HAL
archives-ouvertes.fr

Secure XML-based Network Management in a Multi-source Context

Vincent Cridlig, Radu State, Olivier Festor

▶ **To cite this version:**

Vincent Cridlig, Radu State, Olivier Festor. Secure XML-based Network Management in a Multi-source Context. [Intern report] A04-R-080 || cridlig04a, 2004, 12 p. <inria-00107808>

HAL Id: inria-00107808
https://hal.inria.fr/inria-00107808

Submitted on 19 Oct 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Secure XML-based network management in a multi-source context

V. Cridlig, R. State, O. Festor
LORIA - INRIA Lorraine
615, rue du jardin botanique
54602 Villers-les-Nancy, France
{*cridligv*|*state*|*festor*}*@loria.fr*

Abstract

Mettre l'abstract ici.

Keywords

network management, netconf, security, xml, multi-source

1. Introduction

IP Network Management is most often built on a classical manager/agent model. A manager queries the set of agents under its responsability to get status information and sometimes set parameters on it. In the hierarchical management approach, the network is divided into domains and sub-domains. Each manager is in charge of a sub-domain and reports the events and operations to a master manager.

The first motivation of this paper is to introduce a decentralized management approach where several configuration providers (CP) can manage several devices (EQ). For instance, a configuration at one agent can be made of configuration parts coming from many sources. This needs a minimal organization in particular for security and efficiency reasons. In particular, only authorized entities must be able to access to management data and thus be given the right credentials to perform some operations. Our approach is driven by an efficiency effort with large P2P type functionality.

Second, many XML-based management solutions have emerged over the past few years. XML-native approach like Junoscript and Netconf or hybrid approach like XML/SNMP gateways are representative of an effort to use XML to encode management data over the networks. XML, whose main quality is to be both human and machin-readable, makes it possible to model and organize data hierarchically, to describe data in a formal way with XML Schemas, to exchange easily data over different platforms, to handle data efficiently with a very large panel of available tools (XSLT, SAX, DOM). XML can also be used in network management for these reasons. Moreover XML now integrates security features like digital signature [2] or encryption [7]. This allows the definition of customizable, fine-grained, ie at XML element level, security policies. Network management did not yet address and propose a security architecture for XML based management. A security architecture is expected to provide authentication, confidentiality and access control while remaining scalable and being integrable in existing modules.

Lastly, this multi-provider context suits well for a distributed firewalls configuration. One firewall protecting the internal network can push a set of rules onto the frontal firewall

dynamically when it receives abnormal traffic. This process can be reversed if the frontal firewall wants to configure the filtering rules of the internal firewall. In a multi-homed environnement, a secure configuration between proxies can be thought of.

Our contribution addresses a security framework for large-scale P2P type XML-based management architecture.

The remainder of this paper is organized as follows. Section 2 gives an overview of our approach. Section 3 instantiates our security model through the description of an example. Section 4 presents some related works. Section 5 summarizes our approach and presents some future works.

2. Contribution

2.1 Architecture

Our secure model for Network configuration is architectured around several main entities depicted on Figure 1. While configuration providers act like a data configuration source, the managed devices run a Netconf agent which receives RPC requests from the different configuration provider. For instance, CP_a and CP_b can load different partial configurations onto an agent configuration which we then call a multi-source configuration. These configurations are loaded relying on different access rights meaning that an agent must be able to decrypt incoming configurations and to bind them some access rights. We will describe in this section the mechanisms needed to perform these requirements.

An RBAC manager is responsible for security deploiement among configuration provider and managed devices. This entity provides the different security services: authentication, data integrity, confidentiality and access control. The RBAC manager hosts an RBAC policy which is dynamically deployed on our network entities.

RBAC allows high level and scalable access control process and configuration. The RBAC model consists in a set of users, roles, permissions (operations on resources) and sessions. The originality of RBAC model is that permissions are not granted to users but to roles, thus allowing an easy reconfiguration when a user changes his activity. A role describes a job function within an organization. The permission to role relationships illustrates that a role can perform a set of operations on objects (privileges). In the same way, the user to role relationships describes the available function a user is allowed to endorse in the organization. Lastly, a session gathers for each user his set of currently activated role, on which behalf he can interact with the system. In this paper, we consider the NIST (National Institute of Standards and Technologies) RBAC (Role-Based Access Control) model which gathers the most commonly admitted ideas and experience of previous RBAC models. This standard provides a full description of all the features that should implement an RBAC system. RBAC model allows the description of complex authorization policies while reducing errors and costs of administration. Introduction of administrative roles and role hierarchy made it possible to reduce considerably the amount of associations representing permissions to users allocation.

