

2 Introduction

Nous allons détailler dans ce chapitre les outils et les bibliothèques utilisées pour réaliser notre simulateur «SimRF-Auth ». Pour cela, nous avons basé sur la conception détaillée dans le chapitre précédent.

1. Outils et langages utilisés

1.1. Java :

Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C. Ses caractéristiques ainsi que la richesse de son écosystème et de sa communauté lui ont permis d'être très largement utilisé pour le développement d'applications de types très disparates. Java est notamment largement utilisé pour le développement d'applications d'entreprise et mobiles. [Dou10]

1.2. EDI NetBeans :

Afin d'implémenter notre simulateur, nous avons utilisé NetBeans comme EDI (Environnement Développement intégré). L'EDI NetBeans est un environnement de développement - un outil pour les programmeurs pour écrire, compiler, déboguer et déployer des programmes. Il est écrit en Java - mais peut supporter n'importe quel langage de programmation. Il y a également un grand nombre de modules pour étendre l'EDI NetBeans. L'EDI NetBeans est un produit gratuit, sans aucune restriction quant à son usage. [Ne11]

1.3. JCE (Java Cryptography Extension) :

Le Java TM Cryptography Extension (JCE) fournit un cadre et des implémentations de chiffrement, génération de clés et d'accord de clé, et Message code d'authentification (MAC) des algorithmes. Support du cryptage comprend symétriques, asymétriques, bloc, et de chiffrement à flot. Le logiciel supporte aussi les flux sécurisés et les objets scellés.

JCE était auparavant un package optionnel (extension) à la Java TM SDK 2, Standard Edition (Java 2 SDK), les versions 1.2.x et 1.3.x. JCE a maintenant été intégrée dans le SDK Java 2, v 1.4. [ArCr]

1.4. Système de gestion de base de données (SGBD) :

Pour la manipulation de l'ensemble des tables constituant notre base de données, nous avons choisi MySQL Administrator 1.2.16 comme système de gestion de base de données.

MySQL est un système de gestion de base de données (SGBD). Selon le type d'application, sa licence est libre ou propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle, Informix et Microsoft SQL Server. [MyAd]

MySQL AB a été acheté le 16 janvier 2008 par Sun Microsystems pour un milliard de dollars américains [My08]. En 2009, Sun Microsystems a été acquis par Oracle Corporation, mettant entre les mains d'une même société les deux produits concurrents que sont Oracle Database et MySQL. Ce rachat a été autorisé par la Commission européenne le 21 janvier 2010 [UEOr, EuCU].

Depuis mai 2009, son créateur Michael Widenius a créé MariaDB pour continuer son développement en tant que projet Open Source.

1.5. MySQL-Connector-java-3.1.12 :

MySQL fournit la connectivité pour les applications clientes développées dans le langage de programmation Java par le biais d'un pilote JDBC, qui est appelé MySQL Connector / J.

MySQL Connector/J est un pilote JDBC Type 4. Différentes versions sont disponibles qui sont compatibles avec la version 3.0 et JDBC JDBC spécifications 4.0. Le Type 4 désignation signifie que le pilote est pure-Java mise en œuvre du protocole MySQL et ne s'appuie pas sur les bibliothèques clientes MySQL.

Pour les grands programmes qui utilisent des modèles de conception communes d'accès aux données, pensez à utiliser l'un des frameworks de persistance populaires tels que Hibernate , modèles Spring JDBC ou iBATIS Cartes SQL afin de réduire la quantité de code JDBC pour vous à déboguer, air, sécurisé, et maintenir. [MyC]

Cette section n'est pas conçue pour être un tutoriel complet JDBC. Si vous avez besoin de plus amples renseignements sur l'utilisation de JDBC, vous pourriez être intéressé par les tutoriels en ligne suivants qui sont plus en profondeur que l'information présentée ici:

- Notions de base JDBC : un tutoriel de Sun couvrant débutants sujets dans JDBC
- JDBC Short Course : Un examen plus approfondi tutoriel de Sun et JGuru

2. Pseudo-code principal:

2.1. Fonction de hachage :

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public static String hashage(String msg,String type) {
    // type=="MD5" or "SHA1": type méthode d'utilisé dans le
    cryptage
    byte[] uniqueMsg = msg.getBytes();
    byte[] hash = null;
    try {
        hash =
        MessageDigest.getInstance(type).digest(uniqueMsg);
    } catch (NoSuchAlgorithmException e) {
        System.out.print(e.getMessage());
    }
    StringBuffer hashString = new StringBuffer();
    // convert le resultat vers texte
    for ( int i = 0; i < hash.length; ++i ) {
        String hex = Integer.toHexString(hash[i]);
        if ( hex.length() == 1 ) {
            hashString.append('0');
            hashString.append(hex.charAt(hex.length()-1));
        } else {
            hashString.append(hex.substring(hex.length()-2));
        }
    }
    return hashString.toString();
}
```

2.2. Nonce (Générateur de nombre pseudo aléatoire)

