Fraud Detection and Prevention in Smart card Based Environments Using Artificial Intelligence

Wael William Malek[1], Keith Mayes[2], Kostas Markantonakis[3]

Royal Holloway, University of London,
Egham, Surrey, TW20 0EX, United Kingdom
`<ww.malek@gmail.com><(Keith.Mayes,K.Markantonakis)@rhul.ac.uk>`

Abstract. This paper discusses the development and research for the detection of fraud in Smart-Card environments by using artificial intelligence. The current research deals with behaviour based detection engine, which will detect abnormalities by learning the usual behaviour of the user and detecting new unusual behaviours. The behaviour-based detection engines is based on 'Neural Networks'. This work considers the feasibility of implementing 'Neural Network' fraud engine on a Smart card platforms.

1 Introduction

There are many reasons for researching fraud detection mechanisms. Certain types of fraud are still very hard to detect by current technical security measures. Telecommunication and other Smart card based industries, such as payment cards, are still very vulnerable to fraud. The use of sophisticated fraud detection techniques can assist in early detection and prevention. Neural Networks and behaviour-based detection engines add true artificial intelligence to the security defences of the system, rather than the more conventional signature based security measures. With intelligent security in place, the development cycle of industry applications could go further and faster and take bold steps towards the future.

In researching this paper a Fraud Engine based on an Artificial Neural Network (ANN) was implemented on a Smart card in order to asses the performance and the general feasibility of this approach. The motivation was that, the intelligent behaviour-based security mechanisms can provide added protection for critical systems. Of particular interest is the real-time detection and reaction to fraudulent behaviours [1]. Any suspicious or unusual activities are captured and prevented instantly. With real artificial intelligence implemented using Neural Networks, behaviour based security mechanisms promise to be at the same step as the attacker and not a step behind. Using this kind of approach to security brings it down to the personal level, meaning fraud should be detectable for every single user or customer depending on his usage characteristics.

The Smart card environment is very limited, e.g. it has very limited memory and processing power and even modern Java Cards still have a restricted Java

Card Runtime Environment(JCRE). In this paper the possibility of using Artificial Neural Networks to detect fraud and unusual behaviour is investigated. The fraud engine implementation is tested on a PC environment and we will discuss the feasibility of the implementation on a Java Card. In the next section, we will discuss Neural Networks concept and how the brain works. In section 3, we will discuss the design and implementation challenges related to implementing an ANN on a smart card. In section 4, we will discuss the functionality of a fraud engine based on ANN. Finally, we will discuss the results and the benefits of using such a tool as well as the future suggested work.

2 Neural Networks Concepts

It can be argued that a behavioural based fraud detection engine protects and provides added security to Smart card based systems. It is considered necessary to add such a measure to security because Smart card applications such as banking cards and SIM/USIM mobile cards have become a crucial part of everyday life. Since, Smart Cards have become a part of our critical transactions, they can learn our habits and behaviours as we use them. A behavioural based security measure is an attractive method of protection but requires a sensitive but reliable detector. The proposed security mechanism is heavily dependent on Neural Networks and the degrees of intelligence they provide, as they learn the usual behaviour and are able to easily detect and stop the unusual ones. The idea behind the Neural Networks is to provide an artificial brain that will learn the same way that the human brain learns [2]. According to the required knowledge it will start to make decisions when anomalies are detected.

2.1 How the Brain Works

Before considering an artificially intelligent mechanism it is necessary to first understand a little about how the human brain works. The human brain has about 100 billion cells [3]. Most of these cells are called neurons. The interaction of multiple cells is called a neural network or biological neural network. A neuron is an on/off switch and is either in a resting state (off) or it is shooting an electrical impulse down a wire (on) [4]. It has a cell body, a long wire which is called an Axon, and at the very end it has a little part that shoots out a chemical. This chemical goes across a gap which is called Synapse where it triggers another neuron to send a message as shown in Figure 1 [5, 6]. There are are many neurons sending messages using Axons to communicate with the neighboring cells [7]. As the message goes down an Axon to a neighboring cell it could pass through a lot of other cells on the way until it stops at a certain destination as shown in Figure 1. The establishment of this route, which starts from a starting cell to a destination cell, when activated is called learning [6]. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes [3].

