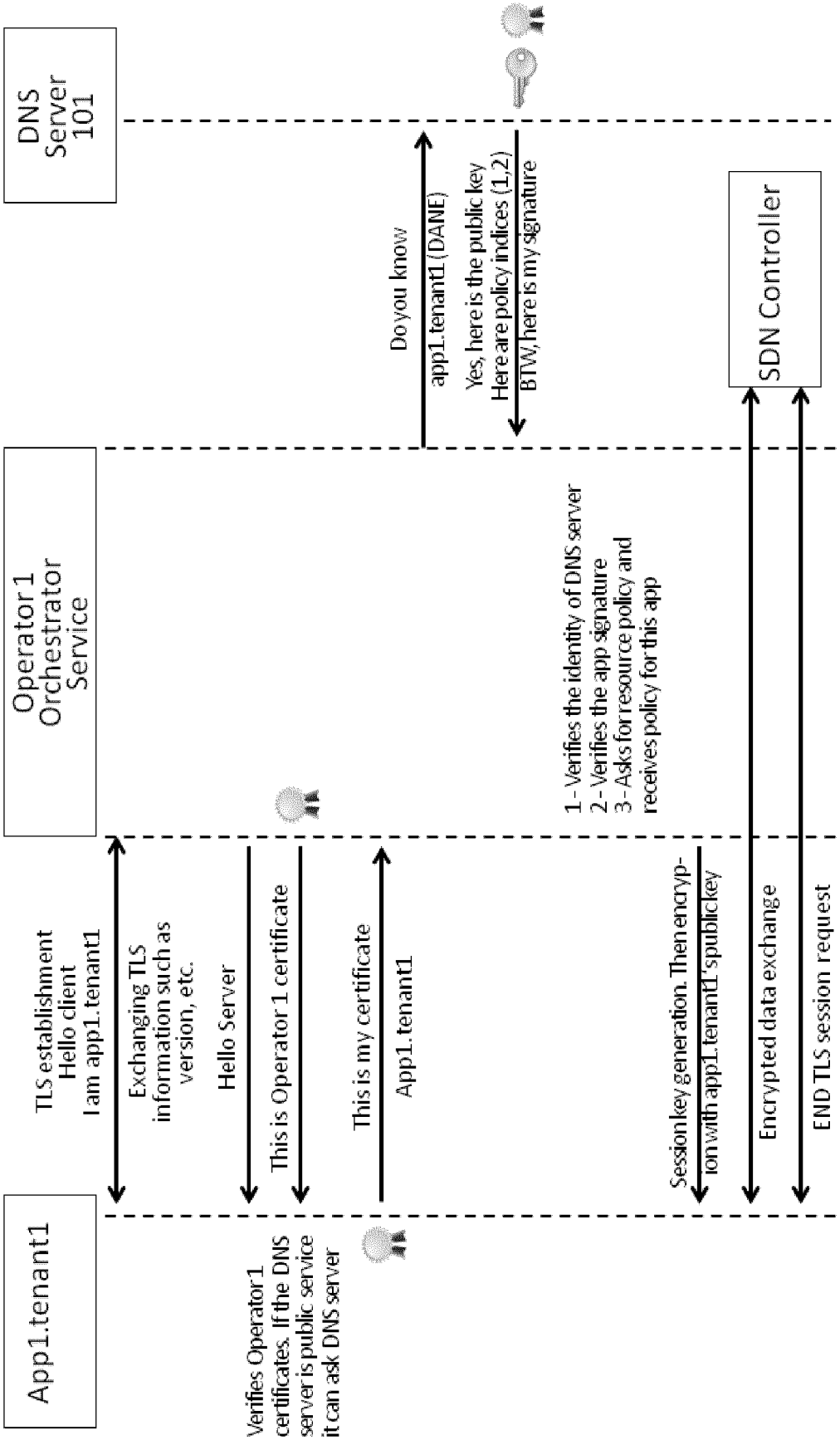
App1.tenant1
Operator 1 Orchestrator Service
DNS Server 101
SDN Controller
TLS establishment Hello client I am app1.tenant1
Exchanging TLS information such as version, etc.
Hello Server
This is Operator 1 certificate
This is my certificate
App1.tenant1
Verifies Operator 1 certificates. If the DNS server is public service it can ask DNS server
Do you know app1.tenant1 (DANE)
Yes, here is the public key
Here are policy indices (1,2)
BTW, here is my signature
1 - Verifies the identity of DNS server
2 - Verifies the app signature
3 - Asks for resource policy and receives policy for this app
Session key generation. Then encryption with app1.tenant1's public key
Encrypted data exchange
END TLS session request