```
import java.util.Random;
public static int Nonce(){
    Random r1 = null;
    r1=new Random();
    int s1=    r1.nextInt(1000000000);
    return s1;
}

//la valeur de nonce entre (1 et 0);
public static int Nonce0_1(){
    Random r1 = null;
    r1=new Random();
    int s1=    r1.nextInt(2);
    return s1;
}
```

2.3. Concaténation :

```
public static String concatenation(String x1,String x2){
    return x1+x1;
}
```

2.4. Opérateur ou-exclusif :

```
import java.math.BigInteger;
public static int Xors(int x1,int x2){
    return x1^x2;
}

public static BigInteger Xorsbig(BigInteger
x1,BigInteger x2){
    return  x1.xor(x2);
}
```

3. Algorithme de simulation des activités (ex. protocole LAK):

3.1. Algorithme:

```

Debut
|   Initialisations ( ) ;
|
|   Tant que N-Activité < N-max-Activité Faire
|   |   Extraire l'activité de l'ETAT-SIMULATION;
|   |
|   |   Suivant (Activité) Faire
|   |   |   READER: READER();
|   |   |   ATTAQUE: ATTAQUE();// dans Simulation d'Attaque
|   |   |   TAG: TAG();
|   |   |   SERVER: SERVER();
|   |   Fin cas
|   Fin tant que
|
|   Analyser le temps pour chaque activité;
|   Imprimer_Resultats( ) ;
Fin

```

3.2. Programme :

```

final Thread th2=new Thread (new Runnable() {
    public void run() {
        try {
            for (int i = 0; i < 7; i++) {
                switch(ETATSIM){
                    case 0: System.exit(0);
                    case 1: {jTable2.getModel().setValueAt("READER",i,0);
                        //Imprimer_Resultats( ) ;
                        // Affiche le temps en nano seconde
                        long start = System.nanoTime();
                        jTable2.getModel().setValueAt(start, i, 1);
                        rader();
                        long finish = System.nanoTime();
                        jTable2.getModel().setValueAt(finish, i, 2);
                        long durationA = finish - start;
                        Table2.getModel().setValueAt(durationA,i,3);
                        Thread.sleep(5000);
                        break; }
                    case 2: {jTable2.getModel().setValueAt("TAG", i, 0);
                        long start = System.nanoTime();
                        jTable2.getModel().setValueAt(start, i, 1);
                        TAG();
                        long finish = System.nanoTime();
                        jTable2.getModel().setValueAt(finish, i, 2);
                        long durationA = finish - start;
                        jTable2.getModel().setValueAt(durationA, i, 3);
                        Thread.sleep(6000);
                        break; }
                }
            }
        }
    }
});

```

```

        case3:{jTable2.getModel().setValueAt("SERVER",i,0);
            long start = System.nanoTime();
            jTable2.getModel().setValueAt(start, i, 1);
            server();
            long finish = System.nanoTime();
            jTable2.getModel().setValueAt(finish, i, 2);
            long durationA = finish - start;
            jTable2.getModel().setValueAt(durationA,i,3);
            Thread.sleep(3500);
            break; }
        case 4: { // dans Simulation d'Attaque
            jTable2.getModel().setValueAt("TAG",i,0);
            long start = System.nanoTime();
            jTable2.getModel().setValueAt(start, i, 1);
            ATTAQUE();
            long finish = System.nanoTime();
            jTable2.getModel().setValueAt(finish, i, 2);
            long durationA = finish - start;
            jTable2.getModel().setValueAt(durationA,i,3);
            Thread.sleep(3000);
            break; }
        default:i=7;        } }
    } catch (InterruptedException ex) {

Logger.getLogger(Protocole_LAK.class.getName()).log(Level.SEVERE, null,ex);
    }
    throw new UnsupportedOperationException("Not supported yet.");
}

});

```

4. Algorithme de simulation des événements pour plusieurs clients (protocole CRAP):

Structure de données:

Event_List = liste d'enregistrements composés de :

- Event_Type \in { Arrivee, Debut_Service, Depart };
- Tevent : reel.

On utilise deux listes de chaînes:

- Waiting_List: Contient les clients qui arrivent;
- Finished_List: Contient les clients ayant terminés le service.

Algorithme:

```

Debut
Initialisations ( ) ;
Tant que Ntag < Nmax Faire
Extraire le premier événement de l'Event_List ;
Suivant (Event_Type) Faire

```

```
Arrivee : Arriver( ) ;
Debut_service : Debuter_Service( ) ;
Depart : partir( ) ;
Fin cas
Fin tant que
Imprimer_Resultats( ) ;
Fin