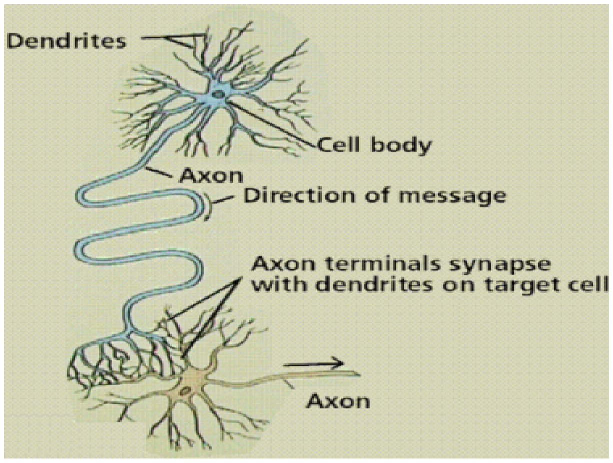


Fig. 1. Neuron Cell Connection [18].

According to the concept on how learning happens in the brain, scientists were able to create an Artificial Neural Network (ANN) based on the understanding of how the brain's Biological Neural Network works [5]. The aim was to simulate the human brain on an application and employ it to provide the advantages of the human brain and make it available to maximize the performance of systems and applications or even the actual computers.

Artificial Neural Networks are typically composed of interconnected units, similar to the brain. There are input units which serve as the denderites in the brain [8]. The function of the synapse is modeled by a modifiable weight, which is associated with each connection between the dendrites (input units) and other units (other cells). Each unit converts the pattern of incoming input (experience) that it receives from the other units into a single outgoing activity that it broadcasts to all other units. It multiplies each incoming input by the weight of the connection and adds together all these weighted inputs to get a quantity called the total input [8] Figure 2.

The behaviour of an ANN depends on both the weights and the input-output function (transfer function) that is specified for the units. When the inputs are received through the input units, the output is calculated and all the weights in the middle, between the input units and the output units, are modified. Through this process learning happens. In other words, learning take place in an ANN by modifying the weights according to some learning mathematical calculations and functions, which take place at every unit. The behaviour of the output units depend heavily on the activity of the hidden units and the weights between the hidden and output units [9].

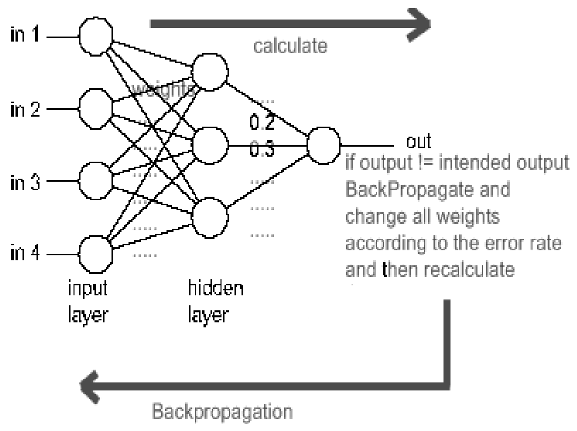


Fig. 2. Artificial Neural Network.

2.2 Artificial Neural Network Advantages

The fact that the ANN can detect patterns, identify familiar ones, as well as identify anomalies and give a best guess is a remarkable advantage over any other conventional design. In addition, the ANN will not need a constant increase in size as it gets smarter, which makes it a unique approach to designing complex and intelligent applications. The smarter the human brain gets through education, the more memory it holds and the more information it contains, however the brain does not get bigger the more education we get. The same idea works for the ANN. We might decide to implement more hidden layers or hidden units but that does not happen as frequently as assigning new memory slots in a data base design approach. Another advantage is that, to calculate an output for a given input, the ANN goes through series of mathematical functions. These functions are heavily dependent on addition and subtraction in plus some simple multiplication or division that does not take a lot of processor time. In other words, it is fast to calculate and efficient, which makes it suitable for hardware implementation and restricted environments such as Smart Cards platforms.

2.3 Existing Artificial Neural Network Based Systems

There are already systems that are designed to analysis behaviour to detect potential fraud. Such systems, known as Fraud Engines, are based on studying a certain behaviour and reporting if a different behaviour is detected. Neural Fraud Management Systems (NFMS) that are completely automated and state-of-the-art integrated system of neural networks, Fraud Detection Engines and Automatic Modeling Systems [10, 11].

A machine learning, anomaly-detection ANN based system can be used to address the shortcomings of rule-based systems. Rather than having to wait for a new attack to be detected and for a new rule to be written by an expert, these systems automatically and immediately detect unusual behaviour for each user and for groups of users. Behavioural systems are inherently future proof as they can spot new types of attacks the first time that they are executed. An effective anomaly detection system relies on clustering algorithms that are is based on Artificial Neural Networks. A clustering algorithm groups similar transactions into a small number of clusters. Each cluster represents a common pattern of activity. Each time a new transaction is processed by the anomaly detection system, the system tries to fit it into an existing cluster. If a transaction does not fit into any cluster, it is classified as an anomaly [12].

