A Fundamentally Secure Payment Device Interfaced to Regular PCs

Abdelhafid Bouhraoua and Metub Al-Shammari
Computer Engineering Department
King Fahd University Of Petroleum and Minerals (KFUPM)
Dhahran, Saudi Arabia
Email: abouh@kfupm.edu.sa, almsafer9@hotmail.com

***Abstract* — The present contribution introduces a new way for solving the issue of security for payments over the internet. It particularly addresses the issues related to the PC weaknesses like the combination of key loggers and spyware software. The device uses exclusively symmetric encryption (AES) that ties the device directly to the payment server base at fabrication time. The device is connected to the PC through the USB interface from which it takes its power. The platform architecture is built around three entities: a I/O processor (IOP) responsible for the communication and user interface and a management of keys processor (MKP), responsible for all of the messages processing. Encryption is assured by a dedicated hardware engine for increased performance. The device is made known to the payment server at fabrication time through the assignment of a device ID. Both the server and the device will use secret keys known only to the two parties. This way, the authentication and security are guaranteed at the source. The device ID along with the device and server set of keys are assembled in a data storage packet, scrambled, encrypted by a completely secret device internal key, and stored on a local serial EEPROM. Moreover, the EEPROM setting procedure is a one way procedure where no way of reading back the clear device ID and set of keys is available. The strength of this approach is the fact that the device ID is associated with a set of device keys within the payment server database.**

***Index Terms* — Security, AES, Secure Trusted Device, Payment Systems, Nonce**

I. Introduction

The internet today has gained enormous ground in people's life that it is becoming a necessary tool for the daily activities of many. E-commerce is not an exception in this regard. Online commercial transactions requiring, by essence, higher security levels than many other internet-based activities, have nevertheless attracted the increasingly growing number of cyber criminals who everyday are trying to defeat the more and more complex infrastructure.

The development of the *Secure Sockets Layer* (SSL), which was replaced by the *Transport Security Layer* (TSL) has provided the necessary infrastructure that solved the problem of sending a secure message from an anonymous client to a trusted server. The *Secure Electronic Transaction* (SET) [8] protocol was developed to authenticate the servers by using authentication certificates that provides means to make payments to merchants without disclosing client credit card numbers to them. However, SET has not been deployed massively yet due to complex procedures[7]. The most widely used solutions in online payment are based on sending credit card numbers on the SSL secure channel. Some solutions provide a way to authenticate the credit card holder by requiring users to enter a password tied to the bank issuing the credit card.

All of these solutions are focusing on preventing eavesdroppers from accessing sensitive information like credit card numbers and client personal information. They are also addressing *man-in-the-middle* attacks by authenticating the servers. These solutions assume that the personal computer is a safe an trusted device. In reality, with the current security hazards, the PC can no longer be considered as a trusted device[9]. For example, using a combination of spyware software and keystroke logger anyone can remotely access any information typed on the PC keyboard[9]. The current effort aims at addressing this particular issue of making secure payments online in the context of the PC not being a fully trusted device. The approach proposed here takes the path of a hardware device that is connected to the PC and that takes care of processing the payment in conjunction with a trusted server.

This paper is organized into nine sections starting with the current introduction. Section II presents and discusses the previous work focusing on the hardware solutions. The next section details the principle of operation. The transaction sequence is detailed in section IV while the device's processor architecture is presented in section V. Section VI discusses the immunity to the most common attacks. The implementation is presented in section VII. Performance requirements are determined in section VIII. Finally, section IX concludes this paper.

II. Previous Work

Several approaches have been proposed to tackle the problem of secure payment and transactions on the internet through the PC.

The use of a secure display is a new trend in the arena of secure trusted terminals. It is mainly based on providing a secure display terminal that should be used in conjunction with personal mobile phones or PDA to initiate the transactions[2]. Other proposals recommend the use of mobile phones for their ubiquitous nature[3][4][6]. These solutions are based on the belief that mobile phones and PDA are immune against malicious software. This assumption is starting to be shaken with the appearance of the first mobile phone viruses.

Another set of solutions is proposing to modify some of the existing protocols like merging the SET protocol with reputation systems to automatically and securely use the merchant reputation information to add confidence in using the SET protocol[7]. Another solution rely on increasing the amount of information contained within key certificates[10]. The solutions that focus on the protocols are not immune to the PC weaknesses like key logging software and spyware.

