Design and Implementation of a Zero-Knowledge Authentication Framework for Java Card

Ahmed Patel, Universiti Kebangsaan Malaysia, Malaysia and Kingston University, UK

Kenan Kalajdzic, Center for Computing Education, Bosnia and Herzegovina

Laleh Golafshan, Department of Computer Engineering and IT, Science and Research Branch, Islamic Azad University, Fars, Iran

Mona Taghavi, Department of Computer, Science and Research Branch, Islamic Azad University, Tehran, Iran

ABSTRACT

Zero-knowledge authentication protocols are an alternative to authentication protocols based on public key cryptography. Low processing and memory consumption make them especially suitable for implementation in smart card microprocessors, which are severely limited in processing power and memory space. This paper describes a design and implementation of a software library providing smart card application developers with a reliable authentication mechanism based on well-known zero-knowledge authentication schemes. Java Card is used as the target smart card platform implementation based on the evaluation of the Fiat-Shamir (F-S) and Guillou-Quisquater (G-Q) protocols under various performance criteria are presented to show the effectiveness of the implementation and that G-Q is a more efficient protocol.

Keywords: *Authentication, Cryptography, Fiat-Shamir Protocol, Guillou-Quisquater Protocol, Java Card, Security, Smart Cards, Zero-Knowledge Protocols*

1. INTRODUCTION

User authentication is essential in many networked and Internet applications. It is a process by which a user proves his/ her identity to the system, thus proving his/ her rights to use particular information and services. The essence of authentication is the demonstration of either the knowledge of a secret, the possession of a physical object, or the authenticity of a certain human body characteristic.

The most popular mechanism of user authentication is the use of passwords. It is cheap to deploy and easy to use. While suitable for many applications, password authentication is lacking many features necessary for security critical applications. Badly chosen passwords are easy to guess, can be intercepted in transmission and re-used later for impersonating legitimate users. Passwords cannot be used directly to sign digital documents.

DOI: 10.4018/jisp.2011070101

Cryptography offers better methods of authentication, but their use is connected with manipulating secret cryptographic keys, which are difficult to remember. For sensible use, cryptographic keys need to be stored in some well-protected computing devices. For people on the go, such a device has to be small enough to fit into a pocket. Smart cards are probably the most widespread device of this sort.

A Smart card is a credit card sized plastic card with an embedded single-chip microcomputer. The use of special manufacturing technology makes physical tampering or probing of the microcomputer circuitry difficult, although not completely impossible. Smart card microcomputers are characterized by low clock frequencies (around 1 MHz) and small memory capacity (1-16 KB of ROM and less than 1 KB of RAM). Thus, smart cards are portable and small computers with different types of memory. Java Card technology is used in order to enable smart cards for running small applications in secure mode for a variety of environments, such as telephone networks and banking industry (ORACLE, 2010; Chen, 2000) and mobile agent e-marketplaces (Wei & Patel, 2009; Patel, 2010). Typically, it is touched wherever authentication and security are essential to access valuable data.

The limitations of smartcards severely impact the choice of cryptographic techniques available for use in smartcard applications. Currently, only techniques based on symmetric cryptography are in wide use. Although asymmetric (public key) cryptography offers a richer range of functionality, it requires more memory space and processing power than is available in the majority of currently available smartcards.

In the domain of authentication protocols, an alternative to both symmetric and asymmetric cryptography is the use of zero-knowledge proof techniques. Zero-knowledge authentication protocols offer same level of convenience as authentication protocols based on asymmetric cryptography, but require less memory space and processing power. Zero-knowledge protocols consist of two essential parts, the *prover* and *verifier* (Kapron *et al*., 2007). For a more detailed account regarding the background and content of zero-knowledge protocol see published paper by Vadhan (2004).

To validate practical applicability of zero-knowledge techniques in smartcard environment, the authors developed a prototype software library that implements a well-known zero-knowledge authentication protocol. Java Card specification was used as the target smartcard platform. The results of this work are discussed in the rest of this paper.

Section 2 gives a brief overview of smartcard technology and related standards. Section 3 gives an introduction into zero-knowledge proofs and zero-knowledge authentication protocols. Thereafter, the design and implementation of a prototype library based on the evaluated zero-knowledge protocols are discussed in Section 4 and the conclusions given in Section 5.

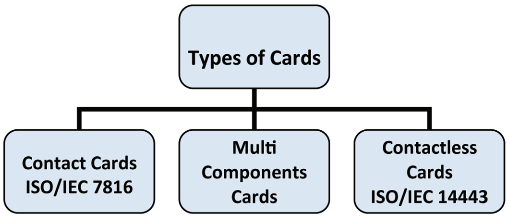
2. SMARTCARDS

A smartcard looks like a normal credit card with a chip embedded in it. Smartcards can be divided into three main categories according to the capabilities of the chip:

- *Memory cards*, which can just store data and have no data processing capabilities.
- *Wired Logic Intelligent Memory cards,* which contain also some built-in logic, usually used to control the access to the memory of the card.
- *Processor cards,* which contain memory and processor and have data processing capabilities.

Smartcards have to communicate with some other devices to gain access to a network. Therefore, they can be plugged into a reader, commonly referred to as a card terminal, or they can operate using Radio Frequencies (RF). In the former type of card, the connection is made when the reader contacts a small golden chip on front of the card whilst the latter (*contactless card*) can communicate via an antenna,

Figure 1. Smartcards types



eliminating the need to insert and remove the card by hand. All that is necessary to start the interaction is to get close enough to a receiver. Contactless cards are practical in applications in which speed is important or in which card insertion/removal may be impractical (an example could be the Subscriber Identity Module (SIM) cards in mobile phones). Some manufacturers are making cards that function in both contact and contactless mode.

All smartcards contain three types of memory: persistent non-mutable memory, persistent mutable memory and non-persistent mutable memory. ROM, EEPROM and RAM are the most widely used memories for the three respective types in the current smartcards.

A typical processor card with contacts has 16KB ROM, 512 bytes of RAM and an eight-bit processor, although the technology is moving towards 16 or 32-bit CPU (Oritz, 2003; ORACLE, 2010).

