

## Dynamic Approach to prohibit the Diffie-Hellman Protocol Interception

<sup>1</sup> M. El Hamzaoui, <sup>2</sup> A. Sekkaki, <sup>3</sup> B. Bensassi, <sup>4</sup> C. B. Westphall, <sup>5</sup> C.M. Westphall

<sup>1,2,3</sup> University Hassan II Ain-Chok, Faculty of sciences, Department of Mathematics & Computer Sciences,  
P.O Box 5366, Maarif, Casablanca. Morocco.

<sup>1</sup> m\_elhamzaoui@yahoo.fr, <sup>2</sup> a\_sekkaki@yahoo.fr, <sup>3</sup> bahloul\_bensassi@yahoo.fr

<sup>4,5</sup> Federal University of Santa Catarina, Network and Management Laboratory, Caixa Postal 476, 88040-970,  
Florianopolis – SC – Brazil

<sup>4</sup> westphal@lrg.ufsc.br, <sup>5</sup> carla@lrg.ufsc.br

### **Abstract:**

*Diffie and Hellman invented in 1976 a protocol bearing their names (Diffie-Hellman) that was at the origin of the cryptography. The most important characteristic of this protocol is that it enables two connected parties to generate a shared secret without having any preliminary information about one other. However, The point of vulnerability of this protocol is the active attack and more precisely the interception of the security information exchanges. In this paper, we present a dynamic Public Key Infrastructure (PKI) to circumvent the Diffie-Hellman vulnerability. The main objective of this PKI environment is to discharge the connected parties from all management tasks. Our PKI environment is composed of a PKI Server (PKIServ) to manage the security inter the connected parties and a Monitoring Service (MS) to automate the PKI environment functioning. A prototype has been implemented with CORBA environment and same experimental results are presented.*

**Keywords:** Cryptography, Diffie-Hellman protocol, PKI, Policy-Based Management, Ponder Language.

### **1. Introduction**

Cryptography plays, nowadays, a significant role in the realization of the security services and mechanisms (ISO 7498-2). Thus, several cryptography algorithms were developed in order to be used in security protocols and applications. For example, IPsec protocol (IP Security) [01] uses security mechanisms that rest on the cryptography such as AH (Authentication Header) [02] and ESP (Encapsulating Security Payload) [03]. The AH Mechanism uses authentication algorithms while ESP (Encapsulating Security Payload) employs encryption algorithms [04].

The cryptography algorithms require the use of the private and public keys and in the case of distributed systems, the management of the keys through a public keys infrastructures will be essential. The main tasks of the cryptography keys' management are generation, distribution, storage and suppression of keys.

The Diffie-Hellman (DH) protocol [05] was the origin of public-key cryptology. The protocol principle is that two people who wish to communicate can exchange the necessary information to generate a shared secret without having any preliminary information about one other. However, the Diffie-Hellman vulnerability point is the active attacks. Thus, an interceptor can sneak between the connected parties and shares with each one of them a different secret. However, the two connected parties think they are sharing the same shared secret.

Several approaches were proposed to circumvent the interception problem. These approaches were based on the public values authentication used to generate the shared secret. However, the problems of these approaches are on the one hand the elimination of a significant advantage of Diffie-Hellman which is the possibility of generating a shared secret without having any preliminary information, and on the other hand these approaches continue the security information between the connected parties.

Policy-based management consists in optimising the managers' efforts by making the distributed system management easy and dynamic as much as possible. The policy-based management

principle [06] is first to define the management policies and to distribute and apply them, then present them as a set rules. Finally, these rules must be decided at Policy Decision Points (PDP) and will be also performed at the considered Policy Enforcement Points (PEPs) which exist on the interest network nodes. To specify policies, we could use the ponder policy specification language [07] which is an important tool for specifying security and management policies for distributed systems.

Our work objective is to solve the interception problem of Diffie-Hellman protocol. To avoid any Security information exchange between connected parties, our solution will be based on the use of an environment of security dynamic management (PKI environment). All security tasks such as the exchanges of public values, the private values generation, the shared secrets calculation, the keys' generation etc... will be dealt with in this environment. The basic elements of our security environment will be the Ponder policy specification language, the PKI Server (PKIServ) and a Monitoring Service (MS) to automate the PKIServ functioning. Our approach will be compared with other DH-based protocols to show

This work will be presented as follows, in the second section we will present the cryptography principle. The third section will display Diffie-Hellman protocol. Concerning the fourth section, it will briefly outline the Ponder Language. The fifth section will include our solution. The comparison of our approach with other DH-based protocols will be the subject of the sixth section. Finally, the conclusion and perspective of this work will be featured in the last section.

## 2. Cryptography principle

Cryptography is important in the realization of the security services and mechanisms (ISO 7498-2). Thus, several cryptography algorithms were developed in order to be used in security protocols and applications.

In cryptographic terminology [08], the message or the data that someone wants to send to a receiver is called plaintext or cleartext. Encryption is the way to hide the message contents from outsiders by encoding it and the encrypted message is called ciphertext. Decryption is the process to retrieve the plaintext from the ciphertext.

