

# SAX: A Privacy Preserving General Purpose Method applied to Detection of Intrusions

François Trusset, Pascal Poncelet, Florent Masegla

► **To cite this version:**

François Trusset, Pascal Poncelet, Florent Masegla. SAX: A Privacy Preserving General Purpose Method applied to Detection of Intrusions. ACM First International Workshop on Privacy and Anonymity for Very Large Datasets, join with CIKM 09, Nov 2009, Hong Kong, China. pp.17-24, 2009. <lirmm-00430646>

**HAL Id: lirmm-00430646**

**<https://hal-lirmm.ccsd.cnrs.fr/lirmm-00430646>**

Submitted on 9 Nov 2009

**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.

# SAX: A Privacy Preserving General Purpose Method applied to Detection of Intrusions

François Trouset  
LG12P - Ecole des Mines  
d'Alès, Parc Scientifique G.  
Besse, 30035 Nîmes, France  
francois.trousset@ema.fr

Pascal Poncelet  
LIRMM UMR CNRS 5506, 161  
Rue Ada, 34392 Montpellier  
Cedex 5, France  
poncelet@lirmm.fr

Florent Masseglia  
INRIA Sophia Antipolis, route  
des Lucioles - BP 93, 06902  
Sophia Antipolis, France  
florent.masseglia@sophia.inria.fr

## ABSTRACT

To overcome the problem of attacks on networks, new Intrusion Detection System (IDS) approaches have been proposed in recent years. They consist in identifying signatures of known attacks to compare them to each request and determine whether it is an attack or not. However, these methods are set to default when the attack is unknown. However, it is frequent that an attack has already been detected by another organization and it would be useful to be able to benefit from this knowledge to enrich the database of signatures. Unfortunately this information is not so easy to obtain. In fact organizations do not necessarily want to spread the information that they have already faced this type of attack. In this paper we propose a new approach to intrusion detection in a collaborative environment but by preserving the privacy of the collaborative organizations. Our approach works for any signature even if it needs a complex program to be detected and insure that no information is disclosed on the content of any of the sites. For this purpose, we have developed a general method (SAX) that allows to compute any algorithm while preserving privacy of data and also of the program code which is computed.

## Categories and Subject Descriptors

D.4.6 [Security and Protection]: Algorithms, Privacy

## General Terms

Algorithms, Security

## Keywords

Intrusion Detection, Attacks, Collaborative Organizations, Privacy

## 1. INTRODUCTION

The fast growing computational Grid environments has increased risk of attack and intrusion. Thus misuse detec-

tion has become a real concern for companies and organizations. Whereas earlier attacks focused on Web servers (often misconfigured or poorly maintained), the most recent ones take advantage of Security service and Web application weaknesses which become more vulnerable [4, 2]. To overcome this problem, new approaches called Intrusion Detection Systems (IDS) have been developed. Installed on networks, they aim to analyze traffic requests and detect malicious behavior (eg Prelude-IDS, Snort). They can be classified into two broad categories (*e.g.* [8, 9]): the *Anomaly Detection Systems* which attempt to detect attacks and the *Abuse Detection Systems* which detects unknown comportement so called *abuse* from a specification of allowed ones.

Recently approaches called Collaborative Intrusion Detection Systems (CIDS) (*e.g.* [1, 12, 5, 7, 11]) have been proposed. In comparison with isolated IDS, CIDS significantly improve time and efficiency of misuse detections by sharing information on attacks between distributed IDS from one or more organizations. The main principle of these approaches is to exchange information using peer to peer links. However the exchanged information are mostly limited to IP addresses of requests (*e.g.* [1, 5, 7]) and consider that data can be freely exchanged among the peers. The last constraint is very strong: companies, for reasons of confidentiality, do not want to spread out that they were attacked and therefore are unwilling to give any information on it.

In this article we propose a secure collaborative detection approach, called SAX (*Secure Algorithm eXecution*), which ensures that private data and programs will not be disclosed. Via our approach, any program from the various collaborative sites can be executed without disclosing any information from the local IDS to the outside. Collaborative Sites are free to work with signatures of attacks or non-attacks and may give information on the type of intrusion detected. Thus, when new request is checked, the response will be one of: it is an attack (with its type if available), it is a non-attack, or unknown (if none of the IDS data leads to a positive or negative conclusion). To our knowledge, very little studies are concerned with this topic of security in such collaborative environment. The only works [10, 7] consider both collaborative and security aspects. In its context, security mainly concerns information on IP addresses and ports. It uses Bloom's filters to manage data exchanges. Our problem is different in that, we want to exchange data, *i.e.* more complex than IP addresses and ports. In fact we want to be able to exchange and execute any algorithm of anomaly detection on the full request.

In previous studies, we focused on those anomalies whose

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM'09, November 2–6, 2009, Hong Kong, China.  
Copyright 2009 ACM 978-1-60558-512-3/09/11 ...\$10.00.

signature detection may be expressed as a regular expressions. But there is still a lot of cases that cannot be expressed like that and which need specific programs to be detected. Instead of handling them one by one, we decided to study a general purpose method able to execute any program while preserving privacy of data and also of the program code (once it has been translated into the adequate formalism). This method is described in this article.

The article is organized as follows. In section 2, we present the problem. An overview of our approach is given in section 3 and 4. The various algorithms are described in section 5. Finally section 6 concludes and presents various perspectives.

## 2. PROBLEM STATEMENT

$DB$  is a database such as  $DB = DB_1 \cup DB_2 \dots \cup DB_D$ . Each database  $DB_i$  is equivalent to a tuple  $\langle id, S_{Alg} \rangle$  where  $id$  is the identifier of the database and  $S_{Alg}$  is a set of anomaly detection programs expressed as a finite state automaton and initial program data. The details are described in section 3. The other part of input data of the program is the request string  $R$  owned by the client site  $S$ . Each program will provide two kinds of value: a three states flag (*True/False/Unknown*) specifying whether the request is an attack or not or has not been identified at all and an integer (the type of the attack) when the request is effectively identified as an attack (*True* state).