Although smartcards are more expensive than ordinary magnetic stripe cards, their use is increasing because of several reasons. Firstly, smartcards are more secure than magnetic stripe cards. In fact, it is easy today to purchase tools needed to hack into confidential data on a magnetic stripe card whilst smartcards are considered tamper resistant. However, unfortunately smartcards are not as tamper resistant as it is believed. Firstly, the technology to read protected memory or reverse-engineer smartcards' CPU is relatively easy, and with the present state of the art, they cannot resist well planned invasive tampering like side-channel signal pickup and differential power analysis (Kocher *et al.,* 1999). Secondly, processor cards with their processing capabilities and increased memory capacity can perform more activities than simple magnetic stripe cards that require a host system to store and process all data, which make them open to tampering.

Smartcards are used to cover the personal secure information, and it is significant that they play a critical role in security systems. Smartcards are the authentication devices that are used to store secret keys because of the lack of secure PCs. On the other hand, cryptographic operations are done via secret key (Herbst *et al.*, 2006).

A. Standards

Several standards for smartcards have been defined by International Standards Organisation (ISO) and the International Electro-technical Commission (IEC). The important ones are shown in Figure 1 based on reading and writing the data from the card, type of chips, and its capacities as well. Here, we restrict our discussion to processor cards with contacts.

1) ISO/IEC 7816

This ISO/IEC 7816 standard covers various aspects of integrated circuit cards with electrical contacts. It consists of the following fourteen parts with a fifteen part numbering - minus part 14 - (International Organization for Standardization, 1987):

- Physical characteristics (Part 1): defines the physical dimensions of contact smartcards and their resistance to static electricity, electromagnetic radiation and mechanical

stress. It also prescribes the physical location of an embossing area.
- Dimension and location of the contacts (Part 2): defines the location, purpose and electrical characteristics of the card's metallic contacts.
- Electronic interface signals and transmission protocols (Part 3): defines the voltage and current requirements for the electrical contacts defined in Part 2 and asynchronous half-duplex character transmission protocol.
- Organisation, security and commands for interchange (Part 4): establishes a set of commands to provide access, security and transmission of card data. Within the basic kernel, for example, are commands to verify access control, secure messaging, read, write and update records.
- Registration of application provider's identifiers (Part 5): defines how to use an application identifier to ascertain the presence of and/or perform the retrieval of an application in a card through data elements and interchange with the integrated circuit card.
- Inter-industry data elements interchange (Part 6): describes encoding rules for data needed in many applications, e.g. name and photograph of the owner, his/her preference of languages, etc.
- Inter-industry commands for Structured Query Language (SQL) (Part 7): describes how to use the database paradigm in cards through the concept of views and the standard SQL command.
- Commands for security operations (Part 8): to facilitate cryptographic operations, complementing commands given in Part 4.
- Commands for card management (Part 9): to facilitate card and file management, e.g. file creation and deletion.
- Electronic signals and answer to reset for synchronous cards (Part 10): specifies the power, signal structures, and the structure for the answer to reset between an integrated circuit card(s) with synchronous transmission and an interface device such as a terminal.
- Personal verification through biometric methods (Part 11): specifies the usage of inter-industry commands and data objects related to personal verification through biometric methods in integrated circuit cards.
- Cards with contacts — USB electrical interface and operating procedures (Part 12): specifies the operating conditions of an integrated circuit card that provides a USB interface.
- Commands for application management in multi-application environment (Part 13): specifies commands for application management in a multi-application environment.
- No Part 14.
- Cryptographic information application (Part 15): specifies a card application which contains information on cryptographic functionality and multiple cryptographic algorithms.

In addition to the above, the ISO/IEC 14443 proximity card standard consists of four parts and related amendments for smart cards on a different physical communication support mechanism (International Organization for Standardization, 2000).

Figures 2 and 3 show the physical appearance of a smartcard as defined in ISO7816 part 1.

Typically smartcard's physical chip appearance in credit card or SIM dimensions has contacts or contactless circuitry as shown in Figure 4.

It houses the computer configuration as shown in Figure 5.

A more comprehensive layout of the computer configuration is shown in Figure 6.

It consists of:

- Central Processing Unit (CPU): which is the heart of the chip
- Security logic: which detects abnormal conditions such as low voltage levels
- Serial I/O interface: which allows contact to the "outside" world

Figure 2. Physical appearance of smartcards

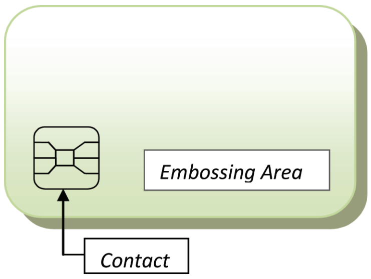


Figure 3.Chip appearance

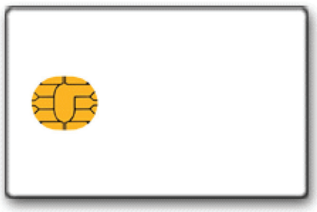


Figure 4. Chip contact circuitry

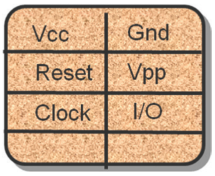


Figure 5. Chip and computer configuration

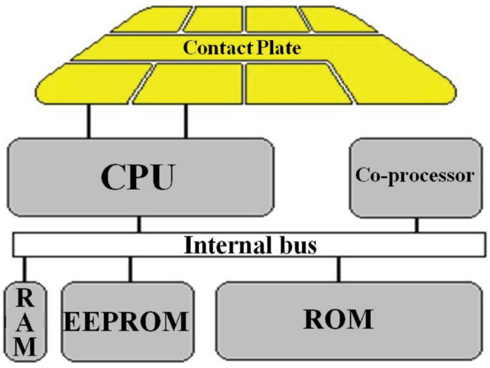
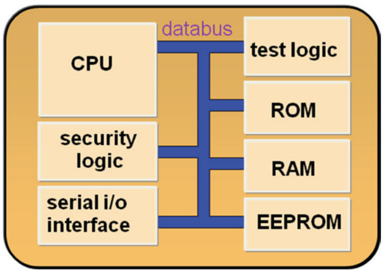


Figure 6. Chip internal circuitry



- Test logic: which permits self-test procedures to run
- ROM consisting of:
  - card operating system
  - self-test procedures
  - typically 16 KBytes
  - future 32/64 KBytes
- RAM consisting of:
  - "scratch pad" of the processor
  - typically 512 bytes (1/2 Kbyte)
  - future 1 KByte
- EEPROM consisting of:
  - cryptographic keys
  - PIN code
  - biometric template
  - balance
  - application code
  - typically 8 or 16 KBytes
  - future 32 KBytes
- Databus: This connects elements of the chip on an 8 or 16 bits wide bus structure

Normally, a smartcard does not contain a power supply, a display, or a keyboard. It interacts with the outside world using the serial communication interface via its eight contact points.

