SECURE APPLICATION UPDATES ON POINT OF SALE DEVICES

Manuel Mendonça
Faculdade de Ciências da Universidade de Lisboa
Bloco C6, Campo Grande, 1749-016 Lisboa - Portugal
Email: mendonca27@sapo.pt

Nuno Ferreira Neves
Faculdade de Ciências da Universidade de Lisboa
Bloco C6, Campo Grande, 1749-016 Lisboa - Portugal
Email: nuno@di.fc.ul.pt

Keywords: Electronic payment systems, point of sale devices, secure application downloads

Abstract: Currently, a large number of electronic transactions are performed with credit or debit cards at terminals located in merchant stores, such as Point of Sale Devices. The success of this form of payment, however, has an associated cost due to the management and maintenance of the many equipments from different generations and manufacturers. In particular, there is an important cost related to the deployment of new software upgrades for the devices, since in most cases human intervention is required. In this paper we describe a secure solution for this problem, where Point of Sale Devices are able to automatically discover and upload new software updates.

1 INTRODUCTION

Even though several electronic payment protocols have been developed for the Internet in the past years (Bellare et al., 2000; Manasse, 1995; Schoenmakers, 1998; MasterCard and Visa Corporations, 1997), currently most of the electronic transactions continue to go through closed banking networks. Several reasons explain the success of these networks, but among them, two are specially important – it is relatively easy to obtain a debit or credit card from a financial institution, and there is a large number of terminals, such as *Point of Sale* (POS) devices or *Automated Teller Machines* (ATM), where these cards are accepted.

At the beginning, when these devices started to be available, they were relatively expensive and had limited capabilities. Basically, they allowed operations like purchase or cash withdraw. However, as the specifications of the equipments employed in a transaction went through a standardization process, many companies began to offer competing terminals at cheaper prices, which resulted in the current situation where most stores have a POS, and ATMs have been widely deployed. Moreover, the type of services provided by these devices has evolved, allowing for instance the payment of electrical or insurance bills, purchase of train or music concert tickets, deposits of checks, or money transfers between accounts.

The success of these networks has a cost associated with the management and maintenance of the considerable number of equipments from different generations and manufacturers. In particular, an important cost arises from keeping the software being run in the devices updated. Contrarily to the hardware that once deployed it stays stable for a long period, the *Electronic Funds Transfer* (EFT) applications can suffer several changes during the lifetime of the devices. Various causes justify the need for software changes, for example, the creation of new services, the year two thousand (potential) bug, or recent events such as the introduction of the euro. Besides these causes for modification, there are always the usual factors related to the maintenance of any software: undetected bugs during the testing phase or security problems.

Nowadays, in most countries, software updates continue to be performed manually. Once a new EFT application has been certified by the entity responsible for the network, the *Payment Network Operator* (PNO), the new version is given to a maintenance company that will be in charge of the rest of the process. This will require a technician to go to the location where the equipment is placed, and then the equipment has to be dismantled so that the chip where the application is stored can be substituted by another. In some occasions, just to save time, it is more cost effective to simply exchange the terminal with a new one.

This procedure has many efficiency problems, some of them related to cost and others to time. For example, in Portugal, the POS network contains around 140.000 devices. If each software update has a fee of 50 euros and takes 20 minutes, then a complete substitution of the network would cost 7 million euros and would take more than 46 thousand person hours. Moreover, from a security point of view, the current procedure is far from the ideal because it requires complete trust on the parties that will do the actual software update and the PNO only controls the operation in a limited way.

The specifications related to the Europay, Master Card & Visa (EMV) (Europay, MasterCard and Visa Corporations, 2000) suggest that a POS should have the capability of uploading new versions of the software. However, these specifications do not provide any concrete solution to the problem. Furthermore, currently available solutions are proprietary. During our research, we found two POS manufacturers claiming to support remote software updates. However, when contacted by us, they did not provide any information about their protocols.