Initialisations ( )
Debut
    Insérer (arrivée, 0 ) dans l'Event_List ; /* mise en place du premier
    évènement primaire*/
    t ← 0 ; /* initialisation de l'horloge de simulation */
    q ← 0 ; /* nombre de clients en attente */
    Nmax /* nombre maximum de clients à traiter */
    qmax ← 0 ; /* nombre maximum de clients en attente dans la queue */
    Ntag ← 0; /* compteur des clients déjà servis */
    Totinact ← 0 ; /* temps total d'inactivité du serveur */
Fin
Procédure Arriver ( ) ;
    Début
        t ← Tevent ;
        Tarr ← t + temps_inter_arrivées;
        Insérer (Arrivée, Tarr) dans L'Event_List;
        q ← q + 1;
        Si q > qmax alors qmax ← q Finsi
        Si Serveur_libre alors
            Insérer (Début_Service, t) dans L'Event_List;
        Finsi
    Fin.
Procédure Débuter_Service ( ) ;
    Début
        t ← Tevent ;
        Serveur_libre ← faux ;
        q ← q - 1;
        Tdep ← t + estimation du temps de service;
        Insérer (Départ , Tdep) dans L'Event_List ;
        Si (t - Tinact ) ≠ 0 alors
            Totinact ← Totinact + (t - Tinact) ;
        Finsi
    Fin.
Procédure Partir ( ) ;
    Début
        t ← Tevent ;
        Serveur_libre ← vrai ;
        Tinact ← t ;
        Nclient ← Nclient + 1 ;
        Si q ≠ 0 alors
            insérer (Début_Service, t ) dans L'Event_List ; Finsi    Fin.
```

5. Implémentation d'application SimRF-Auth:

5.1. Structure du programme :

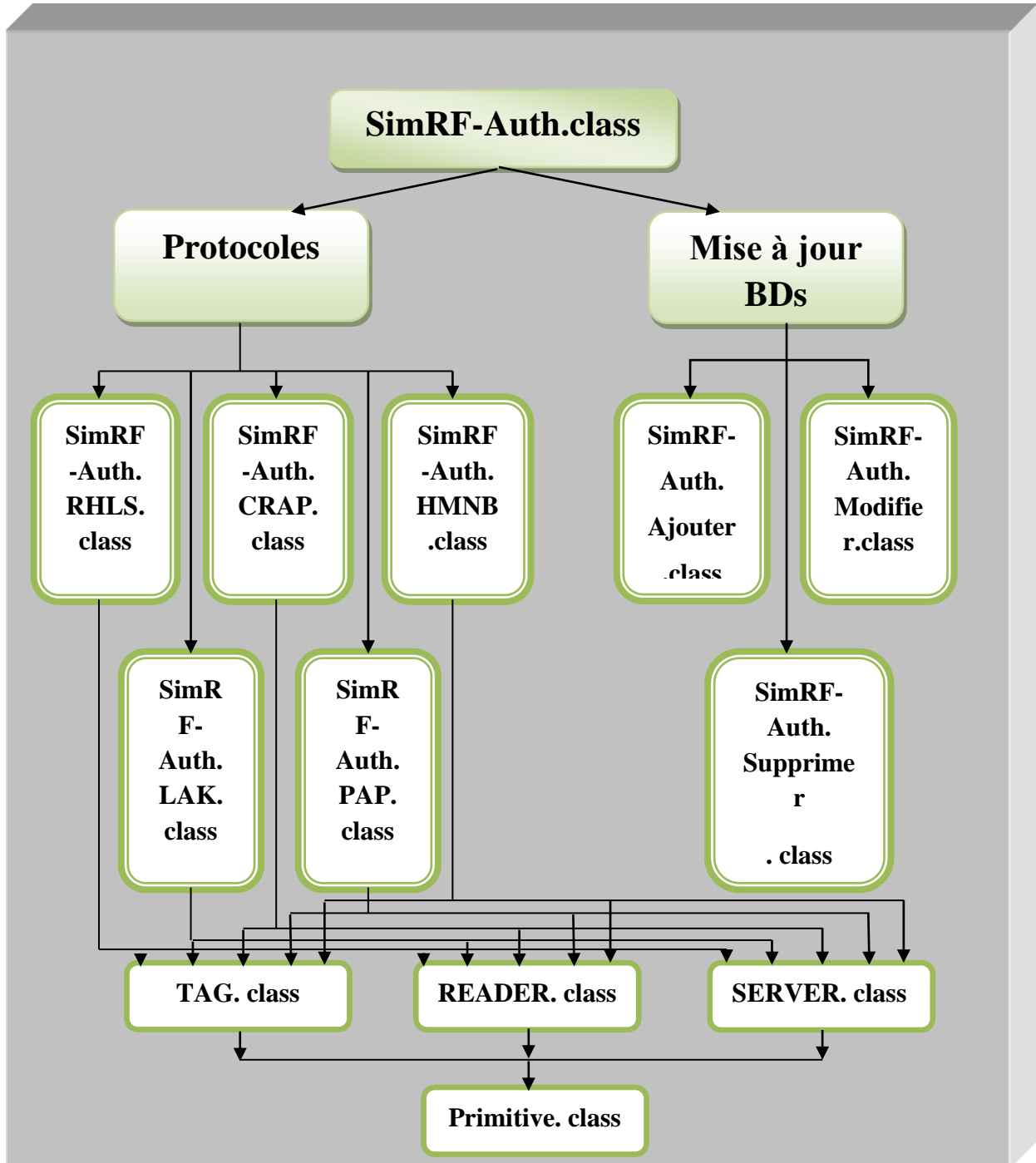


Figure IV.1 : Schéma de relation entre les objets principaux dans la simulation des protocoles.

5.2. Quelques interfaces des modules du système implémenté SimRF-Auth:

- Fenêtre Principale :

Cette fenêtre présente les différentes tâches qui sont exécutés dans le système