Cipher or Encryption/decryption methods are based on the use of keys:

On one hand, the cipher type depends on the used key. Thus, one has to distinguish between two types of cipher :

Asymmetric cipher (public-key cipher) : The encryption key is public in order to be used by anyone to encrypt the exchanged message, whereas the decryption key is private to be used only by the proper recipient to decrypt the message. The encryption key is also called the public key and the decryption key is called the private key.

Symmetric cipher (secret-key cipher): The same key, which is also called secret-key, is used to encrypt and decrypt data.

On other hand, the key type depends on the function that the later can perform [09]:

- Keys to cipher other keys: This type of keys is used to cipher other keys. In the context of public key cryptography, they correspond to the public keys used to cipher the transported ones.
- Master Keys: They are used to generate keys by derivation.
- Keys to cipher data (Session keys): They are generally secret keys which are useful for ciphering exchanged data. They are characterized by their weak lifetime.

Cryptography algorithms require the use of the private and public keys and in the case of distributed systems, the use of public key infrastructures is essential to manage keys (generation, distribution, storage and suppression).

In order to establish secured communications, the use of a mutual authentication and key exchange protocol is very essential (secure protocol)[10]. A secure protocol [11] is a protocol such that the following conditions hold in all cases where one party, say Alice, executes the protocol faithfully and accepts the identity of another party:

- At the time that Alice accepts the other party's identity (before she sends or receives a subsequent message), the other party's record of the partial or full run matches Alice's record.

- It is computationally infeasible for the exchanged key accepted by Alice to be recovered by anyone other than Alice and possibly the party whose identity Alice accepted. (This condition does not apply to authentication without key exchange.)

In addition to being secure, there are other desirable characteristics for a protocol.

- Perfect Forward Secrecy: is the property that disclosure of the long-term secret keying material that is used to derive an agreed ephemeral key does not compromise the secrecy of agreed keys from earlier runs.
- Direct Authentication: When authentication is established by the end of each protocol run, the protocol is direct else it is indirect. An indirect protocol can be modified to be direct by adding an exchange of known messages or messages with redundancy encrypted with the exchanged key.
- No Timestamps: While timestamps are convenient for administrative and documentation purposes, it is desirable in practice to avoid relying on their use for security in authentication protocols.

### 3. Outline on Diffie-Hellman protocol

#### 3.1. Diffie-Hellman principle

The Diffie-Hellman protocol [05] was invented in 1976. It enables two connected parties to generate a shared secret without having any preliminary information about one another. Diffie-Hellman is based on the public key cryptography because it uses public and private values. We will show hereafter (fig.1) the different steps to share a secret by using Diffie-Hellman protocol:

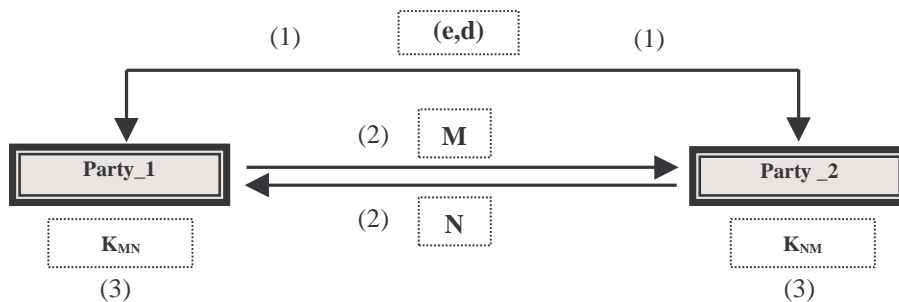


Fig.1 : Diffie-Hellman principle

- (1)- The two communicants agree, first, on two integers  $e$  and  $d$ , such as:  $e$  is a big number,  $(e-1)/2$  is a prime value and  $d$  is primitive to  $e$ .
- (2)- Party\_1 chooses a number  $m$  to be able to calculate the public value  $M$  ( $M=d^m \text{ mod } e$ ). In the same way, the party\_2 chooses a value  $n$  to have a public value  $N$  ( $N = d^n \text{ mod } e$ ). These two public values must be exchanged between the two connected parties.
- (3)- Party\_1, in order to have the shared secret must calculate :  $K_{MN}$  ( $K_{MN} = N^m \text{ mod } e$ ). In the same way, party\_2 has to calculate the shared secret :  $K_{NM}$  ( $K_{NM} = M^n \text{ mod } e$ ).

The shared secret will be then:  $K_{MN} = K_{NM} = d^{mn} \text{ mod } e$ .

Henceforth, we will call DH parameters all values  $e, d, m, M, n, N, K_{MN}$  and  $K_{NM}$ .