The embossing area is reserved to personalize the card, for embossing or laser engraving the name of the owner, the card number or other personal details relevant to the application in which the card is involved.

Among other things, ISO7816 (part 4) also defines a standard data format for interaction between the card and the outside world called APDU (Application Protocol Data Unit). If we consider the communication protocol in terms of master/slave paradigm, the card has always a passive role, waiting for a command APDU from the terminal in which it's inserted. In reply to the command, the card sends a response APDU.

2) Java Card

There is no standard smartcard programming language today. Smartcard companies use different languages to develop smartcard software; code is compiled into machine language and embedded into the chip (Sun Microsystems, 1997). The major problem is non-portability of smartcard software and a small universe of knowledgeable programmers (Coleman, 1998; Peyret, 1995).

How to overcome these problems that slow down the adoption of smartcards in many applications? Java programming language offers a possible solution. Java is an object-oriented programming language that compiles into a platform-independent byte code that can be run on any platform providing a Java byte code interpreter. The idea to give smartcards developers the ability to write applications once and have them run on all platforms led, in November 1996, to the release of the Java Card API Specification. One year later, with the release of Java Card API 2.0, every major vendor of smartcards in the world had licensed the technology (Coleman, 1998). For these reasons, Java Card was chosen as the target platform for this project.

The Java Card API is a part of the smallest virtual machine specification for Java. It is designed to allow Java to run on an 8-bit microprocessor, with 8 kilobytes of electrically erasable and programmable read only memory (EEPROM), 16 kilobytes of read only memory (ROM), and 256 bytes of random access memory (RAM) (Chen & Giorgio, 1998).

Java Card programs, called applets, are small enough so that several can fit into the small amounts of memory available on smartcards. Applets can be easily updated, and Java Card functionality can therefore be continually updated as new applications or updates become available.

3. ZERO-KNOWLEDGE PROTOCOLS

Zero-knowledge is one of the most popular, useful and powerful protocol in cryptographic design which was introduced by Goldwasser et al. (1985).

Zero-knowledge protocols, as their name suggest, are cryptographic protocols in which one party (*the prover*) can demonstrate the knowledge of some secret to another party (*the verifier*) without revealing the secret. This way, an eavesdropper, as well as the verifier, can gain no information about the secret and cannot convince a third party that they know the secret. More precisely, the properties of a zero-knowledge protocol are as follows:

- The prover cannot cheat the verifier unless the prover is extremely lucky; By reiterating the protocol, the odds of an impostor passing as a legitimate user can be made as minimal as necessary
- The verifier cannot pretend to be the prover to any third party because during the protocol execution the verifier gains no knowledge of the secret
- The verifier cannot convince a third party of the validity of the authentication proof

A good introduction into the field of zero-knowledge proof and protocols is given by Quisquater *et al*. (1990).

Zero-knowledge proofs that yield nothing but their validity is a must in the methodology of cryptographic protocol design (Goldreich, 1991). They play an important role in cryptography and it is applicable in solving NP (type of problems in computational theory defined as nondeterministic polynomial time) issues through interaction and randomness (Kapron *et al.*, 2007). The zero-knowledge protocol has been used to solve different problems. For instance, Kapron *et al.* (2007) presented a new characterization of zero-knowledge protocols as Non-interactive Instance-dependent Commitment schemes (NIC), and by this knowledge they believed that a NIC has a *V-bit* zero-knowledge protocol. Besides, with regards to previous related works (Vadhan, 2004; Nguyen & Vadhan, 2007; Ong & Vadhan, 2007) it is possible to prove unconditional results about zero-knowledge protocols, which has used zero-knowledge protocols as special bit commitment-schemes.

A. Basic Zero-Knowledge Protocol

Let's consider the basic operation of a zero-knowledge protocol on the following example taken from Schneier (1996).

Assume that the prover knows some information, and furthermore that the information is the solution to a hard problem. The basic protocol consists of several rounds: what is explained below is repeated *n* times.

The prover uses the information he/she knows and a random number to transform the hard problem into another hard problem, one that is isomorphic to the original one. Not all problems and transformations, of course, are suitable for this purpose; the prover must be sure that the verifier cannot deduce any knowledge from the execution of the protocol, even after many iterations of it.

Then, the prover uses the information he/she knows and the random number to solve the new instance of the hard problem, then commits to the solution, using a bit-commitment scheme. This kind of scheme is used when someone wants to commit to a result but does not want to reveal it until sometime later and, meanwhile; the counterpart wants to make sure that the result is not going to be changed after the commitment.

The prover reveals the new problem instance to the verifier, but the verifier cannot use this problem to get any information about the original instance or its solution. At this stage, the verifier asks the prover either to prove that the old and the new instances are isomorphic (i.e. two different solutions to two related problems) or to open the solution to which the prover committed before and show that it's a solution to the new instance. The prover complies.

In this protocol, the verifier does not get any knowledge of the secret information and the prover cannot cheat. Also, the verifier cannot use a transcription of the exchange to convince a third party that the prover knows the secret, because the verifier cannot demonstrate that she did not collude with the prover to build a simulator that fakes the prover's knowledge.

B. Which Problems can be Used in Zero-Knowledge Protocols?

The notion of Zero-Knowledge proof was set forward in 1985 by Goldwasser et al. (1985). One year later Goldwasser (1986) proved that any problem in NP class has a zero-knowledge proof, assuming the existence of one-way functions.

Unfortunately, not all problems in NP class are suitable for a realistic implementation. Like in other cryptographic protocols, the problems most widely used in actual zero-knowledge protocols are the following (Aronsson, 1996):

- the problem of finding discrete logarithms for large natural numbers
- the problem of checking that y is ($x2$ mod n) for some natural number x, if the factors of n are unknown
- the problem of factoring a large natural number which is a product of two or more large primes

C. Real Zero-Knowledge Authentication Protocols

Amos Fiat and Adi Shamir (1986) showed how to utilize zero-knowledge proofs for authentication and generating digital signatures. Their protocol, called Fiat-Shamir, was the first realistic zero-knowledge protocol; a number of other protocols have been developed after this one. This includes Feige-Fiat-Shamir (Micali & Shamir, 1990), Guillou-Quisquater (Guillou & Quisquater, 1988, 1990), Ohta-Okamoto (Ohta & Okamoto, 1990), Beth (Burmester *et al.*, 1992), Schnorr (Schnorr, 1990), and Burmester-Desmedt-Beth (Burmester et al., 1992) protocols. In this paper we review only Fiat-Shamir and Guillou-Quisquater protocols, which are most relevant to the subject of this paper.