Figure IV.2. Fenêtre Principale de système SimRF-Auth.

- Fenêtre de simulation de protocole LAK :

Cette fenêtre concerne en suite de tâche de la simulation du protocole LAK sans attaque.

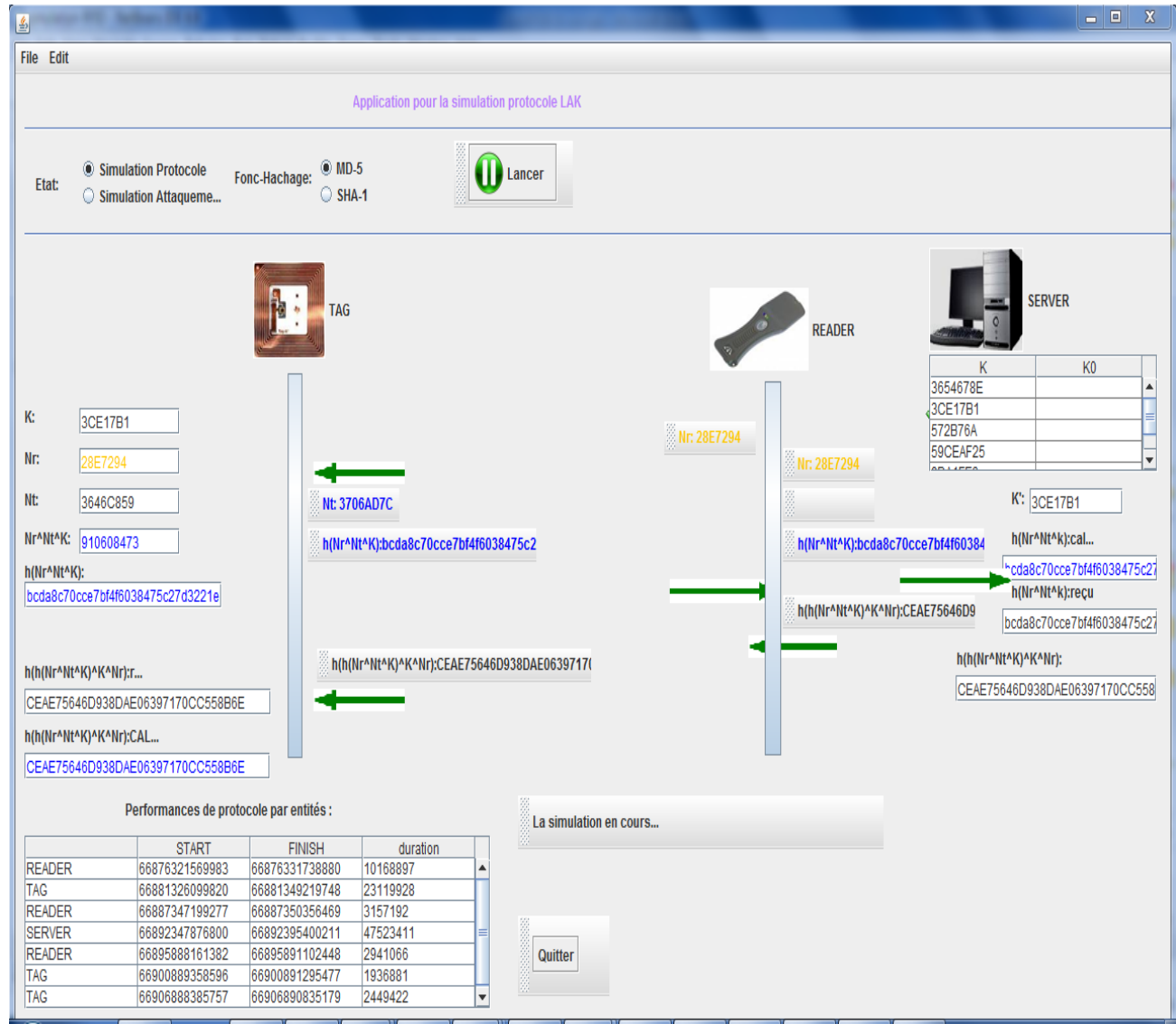


Figure IV. 3 : Fenêtre de protocole LAK « sans attaque ».

- Fenêtre de simulation d'attaque de protocole LAK :

Cette fenêtre concerne en suite de tâche de la simulation du protocole LAK dans l'état d'attaque.

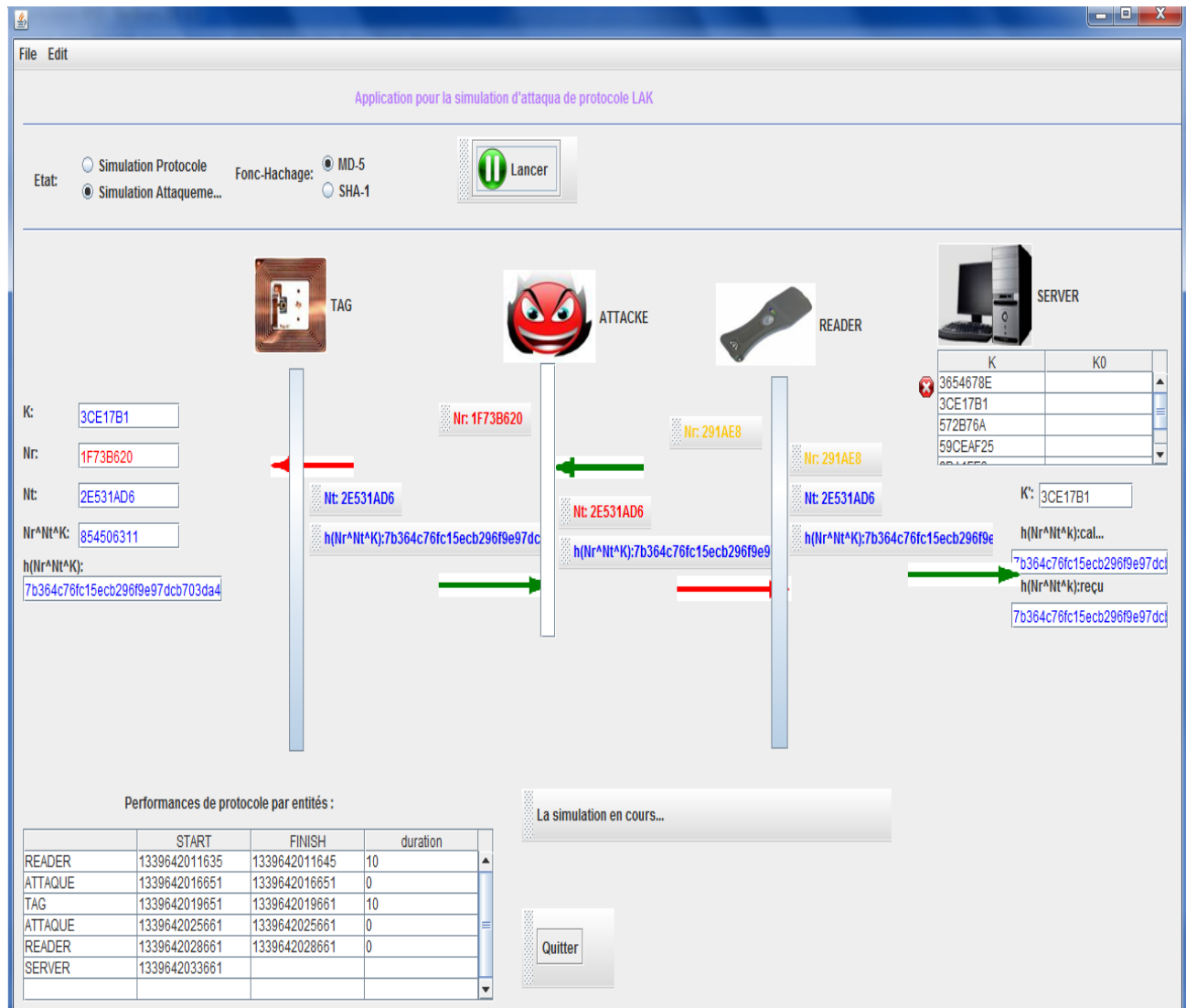


Figure IV.4 : Fenêtre de protocole LAK « avec attaque ».

- Fenêtre de mise à jour des BDs de protocole LAK :

Cette fenêtre concerne à la mise à jour des BDs du protocole LAK (ajouter, modifier et supprimer).

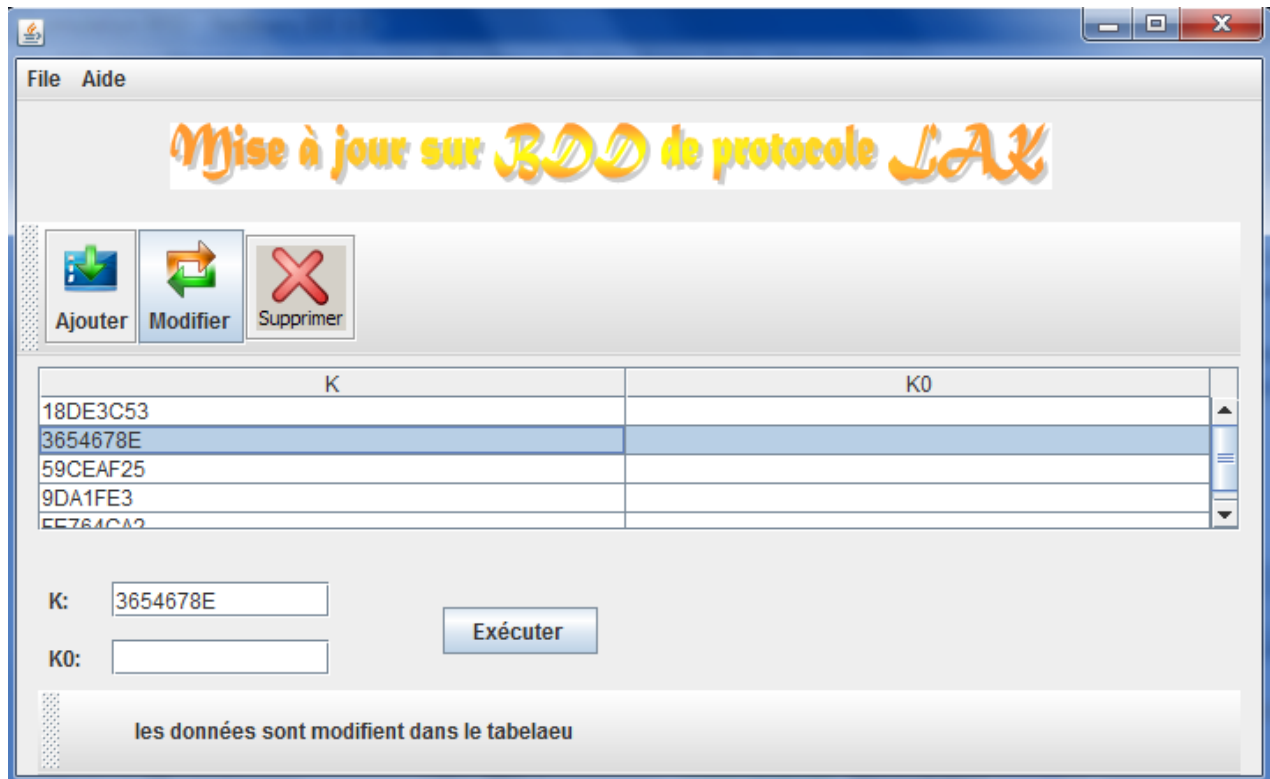


Figure IV. 5 : Fenêtre de la mise à jour des BDs pour le protocole LAK .

- Fenêtre de performances de protocole LAK par entité:

Performances de protocole par entités :				