3 Neural Networks and Smart Cards

The aim of this paper is to examine the feasibility of implementing a neural networks based fraud engine on the Smart card it-self and not on the centralised processing systems, such as Visa or the mobile phone authorisation systems. The fraud engine will be installed on the Smart card in order to evaluate the current card behaviour and analyse the card usage and authorise actions accordingly. The usage behaviour will be stored on the card. If unusual behaviour is detected the carrier device of the Smart card will ask the user a secret question that only the legitimate user can answer. If the secret question is answered properly the requested use will be permitted and the card's ANN will learn the new behaviour so the user will not be prompted again when performing a similar task. In the case of the user not being able to answer the security question, the requested use will not be allowed and the card will be temporarily locked. This implementation guarantees that only the Smart card holder is the one in full control of the card all the time, and if the card is to be stolen, the card cannot be used to commit fraudulent activities. Of course before this scenario can be considered we must first determine if it is indeed feasible to implement the ANN on a smart card.

In this section we review the limitations of the Smart card environment and examine the possibility of implementing an ANN on Smart card. on a Smart card and in particular a card platform supporting the Java Card Run Time Environment (JCRE).

3.1 Design Challenges

In this section we will introduce the challenges that faces the design and the implementation of an Artificial Neural Network on a smart card.

Data Structures As mentioned earlier, to be able to successfully implement an ANN we will need to store the weights in a data structure that will facilitate access and modification of the weights. In the research we used multidimensional arrays to store the different sets of weights, but in JCRE multidimensional arrays are not supported, which means this data structure is no longer viable for use by the ANN tool. JCRE supports one dimension arrays which is the only data structure available and so a conversion from multidimensional arrays to one dimensional arrays is a must.

To be able to implement this conversion successfully the ANN tool's code need to be reviewed carefully to guarantee that the tool is accessing the correct weights at any point in time. The design of the ANN needs to be modified in order generate results as per the original design. In the next section we show the original code and the modified code to demonstrate how the conversion is possible as per Figure 4.

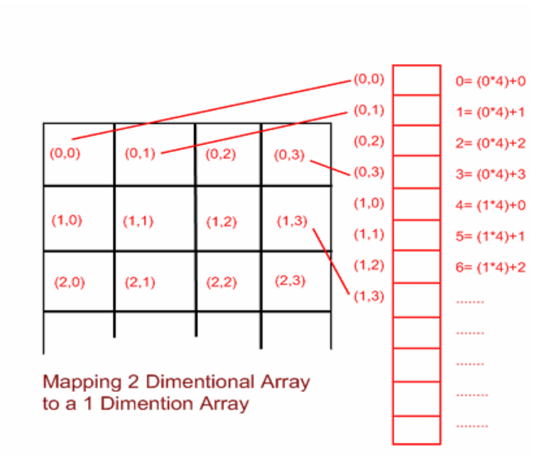


Fig. 3. Mapping two-dimensional array to one-dimensional array.

As shown above, if the mapping process is implemented successfully, we can have an ANN implemented based on a single dimensional array. The access to the weights to the single dimensional array will need to be modified. For example, in the 'for loops' used to access and modify the weights instead of using a double index (2,3) we will only use a multiple of the row plus the column number ($i \times 2 + 3$), where i is the number of rows (Figure 4). By applying this formula we can overcome the first challenge and be able to access the weights correctly and modify them safely.

Complex Math Due to the limited processing power of the Smart card's CPU, complex calculations are not recommended as they will be very time consuming and therefore impractical. In theory, only addition and subtraction take minimal time. Multiplication (which is simply a multiple addition) and division are very time-consuming, and should be avoided if possible. The manipulation of the weights is very simple; it is only a formula with simple math such as division and multiplication. However, the activation function which is used in clustering and mapping behaviours necessitates some demanding maths by using complex functions such as Sigmoid [8]. The Sigmoid function needs to calculate the exponentials of the weights.

The first attempt to solve this complex problem is to implement the exponential function by using simple math only which is possible if the concept of the exponential function is understood correctly. The exponential function, EXP(x), is defined as the sum of the following infinite series [13]:

$$\begin{aligned} \exp(x) &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^i}{i!} + \cdots \\ &= 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^i}{i!} + \cdots \end{aligned}$$

$$(i+1)^{th} \text{ term} = \frac{x^{i+1}}{(i+1)!} = \frac{x \cdot x^i}{(i+1)i!} = \frac{x}{i+1} \cdot \frac{x^i}{i!} = \frac{x}{i+1}(i^{th} \text{ term})$$

Fig. 4. Implementing the Exponential Function [13].