In this paper we propose a solution for the secure update of EFT applications in POS devices. Since we wanted to build a solution that could be used in a real banking environment, we decided to base our work on the Portuguese banking network, that is called *Multibanco*. However, since the architecture and protocols are relatively generic, we feel that our ideas can be applied to other networks (possibly with small changes).

2 BACKGROUND: MULTIBANCO PAYMENT NETWORK

The Multibanco payment network carries most of the electronic transactions made through POS and ATM in Portugal (European Central Bank, 2001). The architecture of the network is based on a client-server configuration, where on one side resides the POS (or ATM) and on the other a server maintained by the PNO. Currently, more than 140.000 devices are linked using different network technologies, ranging from public telephone connections with X.25 to wireless networks like GSM.

The EFT protocol is organized as a request-response transaction. The POS always has the initiative of starting the communication by contacting the PNO server, called the *Payment System Server* (PSS). The message contains several fields, and among them there is the identifier of the selected service. Once the message arrives, the PSS executes the service and then returns a response indicating the success or failure of the operation. A Message Authentication Code (MAC) secures both the request and response against integrity and authenticity attacks. Each POS shares with the PSS a number of symmetric keys that are used to secure the communications. These keys are stored in a secure module of the POS to ensure that they are physically protected from tampering. If the PSS wants to make the POS perform some action, it has first to wait for the contact of the POS. Then, in the response, it can include the code of a transaction that should be executed next.

As an example, lets look at a payment transaction with a debit card in a store. The merchant gets the card from the client, and uses the POS reader to input the data saved in the magnetic stripe of the card. This data includes information about the account number and bank of the client. The vendor types the cash amount using the POS keyboard and lets the user insert her *Personal Identification Number* (PIN). Next, the EFT application executing in the POS constructs a message containing the information related to the transaction: the service code, a sequence number, the device unique identifier, date, amount, client account data, and the PIN encrypted with a key stored in the POS. When a message arrives, the PSS performs several tests to verify the correctness of the received information. For instance, it validates the PIN that was input to make sure that the client is the owner of the card. If the PIN is wrong, the user gets two other tries before the card becomes permanently blocked. On every occasion a test fails, a response is returned to the EFT application indicating the problem. Next, the PSS contacts the client and merchant banks to process the payment. If the client bank authorizes the debit, then the PSS can inform the merchant bank to credit the requested amount on the vendor's account. The banking information about the merchant is known to the PSS because it is associated with the POS unique identifier, which is stored in a PNO database. If all steps are concluded with success, then the PSS can send an OK message to the EFT application. Otherwise, an ERROR message is returned.

3 SECURE APPLICATION UPDATES IN POS

The update of an EFT application requires the execution of two main tasks. First, a new version of the software needs to be produced by some manufacturer and certified by the organization supervising the network (the PNO). The certification procedure is an important step because it not only needs to guarantee that the upgrade satisfies the specification, but also has to ensure that all bugs and/or security problems have been removed (at least as many as possible). As we mentioned in the Introduction, several reasons might contribute for the necessity of a new version of the application.

The second task comprises all activities related to the upload of the application to the POS. Ideally, the PNO should control this process since, in the end, it is responsible for the good performance of the network. Moreover, the whole procedure should be as automatic as possible because this allows rapid deployment of new services, and potentially reduces costs due to human intervention. Since electronic payment transactions will be executed in the POS, the security of the process is also very important.

The proposed update procedure, besides guaranteeing the just mentioned objectives, also has some other interesting characteristics such as: it maintains accurate information about the POS and EFT applications currently deployed, which is an attractive feature if one wants to use the system to support managing decisions.

3.1 System Architecture

The architecture of the payment system is displayed in Figure 1. It includes both the POS in the merchant store and the PSS located in the PNO. In order to support automatic updates of the EFT applications, the PNO has to offer two interfaces to the outside: one to the *Software Manufacturers* (SM) and another to the POS.