#### 3.2. Diffie-Hellman Vulnerability

Diffie Hellman problem is the interception (man-in-the-middle attack) that could occur in the phase of exchange of Diffie-Hellman parameters  $M$  and  $N$  as showed in the following figure (fig.2) :

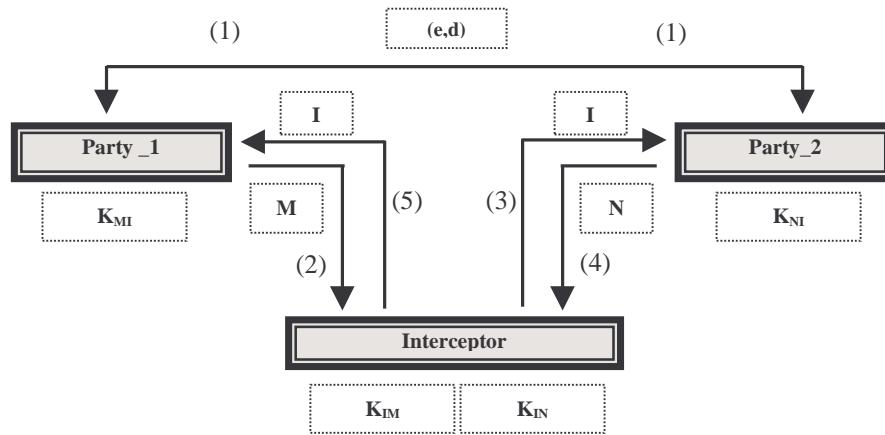


Fig.2: Principal of the interceptor attack

- (1)- The connected parties define initially their integer public values (e,d).
- (2)- Party\_1 chooses his internal secret value m to calculate the public value M ( $M = d^m \text{ mod } e$ ) that will send after to party\_2. The interceptor receives this value and will calculate, by using his internal secret value i, the corresponding shared secret:  $K_{IM} = M^i \text{ mod } e$ .
- (3)- The interceptor calculates, by using his internal secret value i, the public value I ( $I = d^i \text{ mod } e$ ) and will send it afterwards to party\_2 instead of the intercepted value M. At the reception of I, the party\_2 will calculate, by using his internal secret value n, the corresponding shared secret:  $K_{NI} = I^n \text{ mod } e$ .
- (4)- Party\_2 sends his value N ( $N = d^n \text{ mod } e$ ) to party\_1. The interceptor will intercept this value to calculate the corresponding shared secret:  $K_{IN} = N^i \text{ mod } e$ .
- (5)- The interceptor replaces N by I then sends it to party\_1. Party\_1 will calculate the corresponding shared secret:  $K_{MI} = I^m \text{ mod } e$ .

In short, the two connected parties have the impression to have the same shared secret ( $K_{MN} = K_{NM}$ ), but the problem is that each one of them shares a different secret with the interceptor. Thus, on one hand, the interceptor shares with party\_1 the secret  $K_{MI} = K_{IM}$ , and on another hand, the interceptor shares with party\_2 the secret  $K_{NI} = K_{IN}$ .

The interception constitutes a fundamental problem of Diffie-Hellman protocol. This work's objective is to prohibit any kind of Diffie-Hellman parameters interceptions by the use of a PKI environment and Ponder policy specification language.

#### 4. Ponder Policy Specification Language

Ponder [07] is an object-oriented, declarative language for specifying security and management policies for distributed system. Ponder makes the policy-based management of distributed systems easier by offering a greater flexibility thanks to its object-oriented properties.

The Ponder basic characteristics are:

- Access control specification which is based on the deployment of authorisation, delegation, information filtering and refrain policies.
- Obligation policy specification to call upon managers to intervene when a special event occurs in the system.
- Constraints specification to define the conditions under which the policy is valid.
- Composite policies specification to simplify the policy specification task for large distributed systems.

For Ponder all subjects, targets and policies objects are organized in domains. A domain is very similar to a directory or folder on a personal computer, and it is used to partition large systems according to some precise criteria [12]. Domains make distributed systems management very easy and flexible and give the possibility to modify the domains' components without altering management policies.

The organisation in domains of our PKI environment is illustrated on the figure 3 :

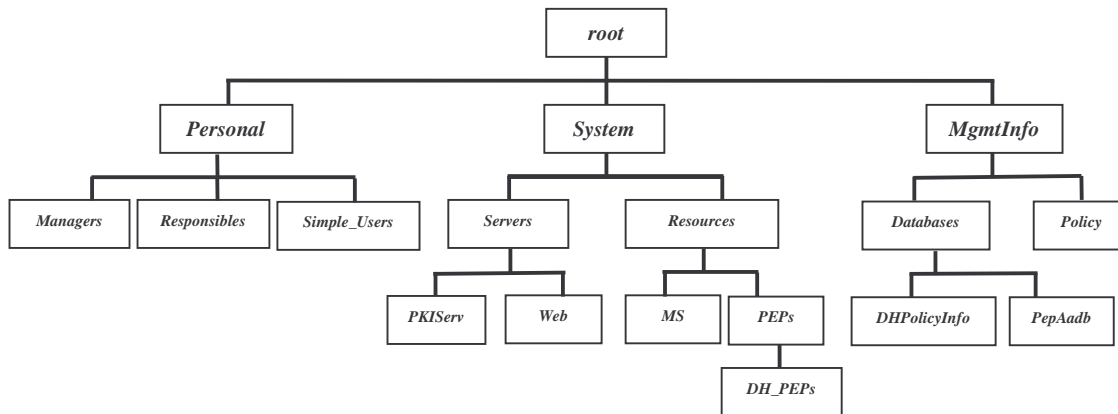


Fig.3: Organization of our PKI environment components

The dynamic management of the PKI environment components such as users, materials, software and responsibilities requires, as it is schematised in fig.3, the use of a root domain. This latter allows to organize the PKI environment components in three main sub-domains : personal, System and MgmtInfo.