One obvious way of writing this program is computing each term directly using the formula (x^i/i!). However, this is not best practice, since both x^i and i! could get very large when x or i is large. One way to overcome this problem is rewriting the term [13] as shown in Figure 5.

Therefore, the (i+1)th term is equal to the product of the i-th term and x/(i+1) [13]. The second design makes it feasible to implement using a simple 'for loop' in addition to some simple math such as addition and multiplication. However, although this implementation was feasible for some computing platforms, it is not suitable for restricted environments such as Smart Cards. The computation time needed to calculate the right exponent is completely dependent on the number fed into the function and the tolerance level. If the tolerance level is low, the time needed to calculate EXP(x) is long, but if the tolerance level is high, the time is less, but the accuracy is poor.

Another proposal is to replace the exponential activation function by the identity function to simplify the procedure, but extra limits are then needed to be added to help the weights not to converge in magnitude and cause an error (Figure 6). As a solution, instead of using Sigmoid functions, which need complex calculations, we use a threshold function that keeps the weights within the required limits (Figure 6).

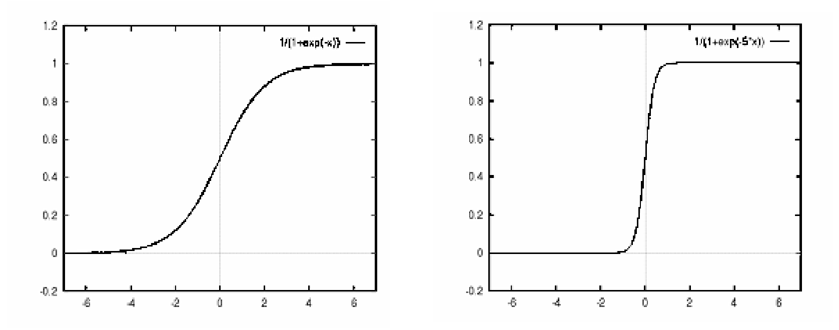


Fig. 5. Sigmoid Function graph [35], Threshold Step Function.

By using the threshold step function we are able to comply and fall within the limitation of the Smart card environment and be able to come a step closer towards implementing a successful application that better cope with a Smart card.

3.2 Implementation Challenges

ANN with Integer Weights The main challenge is that the Smart card environment does not support Float or Double types. Meaning that, it only supports integer and Short types; types with a minimal precision. The entire ANN implementation had to rely on integer maths only. This is a major challenge in the design of the ANN that can affect the results dramatically. Selecting weight precision is one of the important choices when implementing ANN and may be used to trade-off the capabilities of the realised ANN against the implementation cost [14]. A higher weight precision means fewer quantisation errors, while a lower precision leads to simpler designs, greater speed and reductions in area requirements and power consumption, which is ideal for the Smart card environment [14]. By keeping the precision to a minimum, we will be improving efficiency and speed as well as complying with the power restrictions [14].

There has been extensive research in this area proposing advanced algorithms that use only neural networks with integer weights that produces similar accuracy with minimum error rates [15]. The majority of those algorithms use the negative of the gradient of the error function, E(w), as their descent direction. The gradient E(w) can be computed by the BackPropagation [16, 17] of the error through the layers of the network. This calculation, however, is computationally expensive and difficult to implement in hardware [16]. In this paper, a new class of ANN that do not need the floating point weights, has to be researched. Other algorithms, that train neural networks with threshold units, require the learning task to be static. However, in order to train the network 'off-line' in a software simulation and later transfer it to the hardware [18]. Card and so many real life applications may not be static, i.e., input data may continue to change even after the hardware implementation, which is the situation in the case of an ANN on a Smart card, the learning should take place as the user uses the Smart card. In such cases an algorithm capable of continuing training 'on-chip' is needed. There have been some proposed strategies that are capable of continuing the training process in hardware, when threshold activation functions have been used [18].

By using algorithms that only uses integer math in restricted environments such as Smart Cards, we guarantee speed and reduced space requirement. Integer math will result in a reduction in power consumption and maximising the limited performance. An ANN with only integer math will have integer weights only, which is proved to be faster and more accurate compared to networks with floating point weights [19]. However, using only integer math will require all values in the design to be integers such as the error rates and the weight change variables. If such variables are rounded up or down, this might increase or freeze the error rate which will cause inaccuracy in the calculations. By using fixed point math a number can be represented in two parts, an integer part and a fractional part.