**DEFINITION 1.** Given a database  $DB = DB_1 \cup DB_2 \dots \cup DB_D$  and a request string  $R$ , the secure approach in such a collaborative environment consists in finding a program  $Alg$  from  $DB$  such that execution of  $Alg$  upon  $R$  gives a False or True result (identify an attack or a non-attack) while ensuring that the client site does not provide the request string  $R$  to anyone and that none of the databases  $DB_i$  provided any information from its content to anyone (programs and data).

## 3. THE SAX APPROACH

This section will provide an overview of the secure architecture SAX (Secure Algorithm eXecution). It is designed to answer the problem of privacy preserving in a collaborative environment. Inspired by the work of [6], this architecture offers the advantage of achieving the various operations while ensuring that neither party may have access to private data contained in the initial databases. In addition to the client site  $S$  which is responsible to provide the request to be tested, the architecture requires four non-collaborative and semi honest sites [3]: they follow the protocol correctly, but are free to use the information they have collected during the execution of the protocol. These independent sites collect, store and evaluate information in a secure way. The different functions provided by these sites are:

- **The Control Site  $CTRL$ :**  $CTRL$  is used to rule the various operations needed to execute the program. To do this, it interacts with the two non colluding sites  $NC_1$  and  $NC_2$ .
- **Non Colluding Sites  $NC_1$  and  $NC_2$ :** These two symmetric sites collect garbled data from all databases as well as the garbled request to be tested from  $S$ .

Under the control of  $CTRL$  and by interaction with  $PS$ , they perform several secure operations in order to insure that none of them will be able to infer any of the intermediate results or the final result which is returned to site  $S$ .

- **The Processing Site  $PS$ :** This site is used both by  $NC_1$  and  $NC_2$  to process, in a secure way, the various needed operations. Like  $NC_1$  and  $NC_2$ ,  $PS$  also cannot deduce any pertinent value of intermediate or final result from the data it processes.

**Remark:** The data kept by  $NC_1$  ( $\bar{D}$ ) and  $NC_2$  ( $\bar{D}$ ) are random data such that  $\bar{D} \oplus \bar{D} = V$  where  $V$  is the real value. But as  $NC_1$  and  $NC_2$  are non-colluding, the value of  $V$  will never be known by any of the sites. (Details will be explained in the following sections).

### 3.1 Encoding of the program

To perform a secure computation of client site  $S$  data by a program coming from an external site  $DB_i$  (which may be anonymated), we consider the program as a deterministic automaton  $\langle State, Trans, Init, Final \rangle$  where  $State$  is the set of states of the automaton,  $Init$  is the initial state,  $Final$  is the set of final states and  $Trans$  is the set of transitions. Each transition is a quadruplet  $(S_{Initial}, Action, S_{Final0}, S_{Final1})$  meaning that if the automaton is in state  $S_{Initial}$  and that  $Action$  is performed and returns a boolean value then the automaton's current state changes to  $S_{Final0}$  if the boolean value is false or to  $S_{Final1}$  if it is true. The automaton starts with state  $Init$  and ends when reaching any of the final states from  $Final$ .

**Remark:** To ensure that computation is secure, none of the performed action shall return any data to the site  $DB_i$  which gave the program to be computed. This means that only the client site  $S$  gets results.

This encoding implies that all of the states (but final ones) are connected to two other states in the graph of the automaton (in fact  $S_{Final0}$  and  $S_{Final1}$ ). These links are designated by  $\ominus$  leading to  $S_{Final0}$  and  $\oplus$  leading to  $S_{Final1}$ .

Each action is in fact a list of 3-uplets  $(OpNum, Aff_1, Aff_2)$  where  $OpNum$  is the number of the operation to be processed, and  $Aff_1$  and  $Aff_2$  are two random sequences of bits of of same length. Each 3-uplet of the list performs the same action by using different operators which will be effectively executed by  $NC_1$  and  $NC_2$ . There is a list to avoid  $NC_1$  and  $NC_2$  from deducing any link between the value of  $OpNum$  and a "corresponding state" in  $CTRL$ .

The automaton is managed by  $CTRL$  while  $NC_1$ ,  $NC_2$  maintain the memory and performs the operations on the values through  $PS$ . If  $CTRL$  maintains the list of operation numbers to be performed for each node, the protocol to be performed when executing this operation is aimed by  $NC_1$  and  $NC_2$  and data used as input and output of this operation will only be kept (garbled) in  $NC_1$  and  $NC_2$ . Thus  $CTRL$  will know nothing about the real operation that is performed and no more on the values used and produced by the operation.  $NC_1$  and  $NC_2$  also owns a list of *Registers* which contains address in the memory. These register will be used to perform indirect and based memory access. Instead of the data kept in their memory which are garbled, the real values of registers are known by both  $NC_1$  and  $NC_2$ .

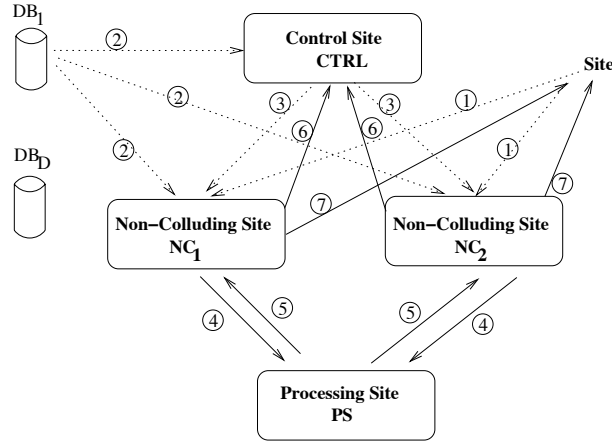


Figure 1: General Architecture of SAX

After Performing the *Action* associated to a state, *CTRL* receives two bits (one from *NC1* and one from *NC2*). If these two bits are identical then *CTRL* follows the link  $\oplus$  otherwise it follows the link  $\ominus$ .