In what follows, we will restrict our discussion on obligation policy. Obligation policy specifies what activities a subject (members of one or several domains) must carry out on a set of target objects (objects of one or several domains) and defines the duties of the policy subject. Obligation policy is triggered by events and is normally interpreted by a manager agent at the subject. Events can be internal, e.g., a timer event, or external events, which could be managed by a monitoring service. The two possible syntaxes to specify an obligation policy are :

- Syntax for direct declaration of obligation policy instance :

```

inst oblig policyName “{“
    on event-specification ;
    Subject [<type>] domain-Scope-Expression ;
    [Target [<type>] domain-Scope-Expression ; ]
    do obligation-action-list ;
    [catch exception-specification ; ]
    [when constraint-Expression ; ] “}”
    
```

The key word **on** specifies the required event. Subject and target are expressed in term of domains. The optional **catch**-clause specifies an exception that is executed if the actions fail to execute for some reason.

- Syntax for declaration and instantiation of authorization policy type :

```

Type oblig policyType “(“ formalParameters “)” “{“ {obligation-policy-parts } “}”
inst oblig policyName= policyType “(“ actualParameters “)” ;
    
```

The authorisation policy type is initially declared, then instantiated.

Ponder is used in many works. Lymberopoulos et al. showed, in [13], how PONDER policies can be implemented and validated for Differentiated Services (DiffSer) by using CIM (Common Information Model) as the modeling framework for network resources as this device independent. They also used, in [14], Ponder language to realize a dynamic adaptation of policies in response to changes could occur within the managed environment. Finally, Damianou et al. presented, in [15], the implementation of an integrated toolkit for the specification, deployment and management of

policies specified in the PONDER language. Concerning our research group, we used Ponder to specify a Virtual Laboratory security policies [16] and manage the distribution of IPSec security policies [17].

### 5. Our approach

The objective of this work is to present a Keys' dynamic management environment which circulates no encoding/decoding information between connected parties in order to prohibit all interceptions between them. Consequently, the Diffie-Hellman parameters will be policy-based managed.

In order to realize a dynamic management environment, we will integrate to this latter a monitoring service which facilitates and automates its functioning.

#### 5.1. Principle of the proposed PKI

Our approach is based on a security policy-based management environment (fig.4) and it consists in discharging the users from all security management tasks as the management of the encryption/decryption keys (generation, distribution and suppression of keys) and the management of the security policy (modification of the policy parameters).

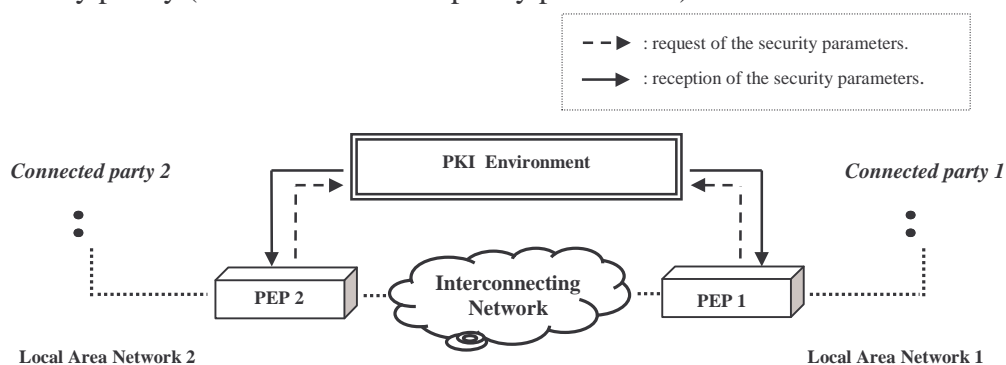


Fig.4: Our proposed PKI environment.

Both the management of the encryption/decryption keys and management of the security policy will be dealt with at the level of a special environment. This environment is our public key infrastructure (PKI) environment (fig.4) that manages automatically the security of the inter connected parties communications. Thus, the PKI environment will calculate, modify, decide, and distribute the DH security policy parameters to apply.

The PEPs of the users' domains that secure the inter connected parties communications must implement the encrypting/decrypting methods correspond to the DH security policy. Thus, when the PKI environment decide the DH security policy parameters to apply, it places them at the disposal of these PEPs (fig.4).

Concerning the implementation (fig.5), the PKI environment is composed of two basic components which are a PKI Server (PKIServ) that contains the PDP to manage the inter-connected parties communications security and a Monitoring Service (MS) to automate the PKIServ functioning :

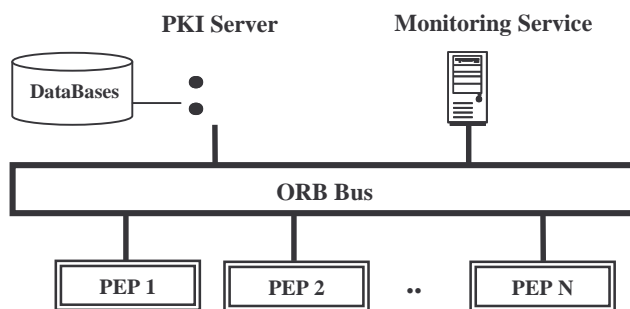


Fig.5: The mains actors of our approach

Moreover, the PKIServ is provided with two databases. The first one contains all necessary information on the PKI environment (*DHPolicyInfo.mdb*) whereas the second contains the necessary information on the PKI Environment PEPs (PEP Authentication and Authorization DataBase : *PepAadb.mdb*).