Fig. 9

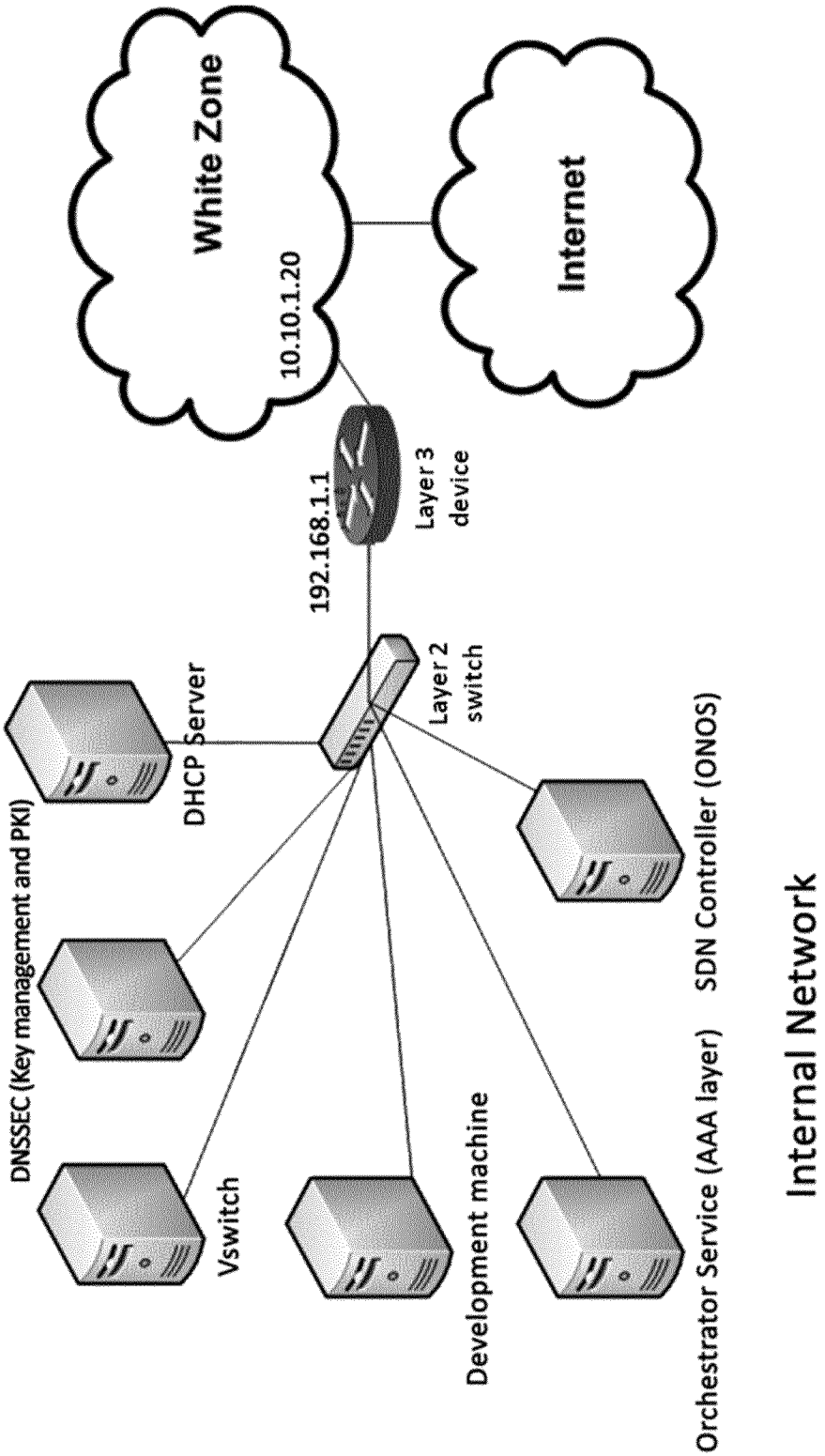
White Zone
10.10.1.20
Internet
192.168.1.1
Layer 3 device
Layer 2 switch
DHCP Server
DNSSEC (Key management and PKI)
Vswitch
Development machine
Orchestrator Service (AAA layer)
SDN Controller (ONOS)
Internal Network