In order to perform authentication, the RBAC manager communicates with a Public Key Infrastructure (PKI) which delivers certificates to configuration provider and man-

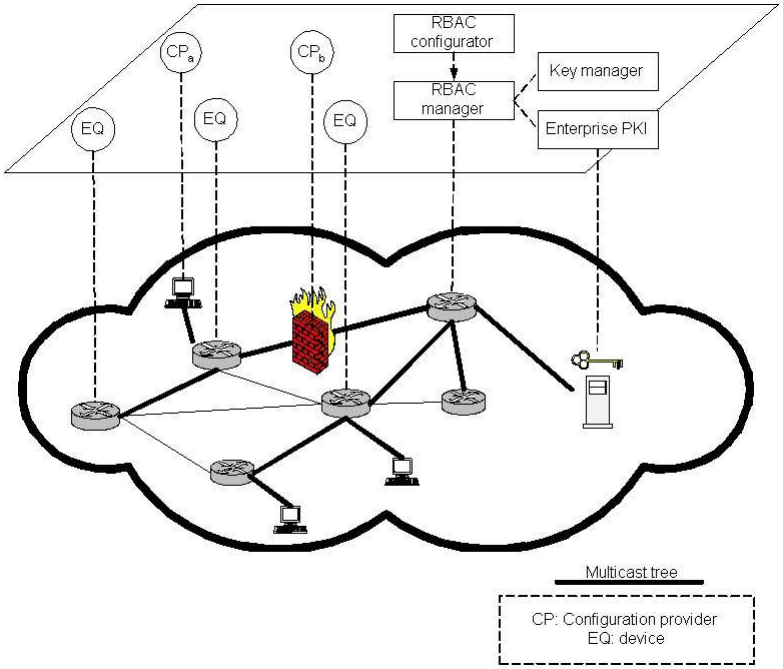

Figure 1: Secure management architecture

aged device. Thus, we assume that all of them (configuration provider and managed device) own a public/private key pair to provide a strong authentication among them. An entity authenticates itself on the RBAC manager by sending a message signed with its private key. Once the entity has been authenticated, the RBAC manager can distribute the keys for a particular role queried by the entity.

We propose here to bind each RBAC conceptual role to a specific key. This means that each entity who wants to perform an operation in the context of a role needs to upload this key. The RBAC manager is responsible for distributing these keys to the different entities. This way, only the entities who possess the role key are able to decrypt and encrypt the message for a particular role. Indeed, when an entity leaves a group, it should not be allowed to access future traffic. That is why we have to change the key group for each entity who owns this key. We call this set of entities a $KeyGroup_{role_i}$.

We give now some definitions to formally describe the credentials and sets used in our architecture:

- E: the set of entities of our architecture,
- $k^e_{public}/k^e_{private}$ where $e \in E$: denotes the public and private keys owned by device e,
- roles(e) where $e \in E$: the set of roles that appears at least on time in the ACL rules of the entity e. This represents all the keys needed by an agent to decrypt and encrypt data for a role. An ACL is bound to each node of the XML configuration in order to allow dynamic access control. An ACL is an XML node containing a set of rules describing

a set of available operations for a given role. $roles(e)$ is built by parsing all the ACL of the XML document configuration in an agent and adding each appearing role in this set. We define ACL XML syntax in the next section. The set roles(e) is used to define the set of keys needed by an entity, each role being linked to a key,

- k_{role_i}: cryptographic key corresponding to the role $role_i$,
- $KeyPath^{e}_{role_i}$: the set of cryptographic key for $role_i$ needed by e for key refreshment. This $KeyPath$ is depicted in grey in the second tree of Figure 2 for entity d. d has endorsed the role $role_A$. kad is the key used to encrypt data when an message is sent under the role $role_A$. The other keys are used to send the new keys in an efficient way. We describe the rekeying mechanism in the key management section. The set $KeyPath^{d}_{role_A}$ is equal to the keys set $\{kad, kcd, kd\}$,
- $KeyGroup_{role_i} = \{e \in E | role_i \in roles(e)\}$: set of devices which belongs to $role_i$.