### 5.2. Ponder Specification of the PKI environment Management Tasks

In order not to use durably the same DH security policy parameters (e,d,m,M,n,N and the shared secret ( $K_{MN}=K_{NM}$ )) and give luck to others to discover our security parameters, we will proceed like this:

We estimate for the DH parameters an application time ( $T_{app}$ ) which is lower than the necessary time to discover them. Moreover, The content of the table *DH\_param* of the database *DHPolicyInfo.mdb*, which contains the DH security policy parameters, must be modify each  $T_{tab}$  ( $T_{tab} = 5 * T_{app}$ ).

The corresponding Ponder specifications are:

```
inst oblig PolicyChangeDHTable {
on      EventModifyDHTable() ;
Subject System/Resources/Servers/PKIServ ;
Target  t = MgmtInfo/Databases/ DHPolicyInfo.mdb ;
do      param[]= selectDHParam() -> supp(t.DH_param)
        registry(t.DH_param, param[]) ;
```

At each  $T_{tab}$ , the *PKIServ* receives, from the *MS*, the event *EventModifyDHTable()*. Then, it selects firstly with a random manner ten new DH security policy parameters. Afterwards, it suppresses the content of the table *DH\_param*. Finally, The *PKIServ* registries these news parameters in the table *DH\_param*.

At the expiry of the applied DH parameters, i.e at each  $T_{app}$ , the following obligation policy must be triggered:

```
inst oblig PolicyChangeDHParam {
on      EventChgtPolicyParam() ;
Subject s = System/Resources/Servers/PKIServ ;
Target  t = MgmtInfo/Databases/DHPolicyInfo.mdb ;
do      DHPolicy_param[]= selectParam(t.DH_param)
        -> registry(t.PEPs_Needs,DHPolicy_param[]) ;
```

At the reception of the event *EventChgtPolicyParam()*, from the *MS*, the subject *PKIServ* selects firstly with a random manner, from the Table *DH\_param* of the database *DHPolicyInfo*, the DH parameters to apply and stores them then in the variable *DHPolicy\_param*. Afterwards, in order to put these parameters in the disposal of the PEPs, they must be stored in the table *PEPs\_Needs* of the same database.

### 5.3. Implementation of the proposed PKI

In our implementation, the PKI environment is composed, as we already mentioned, of two basic components which are the PKI Server (*PKIServ*) to manage the sinter connected parties communications security and a Monitoring Service (*MS*) to automate the *PKIServ* functioning:

The communication *MS-PKIServ* and *PEPs-PKIServ* are all ensured through a ORB bus (CORBA Objects)(fig.5) and they are all in the form of remote methods invocation. These invoked methods are showed through the following *idl* file :

```

module PkiServ {

// Interface of methods invoked by the Monitoring Service:
    interface PkiServ_MS {
        oneway void changeDHPolicyParam();
        oneway void modifyDHParamTable();
    };

// Interface of methods invoked by the PEPs:
    interface PkiServ_PEPs {
        string getDHPolicyParam(in string pep_id, in string pep_passwd, in string pep_secret);
    };
};

```

**Fig.6:** The idl file (PkiMgmt.idl).

Because of the PKI environment databases (*DHPolicyInfo.mdb* and *PepAadb.mdb*) contain static tables, the permanent change of these tables' contents will be essential to do not give to others the opportunity to discover our security parameters. Consequently, a dynamic management of these databases will be important. To automate the management of the entirely PKI environment, we integrated in this latter, as we already mentioned, a monitoring service.

### 5.3.1. Communication inter PKI Server-Monitoring Service :

Firstly, we remember that the parameters of the DH security policy (e,d,m,M,n,M, shared secret ( $K_{MN}=K_{NM}$ )) are stored in the tables *DH\_param* of the database *DHPolicyInfo.mdb*.

In our implementation, we defined two types of the inter MS-PKIServ communications (fig.6) where each one of them has its own reasons to be.

#### i . Change of the applied policy parameters

At the expiry of the application time ( $T_{app}$ ) of the applied parameters, the MS invokes at the level of PKIServ the method *changeDHPolicyParam()* (fig.6). This method selects randomly through its sub-method *selectDHPolicyParam()* (fig.7), from the table *DH\_param*, the new DH parameters to apply:

```

..... // Program
public void changeDHPolicyParam(){
    DHPolicy_param=selectDHPolicyParam();
}
..... // Program

```

**Fig.7:** Implementation of the method *changeDHPolicyParam()*

#### ii. Change of the content of the DH policy parameters table

In order to modify permanently the contents of the static table *DH\_param*, the MS invokes at the level of PKIServ, at each  $T_{tab}$ , the method *modifyDHParamTable()* (fig.6). This method changes dynamically the content of the table *DH\_param*. The corresponding implementation is:

```

..... //program
public void modifyDHParamTable() {
    changeDHParamTable();    \\ Method to change the DH_param Table contents.
}
... // program.