Fig. 10

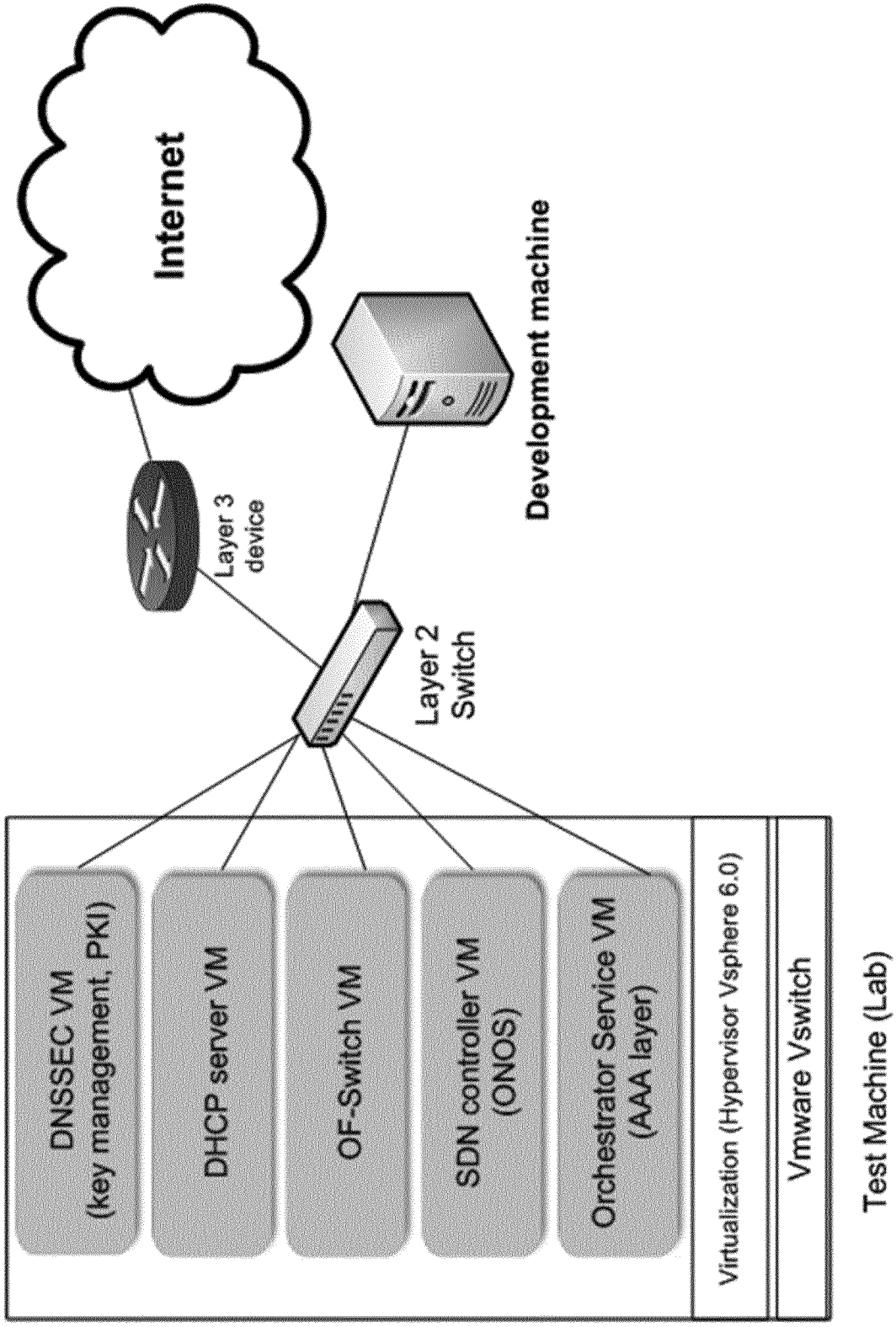
Internet
Development machine
Layer 3 device
Layer 2 Switch
DNSSEC VM (key management, PKI)
DHCP server VM
OF-Switch VM
SDN controller VM (ONOS)
Orchestrator Service VM (AAA layer)
Virtualization (Hypervisor Vsphere 6.0)
Vmware Vswitch
Test Machine (Lab)