	START	FINISH	duration	
READER	62372234143213	62373579611277	1345468064	▲
TAG	62378586397399	62379189904735	603507336	
READER	62385185177547	62385814443001	629265454	
SERVER	62390816841791	62397444521251	6627679460	
READER	62400941249999	62402519705326	1578455327	
TAG	62407525373442	62408571692851	1046319409	
				▼

Figure IV. 6 : performances de protocole LAK par entité

- Fenêtre de la simulation d'arrive de plusieurs TAG sur le protocole CRAP.

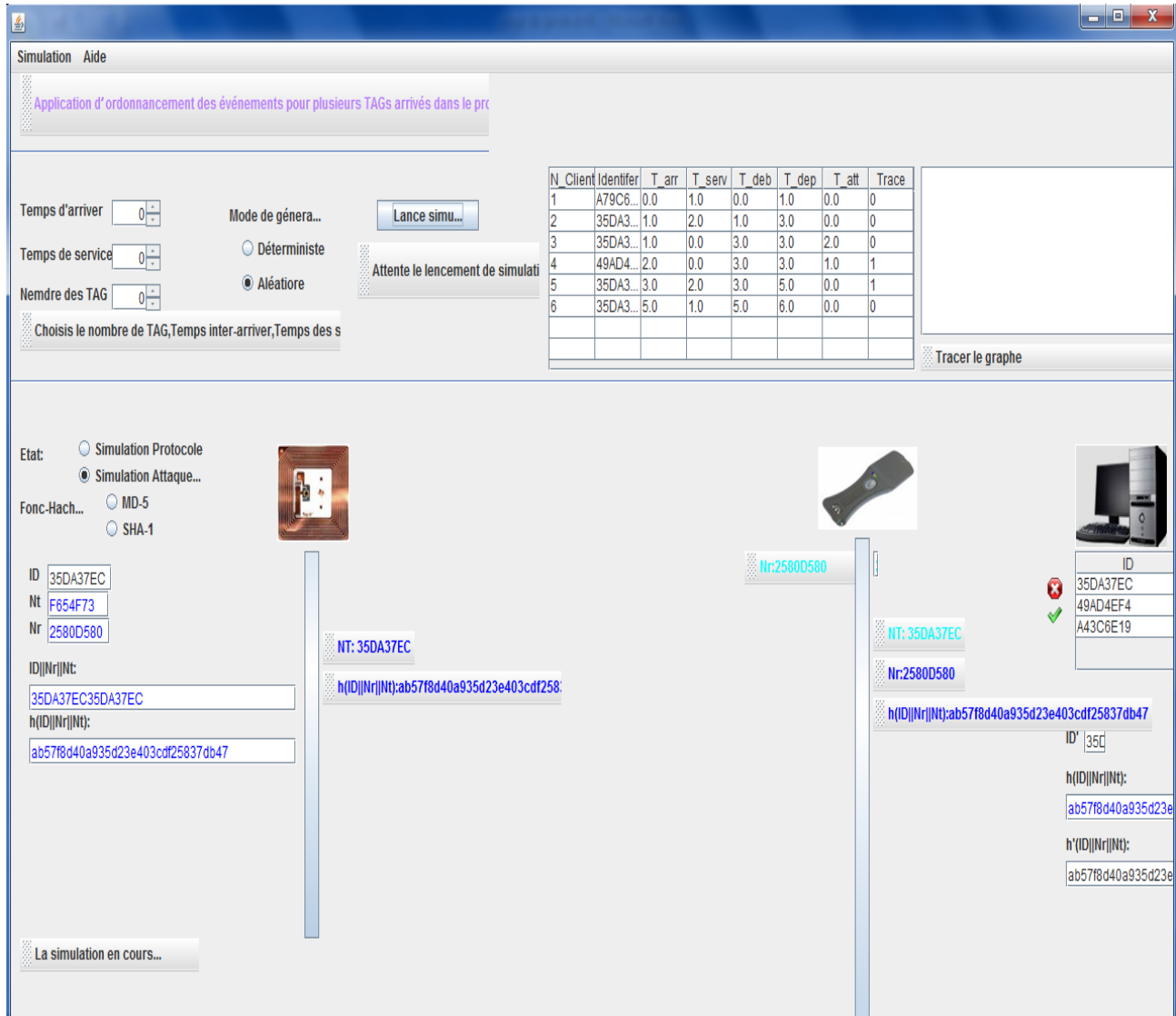


Figure IV. 7 : fenêtre d'ordonnancement de protocole CRAP

- Fenêtre de la simulation d'arrive de plusieurs TAG sur le protocole CRAP (avec attaque).

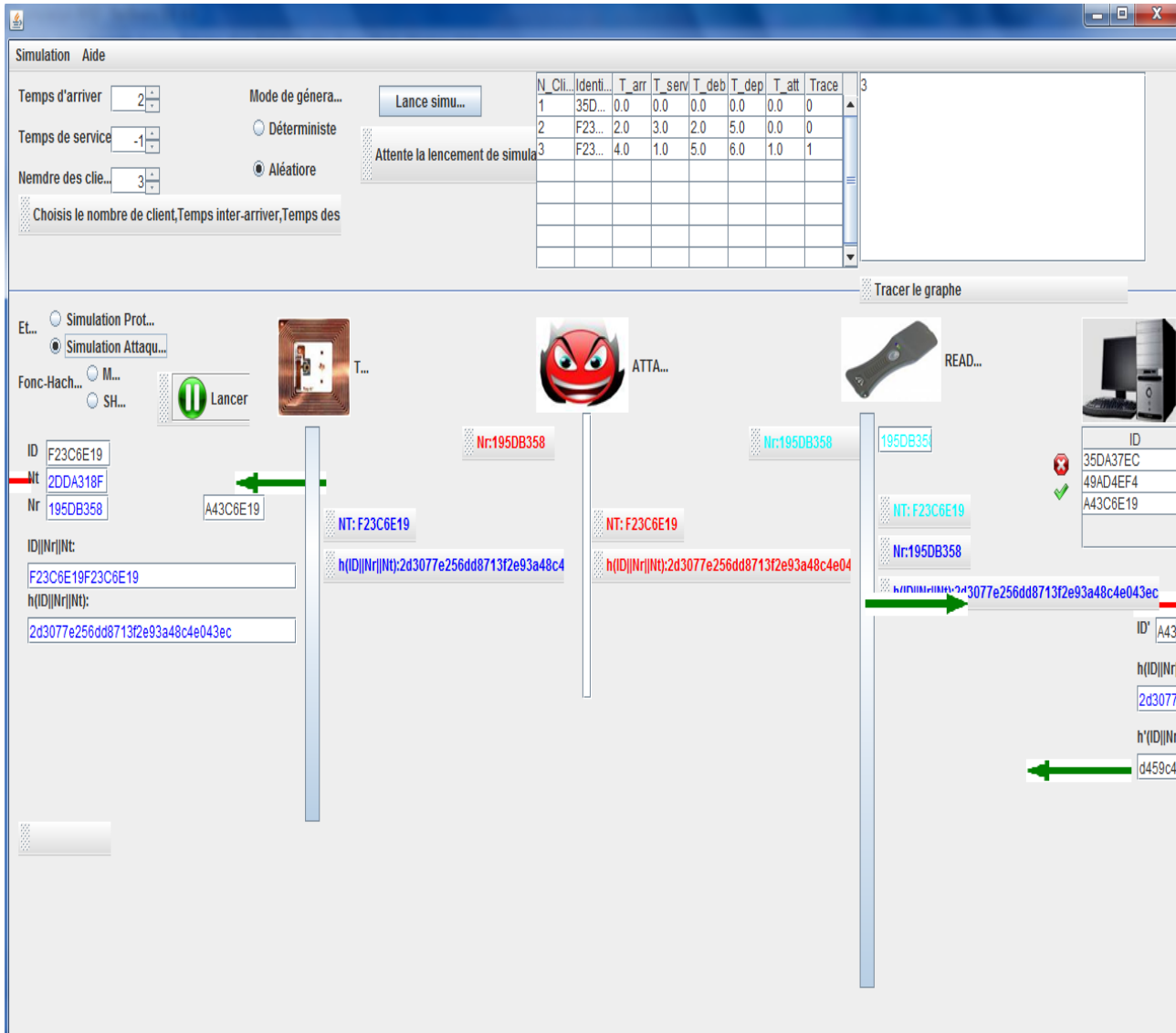


Figure IV.8 : fenêtre d'ordonnancement de protocole CRAP (avec attaque)

6. Résultats expérimentaux :

- Ce Tableau présente le temps d'exécution des différentes opérations du protocole LAK qu'utilise dans Reader, tag et server.