```

**Fig.8:** Implementation of the method *modifyDHParamTable()*



The sub-method `changeDHPParamTable()` (fig.8) removes the contents of the table `DH_param` and records afterwards in it ten new recordings.

### 5.3.2. Communication PKI Server-PEPs

All the PEPs of our PKIServ implement the encrypting/decrypting methods corresponding to the DH security policy. These PEPs call periodically the PKIServ through the invocation of the method `getDHPolicyParam()` (fig.6) to get the security parameters to apply (DH security Policy parameters). The period of the invocation of the method `getDHPolicyParam()` must be too lower than the average time of the change of the DH security policy parameters.

The arguments of the method `getDHPolicyParam()` (fig.6) are the PEP identifier, the PEP password and a secret to reinforce the security. In this context, we could also use certificates to reinforce communications security [18].

The implementation of the method `getDHPolicyParam()` is :

```

..... // program
public String getDHPolicyParam(String peplogin, String peppasswd, String pepsec){

    boolean evalauthen, evalauthor;
    evalauthen = authentication(peplogin,peppasswd);
    evalauthor = authorization(peppasswd,pepsec);
    String resp="";

    if( evalauthen == true){
        if( evalauthor == true) { resp = getSecParam();}
        else { resp = "Warning : Unauthorized PEP";}
    }
    else { resp = "Warning : Failed Authentication ";}

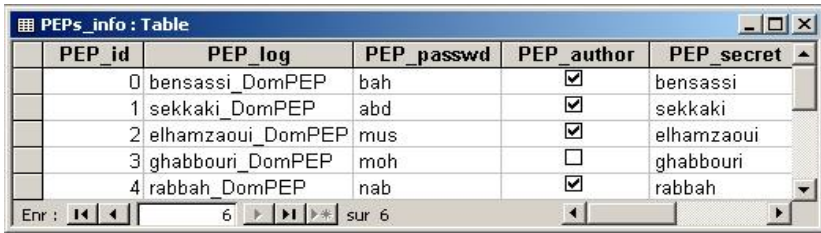
    return resp;

} // End of the method getDHPolicyParam().
..... // program

```

Fig.9: Implementation of the method `getDHPolicyParam`

The PKIServ checks firstly the PEP identity through the method `authentication()` (fig.9) and checks afterwards the PEP authorization through the method `authorization()` (fig.9). The checking of both PEP authentication and PEP authorization is based on the consultation of the table `PEPs_info` (fig.10) of the database `PepAadb.mdb`.



PEP_id	PEP_log	PEP_passwd	PEP_author	PEP_secret
0	bensassi_DomPEP	bah	<input checked="" type="checkbox"/>	bensassi
1	sekkaki_DomPEP	abd	<input checked="" type="checkbox"/>	sekkaki
2	elhamzaoui_DomPEP	mus	<input checked="" type="checkbox"/>	elhamzaoui
3	ghabbouri_DomPEP	moh	<input type="checkbox"/>	ghabbouri
4	rabbah_DomPEP	nab	<input checked="" type="checkbox"/>	rabbah

Fig.10 : Table `PEPs_info`

### 5.4. Scenario of execution

As we previously explained, the method *modifyDHParamTable()* permits to change the content of the static table *DH\_param* of the database *DHPolicyInfo.mdb* which contains the parameters of the DH security policy. An example of the invocation of this method gave us the following result:

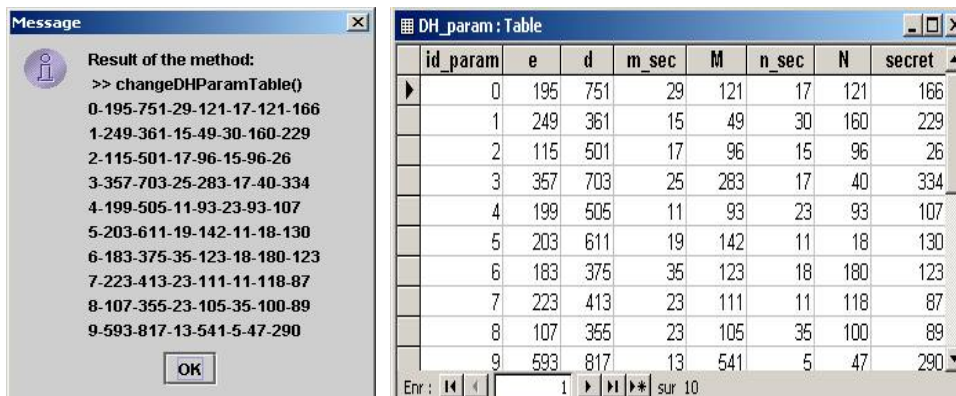


Fig.11: Modification of the content of the table *DH\_param* through the method *modifyDHParamTable()*

All our next examples and executions will be based on the current contents of this table. In order to change de policy parameters, the MS invokes, at each  $T_{app}$ , the method *changeDHPolicyParam()*. Moreover, for each change of the DH security policy parameters, the PKIServ stores these new parameters in the variable *DHPolicy\_param* and registries them afterwards in the table *PEPs\_Needs* of the database *DHPolicyInfo.mdb*. Practically, the execution of these operations gave us:



Fig.12: The result of the method *selectDHPolicyParam()* and the content of the variable *DHPolicyParam*