Fig. 11

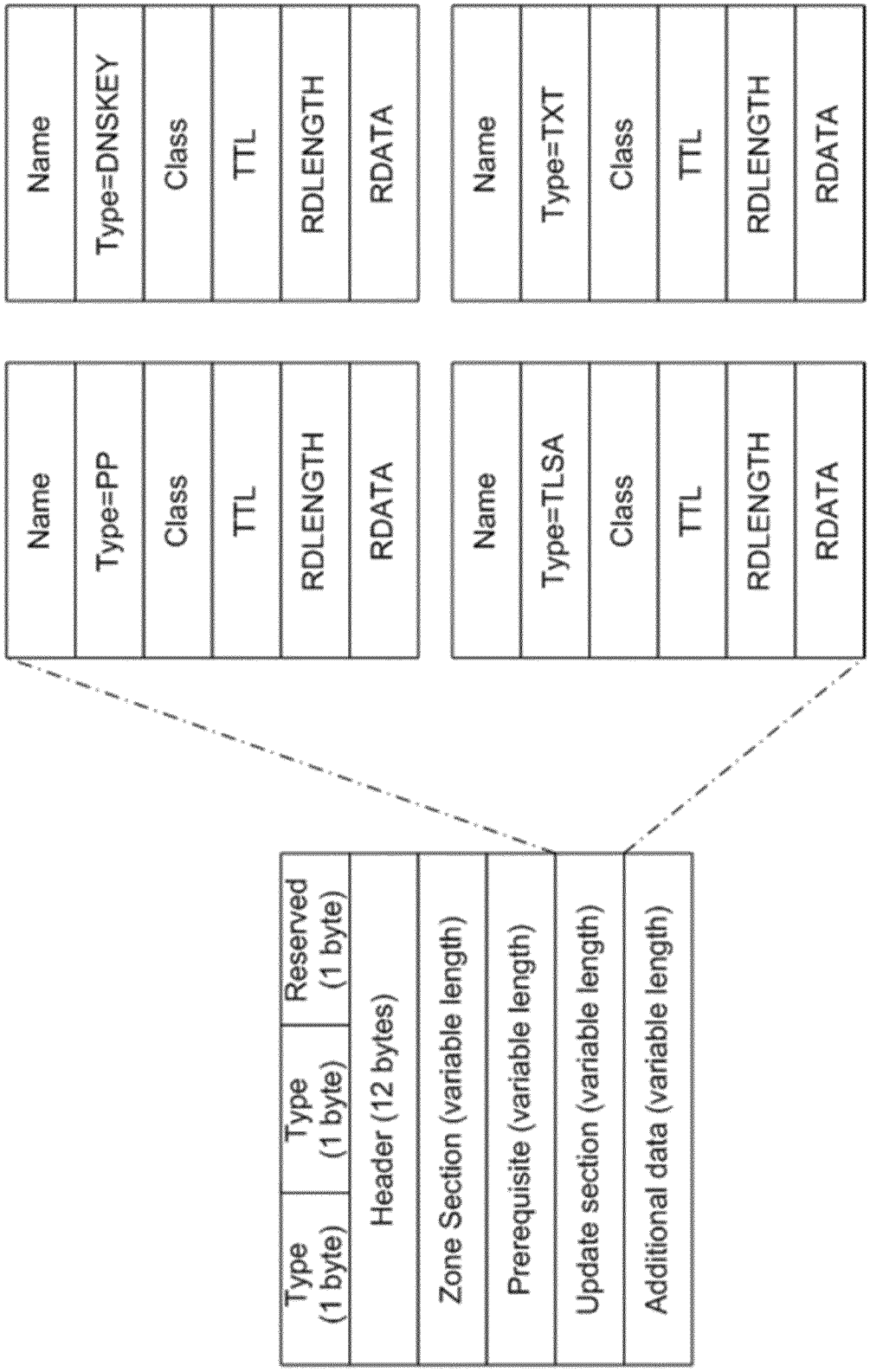
Name
Type=DNSKEY
Class
TTL
RDLENGTH
RDATA
Name
Type=TXT
Class
TTL
RDLENGTH
RDATA
Name
Type=PP
Class
TTL
RDLENGTH
RDATA
Name
Type=TLSA
Class
TTL
RDLENGTH
RDATA
Type (1 byte)
Type (1 byte)
Reserved (1 byte)
Header (12 bytes)
Zone Section (variable length)
Prerequisite (variable length)
Update section (variable length)
Additional data (variable length)