1) Fiat-Shamir Protocol

A trusted process chooses and makes public a modulus n that is the product of two large prime numbers p and q known only to the process. The process then generates for each user the public key $v_1, v_2, ..., v_k$ and the private key $s_1, s_2, ..., s_k$ such that $s_i = v_i^{-1}$ (mod n).

To embed the identity of the user into her or his public key, the trusted process prepares a string I which contains all the relevant information about the user. The process also chooses and makes public a pseudo random function f which maps arbitrary strings to the range $[0,n)$. The function f must be indistinguishable from a truly random function by a polynomially bounded computation. To generate the public key, the process then computes a number of values $v'_j = f(I,j)$, where $j = 1,2,\dots,N$, and I,j means concatenation of I with a string representing j. For the public key, the process selects k values of v'_j for which there is a square root (modulo n). The selected values become $v_1, v_2, \dots, v_k$.

The proof is based on the following protocol (the prover is identified with P, the verifier with V):

1. P sends I to V

2. *V* generates *k* v_j values using same algorithm as the trusted process

The following steps are repeated *t* times:

3. *P* selects a random number *r* from [*0*,*n*) and sends *V* a value of $x = r^2 \pmod n$
4. *V* sends a random binary vector $(e_1, \ldots, e_k)$ to *P*
5. *P* sends to *V*: $y = r \cdot \prod_{e_j=1} s_j \bmod n$
6. *V* checks that $x = y^2 \cdot \prod_{e_j=1} v_j \bmod n$

V accepts *P*'s proof of identity only if all *t* iterations are successful.

V can get no knowledge of the secret key from the protocol. He cannot recover the secret values s_j from the public values v_j, because the calculation of a square root modulus *n* is considered computationally infeasible for large values of *n* and v_j.

A lucky cheater could guess the correct vector $(e_1,\ldots,e_k)$ sending then to *V*:

$$x = r^2 \prod_{e_j=1} v_j \left(\bmod n\right) \qquad \text{and} \qquad y = r.$$

However, the probability of this event is only 2^{-k} per iteration and 2^{-kt} for the whole protocol. *k* and *t* can be chosen to achieve level of security appropriate for a particular application. A digital signature scheme was constructed on the basis of this protocol.

2) Guillou-Quisquater Protocol

$$B^v \cdot J \bmod n = 1$$

where:

$$J = \text{Red}\ \left(I\right)$$

- Red (Redundancy Rule) is a published function, or preferably standardized, that completes *I*, which is half shorter than n, to obtain *J*, the "shadowed identity", that is a number as large as *n*.
- *v* is an exponent, both published by the authority and known to each verifier. *v* must be relatively prime to (*p-1*) and (*q-1*) to ease the operation of calculating the number *B* for the user.
- *n* is known by everyone, but only the authority knows its factorization.

The protocol requires only one round and it consists of the steps seen in Table 1.

The strength of the protocol is in the computational complexity of calculating roots of *v*th power modulo *n*.

Any cheater, having guessed the question *d*, can obviously prepare a pair of *T* and *t* by, firstly, picking *t* at random in $\mathbf{Z}_n$ and, secondly, deducing *T* by computing exactly as the verifier would do. A cheater, however, has only one chance to guess *d*, which means that the level of security is $2^{-|v|}$, where |*v*| is the length of *v* in bits. A digital signature scheme was developed to ascertain and verify this for Guillou-Quisquater protocol as well as for Fiat-Shamir protocol.

D. The Chess Grandmaster Problem

Although the idea behind zero-knowledge proofs of identity is quite powerful, zero-knowledge protocols are not perfect. The man-in-the-middle attack, for example, cannot be avoided as illustrated by the "Chess Grandmaster Problem" described in Goldwasser *et al.* (1985).

To defeat a world championship level grandmaster, someone (let's call her NICE MONA) could set up a two-room game, inviting two grandmasters to play with her. Neither grandmaster knows about the other.

NICE MONA starts the game with the grandmaster that plays with white pieces (the other one plays with black) so that she can see his first move. Then NICE MONA records the move and walks in the other room. Since NICE MONA plays white, she makes the first move in the game with the second grandmaster. She simply repeats the move of the first grandmaster. This continues, until NICE MONA wins one game and loses the other, or both games end in a draw.

This kind of fraud can be used against zero-knowledge proofs of identity: while the prover is proving her identity to the verifier, the verifier can simultaneously prove to another verifier that she is the prover. The only reasonable counterattack to the man-in-the-middle problem is imposing time limits for the replies.

4. DESIGN AND IMPLEMENTATION OF AN AUTHENTICATION LIBRARY FOR JAVA CARD

In this section, we discuss our major design decisions and the architecture of the authentication library for Java Card that implements a zero-knowledge authentication protocol.

A. Evaluated Choice of the Protocol

All public key protocols and the majority of analysed zero-knowledge protocols suffer from the problem of key integrity. In other words, a key has to be bound to the identity of its owner by means of a key certificate issued by an authorized trusted third party. Unfortunately, key certificates, especially those conforming to ITU-T X.509 Recommendation (ITU-T, 2005), are quite big and can easily take up to 1.5 to 2 Kbytes each (Meckley, 1998). Although storing such certificates inside smartcards makes perfect sense, it is problematic due to the small amount of memory available in modern smartcards. Some zero-knowledge protocols (the identity based ones) seem to solve the problem. In these protocols, the public key is generated from the identity of the user, which eliminates the need for certificates (Schneier, 2007).

Three of the well-known zero-knowledge protocols, Fiat-Shamir, Guillou-Quisquater and Beth, are identity based. The public key is derived from the identity of the user via a publicly known pseudo-random one-way function. The verifier knows the function as well as the prover and can generate the public key of the prover from the identity of the prover.

Only the process issuing secret keys can calculate the prover's secret key (or keys) on the basis of the secret information it has (in Fiat-Shamir and Guillou-Quisquater protocols this information are the factors of the modulus n). The verifier has no access to this information, thus, identity based public key generation does not reduce the security of the protocols, while eliminating the problem of certificates. After secret key generation, no further interaction with the process is required.