The RBAC manager also interacts with a key manager. The latter manages different key trees (see Figure 2) whose keys are distributed to the configuration provider and managed devices in a multicast manner. These keys are used to authenticate and encrypts Netconf messages between these different peers. They are also used to refresh the keys. A key tree is bound to a group. The leaves of a tree correspond to a particular entity. An entity must be advertised of all the keys from its leaf to the root so that the key refreshment will be more efficient. The PKI is used to exchange the initial key of an entity. Once this key has been received by an entity, the public/private key are no longer used in an effort to optimize the efficiency of our secure architecture. We discuss this at the end of this section.

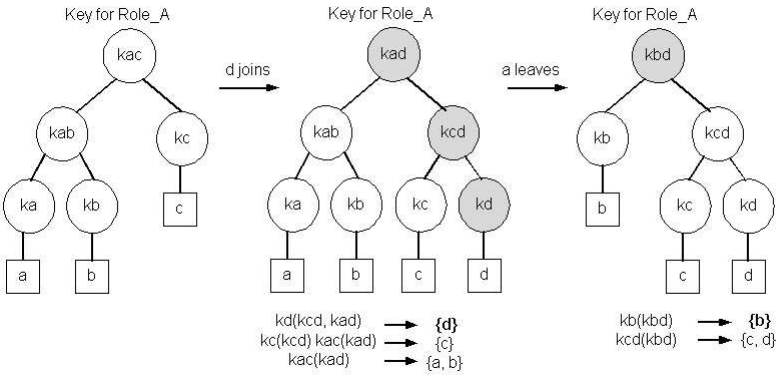

Figure 2: Key tree representation

Finally, an RBAC configurator sets the access control policy. RBAC manager maintains associations linking users and roles. When an entity asks for a role, the RBAC manager looks up its user-to-role database to decide if the user can endorse the role.

In fact, we use a key tree for each role in order to provide an efficient key refreshment. This makes it possible to refresh the k_{role_i} of the entities of $KeyGroup_{role_i}$ without using

their public/private keys. In order to refresh the key group in an efficient way, we use an LKH algorithm to update a minimal set of keys in a multicast way.

We now describe the interactions between the configuration providers, the managed devices and the RBAC manager:

- When a configuration provider wants to perform some operation on a managed devices, it must first endorse a role. In order to be given a role, a configuration provider must send a request to the RBAC manager. This request contains the roleID of the desired role. It is encrypted with the public key of the RBAC manager and authenticated with the private key of the configuration provider. Formally, the message to be sent to the RBAC manager in order to apply for a role *roleID* is the following:

$$k_{public}^{RBACmanager}(roleID, k_{private}^{EQ}(roleID))$$

  This is possible thanks to the PKI which delivers certificates to both the configuration provider and managed devices,
- If the message is valid, the RBAC manager retrieves the user name which is contained in the certificate and searches for a valid entry in its user-to-role associations set. If the asked role (roleID) can be endorsed by this user, the RBAC manager queries the key manager to get the different keys linked to this particular group. Formally, the response to be sent to the device EQ is the following:

$$k_{public}^{EQ}(KeyPath_{role_i}^{EQ}, k_{private}^{RBACmanager}(KeyPath_{role_i}^{EQ}))$$

  Key management is described in the Key Management section,
- Once a role has been endorsed, a Netconf manager is able to send secure messages to all devices. Secure here means both encrypted and authenticated with the role keys k_{role_i}. The agent also have an access control system to limit the operation a role is allowed to perform. Formally, a request sent from an entity acting as a manager to an agent under a role $role_i$ is the following:

$$k_{role_i}(< rpc > ... < /rpc >)$$

  These services are described in the next section,
- When a user leaves a group, the RBAC manager tells the key manager to remove the user from the corresponding tree. All the keys from the user key to the root of the tree must be refreshed so that the user can no longer perform operations on the managed devices.

2.2 Secure Netconf

In order to secure Netconf, managers and agents need to have a shared secret. Since we want to perform both authentication and encryption services, we need two keys. Once a manager has been authenticated to the RBAC manager, it communicates with the agent through a role that he endorsed beforehand. The manager owns the two keys k_{role}^{priv} and k_{role}^{auth} for this role. Each Netconf message is sent in the context of a role, not user. This makes it easier for both agents to perform access control and key manager to distribute and refresh keys.