Solution - Integer Math and Bit Manipulation Every unsigned integer in the new proposed design is represented by 8 bits. The lowest order 3 bits represent the fractional part of a number and the remaining 5 bits represent the integer part, for example [19]:

0 = 00000.000 , 2 = 00010.000, 1.25 = 00001.010

So that the full range of this number is:
00000.000 to 11111.111 = 0 to 31.875
Now we can use normal integer arithmetic operations and bit
shift the operands or result to acquire the correct answer.
For example [19],
Multiplication
00001.010 * 00010.000 (1.25 * 2.0) = 10100.000
The integer maths has [effectively] performed the
multiplication with an extra factor of 2^3 or 1000 binary,
which we can now adjust for by shifting the answer
3 bits to the right:
10100.000 >> 3 = 00010.100 (2.5 decimal).[19]
Division
00001.010 / 00010.000 (1.25 / 2.0)
If we wish to keep the same level of precision in the
answer then we must first shift the numerator 3
bits to the left,
00001.010 << 3 = 01010.000

Now we can perform the integer division,
01010.000 / 00010.000 = 00000.101 (0.625).[19]
Addition / Subtraction
00001.010 +00010.000 (1.25 + 2) = 00011.010 (3.25)
This is the correct answer. No manipulation required.[19]

The bit shifting technique is the ideal way of implementing ANN on a Smart card, as it will save both power and computation time. The bit shifting techniques could be incorporated and calculated in the activation function so we do not need to perform the bit shifts in the ANN actual calculations. From experimental results it has been proved that this design is actually slightly faster than any ANN using floating point weights [16], which is a great achievement that emphasises the idea of having an ANN on a Smart card.

4 The Fraud Engine

4.1 Design

The design of the fraud engine is based on a neural network with back propagation learning algorithm. The neural network should receive some inputs and produce a decision (Approve/Reject), Figure 3. The inputs will depend on the use of the tool e.g. if it is used as a Telecommunication fraud engine, the inputs could be a phone number and a time of call, or duration of call and a destination country. If the fraud engine is used for a payment card, the inputs could be an amount of a transaction and a time of transaction. The inputs to the fraud engine must define the user behaviour and be crucial to the purpose and use of the Smart card. The decision is based mainly on the ANN weights that have been learnt and optimised from previous. If the Smart card is used for the first time it will start recording the first few activities which will reflect its decision on the future ones.

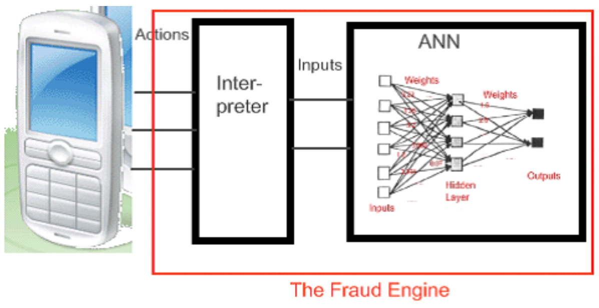


Fig. 6. Our proposed design of a fraud engine.

4.2 How will it work?

The weights are originally randomly initialized and stored in a multidimensional array. The network has 3 layers i.e. the input layer where the inputs are fed to the network, the hidden layer where extra cells and neurons exist and the output layer which is the decision enforcement point. The inputs to the network are specified by the environment of the Smart card e.g. the telecommunication SIM cards might have inputs such as call information and financial Smart Cards such as credit cards might have transaction information as inputs. The inputs are fed to the network and the learning process starts. After the first few transactions the network starts to build an idea about the user and his behaviour which defines the thresholds for the user's actions. As the learning process progresses the weights change and they have different magnitudes that reflect the network's new state. Future behaviours are reflected by new inputs which are fed to the network, if the behaviour is recognized the output is positive and approval of the behaviour is guaranteed otherwise the user will be asked for security questions which he should answer correctly if he is the genuine owner of the Smart card at the time. If the answer is correct, the action is permitted and the new behaviour is learned, otherwise the action is prohibited.