No interaction with the prover will enable verifiers to reproduce prover's secret, and even the knowledge of the prover's secret will not enable adversaries to create new identities or to modify existing ones without the help of the key issuing process (Fiat & Shamir, 1986).

Including the serial number of the smartcard as part of the identity ensures that if the user's secret is compromised, new public and secret keys can be generated for the replacement card (Guillou & Quisquater, 1990).

Beth's protocol does not offer a digital signature scheme, so we restricted our choice of protocol to Fiat-Shamir (F-S) and Guillou-Quisquater (G-Q).

An authentication scheme should be both secure and efficient, so that security overhead is minimized. Efficiency is particularly important in the context of smartcards, whose computational power and memory are severely restricted.

As the minimum security level recommended for authentication schemes is 2^{-20}, the theoretical performances of protocols are compared at this level. The criteria by which we compare these protocols are transmission cost (the amount of transmitted data, without considering the communication overhead), number of modular multiplication, and memory required.

Modular multiplication is one of the slowest operations performed, taking 0.5 sec on average using modern smartcard technology. Thus, we approximate the processing power required for each protocol by the number of modular multiplications required as shown in Table 2.

The number of modular multiplications is calculated on average for Fiat-Shamir (considering that a random vector has on average the

Table 1. Protocol steps

1. *P* transmits its identity *I* and a test number *T* which is the v^{th} power in $\mathbf{Z}_n$ of an integer *r* picked at random in $\mathbf{Z}_n$*
$$T = r^v \bmod n$$
2. *V* asks a question *d* which is an integer picked at random from *0* to *v-1*
3. *P* sends a witness number *t* which is the product in $\mathbf{Z}_n$ of the integer *r* by the d^{th} power of the authentication number *B*
$$t = r \cdot B^d \bmod n$$
4. *V* verifies that the product of the d^{th} power of the shadowed identity *J* by the v^{th} power of witness *t*, it's equal to *T*
$$J^d \cdot t^v \bmod n = J^d \cdot \left(r \cdot B^d\right)^v \bmod n = \left(J \cdot B^v\right)^d \cdot r^v \bmod n = T$$

same number of '0' and '1'), and in the worst case for Guillou-Quisquater. Exponentiation modulo *n* is approximated with 3/2•($\log_2$*exponent*) modular multiplications.

Table 3 shows the respective values for a level of security 2^{-20}, for $|n| = 512$.

The Guillou-Quisquater protocol minimizes the communication cost and the memory cost at the price of more computations (only 3 times Fiat-Shamir), which will be acceptable assuming the future growth of processing power in the new generation of smartcards.

Based on Table 3, it can be seen that the Guillou-Quisquater protocol has an effective performance in minimizing the memory and communication cost based on control parameter *V*. Another protocol, Fiat-Shamir, is tested according to two key parameters *t* and *k*. These two parameters are presented by the number of iterations of the basic protocol and the number of secret keys, respectively. Figure 7 shows the comparison among transmission cost, number of modular multiplications, and memory requirements to measure the performance of these protocols based on control parameters. Regarding these key factors, the suitable protocol is chosen. Although the rate of the bit transmitted of Fait-Shamir is acceptable on average with fewer numbers of iteration (*t = 1*), but the rate of memory requirements is high. In contrast, Guillou-Quisquater provides low memory requirements in terms of length of bits are 20 and consideration to the rate of transmitted data. It can be seen the relation between parameters and key factors for protocols. It is noticeable that the *t* parameter is consequential in computing cost transmission and processing power required for Fiat-Shamir, whilst the increase of *k* parameter (the number of secret key) will enhance the rate of memory requirements.

The number of bits transmitted is considered as an important factor to choose a useful protocol for implementation. So, for investigation of this factor, control parameters have been tested with different values to compare two existing protocols. As the chart shows (Figure8), the number of bits transmitted has a significant increase when the number of iterations of the protocol has changed from low value to high value for Fiat-Shamir protocol. In contrast, the number of bits transmitted of Guillou-Quisquater protocol is computed based on the length in bits of the public exponent.

It is noticeable that Guillou-Quisquater protocol needs less memory in comparison with Fiat-Shamir, basically (Figure 9). Therefore, we chose Guillou-Quisquater protocol for implementation in the authentication library based on the efficiency and security of the protocol. The performance is measured regarding to key factors at 2^{-20} minimum security level.

B. Operational Scenarios

The functionality for the prototype authentication library was specified as two operational

Table 2. Evaluation of F-S and G-Q protocols under performance criteria

	Fiat-Shamir	Guillou-Quisquater
No. of bit transmitted	$t\cdot(2\cdot\lvert n\rvert + k)$	$2\cdot\lvert n\rvert + \lvert v\rvert$
No. of modular multiplications (prover)	$t\cdot(k+2)/2$	$3\cdot\lvert v\rvert + 1$
No. of modular multiplications (verifier)	$t\cdot(k+2)/2$	$3\cdot\lvert v\rvert + 1$
Memory requirements	$\mathrm{k}\cdot\lvert\mathrm{n}\rvert$	$\lvert n\rvert$
Security level	2^{-kt}	$2^{-\lvert v\rvert}$

Notation: t = the number of iterations of the basic protocol
n = the modulus
$|n|$ = the length in bits of n (usually 512)
k = the number of secret keys
v = the public exponent
$|v|$ = the length in bits of v

*Table 3. F-S and G-Q for a 2^{-20} level of security (*values recommended by Fiat & Shamir, 1986)*

Parameters & Key factors	t	k	No. of bits transmitted	No. of modular multiplications	Memory requirements
Fiat-Shamir	1	20	1044	11	10240
Fiat-Shamir	2	10	2068	12	5120
Fiat-Shamir (*)	4	5	4116	14	2560
Fiat-Shamir	5	4	5140	20	2048
Fiat-Shamir	10	2	10260	20	1024
Fiat-Shamir	20	1	20500	30	512
Guillou-Quisquater	$\lvert v\rvert = 20$		1044	61	512

scenarios. Given below is a summary of these two scenarios.

1) Initialization Scenario

Actors: authentication process, user.

During initialization, the authentication process must generate the exponent v, the two prime numbers p and q and their product n according to the desired security level. The process then, on a user request, must perform user initialization. To do so the process calculates the user public key and private key from the user's identity. The process generates a Private Identification Number (PIN) for the user as well. At the end of the initialization, a card is issued.