Each Netconf message emitted by a manager is encrypted with the key k_{role}^{priv} and

stored in an <EncryptedData> element in accordance with the W3C recommendation XML-Encryption.

Concerning authentication service, an XML element is then signed using the key k_{role}^{auth}. The result is a <Signature> element in accordance with the W3C recommendation XML-DS. It contains the description of the algorithms used to build the signature as well as the information to retrieve the credentials that must be used to verify the signature (certificates, keys, ...). Figure 3 illustrates the XML encapsulation using XML authentication and encryption process.

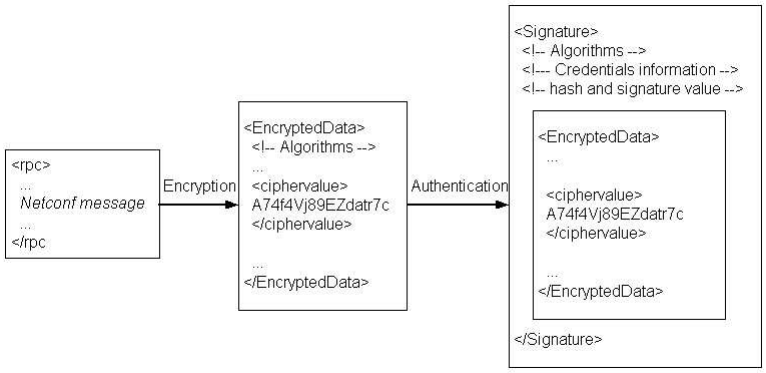

Figure 3: Netconf message encapsulation

Many different access control models have been proposed to protect XML data. Most of them are role-based [8, 1]. These models are recognized to be scalable thanks to different properties like role concept by itself and role hierarchies. However, it most often uses an external formalization with XPath references to the protected document. It remains quite difficult for an access control administrator to see what is currently accessible or not for a role because he has to parse all the permissions. We propose an XML integrated role-based Access Control List (ACL) mechanism. Each node of the XML document representing the device configuration is protected by a set of access control rules. In concrete terms, a rule describes an operation a role is allowed to perform. We organize the Netconf operations in terms of *Read*, *Write*, *Lock*. The fact that a role is allowed to perform a particular operation on a node does not mean that it is allowed to perform the same operation on the children or parent nodes. This greatly simplifies the process of access control and gives a fine-grained level of granularity. Each node is responsible for its own access control. However, an ACL editor application may give access to all parent node automatically when giving access to a particular node depending on the preferences of the rule editor. Figure 2.2 shows how we add access control list to an XML node. It defines that *myFirstRole* can read and write at this node level but does not subsume anything for the parent and children nodes. This can be compared with a Unix file system access control model. The second rule states that the role *mySecondRole* can only read this XML node. The fact that we use roles considerably limits the amount of <rule> tag in the ACL.

```
<interface>
  <ACL>
    <rule roleRef="myFirstRole" operation="RW"/>
    <rule roleRef="mySecondRole" operation="R"/>
  </ACL>
  <name>Ethernet0/0</name>
  <mtu>1500</mtu>
</interface>
```

Figure 4: An ACL for a particular node

Using this model, we are now able to build a Netconf XML response depending on the access rights of a role. First, in the case of a *Read* operation, we build the resulting XML document without taking care of the access control rules. This gives a subtree of the whole configuration tree. Then we prune the subtrees for which the role can not perform the requested operation. The next step consists in encrypting the tree with the key corresponding to the role in use: k_{role}^{priv}. Once this is done, we build an XML-Signature using k_{role}^{auth}. In the case of a *Write* operation, we do not have to build an XML document but we have to verify that the role is allowed to write in the requested node(s).

We assume that we trust the authenticated entities meaning that these entities, having some role, will not use their privilege (their role keys) to spoof another entity having the same role. In particular, we assume that an entity will not intercept a request and send back a response in place of another agent.

2.3 Key management

The Key Manager is the entity responsible for managing key trees for each group. We bind a key tree to each role, the root key serving as the role key. Since several members can endorse the same role simultaneously, all of them must receive the key for that role. A member can join or leave a group at all instant so a key refreshment mechanism is needed to deter newly arrived member from decrypting past messages and past member from de(en)crypting new messages.