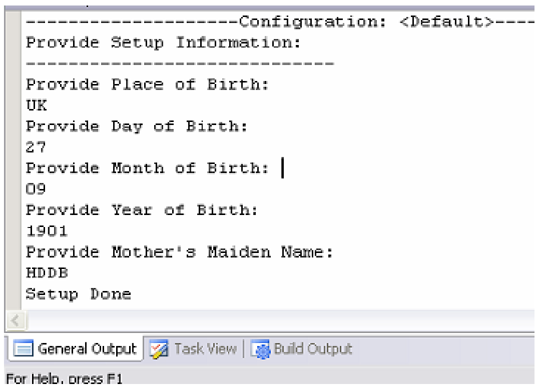
4.3 The Fraud Engine Tool

In this example we will assume that this tool is implemented for use on a mobile phone to provide a behavioural based fraud detection engine. The inputs could be assigned by the manufacturer of the phone, but in this example we will use 2 inputs only; the time of the call and the call destination. Ideally the time of the call should be easily understood and interpreted by the Fraud Engine, but we might have some complications with (AM/PM) timing. For simplicity we will divide the day into 4 hour intervals and we will refer to them as Time Zones and they will be numbered as follows:

12:00:00 PM until 04:00:00 PM: Zone 1
04:00:01 PM until 08:00:00 PM: Zone 2
08:00:01 PM until 12:00:00 AM: Zone 3
12:00:01 AM until 04:00:00 AM: Zone 4
04:00:01 AM until 08:00:00 AM: Zone 5
08:00:01 AM until 12:00:00 AM: Zone 6

The time zone will be one of the inputs to the fraud engine, the second input will be another number reflecting the destination of the call which will be called the Destination Number. It could be represented by another set of numbers which reflect the destination country or even the city. Assuming it will be ranging from (0..9), where 0, 1 are reserved for local calls and (2..5) for continental calls and (6..9) for international calls. When the user attempts to make a phone call, both the number called and the local time of the call are translated by the phone into the corresponding Destination Number and Time Zone which will be fed into the ANN for processing.

During the setup process of the phone with the new SIM Card, the phone should be able to store securely some information about the SIM Card holder such as an OIN, birth date, age, address, postal code, house number, a memorable word, a memorable day or any piece of information that the user could remember later on and be able to be authenticated to the phone or the SIM Card if requested. After the setup process, the information retrieved should be stored locally and securely on the mobile phone, Figure.

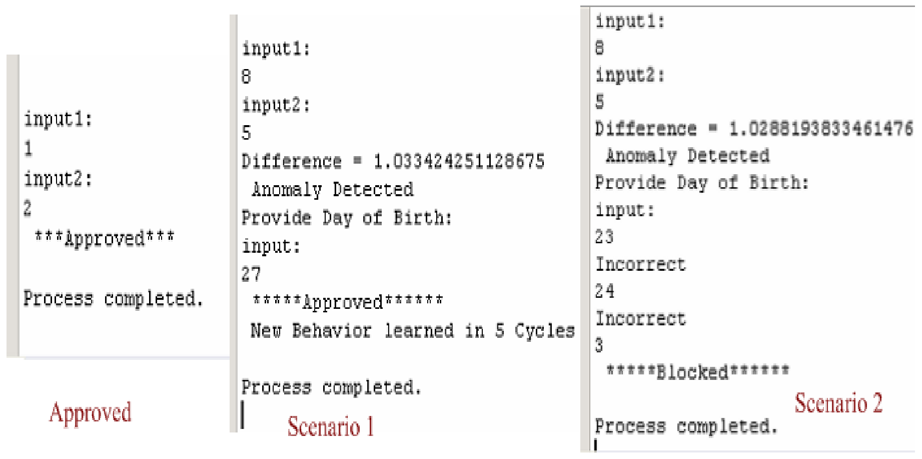


```
--------------------Configuration: <Default>----
Provide Setup Information:
-----------------------------
Provide Place of Birth:
UK
Provide Day of Birth:
27
Provide Month of Birth:
09
Provide Year of Birth:
1901
Provide Mother's Maiden Name:
HDDB
Setup Done
```

Fig. 7. Provide setup information, which is done only once at the beginning and the information is then kept secret.