Fig. 12

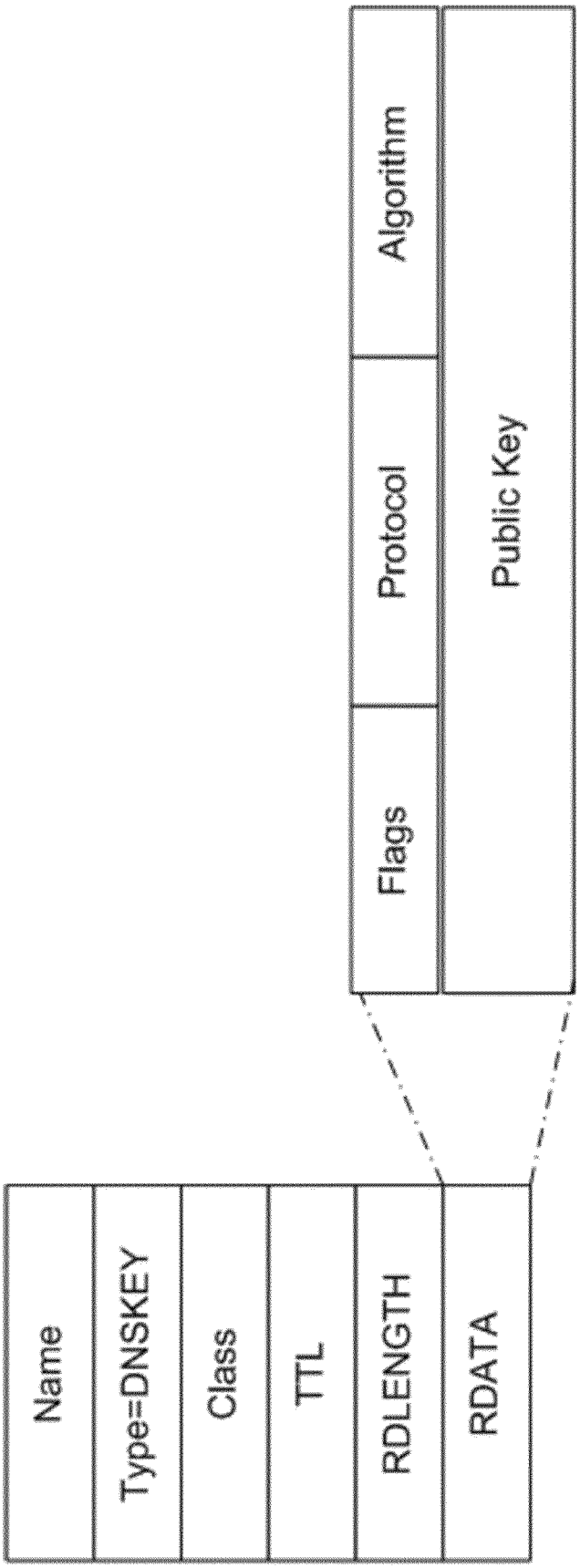
Name
Type=DNSKEY
Class
TTL
RDLENGTH
RDATA
Flags
Protocol
Algorithm
Public Key

Fig. 13

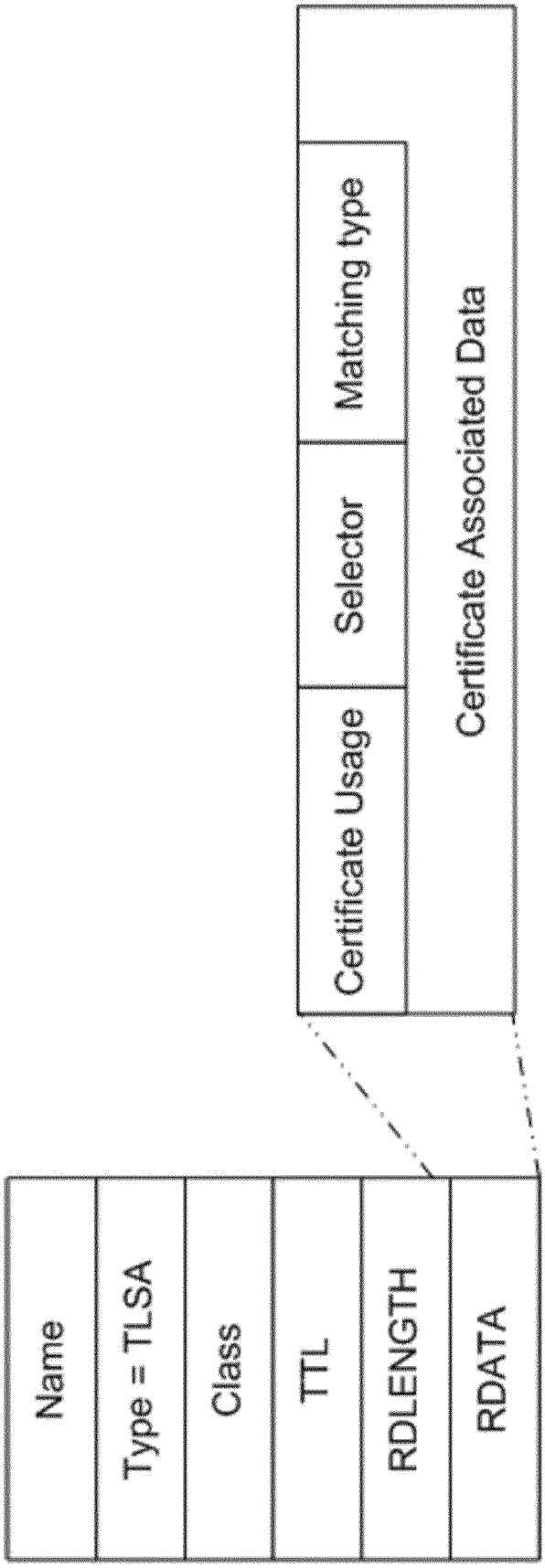
Name
Type = TLSA
Class
TTL
RDLENGTH
RDATA
Certificate Usage
Selector
Matching type
Certificate Associated Data