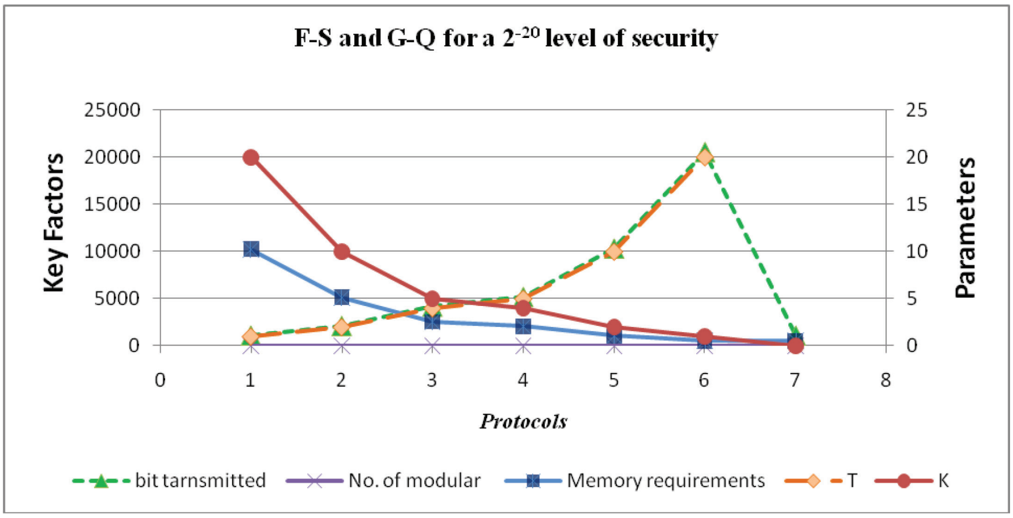
2) User Authentication Scenario

Actors: user, user interface, card reader, verifier process, Java Card applet (prover).

The user authentication scenario is shown in Figure 10. The arrows denote the direction of the information flow and the names on the arrows denote the corresponding data elements.

The user enters PIN through the user interface software, which starts the verification process. The reader interface software operates card reader hardware and provides a means for

Figure 7. Comparison of key factors of protocols



communication between the applet, the user interface and the verifier process. The verification process begins by the reader communicating an authentication request (containing PIN) to the applet. After that, the verifier and the applet perform Guillou-Quisquater protocol as described previously.

C. Library Implementation

The implemented prototype library consists of five Java packages:

- Package applets.lib contains BigInt and RandomBigInt classes implementing arbitrary precision integer arithmetic.
- Package applets.Auth contains Auth class, a JavaCard applet that implements the prover functionality of the Guillou-Quisquater protocol.
- Package verifier contains the Verifier class that starts a daemon process servicing authentication requests submitted via custom protocol running on top of TCP/IP.
- Package auth_process contains SystemInitialization class that calculates private and public keys and communicates them to the prover (Auth applet) and the verifier process respectively.
- Package cardReader contains CardReaderInterface class that specifies the interface to the card reader device for Java applications and provides communication between the JavaCard applet and the verifier process. The implementation of CardReaderInterface is platform dependent. In our case, it was implemented using socket interface to Java Card platform Workstation Development Environment (JCWDE) simulator and to the verifier process.
- Package userInterface contains a set of classes and a standalone Java application providing a window based interface to the user. It allows the user to input the PIN code, after which it then starts the authentication process.

The application based on this authentication library must implement classes derived from (or using) the applet and verifier components. In addition, it may have to provide an implementation for the reader interface component.

D. Implementation Difficulties

Number of technical problems arose during implementation. These may be of interest to other Java Card developers.

Figure 8. Presentation of number of bit transmitted

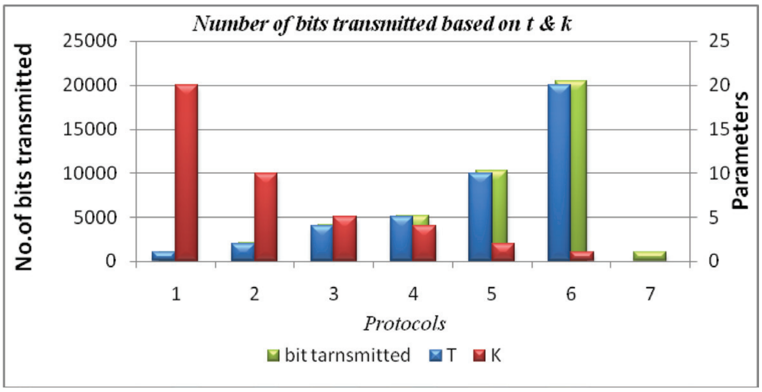


Figure 9. Memory requirements

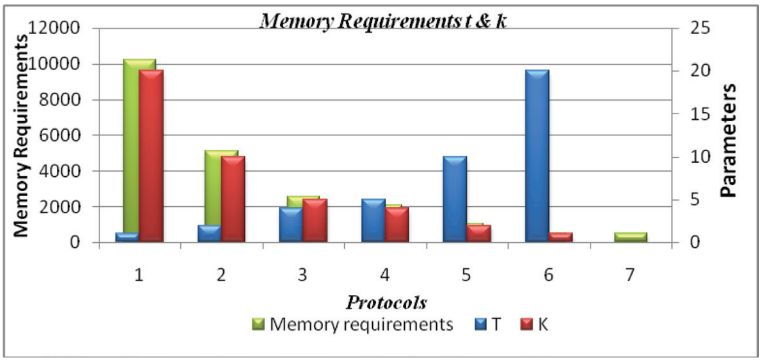
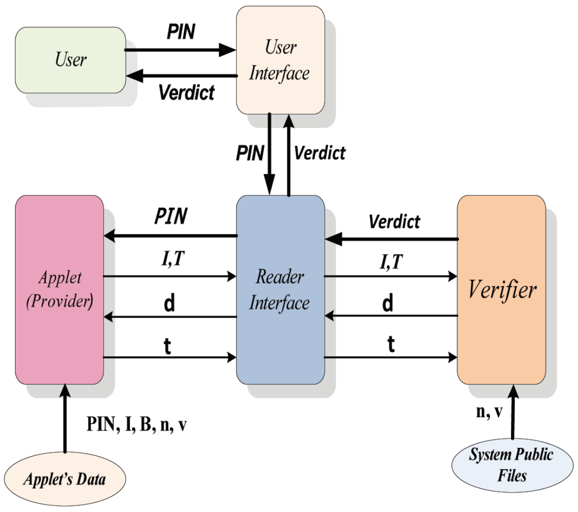


Figure 10. User authentication scenario



Java Cards support only bytes and shorts (), a 16-bit signed two's complement integer to save memory in large arrays in situations where the memory savings actually matters; while the applets in our application need to handle numbers up to 64 bits long. The solution was to implement a library class, residing on the "card", for arbitrary-precision nonnegative integers. The class provides all necessary integer operations for the implementation of the protocol. Modular arithmetic operations have been provided for computing residues and for exponentiation. All operations implement algorithms described by Knuth (1997) with radix 256.