III. Principle Of Operation

The main idea of this proposal is to perform all the transactions that contain sensitive information such as payment amount, credit card numbers or merchant ID within an external device attached to the PC. This device is made known to the payment server at fabrication time through the assignment of a device ID. Both the server and the device will use secret keys known only to the two parties. This way, the authentication and security are guaranteed at the source. The encryption algorithm used is the *Advanced Encryption Standard,* AES [] adopted by the NIST in 2001. The AES algorithm is used for all the encryptions. AES is a symmetric key algorithm which means the same key is needed on both sides of the secure channel to cipher and decipher the messages.

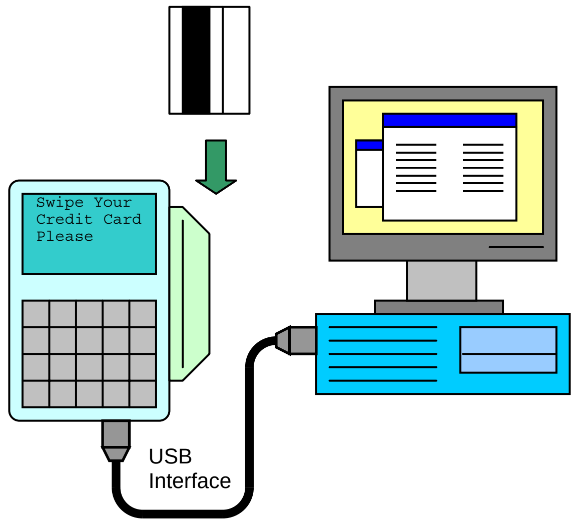

Figure 1 – Principle of Operation

The device comprises a display screen used to display messages to the user and a keypad used to accept user's input securely instead of using the insecure PC keyboard. The messages displayed on the device screen are simple prompts inviting the user to enter credit card numbers or to insert the credit card in the card readers for the versions equipped with card readers. Other messages like confirmation and acknowledge messages from the server are also displayed on the device.

Communication with the PC is realized through simple interface protocols such as USB or simple RS-232. However, USB is preferred as most recent PCs do not provide RS-232 ports. Another advantage of the USB interface is its power capability. The device can therefore be directly powered out of the USB interface reducing the overhead and size.

The fact that a device may have a single secret key may be sensitive to dictionary attacks. To remove such a weakness, both the device and the server will be associated to several keys. Every transaction, both the device and the server will randomly select one of the key to cipher the messages. Therefore, the device should memorize two lists of keys: the device list and the server list. This will introduce a little overhead since the deciphering of the messages will try each key until it finds the proper one that decrypts the message. For this reason messages will be CRC protected and start with a fixed pattern so that the successful deciphering will be easily identified on the receiver's side (device or server).

IV. Transaction Sequence

The transaction sequence starts when the user initiates the process by checking out of a merchant store. Actually after selecting one of the related methods of payment (method that is tied to the use of the payment server that uses the device), the browser starts by sending a first message *M1* to the payment server. Figure 2 shows the transaction sequence showing the different messages sent and the different entities involved.

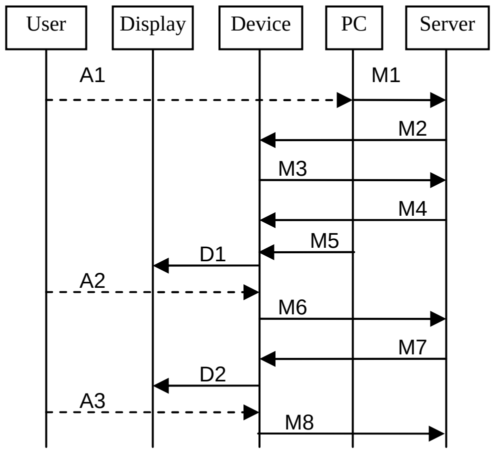

Figure 2 – Transaction Sequence Chart

The following list details the content and the encryption status and key of every message. The following notations are used throughout this section and when needed in the rest of the paper:

- *E(M, K)*: represents a message *M* encrypted with the key *K*. It actually represents a message *M* already in its *cipher text* mode after it has been encrypted with key *K*.

- K_S represents a payment server key
- K_D represents a device key
- K_N represents a randomly generated key used once and commonly referred to as *nonce* in the security terminology. In the present scheme, the key is generated by the server.