The interface to the SM is provided by a new server, called the *Certification System Server* (CSS). CSS is in charge of all exchanges with the SM, and in particular coordinates the certification process. Whenever a new application is developed, it should be submitted through the CSS for validation. Then, a number of tests are performed to ensure the quality of the software, and a report is returned. Ideally, the analysis would be done by machines in a controlled testing environment, however, for the time being, one would expect (and accept) some human intervention. The communication between the CSS and the SM needs to be secured to prevent attacks that could try, for instance, to change the software upgrade.

At first, one might feel tempted to re-use the PSS as the interface to the POS for the application updates. In practice, this solution has some problems because it takes a reasonable interval of time to download an application (see Section 4), which could degrade significantly the performance of the PSS (do not forget that the main task of the PSS is to accept payment transactions). Therefore, we introduce a new server, called the *Application Distribution Server* (ADS), that will store the code sent by the CSS and transfer the updates to the POS.

The EMV specification for cards with a chip uses asymmetric cryptography to secure the electronic payment transactions (Europay, MasterCard and Visa Corporations, 2000). Therefore, since it is expectable in the near future the replacement of magnetic stripe cards with chip cards, we decided also to use asymmetric cryptography to protect the transactions of the update protocol. The *Certification Authority* (CA) plays an important role in the security of the whole system because it manages the certificates with the public keys of the various components.

The CA has to perform several functions: it needs to reliably authenticate the entities that require the creation of new certificates (as defined by the security policy); it generates the certificate revocation lists (CRL) whenever a certificate needs to be cancelled (e.g., the private key was compromised); and it also carries out other management functions, such as the archival of all generated data.

3.2 Update Protocol

The update procedure of the EFT applications is implemented by a set of sub-protocols, each one responsible for a specific task (see Figure 1). Due to space limitations it is not possible to provide all details about the various fields of the messages and their validation process. However, since most messages are protected using the same mechanism, we will give here a generic description (the only exception are the EFT protocol messages that are secured using the standard MAC scheme, as mentioned in Section 2). Whenever a component A sends a message $data$ to a component B, it constructs a message with $< data, E(KrA, Hash(data)), CERT\_KuA >$, where $E(KrA, Hash(data))$ is a signature and means enciphering an hash of the $data$ ($Hash(data)$) with the private key KrA of A. The message also includes all certificates with the public keys necessary to the validation of the signature, which in this case is $CERT\_KuA$. Once a message arrives, the receiver makes the following verifications: $CERT\_KuA$ is validated using the public key of the CA that is stored in $CERT\_KuCA$; the signature is verified by deciphering $E(KrA, Hash(data))$ with KuA, and comparing the result with the hash of the received $data$. If they are equal then the message is correct.

Certificate Distribution Protocol This protocol distributes the cryptographic keys and certificates produced by the CA. These keys and certificates are used to ensure the authentication and non-repudiation of the information exchanged among the components of the architecture. The CSS, ADS and SM should execute the following actions:

- Obtain a copy of the certificate with the public key of the CA (CERT_KuCA).
- Generate a pair of asymmetric keys, public and private key, and store them securely ((KuXXX, KrXXX) where XXX is either CSS, ADS, or SM).

- Ask the CA for the creation of a certificate containing the public key (CERT_KuXXX where XXX is either CSS, ADS, or SM).

The certificate with the public key of the CA has to be securely distributed through the components of the architecture because all signature validations will depend on the correctness of this key.

The creation of a certificate will typically require human intervention due to some authentication/authorization steps. The security policy might impose, for instance, the need for an explicit authorization from the PNO before a request for the creation of a certificate can be made. This type of requirement is interesting because if applied to the SM, it would limit who can produce correctly signed software.