Several rekeying algorithms have already been proposed among which Marks, logical key hierarchy (LKH) [9] [11] [10] and OFT. We propose to use LKH which best suits to our needs. Marks implies that the time membership is known at the beginning which is not possible in our case. LKH+ proposes a rekeying algorithm with a log(n) complexity for both join and leave rekeying. The key group (key role in our case) is the root node of the tree. Each member is a leaf and knows all the keys from its own key to the root. Its own key is distributed by the Key Manager using the public/private key. The other keys are sent in the same way as initializing of LKH. An important aspect in this algorithm is that a member must know all the keys from its own key to the root. We previously called this key set a *KeyPath*.

We describe in this paragraph the algorithm from the LKH approach to manage a member join or leave messages. Figure 2 displays three trees corresponding to the same role: the first one is the initial key tree and contains the three members a, b, c. If a new entity d wants to join the role, the key tree must be updated as efficiently as possible. D must be given the root key to be able to handle future messages. This root key is changed so that d can not read old messages. We use here a PKI to communicate the symmetric

```
<rpc message-id="107" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config xmlns="http://example.com/schema/1.2/config">
      <interface>
        <name>Ethernet0/0</name>
        <mtu>1500</mtu>
      </interface>
    </config>
  </edit-config>
</rpc>
```

Figure 5: A Netconf message

key kd for entity d encrypted with its public key. He is the only one who can decrypt this initial message. Then the other keys of the set $KeyPath^{d}_{role_A}$ are sent to d and to the other entities as depicted on the figure. The transition from the second to the third tree illustrates a leave rekeying. Entity a knows ka, kab and kad. Since ka and kab disappeared from the tree when a leaves the role, only kad should be updated so that a can not access future traffic. Therefore the key manager generates a new key kbd and sends it to all the entities under this key, namely b, c and d in such a way that the number of message is minimal. For instance, a single message is needed to send kbd to both c and d entities as they both know kcd. The two messages are given under the third tree.

We use these trees in order not to encrypt new role keys with the public keys of the entities. This would be too expensive.

3. Implementation and design

The Netconf protocol [6] allows network devices management. It is based on eXtensible Markup Language (XML) [4] in order to provide interoperability, to carry easily hierarchical data and to benefit from a large panel of existing tools allowing XML documents handling such as XSLT [5]. Each device runs a Netconf agent providing an API to manage its configurations. XML-formated configurations can be loaded dynamically onto Netconf agents or can be queried from them. The Netconf protocol uses a RPC paradigm (`<rpc>` and `<rpc-reply>`) to embed management operations. It defines a set of operations to get, copy, edit data... Data set is divided into two classes: state data, not editable, representing the status information of the managed device, and configuration data which is editable and allows to modify the behavior of the device. Figure 3 shows a sample Netconf request that set the MTU to 1500 on the interface whose name is Ethernet0/0 on the running configuration. A Netconf session is open with the advertisement of the peers capabilities, which are used to extend the basic set of operations and data introduced in [6]. A session can be explicitly closed or killed. An agent can also have several configurations (running, candidate, start-up), each of them being accessible.

The running configuration is the currently active configuration. Such a protocol introduces security issues and therefore, only authorized principals must be able to set up network devices configurations. The application protocol on which Netconf lays is responsible for providing authentication, data integrity, and privacy services. However, a

fine-grained access control model can be defined only at the Netconf level since it depends on the data that must be protected.

We propose to extend this modular architecture with a security module providing a set of services: managing ACL rules, authenticating and encrypting messages and managing keys dynamically. The first service deals with ACL rules. The module is able to parse dynamically the ACL embedded in the data configuration tree. When a request is received, the resulting document is built firstly without taking care of the ACL rules. Then the module is called to prune the subtrees which are not available for that role, operation. It is also possible to validate the ACL rules.

Once the response document has been built, it is transmitted to the Authentication and Encryption service. The module retrieves the key k_{role_i} which is the key corresponding to role $role_i$. Then the XML document is encrypted with this key so that only the entities from the set $KeyGroup_i$ can decrypt the message. The encrypted document is valid regarding the XML-Encryption recommendation.

The third service provided by our security module is the dynamic management of keys. This service is responsible for managing KeyPath coming from the RBAC manager.