For each operation *OpNum* of *CTRL*, *NC1* and *NC2* have a corresponding 5-uplet (*Protos*, *NCA*, *Arg1*, *Arg2*, *Res*, *Incr*) where *Protos* is a list of binary operations (*p* elements), *NCA* is a random number ( $1+m+n+p+q$  bits), *Arg1* is the list of values from the memory that may potentially be used as first argument of operation (*n* elements), *Arg2* is similar to *Arg1* but for the second argument (*m* elements), *Res* is the list of potential storages for the result in the memory (*q* elements), and *Incr* is the list (*r* elements) of couples (*Register*, *IncrVal*) giving increments to perform on registers at the end of the operation. In fact the *Arg1*, *Arg2* and *Res* are couples (*Position*, *Length*) where *Length* is the number of bits of the value and *Position* is either a position in the memory or a register number *Reg* and an *Offset* to perform indirect or based memory addressing.

**Remark:** there is several ways to implement an action that does only register increments, but it also exists a non operation prototype named *NOPS* designed for this purpose.

**Remark:** *Aff1*, *Aff2* are such that  $Aff = Aff1 \oplus Aff2 \oplus NCA \oplus NCA$  where  $NCA$  is the part of *NCA* owned by *NC1* and  $NCA$  is the part owned by *NC2*. *Aff* is a list of boolean flags (true/false) that are used to specify which affectation shall be effectively done (see section 3.3). We will use the notations  $Aff^+ = Aff1 \oplus NCA$  owned by *NC1* and  $Aff^- = Aff2 \oplus NCA$  owned by *NC2*.

In fact  $Aff1 = Aff^+ \oplus RND1$ ,  $Aff2 = Aff^- \oplus RND2$ ,  $NCA = RND1 \oplus RND3$  is owned by *NC1* and  $NCA = RND2 \oplus RND3$  is owned by *NC2* where *RND1*, *RND2* and *RND3* are generated by *DBi* and  $Aff = Aff^+ \oplus Aff^-$ .

### 3.2 Initialization of the processus and global processing

The exchange of data between the different sites is done by using the secure method  $SEND^S(\vec{v}|\vec{v}')$  which sends the vector of bits  $V = \vec{v} \oplus \vec{v}'$  to *NC1* and *NC2*. It is defined

in order to send  $\vec{v}'$  to *NC1* and  $\vec{v}$  to *NC2* (or vice versa). A random vector *R* is used to garble transmitted data such that  $\vec{v}' = R$  and  $\vec{v} = V \oplus R$  or vice versa. This method is used in particular to send the data from the databases *DBi* and to send the request from site *S*. Thus, the process described in figure 1 starts in the following way.

\* First, the site *S* sends its request to *NC1* and *NC2* using the  $SEND^S$  method (See arrow number 1 in figure 1). More precisely, the request *R* is taken in its boolean form: a vector of bits.

\* A random vector of bits *AR* is then generated with the same size as the request *R* to compute the new vector  $Z_R = AR \oplus R$ . *ZR* is sent to *NC1* and *AR* to *NC2* (or vice versa).

\* Each database *DBi* sends the transition matrix of its automaton to *CTRL* (See arrows number 2) The size of the needed memory, the number of needed address registers and their initial value are naturally sent to *NC1* and *NC2* as their real values are known by both sites. The initial value of the memory is sent to *NC1* and *NC2* by using  $SEND^S$ . *DBi* also sends the list of actions to be performed (5-uplets) to *NC1* and *NC2*.

From this point, the computation of the request is done under the control of *CTRL*. It will ask *NC1* and *NC2* execute action (See arrow number 3).

\* The action is effectively performed securely by *NC1* and *NC2* through *PS* (See arrow number 4 and 5).

\* The result of the action is a boolean divided in two parts, one is owned by *NC1* and the other by *NC2* such that none are able to infer its real value. Both parts are then securely returned to *CTRL* (see arrows 5 and 6)

\* While *NC1* and *NC2* are updating the value of the different registers as needed by the action, *CTRL* compares the two values he received and updates the current state of the automate in consequence.

\* The process is repeated under control of *CTRL* unless the automaton moves into a final state.

\* When entering a Final state, the result value previously stored by the program by executing the secure action  $AFFECT^S$  is aggregated or sent to the client site *S* by *NC1* and *NC2* (See arrow number 7) under the control of *CTRL*.

This choice depends of the configuration of the anonymity behaviour (See section 4).

### 3.3 Execution of action

When *CTRL* enter a non final state, he starts by randomly selecting one of the 3-uplets ( $OpNum$ ,  $Aff_1$ ,  $Aff_2$ ) from the list associated to this state. It then generates a random number  $ROP$  of same length than  $Aff_1$  and  $Aff_2$ . Then *CTRL* sends  $\bar{A}_R = Aff_1 \oplus ROP$  to  $NC_1$  and  $\bar{A}_R = Aff_2 \oplus ROP$  to  $NC_2$ .

$NC_1$  (respectively  $NC_2$ ) computes the vectors of bits  $\bar{A} = \bar{A}_R \oplus NC_1 \bar{C}_A$  (respectively  $\bar{A} = \bar{A}_R \oplus NC_2 \bar{C}_A$ ) and extract  $\bar{B}^0 = \bar{A}[0]$ ,  $\bar{B}^1 = \bar{A}[1 \dots n]$ ,  $\bar{B}^2 = \bar{A}[n + 1 \dots n + m]$ ,  $\bar{B}^3 = \bar{A}[n + m + 1 \dots n + m + p]$ ,  $\bar{B}^4 = \bar{A}[n + m + p + 1 \dots n + m + p + q]$  (and repectively  $NC_2$  extract then  $\bar{B}$  parts from  $\bar{A}$ ). We will use the notation  $\bar{B}_j^i$  for  $\bar{B}^i[j]$  which is the  $j^{\text{th}}$  bit of the vector of bits  $\bar{B}^i$  and similarly for the  $\bar{b}$  form.