Primitive	GNPA		Concaténa- tion	Xor		Fonction de hachage	
	Nonce	NonceID		Xori		SHA-1	MD5
Temps(ns)	3855063	40695	791910	9349	11548	474596	643977

Tableau IV. 1 : performances de primitive.

- Ce Tableau représente l'ordonnancement des événements pour plusieurs TAGs arrivés dans le protocole LAK

N_Ta g	Identifier	T_Arri ve	T_Servic e	T_ débu t	T_dépar t	T_Attent e	N_C_At t
1	49AD4EF4	0.0	1.0	0.0	0.1	0.0	0
2	35DA37EC	2.0	3.0	2.0	5.0	0.0	0
3	23C8E11C	4.0	2.0	5.0	7.0	1.0	1
4	35DA37EC	6.0	4.0	7.0	11.0	1.0	1
5	F23C6E19	8.0	4.0	11.0	15.0	3.0	1
6	49AD4EF4	10.0	3.0	15.0	18.0	1.0	2

Tableau IV. 2 l'ordonnancement des événements sur le protocole LAK.

- Le tableau ci-dessous représente le temps(ns) d'exécution des différents protocoles par l'entité avec l'ordre de l'identificateur est 1 la fonction de hachage utilisé est MD5.

Entité Protocole	Reader	Server	Tag	Total
LAK	15748014	44938704	21317232	82003950
RLHS	1291253	42686160	7278425	51255838
HMNB	42668562	53911487	28991612	125571661
CRAP	30346658	69421927	23037439	122806024
PAP	10320130	56831655	20170063	87321848

Tableau IV. 3 Temps d'exécution des protocoles d'authentification (en nanoseconde)

- Le tableau ci-dessous représente le temps(ns) d'exécution des différents protocoles par l'entité avec l'ordre de l'identificateur est 3 la fonction de hachage utilisé est MD5.

Entité \ Protocole		Reader	Server	Tag	Total
LAK	Sans Attaque	16514079	50899478	23627520	91041077
	Avec Attaque	14108953	52114290	12753604	78976847
RLHS		3180840	56344960	25658440	85184240
HMNB		41580235	62663194	29732379	133975808
CRAP		15343261	83054882	13916172	112314315
PAP		14147146	78202233	21024665	113374044

Tableau IV.4 Temps d'exécution des protocoles d'authentification (en nanoseconde)