Figure 6 depicts the architecture implemented in our prototype. The entity acting as a manager is represented at the top of the figure. It consists in three layers. The top layer provides a graphical tree representation of the XML configuration data for the administrator. The intermediate layer is a DOM parser that is able to build a DOM structure from the received XML data or to serialize requests from the administrator. The bottom layer is the communication layer which is in charge of sending and receiving Netconf messages from the network. The manager also contains a database that contains all the keys needed depending on all the roles a manager has endorsed.

The agent is represented at the bottom of the figure. The Networking layer is used to emit and receive Netconf message over the network. The parser layer is in charge of extracting the different parts of the Netconf messages among which the target configuration and the operation. Then the request dispatcher forwards the request to the right module depending on the name of the root tag element under `<config>`. For instance, the message on Figure 3 is handled by the interface module. We now detail the main interactions of the process on Figure 3:

- First, *interaction 1* depicts how a module registers to the Module subscription handler. The interfaces module tells the module subscription handler that it is responsible for all requests containing the `<interfaces>` element as root,
- In *Interaction 2*, the Key Manager sends the KeyPaths to each Netconf entity e for all their available roles: $\forall role \in roles(e), KeyPath^{e}_{role}$. Once the two previous interactions are performed, it is possible to send secure requests,
- *Interaction 3* shows the level in which we use the key corresponding to a role to encrypt and authenticate data,
- *Interaction 4* shows the message transmitted over the network. It is embedded into a `<Signature>` element after being encrypted into an `<EncryptedData>` element,
- *Interaction 5* depicts the interaction with the Security Manager to decrypt and authenticate the incoming message. The role to use is added in the KeyInfo element of the XML Signature,

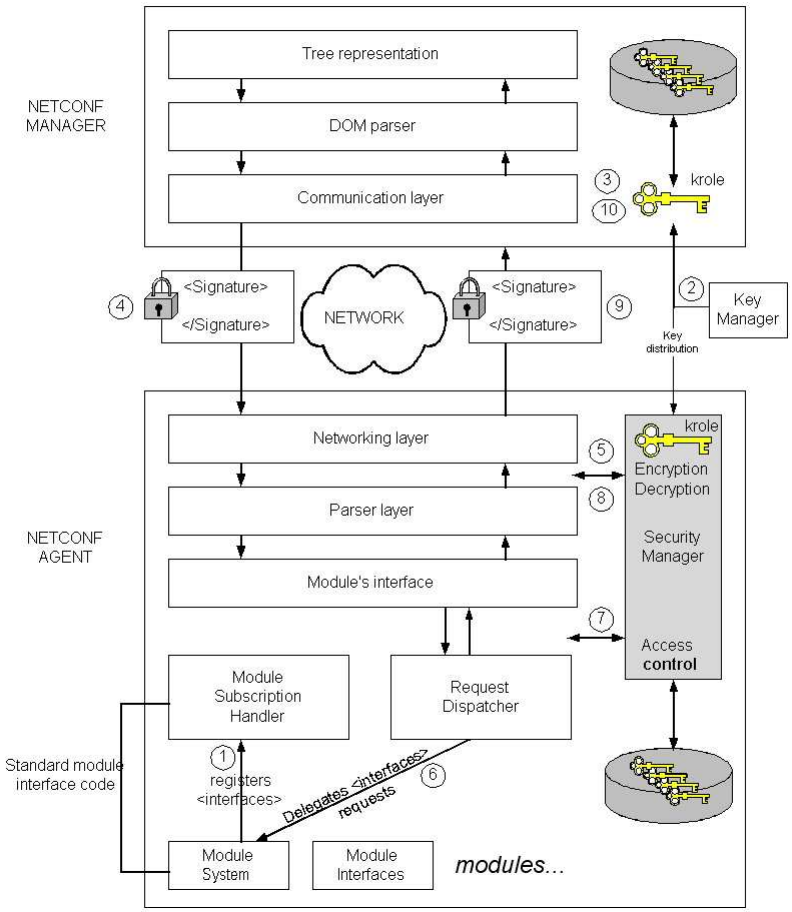

Figure 6: Netconf secure architecture

- *Interaction 6* shows how the request dispatcher delegates the handling of the request to the module on the base of the data contained in the request,
- In *interaction 7*, the security manager receives the XML response containing ACLs from the Request dispatcher and uses these ACLs to prune the XML response. This mechanisms depends on the current Netconf operation and the role in use,
- Once the document is reformatted, it is encrypted and signed in *interaction 8* in the same way as during *interaction 3*,
- *Interaction 9* shows the transport over the network to the destination,
- *Interaction 10* emphasizes the decryption and validation process on the manager side.