Concerning the PEPs, they get the security parameters to apply through a simple invocation of the method *geDHPolicyParam()*. In the reception of this invocation, the PKIServ consults initially the table *PEPs\_info* (fig.10) of the database *PepAadb.mdb* to check the identity and the authorization of the corresponding PEP. The possible replies that could receive the PEPs are :

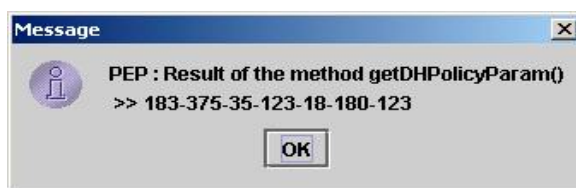
- Failed Authentication or Failed Authorization : They correspond respectively to the case where the PEP is not declared at the level of the table *PEPs\_info* (fig.12) and to the case where the PEP is declared in the table *PEPs\_info* but it is not authorized to use the PKI environment. An example of these two replies is given through the following figures :



Fig.13: Failed authentication and failed Authorization replies

- Correct Authentication and Authorization : If the PEP is declared and authorized at the level of our PKIServ, this latter gives it the security parameters to apply which are stored in the table *PEPs\_Needs* of the database *DHPolicyInfo.mdb*. In our example, we invoked the method

`getDHPolicyParam()` with a correct authentication/authorization arguments and the obtained result is :



*Fig.14:* Result obtained by a correct invocation of the method `getDHPolicyParam()`

## 6. Evaluation of our work

DH protocol is used in many mutual authentication and key exchange protocols that are characterized by their imposed prerequisites (preliminary shared secret, public key infrastructure) and verified properties (direct authentication, PFS).

In this section, we present three mutual authentication and key exchange protocols that are used on IP Networks. They are DH-based protocols and characterized by the employ of public values authentication to generate shared secret. On one hand, they either use authenticated public values through certificates or authenticate public values after having exchanged them. On other hand, they are either connection oriented protocols or connectionless protocols:

- If the protocol is connection oriented, it will need a protocol of establishment of authenticated session key before the communication and the resulting key is then used to secure the IP traffic. Because of IP protocol is connectionless, the establishment and management of a pseudo session layer under IP constitute a main disadvantage of this solution.

- If the protocol is connectionless, the management of stateless keys is essential and then any connection is required. This approach consists in transmitting the key used to cipher the packet in the packet itself. The disadvantage of this approach is the add of data to each transmitted packet.

SKIP (Simple Key management for Internet Protocols) protocol [19], which is a connectionless protocol, is integrated on IP level. SKIP does not require any preliminary messages exchange before sending a ciphered packet because each packet transports the necessary information to be deciphered later. It is also based on the generation of DH shared secret with authenticated public values.

SKIP uses DH 1024 bit public key based authentication algorithms for long term key (shared secret). This shared secret is then used to derive a secret key (40 to 256 bits) for ciphering keys. Concerning the keys exchange, the secret key is used to cipher a session key (packet key) that is afterwards employed to manage two keys; a packet ciphering key and a packet authentication key.

Skip does not provide the PFS property. Indeed, if the secret key is discovered, the whole of the used session keys will be compromised. However, the SKIP extension (SKIP PFS) provides this property.

Photuris protocol [20][21] is an connection oriented protocol that requires preliminary exchanges before each ciphered message transmission. Photuris is based on the generation of a DH shared secret of a weak lifetime. This secret is used to generate the necessary session keys to protect the coming exchanges.

To prohibit the interception problem, Photuris authenticates, using a short term secret, the values used to generate the shared secret after their exchange. Therefore, Photuris provides the PFS property. Photuris introduces also the concept of cookies to forbid certain denial of service attacks. This protocol comprises three phases [09] : cookies exchanges like mechanism to counter attacks, and public values and identities exchanges.

Oakley protocol [22] proposes several distinct modes of keys exchange, uses Photuris cookies mechanisms, and does not require any DH shared secret calculation before the protocol end. It is distinguished from the other protocols by the fact that it explicitly allows third parties to agree with each other on the keys exchange mechanisms, cipher and authentication.

In sum, the main properties of these three protocols could be summarised in the following table:

Protocol	Connection oriented	PFS property	prerequisites	Secret lifetime
SKIP	no	No Yes(SKIP PFS)	No	long
Photuris	yes	Yes	Yes	weak
Oakley	yes	Yes	No	-

**Table 1** : The main properties of SKIP, Photuris, and Oakley protocols

Concerning our approach, it preserved the main DH protocol characteristic with the fact that the majority of the key management tasks are dynamically dealt with on the level of a PKI environment. Our approach avoids the preliminary exchange of security information which is at the origin of all attacks and threats and also replaces the security information exchange by an automatic management based on the methods invocation that work on behalf of connected parties. The secret lifetime strongly depends on the period of the invocation of the **PKI Server** by the MS. This invocation is realized through the method **changeDHPolicyParam()** (fig.6) that changes the applied DH parameters. This management task means that our approach could easily provide the PFS property.

Concerning the authentication, it is assured through the method **getDHPolicyParam()** of the interface **PkiServ\_PEPs** (fig.6) that requires, as argument, the PEP identifier and password plus an authentication secret.