Now let  $\bar{v}^1$  and  $\bar{v}^2$  (respectively  $\bar{v}^1$  and  $\bar{v}^2$ ) be two distinct vectors of bits of length greater or equal to the lengths on any of the values in  $\bar{A}_{rg}^1$  for  $\bar{v}^1$  and  $\bar{A}_{rg}^2$  for  $\bar{v}^2$  used to store the actual arguments of operations. We shall also use  $\bar{r}$  and  $\bar{b}$  (respectively  $\bar{r}$  and  $\bar{b}$ ) to store the computed results of the operation.

Now  $NC_1$  (respectively  $NC_2$ ) computes the following secure instructions (See notations in section 5):

```

foreach  $i \in [1..n]$  do  $AFFECT^s(\bar{v}^1, \bar{B}_i^1, \bar{A}_{rg_i^1} \bar{v}^1, \bar{B}_i^1, \bar{A}_{rg_i^1})$ ;
foreach  $i \in [1..m]$  do  $AFFECT^s(\bar{v}^2, \bar{B}_i^2, \bar{A}_{rg_i^2} \bar{v}^2, \bar{B}_i^2, \bar{A}_{rg_i^2})$ ;
 $(\bar{b} | \bar{r}) \leftarrow (\bar{B}^0 | \bar{B}^0)$ ;
foreach  $i \in [1 \dots p]$  do
   $(\bar{r}_X, \bar{b}_X | \bar{r}_X, \bar{b}_X) \leftarrow Proto_i(\bar{v}^1, \bar{v}^2 | \bar{v}^1, \bar{v}^2)$ ;
   $AFFECT^s(\bar{r}, \bar{B}_i^3, \bar{r}_X | \bar{r}, \bar{B}_i^3, \bar{r}_X)$ ;
   $AFFECT^s(\bar{b}, \bar{B}_i^3, \bar{b}_X | \bar{b}, \bar{B}_i^3, \bar{b}_X)$ ;
foreach  $i \in [1..q]$  do  $AFFECT^s(\bar{r}_{esi}, \bar{B}_i^4, \bar{r} | \bar{r}_{esi}, \bar{B}_i^4, \bar{r})$ ;
foreach  $i \in [1 \dots r]$  do
   $| Register[Reg_i] = Register[Reg_i] + IncrVal_i$ 

```

**Remark:** There exists more complex operators on indexes but incrementation is sufficient for our purpose as we shall care that operations on indexes are not private:  $NC_1$  and  $NC_2$  knows their real value. We shall also notice that operation on indexes shall never combine data stored in the memory otherwise  $NC_1$  and  $NC_2$  will gain access to the real values of data stored in their memory.

### 3.4 Proof of privacy

PROPERTY 1. *This implementation allows to compute securely any program while encoded in the correct form.*

*Proof:* The secure operations computed may be any boolean operation. We have encoded the operation and ( $\wedge^S$ ) and not ( $\neg^S$ ) that form a generator system of the booleans. That means that any other boolean operation may be encoded securely. If in theory, we consider the memory maintained by  $NC_1$  and  $NC_2$  to be infinite it is easy to encode any Turing Machine using *CTRL*,  $NC_1$ ,  $NC_2$  and *PS*. To do that we only need one *Register*, and to decompose each transaction of the Turing Machine in three single transactions of our automaton, the first consists in testing the value at the current (*Register*) position by using *COMPARE*<sup>S</sup>, the second

stores new data at the current position using *AFFECT*<sup>S</sup> and the third increments the *Register* with value 1 ou -1 using operator *NOP*<sup>S</sup>.

PROPERTY 2. *This secure execution of a program prohibits DB to know any of the client's data.*

*Proof:* The proof is evident as no data is returned to *DB*. On the client side, it may not be true but it depends on the program which is computed by the system which is of the responsibility of *DB*. Thus if the program deliver data coming from *DB* to the client, that means that *DB* (who writes the program) has decided that it shall be so and thus it is a normal result approved by *DB*.

PROPERTY 3. *CTRL does not know any of the incoming and outgoing data and does not even know what processing has been performed on the data.*

*Proof:* *CTRL* does not have access to any of the data managed by  $NC_1$  and  $NC_2$ . It also has only access to operation numbers and does not know to which protocols it corresponds. Further more the values of  $Aff_1$  and  $Aff_2$  are pure random values from which it cannot infer anything, even the length of those may be greater that the one really needed by  $NC_1$  or  $NC_2$ . The only thing that *CTRL* may infere is the sequence of states encountered during a run.

PROPERTY 4. *None of  $NC_1$ ,  $NC_2$  or  $PS$  can infere what processing has been performed.*

*Proof:* At each step, when executing an action,  $NC_1$  and  $NC_2$  perform several operation using various values from the memory and it does not know which of the values will be used as inputs of the operator, nor which result is stored in the memory no more than where it is effectively stored. This is due to the usage of the secure conditional assignment operator *AFFECT*<sup>S</sup>.

Secondly when  $NC_1$  is performing any boolean operation it does not know if  $NC_2$  negates or not any of the value or the result (because to implement the negation, only one from  $NC_1$  and  $NC_2$  has to negate the value which is a property of  $\oplus$  : if  $A = X \oplus Y$  then  $\neg A = \neg X \oplus Y = X \oplus \neg Y$

PROPERTY 5. *None of  $NC_1$ ,  $NC_2$  or  $PS$  can deduce any real value of the incoming or produced data.*

*Proof:*  $NC_1$  and  $NC_2$  own only garbled data at the beginning of the process. During the processing the usage of secure operators insure that they get only garbled data from witch they cannot infere any information on the real values. In the case of *PS* it does not owns any data and gets pure random data from  $NC_1$  and  $NC_2$  and returns random values. (See proof of operators for more details).

**Remark:** The values of  $m$ ,  $n$ ,  $p$ ,  $q$  and  $r$  arte chosen by  $DB_i$  when producing the automata. The cost in time is linearly increasing with the value of  $m + n + p + q + r$ . At the same time, the "randomness" of the resulting program increases by  $m * n * p * q * r$ . When encoding program  $DB_i$  must choose values such that time cost is bearable and garbling of the algorithm is the highest one.