Figure 7 shows the structure a Netconf generic module must implement to be registered in the agent. All of the *NetConf*'s modules must respect the structure representation that

is used by the *netconfd* core to represents a generic module. The API defines several generic function calls that must be implemented by each module. The register function allows the module to register to the Module Subscription handler. The module implements the get_request function that makes use setcfg and getcfg to answer Netconf requests like <get-config> and <edit-config> depending on the request structure. A *NetConf* module only must be compliant with the generic API described just above.

```
typedef struct {
  char *name;                  /* Module's Name */
  char *desc;                  /* Description */
  void *prop;                  /* Module's properties */
  int nr;                      /* Number of module's descriptors */
  notify_f notify;             /* Module's notify() specific function */
  list_f list;                 /* Module's list_properties() function */
  savecfg_f savecfg;           /* Module's save_cfg() specific function */
  restcfg_f restcfg;           /* Module's rest_cfg() specific function */
  get_request_f get_request;   /* Module's get_request() specific function */
} net_module_t;

/* Public Functions */
net_module_t *register_module (void);

/* Private Functions */
int getcfg (void *iface);
int setcfg (char *iface, char *prop, char *value);
int notify (void);
int list_properties (char *iface);
int save_cfg (char *id);
int rest_cfg (char *id);
int get_request (xmlNodePtr tree, netconf_request_t *req, char *msg);
```

Figure 7: Netconf generic module structure

4. Example

5. Related works

We give in this paragraph an overview of the XML security concepts which have been developed mostly for the security needs of web services. Over the past years, some XML languages have been defined to address a set of security requirements. Many existing security mechanisms have been adapted to XML languages so that it is now possible not only to encrypt but also to digitally sign all or parts of an XML document. The World Wide Web consortium published a set of recommendations that describe the syntax of XML tokens and the way of handling such elements. In particular, XML-DigitalSignature [2] describes the syntax for digital signature and its linked information such that the algorithms and keys to use or the way to retrieve them. Moreover, XML-Encryption [7] provides the guidelines to generate encrypted XML elements and to link them with security credentials. XML-Canonicalization aims at describing the preprocessing of an XML element before it is either encrypted or authenticated. Indeed, some equivalent XML elements can be written in quite different ways depending on the application that handles it. Different parts such as white spaces and namespaces can be changed during the different processes. This introduces issues when XML elements need to be hashed for authenti-

cation. XML-Canonicalization [3] defines a set of rules to format the elements in such a way that it will always produce the same hash.

6. Future works

7. Acknowledgment

We would like to thank Benjamin Zores for the development of YENCA which is an implementation of the Netconf protocol agent. We also thank Isabelle Chrisment for fruitfull discussions on key group exchange.

References

[1] Anne Anderson. Xacml profile for role based access control (rbac). OASIS Committee Draft, February 2004.

[2] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. Xml-signature syntax and processing. W3C Recommendation, February 2002.

[3] John Boyer, Donald E. Eastlake, and Joseph Reagle. Exclusive xml canonicalization version 1.0. W3C Recommendation, July 2002.

[4] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible markup language (xml) 1.0 (third edition). W3C Recommendation, February 2004.

[5] James Clark. Xsl transformations (xslt) version 1.0. W3C Recommendation, November 1999.

[6] R. Enns. Netconf configuration protocol. Internet Draft, October 2003.

[7] Takeshi Imamura, Blair Dillaway, and Ed Simon. Xml encryption syntax and processing. W3C Recommendation, December 2002.

[8] R. Kuhn. Role based access control. NIST Standard Draft, April 2003.

[9] Marcel Waldvogel, Germano Caronni, Dan Sun, Nathalie Weiler, and Bernhard Plattner. The VersaKey framework: Versatile group key management. *IEEE Journal on Selected Areas in Communications*, 17(9):1614–1631, September 1999.

[10] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. RFC 2627 (informational), June 1999.

[11] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure group communications using key graphs. *IEEE/ACM Trans. Netw.*, 8(1):16–30, 2000.