POS Set-up Protocol The manufacturer of the POS needs to execute some set-up operations before sending the device to a store. These operations include the installation of the basic run-time software, the bootstrap code and the application loader, and the secure storage of the following keys:

- A pair of asymmetric keys of the POS (KuPOS, KrPOS).
- A copy of the certificate with the public key of the POS (CERT_KuPOS). This certificate can be signed with the private key of the SM (instead of the CA).
- A copy of the certificate with the public key of the manufacturer (CERT_KuSM).
- A copy of the certificate with the public key of the CA (CERT_KuCA).

Each device has a distinct pair of keys which ensures that a POS compromise will not affect the rest of the network. To reduce the risk of key disclosure, the local copies of the POS private keys should be destroyed by the SM after their storage in the security module.

The SM is responsible for the generation of the certificate with the public of the POS. This option is interesting from an economic and efficiency point of view because it is simpler to produce a certificate locally (without the intervention of the CA) with a more relaxed the security policy. However, the validation of a POS signature will require the possession of both certificates of the SM and CA (CERT_KuCA verifies CERT_KuSM, and CERT_KuSM verifies CERT_KuPOS).

The inclusion of the certificate of the manufacturer is used to restrict who can create new versions of the software – a POS will only accept upgrades signed by the same manufacturer. At first, this might look like an unnecessary limitation of the protocol. However, it has important security implications because it prevents certain types of attacks. For example, a bad CSS is unable to substitute a new update with a malicious one, signed by a friendly (and also malicious) SM.

Application Transfer Protocol This protocol defines how new versions of the EFT applications and certification reports are exchanged between the SMs and the CSS. Whenever a new upgrade is produced, the SM sends a signed copy of the code to the CSS for approval. Then, the code goes through a testing phase to guarantee that it works as expected. Next, the CSS returns a signed report to the SM stating if the new version of the software was accepted or not by the certification tests (in the last case it also includes a list with a description of the failed tests).

From a security perspective, the certification procedure is particularly important because it can constrain the type of attacks that can be executed by an adversary SM. Basically, with an exhaustive set of tests, one can prevent bad software from being inserted in some POS of the network.

If the CSS is controlled by an adversary, even if momentarily, she (or he) will not be able to produce correctly signed applications, at most she could try to substitute the upgrade with a previous one (do not forget that a POS only accepts updates from the its own manufacturer). However, since we associate an increasing version number to each upgrade, it is possible to prevent this attack with a simple rule enforced at the POS – it only accepts updates with a larger version number.

Internal Management Protocol This protocol comprises all actions internal to the PNO, necessary to support the software updates. Basically two tasks have to be accomplished:

- Code transfer to the ADS: the CSS sends to each ADS a signed message with a copy of the code and a list of POS models that should be updated. After storing the code, the ADS is ready to receive requests from the POS.
- Notify the PSS about the upgrade: The CSS sends a signed message to the PSS containing the list of POS models that should updated, the version number of the code, and a signature of the code done by the SM (i.e, based on the KrSM). This information is then stored in the database of the PSS.

During normal operation, a POS only communicates with the PSS. Therefore, it must be the PSS who will inform the terminal that it must upload a new version of the EFT application.

EFT Protocol In the standard EFT protocol, only the POS can initiate the communication by sending a request to the PSS. In the response, the PSS can demand the execution of an operation by indicating that a given transaction should be carried out following the current one. Therefore, we had to introduce four new EFT transactions to inform the POS about new updates and for some management activities. The new EFT transactions are:

- Begin Update Transaction (BUT): indicates that a new upgrade is available and provides the following information: configuration data about the ADS that should be contacted (e.g., communication ports), version number of the code, and the signature of the code made by the SM.
- End Update Transaction (EUT): serves to maintain audit information at the PSS database about which software upgrades were made in the past. After completing the download and the application restart, the POS contacts the PSS to indicate that the update was a success (or that there was an error).
- Key Version Transaction (KVT): POS sends the versions of the keys (and certificates) that are stored in its security module.
- Update Keys Transaction (UKT): the PSS can update the keys saved in the POS. This transaction has to be performed with some caution because a bad PSS could use it to compromise the whole network. There are basically these types of updates:
  - new POS keys: to avoid tampering, the SM should first encrypt the keys with the previous public key of the POS, and then sign them. If the cause for the keys exchange was the compromise of the POS, then a different mechanism would have to be used. An attack as severe as this one would probably require the manual substitution of the POS since it could no longer be trusted.
  - renewal of CERT_KuPOS: typically, certificates are only valid during an interval of time. Therefore, periodically a new version of the certificate with the current POS public key must be produced and distributed by the SM (or CA).
  - renewal of CERT_KuSM: for the same reason, a new version of the SM certificate has to be periodically distributed. The new certificate could contain a different public key, however, it should have the same SM identification and should be correctly signed by the CA.
  - renewal of CERT_KuCA: due to the same reason as before. If the new version of the certificate has a different public key, then it should be accompanied with some proof demonstrating the knowledge of the previous CA's private key. This scheme allows the substitution of the CA's keys and at the same time prevents attacks where, for instance, the CA's public key is replaced by a malicious PSS.

These new transactions will be included on the standard EFT protocol. Therefore, they will share the regular security infrastructure where messages are protected with a MAC based scheme.

Table 1: Fields of the ATS transaction.

Message Fields	Request POS → ADS	Response ADS → POS
Message Type Identifier (MTI)	304	314
System Trace Audit Number	X	X
Error Indicator		X
Source	POS ident	ADS ident
Destination	ADS ident	POS ident
Signature with KrPOS	X	
Signature with KrADS		X
Next MTI		X
Data Length		X
Transport Data		data block
Current Data Block Number		X
Next Data Block Number	X	

Download Protocol This protocol downloads the software from an ADS to the POS. An example of an application transfer can be observed in Figure 2 (the message format follows the standard ISO 8583 (International Standard Organization, 2003)). To accomplish this task three different types of transactions are executed between the POS and the ADS:

- Begin Transmission Session (BTS): POS sends a message to the assigned ADS providing information about the wanted version of the code and its communication requirements, such as maximum acceptable block size (MBS) and list of supported compression algorithms. The response returned by the ADS includes the total size of the application, and the total number of MBS that will be sent (the application might have to be fragmented if its size is larger than the acceptable MBS).
- Application Transfer Session (ATS): in each exchange pair, the POS requests a specific block of the application and the ADS transmits that block. In case of failure, the POS can always restart the transfer process from the last correctly received block. As an example, we provide in Table 1 a more detailed description of the various fields that must be included in each of the transaction's messages.

- End Transmission Session (ETS): after the arrival of the whole application, the POS confirms the correctness of the upgrade using the signature created by the SM (that was provided by the PSS). Then, it uses this transaction to inform the ADS about the success (or failure) of the transfer. Next, it executes the new software.

There are several causes for the interruption of an application transfer session. For instance, there might be an abnormal delay in the communications that results in a timeout either at the POS or the ADS; or the session might be cancelled because a more recent upgrade is received. In any case, depending on the reason, the POS should be informed with different error codes if it can continue the transfer or it should start from the beginning.

3.3 POS Software Architecture

This section discusses various options for the software architecture of the POS. In general terms an application can be compiled into one of the following forms: *executable object code* or *interpreted code*. The first solution usually leads to better performance because it takes into consideration the particularities of the system for which it was built. But exactly because this dependency, it is not portable. Moreover, the source code must be compiled for each specific system prior to its deployment.

Using an interpreted code approach, the source code of the application is translated into byte codes of a virtual machine. A virtual machine is a theoretical microprocessor with standard characteristics that defines such things as addressing mode, registers, and address space. Once translated, the code can be interpreted by the virtual machine implemented specifically for a particular system (Morrison, 2001).