## 4. ANONYMISATION

There is two main ways to do anonymization: in buffering inputs programs issued by databases  $DB_i$  and execute them randomly or in buffering results before sending them

randomly to the client. The first one essentially introduce costs in term of space (we need to store the automata in *CTRL* and associated data in  $NC_1$  and  $NC_2$ ). The second one essentially introduce time costs (we may process other  $DB_i$ 's programs while the desired result has already been obtained. And we will do that until the client gets this result). A third way is to mix the two approaches and adapt parameters (size of both buffers) to adjust anonymization process according to the needs and bearable costs.

As we are dealing with anomaly detection, the result is not so complex: an algorithm may detect an attack and then give the type of the detected attack, or identify a non attack request or may not be able to identify the request. In fact only the first positive or negative result has to be kept. Then we only need to keep one result in  $NC_1$  and  $NC_2$  in an *Accumulator*. Any new result may be aggregated in the accumulator by computing securely

**if** (*Accumulator* == *unknown*) **then** *Accumulator* = *Result* which may be easily computed by using the operators *COMPARE*<sup>S</sup> and *AFFECT*<sup>S</sup>. At the origin, the accumulator is set securely with the value "unknown". Thus, when anonymization is done by buffering results, we only get time costs, and the parameter is the number of aggregations to be performed before sending an aggregated result to the client.

## 5. THE SECURE ALGORITHMS

In this section, we present the various algorithms used in SAX approach. In order to simplify writing, we consider the following notations:

Let  $(\bar{x}|\bar{x}) \leftarrow h^S(\bar{y}_1 \dots \bar{y}_n | \bar{y}_1 \dots \bar{y}_n)$  be a tripartite computation of any function  $h^S$  between  $NC_1$ ,  $NC_2$  and  $PS$  where  $NC_1$  owns some of the entries  $\bar{y}_1 \dots \bar{y}_n$  and gets part of the result  $\bar{x}$  and similarly  $NC_2$  owns some of the entries  $\bar{y}_1 \dots \bar{y}_n$  and gets part of the result  $\bar{x}$ .

The final result is obtained by applying the binary operator XOR ( $\oplus$ ) between  $\bar{x}$  and  $\bar{x}$ . However, this does not mean that  $NC_1$  sends exactly  $\bar{y}_1 \dots \bar{y}_n$  to  $PS$  and receives the result  $\bar{x}$  from  $PS$ . In fact,  $NC_1$  garbles its inputs  $\bar{y}_1 \dots \bar{y}_n$  by adding random noise and gets  $\bar{y}'_1 \dots \bar{y}'_n$  which are securely sent to  $PS$ . Similarly,  $NC_2$  sends its garbled inputs to  $PS$ .

At the end of the process, both sites receives a part of garbled result from  $PS$  (respectively  $\bar{x}'$  and  $\bar{x}'$ ). This intermediate result may now be used as input of further computation.

We will also use the following simplifications:

1.  $g^S(\bar{x}, \bar{y} | \bar{x}, \bar{y}) \Leftrightarrow g^S(\bar{x} | \bar{x}; \bar{y} | \bar{y})$
2. Si  $h^S()$  is a 2 argument function then  $h^S(\bar{x}_1, \dots, \bar{x}_n | \bar{x}_1, \dots, \bar{x}_n)$  will correspond to  $h^S(h^S(\dots h^S(h^S(\bar{x}_1, \bar{x}_2 | \bar{x}_1, \bar{x}_2); \bar{x}_3 | \bar{x}_3) \dots); \bar{x}_n | \bar{x}_n)$

### 5.1 The algorithm *AFFECT*<sup>S</sup>

The operator *AFFECT*<sup>S</sup> implements a conditional affectation of value to a variable. The non secure equivalent operator *AFFECT* could be specified as follows:

```
AFFECT ( Var , Cond , Value )
⇔ if ( cond ) then Var ← Value;
   else Var ← Var;
fi
```

The secure implementation of this operation (which is shown in algorithm 1 ensure that no one can infer whether the affectation has effectively been done or not. That means that no one can predict the value stored in *Var* unless it already know the value of *Cond* (but  $NC_1$  and  $NC_2$  only knows random values).

The secure operator *AFFECT*<sup>S</sup> shall be used in the following way:

$AFFECT^S(v_{ar}^+, Cond^+, Value^+ | v_{ar}^-, Cond^-, Value^-)$ ;

where  $\dagger$  are owned by  $NC_1$  and  $\bar{\dagger}$  are owned by  $NC_2$ .

The implementation of *AFFECT*<sup>S</sup> require the secure operators

$\bigvee^S(\bar{s}_1, \bar{s}_2 | \bar{s}_1, \bar{s}_2) \rightarrow (\bar{v} | \bar{v})$  and  $\bigwedge^S(\bar{s}_1, \bar{s}_2 | \bar{s}_1, \bar{s}_2) \rightarrow (\bar{v} | \bar{v})$

which implements respectively a secure computation of bitwise operators OR and AND on vectors of bits of same length ( $S_1$  and  $S_2$ ) and returns the sequence  $V$ . At the end of the process of *AFFECT*<sup>S</sup>, the new value computed will be stored in the original variables  $v_{ar}^+$  for  $NC_1$  and  $v_{ar}^-$  for  $NC_2$ .

---

#### Algorithm 1: Algorithm *AFFECT*<sup>S</sup>

---

**Data:**  $Var = v_{ar}^+ \oplus v_{ar}^-$  of length  $n$  is the value of the variable.

//  $Value = v_{ar}^+ \oplus v_{ar}^-$  is the value to be conditionally affected (of length  $n$ ).

//  $Cond = Cond^+ \oplus Cond^-$  is the 1 bit condition true if the affectation shall be done and false otherwise.