The list of messages shown in Figure 2 is defined below:

- *A1* represents the user action that initiates the payment transaction by selecting the payment method on the merchant's web site.
- *M1* is a plain text message that contains a request for transaction
- *M2* is a plain text message that is used to ask for device identification (and authentication)
- *E(M3,* K_S*). M3* contains the device ID.
- *E(M4,* K_D*). M4* generation: Using the device ID, the server will look up the device key K_D in its device database. It will use the device key K_D to encrypt the nonce key K_N.
- *M5* is a plain text message that contains: the merchant ID (within the server's database) and the total charge amount.
- *D1* represents the display/input script communicated by the PC to the device to ask the user to enter its information. It will also display the merchant name and amount. The user is asked to confirm these choices.
- *A2* represents the user series of input actions such as: credit card swiping or amount confirmation
- *E(M6,* K_N*). M6* contains all the sensitive information encrypted with the session key *KN*.
- *E(M7,* K_N*). M7* is a positive/negative acknowledge message that is received from the server to confirm/infirm the transaction.
- *D2* displays the name associated with the merchant ID.
- *A3* represents the user action of selecting to confirm or cancel the current transaction
- *E(M8,* K_N*). M8* contains either the confirmation or the cancellation of the transaction as chosen by the user after being prompted by *D2*.

Messages generated are not directly ciphered but are embedded within a larger fixed size packet containing randomly generated bulk data. This way, using the same device key or server key to cipher these packets will not result in an identical bitstream.

V. Hardware Architecture

A. Overview

The device functionality can be divided into two categories:

- User/PC interface category which relates to the display, user input and communication with the PC
- Encryption and key management category which deals with all the aspects related to creating and managing the messages and the different keys.

In this scope, the proposed architecture is built using three main entities:

- I/O processor, depicted as *IOP*
- Message and key processor or *MKP*
- Encryption/Decryption engine or *EDE*

Figure 3 shows the block diagram of the SoC (*System-on-Chip*) that implements these different entities. The idea of having two different processors, each dedicated to a particular set of tasks, is motivated by a security measure that is to keep the message and key management firmware inaccessible and hardwired while at the same time allowing the IO software to be updated as needed.

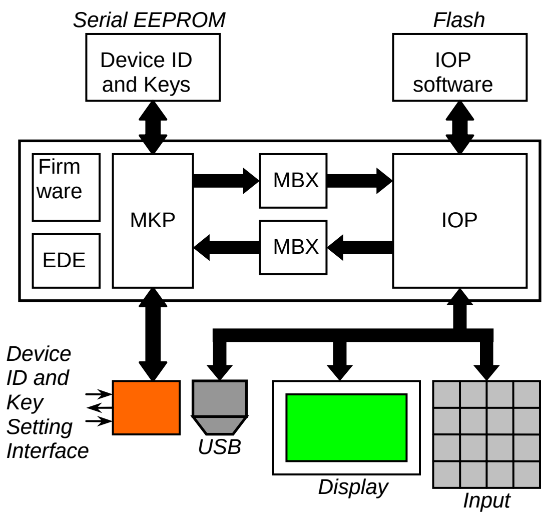

Figure 3 – Hardware block diagram

Communication between the IOP and the MKP happens through a mail box structure. Each processor writes into the mail box of the other processor the message it intends to send to the other processor. A small notification that contains the message length, starting address and type of message is sent separately into a notification register set

The IOP software is stored externally on a flash EEPROM while the device ID and device keys are stored on a serial EEPROM. Both processors are attached to internal temporary storage memories used to execute the programs, to store temporary program variables and help assemble the messages.

B. Securing the device ID and the keys

The contents of the serial EEPROM are encrypted with a device internal key. This way, the device ID and the device and server keys are protected from attempts to recover their values by simply reading the contents of the EEPROM.

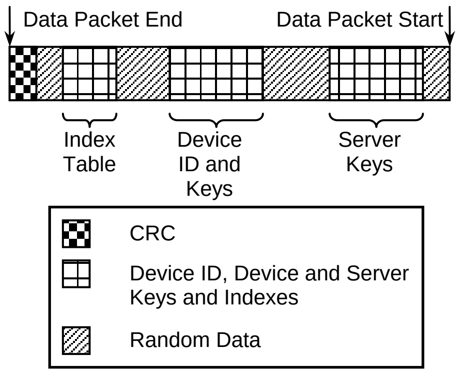

Figure 4 – Storage Data Packet Structure

Another protection barrier is put in the EEPROM by encoding the storage data structure in a way that makes it more difficult to recover in the case the internal key is known. Figure 4 shows the structure of the storage data packet that stores the device ID, the device keys and the server keys on the EEPROM. The main idea is to embed the critical information within a frame of randomly generated data to completely hide it before encryption. Another complexity is added furthermore when the device ID and key list elements are not stored in an increasing index order but are scrambled using a randomly generated order. The right order is kept in an index table. For example, if the list of server keys totals 16 keys, they are not stored as key1 then key2 and so on but can be key7 then key3 then key16. In other terms, the sequential order of the keys is violated on purpose for added protection.