Therefore, in both solutions, at some level, there is always a system dependency, either the compiled machine code or the virtual machine implementation.

Another important aspect that has to be taken into consideration is the loading operation (Hall, 1990). Prior to a program execution the machine code must be loaded into memory. There are two types of machine code: *relocatable* and *non-relocatable* machine code (Intel, 1998). In the first case, every CPU instruction is associated with an offset address instead of an absolute address. When the machine code is loaded into memory a fixed base memory address is given by the operating system (OS). Consequently, it is possible to load the code in (almost) every memory location and to support the concurrent execution of several programs (loaded at different offsets). One disadvantage of this approach is that it typically needs an operating system to load and manage programs and memory. The non-relocatable code solution can be used without an operating system but the program will permanently be located at the same base address.

Taking the above in mind, a correct organization of the POS software is essential to minimize the associated difficulties related to the application management, load and execution, and to take over the full potential of software upgrades minimizing, where possible, the download time (Europay, MasterCard and Visa Corporations, 2000). Several scenarios are possible for the software organization depending on the physical characteristics of the terminal. Let's take a look at two extreme examples: a PC based architecture running a well known OS and an embedded system designed from scratch.

On a PC based architecture, programs can be compiled into executable object code or interpreted code. There are many software manufacturers and several languages that can be used to program the application. The resulting machine code (either from the program itself or from the virtual machine) is typically relocatable and the OS can be programmatically interfaced by APIs to make the memory management. In this scenario the POS software architecture can be very easily organized into the following modules/programs:

- Module that implements the Set-up Protocol;
- Module that implements the Application Transfer Protocol;
- Module that implements the EFT Protocol;
- Common Subroutines.

Upgrades can be done separately and updating each one of the above is almost a matter of receiving, creating, checking, substituting and deleting files.

On the other hand, embedded systems are designed for particular solutions requiring specific software implementations. In this kind of systems the number of software manufactures is very restricted, there are few available development languages and usually no virtual machine or OS is available. Therefore, the easiest solution for a software upgrade is to generate non-relocatable machine code programs and to substitute every line of machine code by the downloaded ones. The following tasks will have to be performed:

- Store the downloaded application into non-volatile memory without erasing the running one;
- Check the correctness of the downloaded application;
- Copy the downloaded application into the address space of the previous application or point to the entry point address of the downloaded application;
- Free unnecessary memory space for future software download.

This solution leads to longer transfer times because the application is not divided into independent modules and must be transferred as a whole. A better (but more expensive) solution is to build an OS for the embedded system, divide the POS application into the previous proposed organization and use relocatable machine code programs. As the modules can be independently upgraded this solution leads to shorter upgrade times.

4 PROTOCOL EVALUATION

The update protocol was implemented and evaluated on a network of PCs. The POS was simulated on a machine with a 1.4 GHz AMD Duron processor, and both the PSS and ADS were simulated on a PC with a 700 MHz Pentium Celeron processor. Each PC had 128 MBytes of RAM and was running Windows XP. The network was a 10 MBits/s Ethernet. In our experimental setting, several types of networks can be simulated, in particular a modem over a telephone line or a wireless GSM connection, by artificially delaying the rate of packet transmission to the operating system. In all measurements, the maximum acceptable block size of the POS was set to 2048 Bytes. The cryptographic algorithms used in the implementation were the SHA hash function and the RSA encryption algorithm (for the digital signatures) with 1024 Bit keys.

Our measurements focus on the transactions directly related to the application download. They include all transactions that need to be executed from the moment a POS finds out that there is an update until the conclusion of the whole process (transactions in dark boxes in Figure 2). The observed elapsed times for application downloads with different sizes, ranging from 100 KBytes to 2 MBytes, are displayed in Figure 3. This figure shows the performance of the protocol for three types of networks. Currently, the most common way to connect POS devices is through a telephone line at 9,6 Kbits/s, therefore, the other two curves provide some indication about protocol's future behavior as faster networks start to be utilized.