1.  $\forall k \in 1..n$   $NC_1, NC_2$  and  $PS$  compute

$$(\bar{w}_k | \bar{w}_k) = \bigwedge^S( \bigvee^S(Cond^+, Value^+ | Cond^-, Value^-); \bigvee^S(\neg Cond^+, Var^+ | Cond^-, Var^-) )$$

2.  $NC_1$  and  $NC_2$  respectively computes  $v_{ar}^+ = \bar{w}$  and  $v_{ar}^- = \bar{w}$ .
- 

**PROPERTY 6.** *AFFECT*<sup>S</sup> prohibits  $NC_1$  and  $NC_2$  to access the value stored in the variable *Var*. They even do not know if the value stored in *Var* has changed or not.

*Proof:* All the values stored in  $NC_1$  and  $NC_2$  are randomized. This count also for *Var*, *Cond* and *Value*. That means that none knows the real value stored in *Var* and *Value*. To know whether *Value* is affected to *Var* or not they shall know the real value of *Cond* but as it is also randomized they cannot infer whether the value of *Var* has been changed or not. In any case, the new value of  $v_{ar}^+$  and  $v_{ar}^-$  are new random values whatever the affectation has been effective or not.

*Complexity:* The methods  $\bigvee^S$  and  $\bigwedge^S$  are used  $2n$  and  $n$  times respectively on one bit. By reusing the complexity of operators  $\bigvee^S$  and  $\bigwedge^S$  (see section 5.2),  $NC_1$  and  $NC_2$  therefore perform  $34n$  binary operations, generate  $6n$  aleatory bits, send  $12n$  bits and receive  $10n$  bits (including parameters).  $PS$  performs  $12n$  binary operations, generates  $3n + 1$  aleatory bits, receives  $12n$  bits ( $6n$  from  $NC_1$  and  $NC_2$  each) and sends  $6n$  bits ( $3n$  to  $NC_1$  and  $NC_2$  each). Obviously this has to be compared with the length of inputs ( $n$  bits).

---

**Algorithm 2:** The Algorithm  $\wedge^S$ 

---

**Data:**  $(\overset{\dagger}{x}, \overset{\dagger}{y} | \bar{x}, \bar{y})$  vector of bit/s are such that  $\overset{\dagger}{x}$  and  $\overset{\dagger}{y}$  are in  $NC_1$ , and  $\bar{x}$  and  $\bar{y}$  are in  $NC_2$

**Result:**  $(A^R | B^R)$  is such that

$$A^R \oplus B^R = (\overset{\dagger}{x} \oplus \bar{x}) \wedge (\overset{\dagger}{y} \oplus \bar{y})$$

- $NC_1$  and  $NC_2$  mutually generate and exchange four random vector of bit/s  $R_A, R'_A, R_B$  and  $R'_B$  such that:  $\overset{\dagger}{x}' = \overset{\dagger}{x} \oplus R_A, \overset{\dagger}{y}' = \overset{\dagger}{y} \oplus R'_A,$   
 $\bar{x}' = \bar{x} \oplus R_B$  and  $\bar{y}' = \bar{y} \oplus R'_B.$
  - $NC_1$  sends  $\overset{\dagger}{x}'$  and  $\overset{\dagger}{y}'$  to  $PS.$
  - $NC_2$  sends  $\bar{x}'$  and  $\bar{y}'$  to  $PS.$
  - $PS$  computes  $\overset{\dagger}{c} = \overset{\dagger}{x}' \wedge \bar{y}'$  and  $\bar{c} = \bar{x}' \wedge \overset{\dagger}{y}'$  and generates a random vector of bit/s  $R_{PS}.$
  - $PS$  sends  $A'_{PS} = \overset{\dagger}{c} \oplus R_{PS}$  to  $NC_1$  and  $B'_{PS} = \bar{c} \oplus R_{PS}$  to  $NC_2.$
  - $NC_1$  computes  $A^R = A'_{PS} \oplus (\overset{\dagger}{x}' \wedge R'_B) \oplus (\overset{\dagger}{y}' \wedge R_B) \oplus (\overset{\dagger}{x}' \wedge \overset{\dagger}{y}') \oplus (R_B \wedge R'_A)$
  - $NC_2$  computes  $B^R = B'_{PS} \oplus (\bar{x}' \wedge R'_A) \oplus (\bar{y}' \wedge R_A) \oplus (\bar{x}' \wedge \bar{y}') \oplus (R_A \wedge R'_B).$
- 

## 5.2 The algorithms $\wedge^S$ and $\vee^S$

In this section, we define algorithms used to implement the secure operator  $\wedge^S$  and  $\vee^S$ , the basic principle of these algorithms is to add uniform random noise to the data which could be deleted from the final result.

The  $\wedge^S$  protocol begins with  $NC_1$  and  $NC_2$  who modify their data by doing XOR them with random values (see step 1 in algorithm).  $NC_1$  and  $NC_2$  share these random values (also see step 1). Garbled data are then send to  $PS$  (step 2 and 3) which is now able to compute  $\wedge$  in a secure way (step 4). In fact,  $PS$  gets only garbled inputs indistinguishable from random and unrelated to each others and thus calculates random values from its point of view.

To avoid  $NC_1$  and  $NC_2$  from inferring the final result, it does XOR with random noise to the values it calculates before sending them back to  $NC_1$  and  $NC_2$  (step 5). Now  $NC_1$  and  $NC_2$  may both obtain their part of the final result by removing the random noise they added on step 1 (see step 6 and 7). The final result is obtained by computing:

$$A^R \oplus B^R = A'_{PS} \oplus (\overset{\dagger}{x}' \wedge R'_B) \oplus (\overset{\dagger}{y}' \wedge R_B) \oplus (\overset{\dagger}{x}' \wedge \overset{\dagger}{y}') \oplus (R_B \wedge R'_A) \oplus B'_{PS} \oplus (\bar{x}' \wedge R'_A) \oplus (\bar{y}' \wedge R_A) \oplus (\bar{x}' \wedge \bar{y}') \oplus (R_A \wedge R'_B)$$

where:

$$A'_{PS} \oplus B'_{PS} = (\overset{\dagger}{x}' \wedge R'_B) \oplus (\overset{\dagger}{y}' \wedge R_B) \oplus (\bar{x}' \wedge R'_A) \oplus (\bar{y}' \wedge R_A) \oplus (\overset{\dagger}{x}' \wedge \overset{\dagger}{y}') \oplus (\bar{x}' \wedge \bar{y}') \oplus (R_A \wedge R'_B) \oplus (R_B \wedge R'_A) \oplus R_{PS} \oplus R_{PS}.$$