Fig. 14

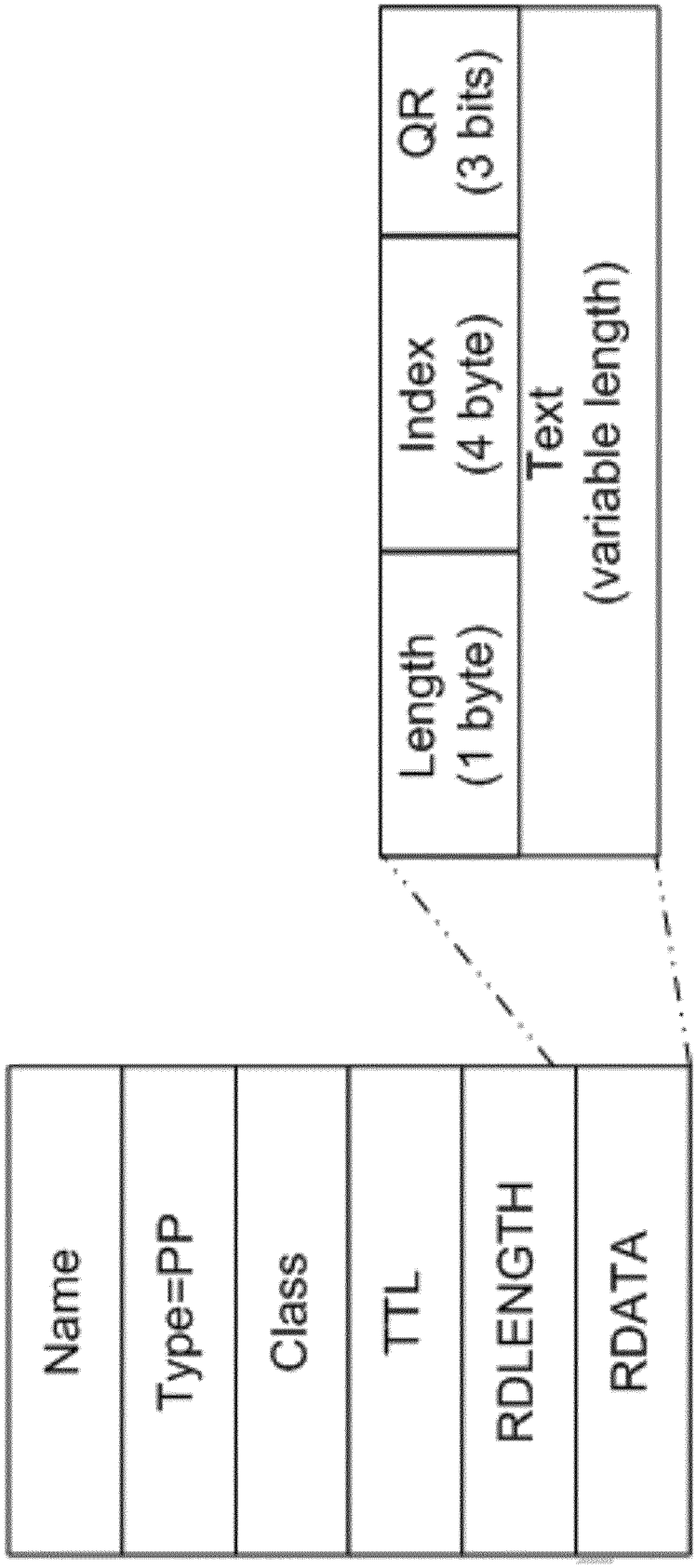
Name
Type=PP
Class
TTL
RDLENGTH
RDATA
Length (1 byte)
Index (4 byte)
QR (3 bits)
Text (variable length)