4.4 Scenarios

In the first scenario, we assume that the user of the mobile and the SIM Card make only local calls between 12:00 PM and 8:00 PM, which means the ANN is trained to accept and approve calls to local destinations on Time Zones 1 and 2 and 3. A call interpreter function translates a call action to two inputs; the destination (0,1) and the time zones (1,2,3) which represent the user's behaviour. The fraud engine was tested by first making a similar call behaviour (Figure 3). Which means the interpreter will translate the similar call behaviour to the inputs (1,2) as for "'Local Call, at 5:30 PM"'. In the second test scenario, we assume that the same user attempts to call China at 5:00AM interpreted by the input pair (8, 5), which is an unexpected behaviour. This anomaly will be detected by the fraud engine and it will not be approved as it is an unexpected behaviour that is not been learned by the ANN. In this case the user will be required to further authenticate himself and prove that he is the legitimate owner. For example, the user may be prompted for a random question based on the reference information stored earlier. If the user is able to give the correct answer the action or the call will be granted and the new behaviour will be learned, otherwise the card could be blocked or the user is transferred automatically to the customer service line Figure 3.



```
input1:
1
input2:
2
 ***Approved***

Process completed.
```

```
input1:
8
input2:
5
Difference = 1.033424251128675
 Anomaly Detected
Provide Day of Birth:
input:
27
 *****Approved******
 New Behavior learned in 5 Cycles

Process completed.
```

```
input1:
8
input2:
5
Difference = 1.0288193833461476
 Anomaly Detected
Provide Day of Birth:
input:
23
Incorrect
24
Incorrect
3
 *****Blocked******

Process completed.
```

Fig. 8. Approved call or expected behaviour.

In these two scenarios we were able to show how a behavioural based fraud engine can defeat fraud and recognise an unusual behaviour, which could help limit the use of stolen Smart Cards and provide a second wall of defence against stolen SIM Cards and mobile phones. As shown earlier it is quite effective and user-tolerant as well. As an enhancement, the security questions could be behaviour based, such as asking the user about the most dialed number, or the last dialed number, or the last call received. These questions provide greater security, but there could be greater inconvenience for the user. Extensive research has to be carried out by the mobile phone manufacturers or the SIM card issuers to find the best questions that provide both convenience and good security.

Apart from the choice of good security questions there are other challenges that need to be addressed such as interoperability. If the SIM card is to be removed from a handset and placed in another, would all the previous behaviours be lost? The answer to this question would depend on the design of the fraud engine. As mentioned earlier, the weights are the heart of the ANN and they reflect the state of the artificial mind. If the weights are to be stored on the phone, all the information would be lost if the SIM card moves to another handset. The solution is for the weights to be stored on the SIM card itself so it is possible to move the SIM card to another handset with a compatible fraud engine that will retrieve the weights and continue to offer the same level of protection.

Another challenge is the tolerance level. Low tolerance from the fraud engine will trigger more security questions which might effect customer satisfaction and cause the customer to switch the tool off, but it will provide higher security. Conversely, high tolerance will achieve better customer satisfaction, but lower the security. The solution to this is to train the ANN offline with the worst case scenarios of fraud as well as to first train for the expected use depending on the purchased calling plan. This initial training will provide a basic protection level that the user will be able to develop and customize according to his own habits. This approach will be useful and effective both in decreasing the alerts and increasing security.

One of the most important challenges is to be able to correctly learn the user behaviour and be able to adapt to the user's life style changes. In modern phones the user can choose different profiles of use. Every profile has ring tones, wallpaper, calling properties such as barring and forwarding, favorite numbers, etc. Every profile may represent a different life style such as work, weekends and personal use. For every profile there could be a different set of weights that represent and reflect the use of the SIM card. Protection could be extended to cover all the different sets of behaviours a user can perform. When the profile is changed the ANN saves the current weights in the previous profile's security domain and loads the new set of weights for the new profile.

5 Conclusion

5.1 Benefits

Issuers have systems that protect them against fraud for which they are liable but these measures may not adequately protect the customer and so added security mechanisms may be desirable. Building good security and protection around every Smart card will decrease the pressure on the Card Issuers of having to counter all problems with centralised security mechanisms. The benefit of the ANN Fraud Engine is to provide a customised security for every single user. It provides mechanisms that are designed to protect every user according to his habits and usage of the SIM Card.

5.2 Results

We used a Java card 2.1 compliment for testing the execution of the fraud engine. It was tested on a simulator machine implemented on a PC. Although there have been some technical difficulties with recording the execution time on the Smart card, an approximation of the relative execution times has been recorded on the simulator. By recording multiple runs and taking an average it was possible to get an estimate of the execution time. The execution time of the Fraud engine when making a decision is 4.24 ms, which was achieved by recording 500 runs at 2.12 seconds. The period of time needed for learning a new behaviour is longer than the time needed to calculate the output of the ANN(make a decision). For the ANN to learn a new behaviour it needs 37.54ms at each cycle. In other words, to calculate the output of the network and get the error rate and then back-propagate to modify the weights it takes 37.45ms. The learning process will need at least 5 cycles to modify the weights efficiently so this execution time needs to be multiplied by 5 to get the overall learning time which is 187.70. The learning process depends on the accuracy of the implementation which means the better the implementation the faster the execution time. One other major accomplishment, is the size of the applet implemented. the size of the applet was 879 bytes on the EEPROM.