5 CONCLUSIONS

The paper describes a solution for the secure and automatic upgrade of EFT applications in POS devices. This solution requires the introduction of a few new components on the current architecture of the payment system, in particular a certification server where software manufactures can send and validate the upgrades, and a set of application storage servers that interact with the POS during the upload. A protocol was also provided to manage the various steps of a software upgrade, from the moment it is produced until it is installed in the POS. The solution was implemented and evaluated on a network of PCs.

6 ACKNOWLEDGEMENTS

This work was partially supported by the FCT, through the Large-Scale Informatic Systems Laboratory (LASIGE) and project POSI/CHS/39815/2001 (COPE).

REFERENCES

Bellare, M., Garay, J., Hauser, R., Herzberg, A., Krawczyk, H., Steiner, M., Tsudik, G., Herreveghen, E. V., and Waidner, M. (2000). Design, implementation, and deployment of the iKP secure electronic payment system. *IEEE Journal on Selected Areas in Communications*, 18(4).

Europay, MasterCard and Visa Corporations (2000). EMV2000 Integrated Circuit Card Specification for Payment Systems.

European Central Bank (2001). Payment and Securities Settlement Systems in the European Union (Blue Book).

Hall, D. V. (1990). *Microprocessors and Interfacing - Programming and Hardware*. McGraw-Hill.

Intel (1998). P6 Family of Processors. Hardware Developer's Manual.

International Standard Organization (2003). ISO 8583 - Financial Transaction Card Originated Messages: Interchange Message Specifications: Part 3 .

Manasse, M. (1995). The Millicent Protocols for Electronic Commerce. In *Proceedings of the 1st USENIX Workshop on Electronic Commerce*.

MasterCard and Visa Corporations (1997). Secure Electronic Transaction (SET) Specification – Book 1: Business Description Version 1.0.

Morrison, M. (2001). *Java 1.1 Unleashed*. SAMS Net.

Schoenmakers, B. (1998). Security Aspects of the Ecash Payment System. In *State of the Art in Applied Cryptography, Course on Computer Security and Industrial Cryptography*, volume 1528 of *Lecture Notes in Computer Science*. Springer-Verlag.

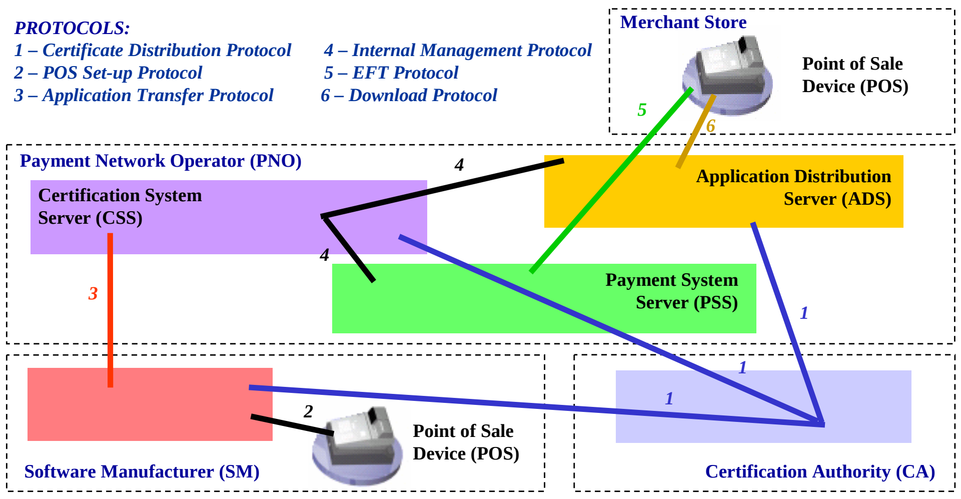
PROTOCOLS:
1 – Certificate Distribution Protocol
2 – POS Set-up Protocol
3 – Application Transfer Protocol
4 – Internal Management Protocol
5 – EFT Protocol
6 – Download Protocol
Merchant Store
Point of Sale Device (POS)
Payment Network Operator (PNO)
Certification System Server (CSS)
Application Distribution Server (ADS)
Payment System Server (PSS)
Software Manufacturer (SM)
Point of Sale Device (POS)
Certification Authority (CA)
1
2
3
4
5
6

Figure 1: Architecture of the payment system.