Using the property of the XOR operator:  $R \oplus R = 0$ , we get the desired result:  $A^R \oplus B^R = \overset{\dagger}{x}' \wedge \overset{\dagger}{y}' \oplus \bar{x}' \wedge \bar{y}' \oplus \bar{x}' \wedge \overset{\dagger}{y}' \oplus \overset{\dagger}{x}' \wedge \bar{y}'$

$\oplus \bar{x}' \wedge \bar{y}'$ . Which is a re-written form of  $(\overset{\dagger}{x}' \oplus \bar{x}') \wedge (\overset{\dagger}{y}' \oplus \bar{y}')$ . However, this operation is never performed by the non collaborative sites and the final result is kept shared between  $NC_1$  and  $NC_2$ .

The  $\vee^S$  protocol is identical to the  $\wedge^S$  protocol except for the last two steps (steps 6 and 7) performed by  $NC_1$  and  $NC_2$ . Thus we get the final result:

$$A^R \oplus B^R = \overset{\dagger}{c}' \oplus (\overset{\dagger}{x}' \wedge R'_B) \oplus (\overset{\dagger}{y}' \wedge R_B) \oplus \overset{\dagger}{x}' \oplus \overset{\dagger}{y}' \oplus (\overset{\dagger}{x}' \wedge \overset{\dagger}{y}') \oplus (R_B \wedge R'_A) \oplus \bar{c}' \oplus (\bar{x}' \wedge R'_A) \oplus (\bar{y}' \wedge R_A) \oplus \bar{x}' \oplus \bar{y}' \oplus (\bar{x}' \wedge \bar{y}') \oplus (R_A \wedge R'_B).$$

This reduce to the desired result:  $A^R \oplus B^R = \overset{\dagger}{x}' \oplus \overset{\dagger}{y}' \oplus (\overset{\dagger}{x}' \wedge \overset{\dagger}{y}') \oplus (\bar{x}' \wedge \bar{y}') \oplus \bar{x}' \oplus \bar{y}' \oplus (\bar{x}' \wedge \bar{y}') \oplus (\bar{x}' \wedge \overset{\dagger}{y}') \oplus (\overset{\dagger}{x}' \wedge \bar{y}')$  which is a re-written form of  $(\overset{\dagger}{x}' \oplus \bar{x}') \vee (\overset{\dagger}{y}' \oplus \bar{y}')$ .

---

**Algorithm 3:** The algorithm  $\vee^S$ 

---

**Data:**  $(\overset{\dagger}{x}, \overset{\dagger}{y} | \bar{x}, \bar{y})$  vector of bit/s such that  $\overset{\dagger}{x}$  et  $\overset{\dagger}{y}$  belongs to  $NC_1$ ,  $\bar{x}$  and  $\bar{y}$  belongs to  $NC_2$ .

**Result:**  $(A^R | B^R)$  is such that

$$A^R \oplus B^R = (\overset{\dagger}{x} \oplus \bar{x}) \vee (\overset{\dagger}{y} \oplus \bar{y}).$$

- 1.5. These steps are same as initial 5 steps of  $\wedge^S$  function.
  6.  $NC_1$  computes  $A^R = A'_{PS} \oplus (\overset{\dagger}{x}' \wedge R'_B) \oplus (\overset{\dagger}{y}' \wedge R_B) \oplus \overset{\dagger}{x}' \oplus \overset{\dagger}{y}' \oplus (\overset{\dagger}{x}' \wedge \overset{\dagger}{y}') \oplus (R_B \wedge R'_A).$
  7.  $NC_2$  computes  $B^R = B'_{PS} \oplus (\bar{x}' \wedge R'_A) \oplus (\bar{y}' \wedge R_A) \oplus \bar{x}' \oplus \bar{y}' \oplus (\bar{x}' \wedge \bar{y}') \oplus (R_A \wedge R'_B).$
- 

**PROPERTY 7.**  $\wedge^S$  and  $\vee^S$  forbid  $NC_1$  to gain any information of private data of  $NC_2$  (and vice versa). Moreover, the  $PS$  learns none of their private inputs.

*Proof:* From the protocol,  $B'_{PS}$  is the only value that  $NC_2$  can learn from the private data of  $NC_1$ . Due to the noise,  $R_{PS}$ , added by  $PS$ ,  $NC_2$  is still not able to deduce the values of  $\overset{\dagger}{x}$  or  $\overset{\dagger}{y}$ . As the roles of  $NC_1$  and  $NC_2$  are interchangeable, the same argument holds for  $NC_1$ , not able to learn the private inputs  $\bar{x}$  or  $\bar{y}$  of  $NC_2$ . However, one key security aspect of not leaking any information to  $PS$  is achieved by randomizing the inputs before transmitting them to the Processing Site. Due to the randomization performed during the initial step, it just infers a stream of uniformly distributed values, and cannot distinguish between a genuine and a random value.

*Complexity:* Length of bit vector is 1: For the operator  $\wedge^S$ ,  $NC_1$  and  $NC_2$  each performs 10 binary operations ( $6 \oplus$  and  $4 \wedge$ ).  $\vee^S$  does two more  $\oplus$  that means 12 binary operations. For both operators  $NC_1$  and  $NC_2$  generate 2 random bits, exchange  $2 \times 2$  random bits and send  $2 \times 1$  bits to  $PS$ .  $PS$  generates 1 random bit and performs 4 binary operation ( $2 \oplus$  and  $2 \wedge$ ) and returns 2 bits to  $NC_1$  and  $NC_2$  each.