5.3 Suggested Future Work

After discussing the previous challenges and the obstacles that could affect having an ANN based fraud engine on a Smart card, it was possible to find solutions to the major problems that face such an idea. After implementing and fine-tuning an ANN algorithm based on the previous findings, it was possible to test it on a Smart card platform. The results were positive in terms of speed and power consumption but with some degradation in accuracy. Therefore, More research is needed in the areas regarding the execution time and the bit shifting algorithm. There are some ways to improve the execution time. The first would be to test it on a real card because the simulator environment might not have given the best indication of the execution time. The host of the simulator machine had limited memory and other processing tasks, which means the Fraud Engine might actually perform faster on the actual Smart Card. By tuning the Algorithm of the ANN further, we might get a better execution time. Another important area to be researched is the Bit shifting algorithm. There is a current bug in the design causing the accuracy to be only 70% due to a flaw in the bit shifting algorithm explained in section 3.2, which means, further tuning of the bit shifting algorithm is needed in order to achieve better accurancy hence better results.

References

[1] Harris, S.: CISSP. Third edition edn. Hardcover. Osborne (2005)
[2] John P. Jesan, D.M.L.: Human brain and neural network behavior: A comparison. http://www.acm.org **1** (2003) 2–5 [accessed on 3/04/2007].
[3] Dewri, R.: Evolutionary neural networks: Design methodologies. http://ai-depot.com/articles/evolutionary-neural-networks-design-methodologies/ **1** (2003) 1–5 [accessed on 9/02/2007].
[4] Eric Davalo, Patrick Naim, A.R.: Neural Networks. MacMillan Education, Limited (1991)
[5] John G. Nicholls, A. Robert Martin, B.G.W.P.A.F.: From Neuron To Brain. Fourth edition edn. Sinauer Associates Inc. (2001)
[6] Stergiou, C.: Neural networks, the human brain and learning. http://www-dse.doc.ic.ac.uk/nd/surprise96/journal/vol2/cs11/article2.html **1** (1996) 1–3 [accessed on 14/06/2007].

[7] Khanna, T.: Foundations of Neural Networks. Addison-Wesley (1990)
[8] Fausett, L.: Fundamentals of Neural Networks. Prentice Hall (1994)
[9] Omid Omidvar, J.D.: Neural Networks and Pattern Recognition. Academic Press (1998)
[10] Dimension, N.: Fraud detection using neural networks and sentinel solutions (smartsoft). http://www.nd.com/resources/smartsoft.html **1** (2006) 1–3 [accessed on 17/06/2007].
[11] Khan, A.: Feedforward Neural Networks with Constrained Weights. PhD thesis, Univ. of Warwick, Dept. of Engineering (1996)
[12] VeriSign: Verisign identity protection fraud detection service an overview. Whitepaper, VeriSign (2006)
[13] University, M.: Exponential function. http://www.cs.mtu.edu/shene/COURSES/cs201/NOTES/chap04/exp.html **1** (2004) 1–3 [accessed on 12/07/2007].
[14] VASSILIS P. PLAGIANAKOS, M.N.V.: Parallel evolutionary training algorithms for hardware-friendly neural networks. Technical report, University of Patras, GR-26110 Patras, Greece (2002)
[15] N.G. Pavlidis, D.K. Tasoulis, V.P.G.N.M.V.: Spiking neural network training using evolutionary algorithms. Technical report, 3Department of Pathology, University Hospital, GR26500 Patras, Greece (2001)
[16] VASSILIS P. PLAGIANAKOS, M.N.V.: Training neural networks with threshold activation functions and constrained integer weights. Technical report, University of Patras, GR-26500, Greece (2001)
[17] G.D. Magoulas, V.P. Plagianakos, M.V.: Hybrid methods using evolutionary algorithms for online training. Technical report, Department of Information Systems and Computing, Brunel University, Uxbridge UB8 3PH (2001)
[18] Sutton, J.Z.P.: Fpga implementations of neural networks - a survey of a decade of progress. Technical report, University of Queensland, Brisbane, Queensland 4072, Australia (2004)
[19] SharpNeat-Developers: An integer based neural network, sharpneat.sourceforge.net. http://www.sharpneat.sourceforge.net/integernetwork.html **1** (2004) 1–5