Fig. 15

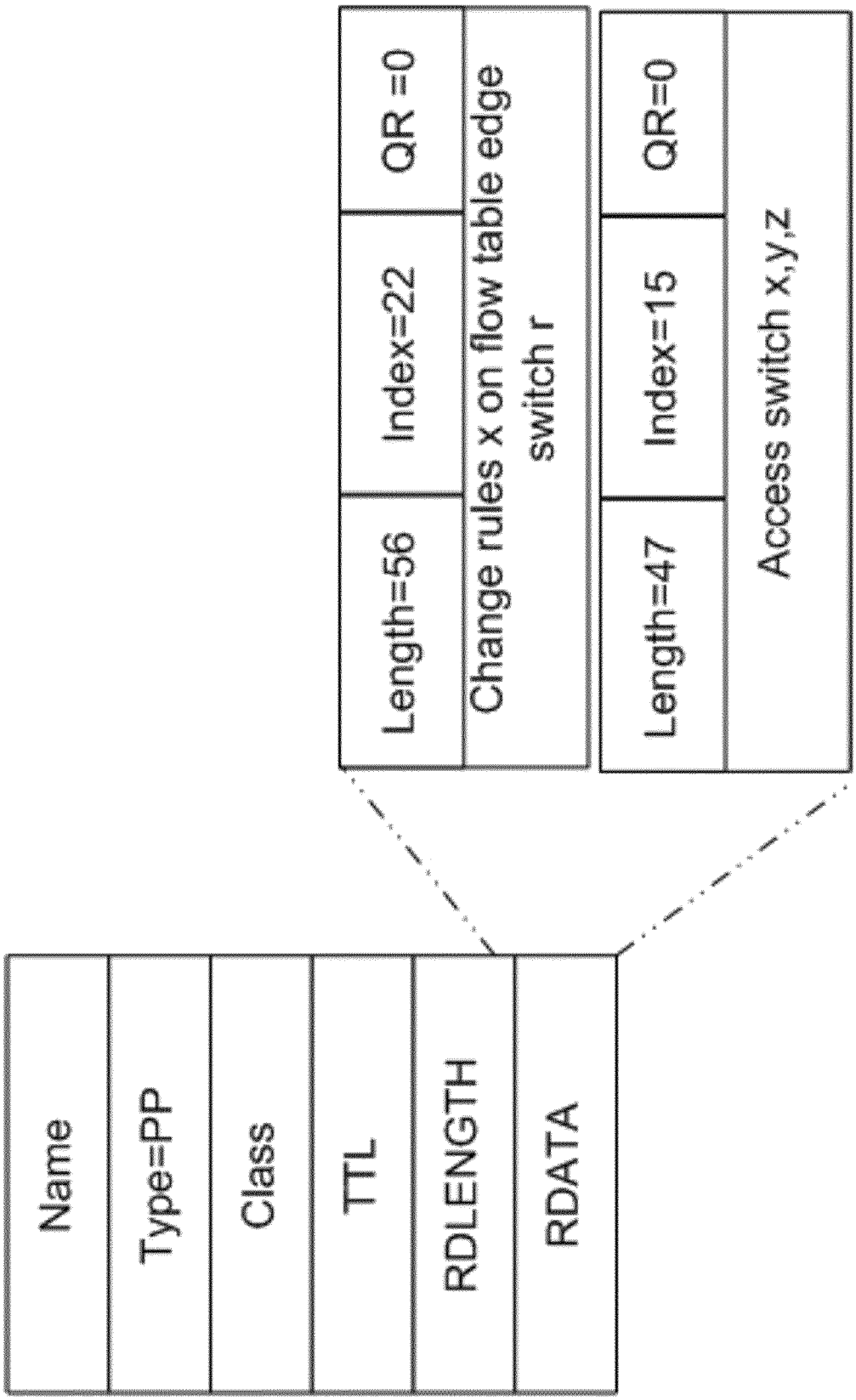
Name
Type=PP
Class
TTL
RDLENGTH
RDATA
Length=56
Index=22
QR =0
Change rules x on flow table edge switch r
Length=47
Index=15
QR=0
Access switch x,y,z