Due to U.S. export regulations on cryptography, the javacardx.crypto package is not included in the JC2RI (Java Card 2.0 Reference Implementation User's Guide), so the class RandomData for generation of pseudo-random arbitrary-precision integers, necessary for the generation of the random numbers, is not available. A class with the same functionality has been implemented.

The simulation requires a suitable implementation of the verifier process that can generate and pass APDUs to the Java Card simulator. The APDU Generator Window for manual generation of APDUs was insufficient for this purpose. A separate Java application employing socket connection to the simulator was developed for this purpose.

Finally, it transpired that the designed library was too heavy for the modern smartcard devices. The code downloaded into the card is 9303 bytes long and requires 478 bytes of variables (assuming $|n| = 512$ bits), which is too much for devices with 16 KB ROM and less than 1 KB RAM. Despite this result, we believe that if the current trends in the smartcard technology are to continue, the future smartcards will be more resourceful and suitable for the developed authentication library. This is also facilitated by Java Card 3 which is a major evolution and upgrade of the Java Card 2 platform. While Java Card 3 enhances the basic security, interoperability, interworking, and multiple-application support in the platform that exploits new higher capacity smartcards hardware features with more and faster memory, higher processing power and far reaching communication capabilities (Allenbach, 2009).

5. CONCLUSION

This paper described the development of a prototype software library providing a zero-knowledge authentication method for smartcards conforming to Java Card specification. In summary, the following goals have been achieved:

- The limitations of smartcards in general and Java Card specification in particular were investigated.
- The problems of zero-knowledge authentication protocols were studied and a comparative analysis of the available protocols was performed, in order to find one most suitable for implementation in a smartcard environment.
- A prototype library implementing the Guillou-Quisquater protocol for use in the Java Card environment was developed, and tested using Java Card simulator provided with the Java Card developer's kit.

The development of the software showed that implementation of zero-knowledge protocols for Java Card programming environment is possible but too unwieldy for existing Java Card devices with limited capacity. Our next big challenge is to experiment zero-knowledge protocols on JAVA Card 3 platform and considering all the issues related to the economics of security and privacy (Katos and Patel, 2008) and how to incorporate evidence-based reputation facilities (Cvrček et al., 2005).

REFERENCES

Allenbach, P. (2009). *Java Card 3: Classic functionality gets a connectivity boost.* Retrieved from http://java.sun.com/developer/technicalArticles/javacard/javacard3/

Aronsson, H. A. (1995). *Zero knowledge protocols and small systems*. Retrieved from http://www.tml.tkk.fi/Opinnot/Tik-110.501/1995/zeroknowledge.html

Beth, T. (1988). Efficient zero-knowledge identification scheme for smart cards. In D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler et al. (Eds.), *Proceedings of the Workshop on Advances in Cryptology* (LNCS 330, pp. 77-84).

Burmester, M., Desmedt, Y., & Beth, T. (1992). Efficient zero-knowledge identification schemes for smart cards. *The Computer Journal*, *35*(1), 21–29. doi:10.1093/comjnl/35.1.21

Chen, Z. (2000). *Java card technology for smart cards: Architecture and programmer's guide*. Upper Saddle River, NJ: Prentice Hall.

Chen, Z., & Giorgio, R. D. (1998). *Understanding Java Card 2.0-Java World.* Retrieved from http://www.javaworld.com/javaworld/jw-03-1998/jw-03-javadev.html

Coleman, A. (1998). *Giving currency to the Java Card API.* Retrieved from http://www.javaworld.com/javaworld/jw-02-1998/jw-02-javacard.html

Cvrček, D., Matyáš, V., & Patel, A. (2005). Evidence processing and privacy issues in evidence-based reputation systems. *Computer Standards & Interfaces*, *27*(5), 533–545. doi:10.1016/j.csi.2005.01.011

Fiat, A., & Shamir, A. (1986). How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko (Ed.), *Proceedings of the Workshop on Advances in Cryptology* (LNCS 263, pp. 186-194).

Goldreich, O., Micali, S., & Wigderson, A. (1991). Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, *38*(3). doi:10.1145/116825.116852

Goldwasser, S., Micali, S., & Rackoff, C. (1985). The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing* (pp. 291-304).

Guillou, L., & Quisquater, J. J. (1988). A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler et al. (Eds.), *Proceedings of the Workshop on Advances in Cryptology* (LNCS 330, pp. 123-128).

Guillou, L., & Quisquater, J. J. (1990). A "paradoxical" dentity-based signature scheme resulting from zero-knowledge. In S. Goldwasser (Ed.), *Proceedings of the Workshop on Advances in Cryptology* (LNCS 403, pp. 216-231).

Herbst, C., Oswald, E., & Mangard, S. (2006). An AES smart card mplementation resistant to power analysis attacks. In J. Zhou, M. Yung, & F. Bao (Eds.), *Proceedings of the 4th International Conference on Applied Cryptography and Network Security* (LNCS, 3989, pp. 239-252).

http://www.oracle.com/technetwork/java/javacard/javacard1-139251.html

International Organization for Standardization. (1987). *ISO/IEC 7816: Electronic identification cards with contacts, especially smart cards 15 minus 1 Part Series*. Geneva, Switzerland: International Standards Organisation (ISO) and the International Electrotechnical Commission (IEC).

International Organization for Standardization. (2000). *ISO/IEC 14443: Proximity cards (PICCs) 4 Part Series*. Geneva, Switzerland: International Standards Organisation (ISO) and the International Electrotechnical Commission (IEC).

ITU-T. (2005). *ITU-T recommendation X.509/ISO/IEC 9594-8: Information technology. Open systems interconnection - The directory: Public-key and attribute certificate frameworks*. Retrieved from http://www.infosecurity.org.cn/content/pki_pmi/x509v4.pdf

Kapron, B., Malka, L., & Srinivasan, V. (2007). A characterization of non-interactive instance-dependent commitment-schemes (NIC). In *Proceedings of the 34th International EATCS Colloquium on Automata, Languages and Programming* (pp. 328-339).