The device ID and keys setting procedure is also designed to protect from unauthorized readings of the EEPROM. The setting procedure is a one-way procedure that cannot read the EEPROM contents for verification. It sends a CRC protected string of defined length. The string is encoded, encrypted and written into the EEPROM. A copy of the initial string is kept internally in the device during the procedure. The EEPROM content is read back, decrypted and decoded. The resulting string is compared with the initial string for compatibility. The operation is repeated several times to ensure a high level of confidence in the data stored in the EEPROM. This verification is completely carried out by the device internally. At the end of the verification, a positive/negative notification is sent back to the external setting equipment.

C. The Display, Keypad and Card Reader

The IOP is in charge of communication and user interface. For added flexibility, the display uses simplified HTML format to display the forms and fields to the user. The HTML format is versatile and suitable for describing language independent, customizable user interface.

VI. COUNTERING ATTACKS

A. Playback Attack

Playback attacks are prevented through the use of sophisticated methods based on: message scrambling and random key selection. These two measures will produce, for example, a different message value when sending the same device ID at different times which makes the task of memorizing all the combinations unrealistic.

B. Man-in-the-middle Attacks

The proposed setting is immune to man-in-the-middle attacks because it needs prior knowledge of the secret keys needed to decipher the received messages in order to provide answers.

C. Reverse Engineering Attack

The idea of reverse engineering poses many problems. The level of technology to reverse engineer a chip in today’s technology needed is way beyond the reach of common hackers. The internal device key used to encrypt the storage data packet is randomly generated and engraved in the silicon, either through the use of randomly generated metal or via masks or through the use of fuse PROMs. Even if this step is achieved it will only give the pirates the device IDs of the devices it can physically access after they reverse engineer the scrambling methods and encryption keys. For the other devices, the internal encryption key has to be guessed if no reverse engineering is possible. It is important to mention that the reverse engineering methods are destructive methods where all the layers are removed one by one and high precision photographs are taken at every step. The process of recovering the functionality from the transistor level may take years that are sufficient to introduce modifications onto the architecture.

VII. IMPLEMENTATION

A preliminary implementation, which goal was mainly to test the protocol and the transaction sequence, has been achieved. This implementation was realized using the Rabbit RCM3700 microprocessor core using the Dynamic C[1] specific programming language. The USB interface is a simple USB-to-serial device that provides seamless connectivity to the USB without dealing with the USB protocol programming details. Figure 5 shows a glimpse of the RCM3700 attached to the USB-to-serial converter through which it interfaces to the host PC (not shown).

[1] © Rabbit Semiconductor Corporation

The implementation has been completed and tested along with the server side application, developed using C# and .NET. The transactions have been fully tested and verified.

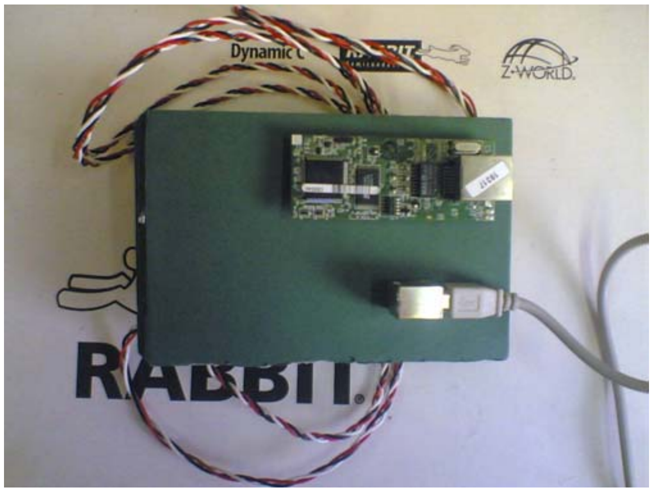

Figure 5 – Implementation

VIII. Performance Requirements

On the terminal side the processing speed requirement falls within the realization of the message processing and encryption within few hundreds of milliseconds which is enough for being unnoticed at the human level. This requirement means a processor speed less than 10 MHz for it to execute, at most, around one million instructions to achieve that. Given the fact that the AES encryption is performed within a dedicated hardware engine, this requirement can be lowered significantly.