Moreover, our approach could be generalized to manage the inter-domain communications security and could be also opened on the environment security users [23]. The opening on the users means that the latter could establish automatically secured channels between them and they could also apply their desired security parameters.

## 7. Conclusion

The objective of our work was the presentation of an approach to solve the Diffie-Hellman vulnerability problem by preventing the interceptions between the connected parties. Our approach is based on the idea not to leave any encoding/decoding information circulate between connected parties and also to treat all actions that the connected parties used to do on Diffie-Hellman parameters at the level of a Keys' management environment. This environment decides and applies, in an intelligent and dynamic way, the Diffie-Hellman parameters.

Solutions of the Diffie-Hellman vulnerability problem were also brought by other approaches but the problem is that they continue to exchange security information between the connected parties by increasing the security level and improving the used mechanisms. However, the fact of continuing to exchange the security information between the connected parties could constitute a future threats of all these solutions.

The improvement of our work will be based on two main ideas. The first one will consist in extending our approach to support other asymmetric algorithms and symmetric cryptography whereas the second one will consist in using this final resulted solution to secure the inter-domain communications.

## 8. References

1. S. Kent, R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, November 1998.
2. S. Kent, and R. Atkinson. IP Authentication Header. RFC 2402, November 1998.
3. S. Kent, R. Atkinson. IP Encapsulating Security Payload (ESP). RFC 2406, November 1998.
4. D. Piper. The Internet IP Security Domain of Interpretation for ISAKMP. RFC 2407, November 1998.
5. W. Diffie, M.E. Hellman. New directions in cryptography, *IEEE Transactions on Information Theory*, IT-22, 1976, pp.644-654.
6. R. Yavatkar, D. Pendarakis, R. Guerin, A Framework for Policy-based Admission Control, RFC 2753, January 2000
7. N. Damianou, N. Dulay, E. Lupu, M. Sloman, The Ponder Policy Specification language, Proc. Policy 2001, International Workshop on Policies for Distributed Systems and Networks, Bristol, United Kingdom, January 29-31, 2001.
8. hsc: <http://www.hsc.fr/ressources/articles/ipsec-tech/>, last update : 23 October 2002.
9. SSH Communications Security: <http://www.ssh.com/support/cryptography/introduction/terminology.html>, Last update: 2006.
10. Siaw-Lynn Ng and Chris Mitchell. Comments on mutual authentication and key exchange protocols for low power wireless communications. *IEEE COMMUNICATIONS LETTERS*, 2003.
11. Diffie Whitfield, Van Oorschot Paul C., Wiener Michael J.. Authentication and authenticated Key Echanges, Designs, Codes and Cryptography. 2, Kluwer Academic Publishers, Mars 1992, pp. 107-125.
12. M.S. Sloman. Policy Driven Management for Distributed Systems. *Journal of Network and Systems Management*, 2(4), Plenum Press, 1994, pp. 333-360.
13. L. Lymberopoulos, E. Lupu, M. Sloman, PONDER Policy Implementation and Validation in a CIM and Differentiated Services Framework, 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea, May 2004.
14. L. Lymberopoulos, E. Lupu, M. Sloman. An Adaptive Policy-Based Framework for Network Services Management. *Journal of Network and Systems Management*, September 2003. Vol.11, No.3, pp. 277-303.
15. N. Damianou, N. Dulay, E.C. Lupu, M. Sloman. Tools for Domain-Based Management of Distributed Systems. IEEE/IFIP Network operations and management symposium (NOMS2002), Florence, Italy, 15-19 April 2002, pp. 213-218.
16. A. Sekkaki, M. El Hamzaoui, and B. Bensassi. Policy-based Management of a Virtual Laboratory Communications Security. in: Proc. The First IEEE International Workshop on Broadband Convergence Networks (BCN2006), Vancouver, Canada, April 7, 2006, pp.199-204.
17. M. El Hamzaoui, A. Sekkaki, B. Bensassi. Infrastructure to manage the Distribution of IPSec Security Policies. in: Proc. GRES'2006 - Gestion de REseau et de Service, 7ème Colloque Francophone, Bordeaux-France, 09-12 May 2006, pp. 315-326.
18. C.B. Westphall, A. Sekkaki, L.M. Alvarez, W.T. Watanabe. Extending TINA Secure On- Line Accounting Services. *Journal of Network and Systems Management (JNSM)*, December 2003, Vol.11, No.4.
19. G. Montenegro, V. Gupta. Sun's SKIP Firewall Traversal for Mobile IP. RFC 2356, June 1998.
20. W.A. Simpson, Karn Phil. Photuris: Session-Key Management Protocol. RFC 2522, Mars 1999.
21. W.A. Simpson, Karn Phil, Photuris: Extended Schemes and Attributes, RFC 2523, Mars 1999.
22. H. Orman, The OAKLEY Key Determination Protocol, RFC 2412, November 1998.
23. M. El Hamzaoui, A. Sekkaki, B. Bensassi, Policy-Based Management of the inter-Domain communications Security, IEEE/IFIP 4th Latin American Network Operations and Management Symposium (LANOMS), Porto Alegre, Brazil, 29-31 August 2005, pp. 269-274.

---

**Article received:** 2006-06-13