Fig. 16

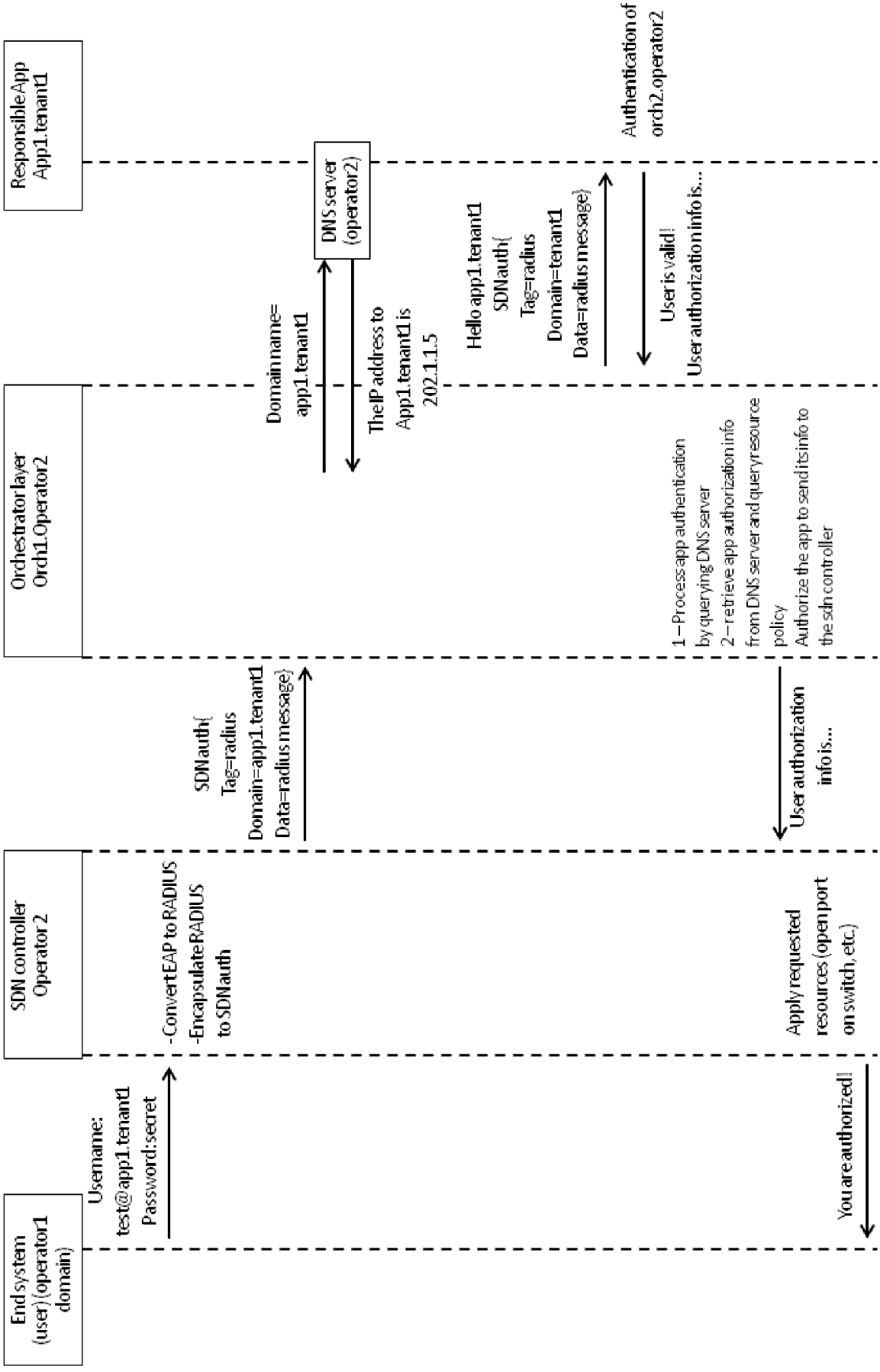
End system (user) (operator1 domain)
SDN controller Operator 2
Orchestrator layer Orch1.Operator2
Responsible App App1.tenant1
Username: test@app1.tenant1 Password:secret
-Convert EAP to RADIUS -Encapsulate RADIUS to SDNauth
SDNauth{ Tag=radius Domain=app1.tenant1 Data=radius message}
Domain name= app1.tenant1
DNS server (operator2)
The IP address to App1.tenant1 is 202.1.1.5
Hello app1.tenant1 SDNauth{ Tag=radius Domain=tenant1 Data=radius message}
Authentication of orch2.operator2
User is valid! User authorization info is...
1–Process app authentication by querying DNS server
2–retrieve app authorization info from DNS server and query resource policy
Authorize the app to send its info to the sdn controller
User authorization info is...
Apply requested resources (open port on switch, etc.)
You are authorized!

Fig. 17

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2015/063763

A. CLASSIFICATION OF SUBJECT MATTER
INV. H04L29/06 H04L9/08 H04L29/12 H04W12/04
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
H04L H04W

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, COMPENDEX, INSPEC, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2013/262860 A1 (LAMB RICHARD [US]) 3 October 2013 (2013-10-03) the whole document	1-15
A	EP 2 518 970 A1 (VERISIGN INC [US]) 31 October 2012 (2012-10-31) abstract paragraph [0023] - paragraph [0024] paragraph [0055] - paragraph [0060] paragraph [0082] - paragraph [0106] claim 1	1-15

-/--

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
5 February 2016	12/02/2016

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Lebas, Yves

2

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	"DFN Mitteilungen Ausgabe 88 ¦ Mai 2015 In fünf Schritten in die DFN-Cloud X-WiN - GÉANT - SINET: Globales VPN für deutsch-japanische Weltraum-Mission Erkennung und Bearbeitung von DDoS-Angriffen im X-WiN", , 30 May 2015 (2015-05-30), XP055247917, Retrieved from the Internet: URL:https://www.dfn.de/fileadmin/5Presse/DFNMitteilungen/DFN_Mitteilungen_88.pdf [retrieved on 2016-02-05] page 22 - page 28 -----	1-15

2

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2015/063763

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2013262860 A1	03-10-2013	NONE	
EP 2518970 A1	31-10-2012	AU 2012202458 A1	15-11-2012
		BR 102012010250 A2	04-02-2014
		CA 2775560 A1	29-10-2012
		EP 2518970 A1	31-10-2012
		JP 2012235462 A	29-11-2012
		US 2012278626 A1	01-11-2012