### 5.3 The Algorithm $COMPARE^S$

---

**Algorithm 4:** The Algorithm  $COMPARE^S$ 


---

**Data:** Half part of  $V$  and  $W$  is owned by  $NC_1$  and the other part is owned by  $NC_2$

**Result:**  $(\overset{\pm}{R}|\bar{R})$  is such that  $\overset{\pm}{R} \oplus \bar{R} = 1$  if  $V = W$  else 0

1.  $NC_1$  computes  $X \leftarrow \overset{\pm}{v} \oplus \overset{\pm}{w}$  where  $X = (X_1, X_2, \dots, X_l)$  and  $l$  is the length of vector  $V$  and  $W$ .
  2.  $NC_2$  computes  $Y \leftarrow \bar{v} \oplus \bar{w}$  where  $Y = (Y_1, Y_2, \dots, Y_l)$ .
  3.  $(\overset{\pm}{R}|\bar{R}) \leftarrow OR^S(X_1, X_2, \dots, X_l|Y_1, Y_2, \dots, Y_l)$
  4. There is two ways to gets the result either  $NC_1$ ,  $NC_2$  perform  $(\overset{\pm}{R}|\bar{R}) \leftarrow (\neg \overset{\pm}{R}|\bar{R})$  or  $(\overset{\pm}{R}|\bar{R}) \leftarrow (\overset{\pm}{R}|\neg \bar{R})$ .
- 

PROPERTY 8.  $NC_1$  and  $NC_2$  gain no information of the real values which are compared and of the result of the comparison.

*Proof:* The input data sent to  $NC_1$  and  $NC_2$  are garbled with random values. Thus they cannot distinguish them from random values. In the same way, all values returned by  $\bigvee^S$  are also garbled with unrelated random bits. Thus  $NC_1$  and  $NC_2$  only gets random values and then cannot infer the actual values of the inputs or results. If  $PS$  keeps history of intermediate results, it might deduce a part of the aleatory bits that were used to encode its results sent to  $NC_1$  and  $NC_2$ . However, this gives no information of actual data.

*Complexity:* Length of bit vector is  $l$ :  $COMPARE^S$  executes  $l \oplus$  operations and  $l - 1 \bigvee^S$ . Thus  $NC_1$  and  $NC_2$  compute  $13l - 12$  binary operations (plus 1 negation in  $NC_1$  or  $NC_2$ ), generate  $2l - 2$  aleatory bits, receive  $4l - 3$  bits (including inputs) and send  $5l - 4$  bits (including the result). On  $PS$  side,  $PS$  computes  $4l - 4$  binary operations, generates  $l - 1$  aleatory bits, receives  $4l - 4$  bits and sends  $2l - 2$  bits.

### 5.4 The Algorithm $\neg^S$ and $NOP^S$

Both are very simple to implement, by negating or not the value in  $NC_1$  and/or  $NC_2$ : If none or both negate the value it is a  $NOP^S$  operation but if only one does it, it is a  $\neg^S$  operation. In the point of view of  $NC_1$  (respectively  $NC_2$ ), there is no way to know which operation is really executed as none of them knows whether or not the other site negates or not its value. We also may implement these operation by involving  $PS$  in the process to increase the confusion of  $NC_1$  and  $NC_2$  but it will consume extra time.

## 6. CONCLUSION

In this paper, we proposed a new approach to do any computation in a collaborative environment while preserving privacy and applied it to the intrusion detection problematic. Via our approach an application can use knowledge from foreign databases to identify whether a request corresponds to an attack or not. We have demonstrated that the proposed architecture ensured that it is impossible to identify which database has given the answer and that none of the

internal components of the architecture can infer knowledge on the databases or on the request from the data they got. Our approach may also provide the type of the attack when they are specified in the databases. We also demonstrate that the cost of this secure computation is linear with the one of the same computation executed in a non-secure way.

The approach also seems to be very scalable and one of our current focus is to study in which cases each site is required depending of the needs of programs code privacy or anonymity of sites. The second one concern the ability to detect when one or more of the non-colluding site does not conform to the edicted protocol. And another one concern the efficiency of operators (*i.e* introduce more powerfull and efficient operators like arithmetic ones) and the definition of a programming langage upon the SAX architecture to simplify the task of encoding.

## 7. REFERENCES

- [1] F. Cuppens and A. Mieke. Alert correlation in a cooperative intrusion detection framework. In *Proc. of the IEEE International Conference on Networks (ICON 2005)*, pages 118–123, 2005.
- [2] T. Escamilla. *Intrusion Detection: Network Security beyond the firewall*. John Wiley and Sons, Ed., 1998.
- [3] O. Goldreich. Secure multi-party computation. In *citeseer.ist.psu.edu/goldreich98secure.html*, 2000.
- [4] R. Heady, G. Luger, A. Maccabe, and M. Servilla. The architecture of a network level intrusion detection system. *Technical Report CS9020*, 1990.
- [5] R. Janakiraman, M. Waldvoege, and Q. Zhang. Indra: a peer-to-peer approach to network intrusion detection and prevention. In *Proc. of the 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2003.
- [6] M. Kantarcioglu and J. Vaidya. An architecture for privacy-preserving mining of client information. In *Proc. of the Workshop on Privacy*, pages 27–42, 2002.
- [7] M. Locasto, J. Parekh, A. Keromytis, and S. Stolfo. Towards collaborative security and p2p intrusion detection. In *Proc. of the 2005 IEEE Workshop on Information Assurance and Security*, 2005.
- [8] J. McHugh, A. Christie, and J. Allen. Defending yourself: the role of intrusion detection systems. *IEEE Software*, pages 42–51, Sep/Oct 2000.
- [9] P. Proctor. *Practical Intrusion Detection Handbook*. Prentice-Hall, 2001.
- [10] K. Wang, G. Cretu, and S. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proc. of the 8th International Symposium on Recent Advances in Intrusion Detection*, 2005.
- [11] G. Zhang and M. Parashar. Cooperative defence against ddos attacks. *Journal of Research and Practice in Information Technology*, 38(1), 2006.
- [12] C. V. Zhou, S. Karunasekera, and C. Leckie. Evaluation of a decentralized architecture for large scale collaborative intrusion detection. In *Proc. of the 10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007.