On the server side, the overhead lies within the fact that to every device ID, an entire set of keys is associated. This translates into multiple attempts to decipher received messages by trying every key in the set until the used one is found and the message is successfully deciphered. This handicap can easily be addressed with the use of hardware encryption/decryption engines. An example of such devices can reach up to 500 Gbits/s[11] per chip. At these rates, a single chip can easily accommodate transactions originating from up to 64 million different devices with messages of 512 bits (64 bytes) and 16 keys per set.

IX. Conclusions and Future Directions

A different approach for online payment systems has been proposed. The approach is based on the use of an external device, connected to the PC through USB, and where all the transaction processing is performed. This device uses secret server keys, set during fabrication, to communicate with the servers. It also has a unique ID that identifies it within the server. This ID is used by the server to retrieve the set of keys associated with the device. It will select one of the device keys to send a randomly generated session key that will be the base for the subsequent steps of the payment transactions.

The set of transactions has been defined in detail. A series of additional countermeasures have been specified for avoiding playback attacks. The device's hardware architecture has been presented showing the partitioning between the I/O related tasks and the message processing tasks. An external EEPROM is used to store the critical device information (device ID, device key set and server key set). The procedure to set these critical parameters has been presented. The method for protecting the information stored in the EEPROM has been presented. Different attack scenarios have been envisaged and related risks have been qualitatively evaluated.

An initial implementation has been realized that was exclusively used to test the different transactions and to show a demonstrator. In the future a more thorough implementation using an FPGA platform will be carried out. It will also constitute a test platform used to experiment more types of attacks.

Acknowledgment

Facilities support by King Fahd University of Petroleum and Minerals is highly appreciated by the authors.

References

[1] *Advanced Encryption Standard (AES),* Nov. 26, 2001.

[2] S. Ghotra, B. Mandhan, S. Shang, C. Wei, Y. Song and C. Steketee, "Secure Display and Secure Transactions Using a Handset", In *Proceedings of the Sixth International Conference on The Management of Mobile Business (ICMB 2007), Toronto, Canada,* July 2007, pp 51-58.

[3] A. Oprea, D. Balfanz, G. Durfee, et al., "Securing a Remote Terminal Application with a Mobile Trusted Device", *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04), IEEE Computer Society*.

[4] J. Porras, P. Jäppinen, P. Hiirsalmi, et al., "Personal Trusted Device in Personal Communications", *1st International Symposium on Wireless Communication Systems* Mauritius, 2004, pp. 388-392.

[5] Z. Djuric, "IPS – Secure Internet Payment System", In *Proceedings of the International Conference on Information Technology, Coding and Computing (ITCC'05), Seoul, Korea,* May 2005, Vol. 1 pp 425-430.

[6] B. Althen, G. Enste, B. Nebelung, "Innovative Secure Payments on the Internet using the German Electronic Purse", *In Proceedings 12th Annual Computer Security Applications Conference, San Diego, USA*, December 1996, pp 88-93.

[7] M. Kinateder and K. Rothermel, "Bringing Confidence to the Web – Combining the Power of SET and Reputation Systems", *In Proceedings of the First IEEE Consumer Communications and Networking Conference, Las Vegas, USA,* January 2004, pp 545-550.

[8] Visa International and Mastercard International, "SET Secure Electronic Transaction Specification Book 3: Formal Protocol Definition," May 1997.

[9] D. Sullivan, *The Definitive Guide to Security Management*, 1 ed, Realtimepublishers, Santa Rosa, 2004.

[10] Blerim Rexha, "Increasing User Privacy in Online Transactions with X.509 v3 Certificate Private Extensions and Smartcards", *Proceedings of the Seventh IEEE International Conference on E-Commerce Technology (CEC'05),* 19-22 July 2005, pp 293-300.

[11] A. Bouhraoua, "Design Feasibility Study For a 500 Gbits/s AES Cypher/Decypher Engine", *Proceedings of the International Conference on Microelectronics (ICM'06),* 16-19 Dec. 2006, pp 190-193.