Katos, V., & Patel, A. (2008). A Partial Equilibrium View on Security and Privacy. *Information Management & Computer Security*, *16*(1), 74–83. doi:10.1108/09685220810862760

Knuth, D. E. (1997). The art of computer programming: *Vol. 2. Seminumerical algorithms* (3rd ed.). Reading, MA: Addison-Wesley.

Kocher, P., Jaffe, J., & Jun, B. (1999). Differential power analysis. In M. J. Wiener (Ed.), *Proceedings of the Workshop on Advances in Cryptology* (LNCS 1666, pp. 388-397).

Meckley, J. (1998). *Definition - Smart card.* Retrieved from http://searchsecurity.techtarget.com/definition/smart-card

Micali, S., & Shamir, A. (1990). An improvement of the Fiat-Shamir identification and signature scheme. In S. Goldwasser (Ed.), *Proceedings of the Workshop on Advances in Cryptology* (LNCS 403, pp. 244-247).

Nguyen, M.-H., & Vadhan, S. (2006, May 21-23). Zero knowledge with efficient provers. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, Seattle, WA (pp. 287-295).

Ohta, K., & Okamoto, T. (1990). A modification of the Fiat-Shamir scheme. In S. Goldwasser (Ed.), *Proceedings of the Workshop on Advances in Cryptology* (LNCS 403, pp. 232-243).

Ong, S., & Vadhan, S. (2007). Zero knowledge and soundness are symmetric. In M. Naor (Ed.), *Proceedings of the 26th Annual International Conference on Advances in Cryptology* (LNCS 4515, pp. 187-209).

Oracle. (2010). *Java Card platform specification 2.2.2*. Retrieved from http://java.sun.com/javacard/specs.html

Oracle. (2010). *Smart Card overview-Chip comparisons*. Retrieved from http://www.oracle.com/technetwork/java/javacard/documentation/smart-cards-136372.html#chart

Oritz, C. E. (2003). *An introduction to JAVA Card technology – Part 1*. Retrieved from.

Patel, A. (2010). Concept of mobile agent-based electronic marketplace – Safety measures . In Lee, I. (Ed.), *Encyclopedia of e-business development and management in the digital economy* (*Vol. 1*, pp. 252–264). Hershey, PA: IGI Global.

Peyret, P. (1995). *Which Smart Card technologies will you need to ride the information highway safely*? Retrieved from http://www.gemalto.com/gemplus/index.html

Quisquater, J. J., Quisquater, M., Guillou, L., Guillou, M., Guillou, G., Guillou, A., et al. (1990). How to explain zero-knowledge protocols to your children. In G. Brassard (Ed.), *Proceedings of the Workshop on Advances in Cryptology* (LNCS 435, pp. 628-631).

Schneier, B. (1996). *Applied cryptography: Protocols, algorithms, and source code in C* (2nd ed.). New York, NY: John Wiley & Sons.

Schnorr, C. P. (1990). Efficient identification and signatures for smart cards. In G. Brassard (Ed.), *Proceedings of the Workshop on Advances in Cryptology* (LNCS 435, pp. 239-251).

Sun Microsystems. (1997). *Java Card 2.0 reference implementation user's guide Java Card 2.0 programming concept*. Retrieved from http://www.it.iitb.ac.in/~satish/phd/smartcard/usinix_99/javacardapi21/jc2ri-users-guide.pdf

Vadhan, S. P. (2004). An unconditional study of computational zero knowledge. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science* (pp. 176-185).

Wei, Q., & Patel, A. (2009). A secure and trustworthy framework for mobile agent-based e-marketplace with digital forensics and security protocols. *International Journal of Mobile Computing and Multimedia Communications*, *1*(3), 8–26. doi:10.4018/jmcmc.2009070102

Ahmed Patel received his MSc and PhD degrees in Computer Science from Trinity College Dublin (TCD) in 1978 and 1984 respectively, specializing in the design, implementation and performance analysis of packet switched networks. He is a Professor in Computer Science at Universiti Kebangsaan Malaysia. He is visiting professor at Kingston University in the UK. He has published over two and ten hundred technical and scientific papers and co-authored several books. He is currently involved in the R&D of cybercrime investigations and forensic computing, intrusion detection & prevention systems, cloud computing autonomic computing, Web search engines, e-commerce and developing a framework and architecture of a comprehensive quality of service facility for networking protocols and advanced services. He is a member of the Editorial Advisory Board of the following International Journals: (i) Computer Standards & Interface, (ii) Information Management & Computer Security and (iii) Cyber Criminology.

Kenan Kalajdzic received his BSc degree in Electrical Engineering and his MSc in Telecommunications and Computer Science from University of Sarajevo, Bosnia and Herzegovina. His interests span a wide range of topics in the area of operating systems and computer security. He is currently working as a lecturer at the Center for Computing Education in Sarajevo, and as an external visiting researcher with Prof. Ahmed Patel at the Universiti Kebangsaan Malaysia. He has published 4 papers. He is a reviewer of papers for Computer Standards & Interface Journal.

Laleh Golafshan received her B.S. from Islamic Azad University, Iran in year 2005, and her M.S. degree in (Computer Science) from Universiti Kebangsaan Malaysia (UKM), Department of Computer Science in 2011. Prior to her Masters degree, she ran a Private IT Training College in Shiraz. Her research interests are data mining, optimization and classification; and undertaking research in software engineering and computer security in collaboration with Prof. Dr. Ahmed Patel. Currently, she teaches computer engineering and IT courses as an instructor in Islamic Azad University Fars Science and Research Branch and also performs researcher at this university.

Nice Mona Taghavi, a.k.a. CMT, received her B.Sc. degree in Information Technology from Parand Islamic Azad University of Iran in 2007. Besides her involvement in several Iranian national ICT research projects, she had worked for an IT consulting and project managing company which was responsible for overseeing and preparing some of the technical reports for the Supreme Council of Information and Communication Technology (SCICT) of Iran programme. Currently, she is pursuing her MSc in Information Systems at Universiti Kebangsaan Malaysia and undertaking research in cooperation with Prof. Dr. Ahmed Patel in advanced secure Web-based information systems and Secure Mobile Agent-based E-Marketplace Systems. She has published 4 papers. She is a reviewer of papers for Computer Standards & Interface Journal.