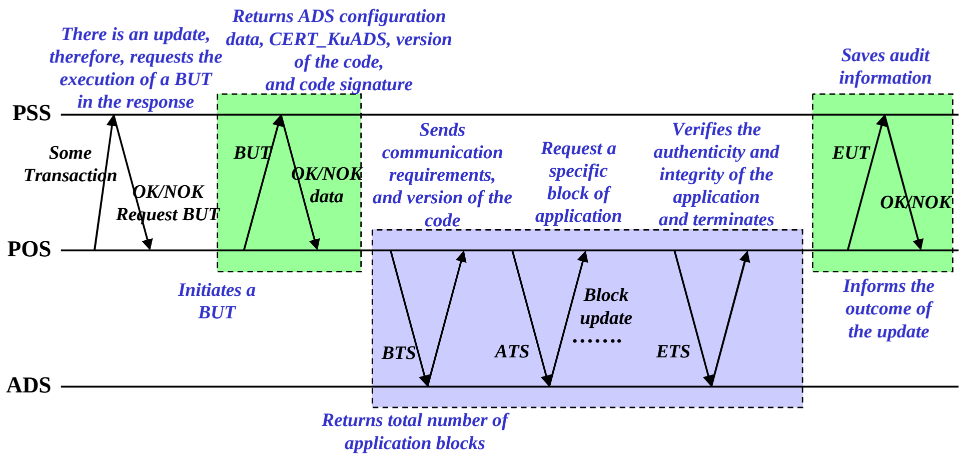
PSS
POS
ADS
There is an update, therefore, requests the execution of a BUT in the response
Returns ADS configuration data, CERT_KuADS, version of the code, and code signature
Saves audit information
Some Transaction
OK/NOK Request BUT
BUT
OK/NOK data
Sends communication requirements, and version of the code
Request a specific block of application
Verifies the authenticity and integrity of the application and terminates
EUT
OK/NOK
Initiates a BUT
Informs the outcome of the update
BTS
ATS
Block update …….
ETS
Returns total number of application blocks

Figure 2: The update of an application.

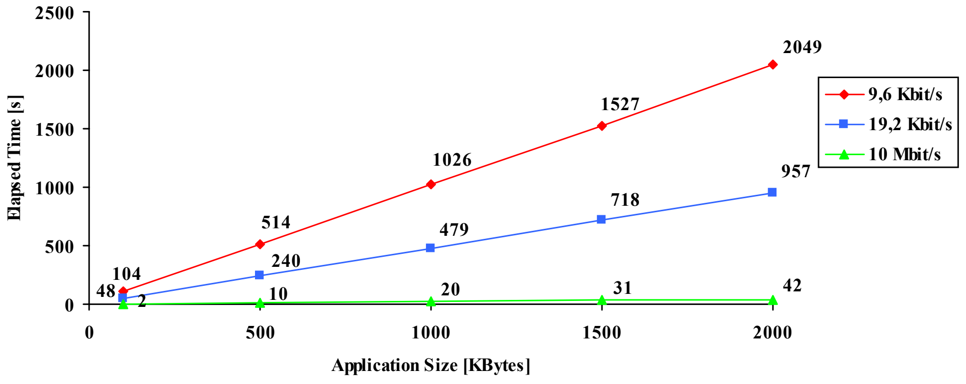

Figure 3: Elapsed time for the update of an application.