International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 04 Issue: 08 | Aug -2017 www.irjet.net p-ISSN: 2395-0072

Application to Quickly and Safely Store and Recover Credit Card's Information, using Tokenization and Following the PCI Standards

Daniela De Vivo [1], Eric Gamess [2]

[1] *School of Computer Science, Central University of Venezuela, Caracas, Venezuela*
[2] *Department of Mathematical, Computing, and Information Sciences, Jacksonville State University, AL, USA*

Abstract - *Nowadays, with the ubiquity of information and communication technologies, most of the companies are stepping toward offering their products and services online (using applications or websites), because it is the easiest and fastest way to do it, and they can reach clients all over the world. In general, the payment of these products and services is realized online by customers by entering the information of their bank card on a website. When payments have to be done fast or when it is very repetitive, introducing the payment information of the card can be unpractical and can slow down the whole process. To tackle this issue, we propose a vault application that allows the storage and recovery of the payment information by using tokens and that follows the Payment Card Industry (PCI) standards. The main objective of this application is to save, in a safe location, the payment information of customers, and allow them to recover that information, with a unique safe identifier, at the moment of the payment. This application will relieve customers to repetitively introduce their card information every time they make a purchase, and will speed up the payment process.*

Key Words: Web Service, Online Payment System, Security, ISO 8583, JSON, XML, PCI DSS, Vault Payment.

1. INTRODUCTION

Companies are constantly looking for raising their sales, by increasing their number of customers. Hence, most of the companies has developed web sites and applications to sale their services and products online. Online sales lead to online payments, where customers sometime have a short amount of time to introduce their cards information or they have to do it for every transaction. This process can be stressful and repetitive, and it forces customers to have their cards with them all the time. Therefore, a possible solution is to save the information of the card and to recover it when needed. However, it is not that simple, this kind of data is very sensitive and it must be stored in a safe place and only used by verified and authorized applications.

In this paper, we introduce in detail our solution for this problem, a vault application that follows the PCI standards [1][2] and saves the information in a safe server. The card's information will be only introduced once and will be associated with a customer. Later, the information will be recovered when the customer needs to realize a payment but, to keep a good security level, the sensitive information will not be send to the commerce's application, that is, it will only be sent to the payment channel. The vault application communicates with the commerce´s application through ISO 8583 [3], JSON [4], or XML [5] requests/responses, which make it compatible with the majority of actual commercial systems. For the communication between our system and the payment system, we use JSON. Our goal was to develop an accessible, adaptable, and multiplatform application.

The rest of the paper is structured as follows. We present related work in Section 2. We briefly discuss encryption in Section 3; a way to force confidentiality in data storage and transmission. In Section 4, tokenization and authentication are introduced; while the tools, languages, and standards that were used in the implementation of the application are described in Section 5. We explain the architecture and development process followed to build our application in Section 6. To certify our application, we discuss the validation tests that we run again our system and the results obtained in Section 7. Finally, Section 8 concludes the paper and gives directions for future work in this area.

2. RELATED WORK

Worldwide, important companies of payments have done similar work. However, some of these companies do not offer their services in Venezuela or have solutions that do not satisfy all the requirements. The Visa Token Service (VTS) [6] is a security technology that replaces the account information with a digital identifier, a token. This token allows merchants and wallet providers to keep the account information without risk, so they can provide their customers safe ways to shop online. Visa Tokens also offer the Visa Digital Enablement Program (VDEP) [7] that allows merchants to use payment solutions with VTS through payment methods such as: (1) Android Pay, (2) Samsung Pay, and (3) Visa Checkout. For token provisioning, a consumer enrolls his/her Visa account with a digital payment provider (mostly, mobile wallets), this provider will request a token from VTS and Visa might share this token with the issuing bank. For token usage, the digital payment provider passes the token as part of an authorization request, Visa receives the token and sends it along with the account or card information to the issuer, the issuer accepts or declines the request and sends its response back to Visa, that forwards the response to the provider.

As Visa, MasterCard has also its tokenization system. MasterCard offers two solutions: (1) Payment Tokenization and (2) Gateway Services Payment and Card Tokenization [8]. Both have the same goal, transforming sensitive information into tokens. In the first solution, MasterCard associates a unique transaction reference once a successful

transaction is done, and then this authorization reference is used in place of the card number, and the merchant only needs to capture the card security code (CVV) and the expiration date. Every reference has a pre-set expiration date of 13 months. On the other hand, in Card Tokenization, a merchant will receive a token in response to an authorization request. Tokens will be used in place of the cards number, and the merchant will only have to ask for the card security code and expiration date. Each token has a pre-set expiration date of 48 months.

The previous solutions have some issues. First of all, tokenization done by Visa or MasterCard forces every transaction to pass through one additional step going to their system and waiting for them to make the detokenization. Another problem could be that companies like Visa and MasterCard might not accept all payment services. These kind of solution, when token is associated to the card company, are more likely to be used in digital wallets.

3. ENCRYPTION

Encryption is the transformation of a message, regardless of its linguistic structure, to make it incomprehensible. In an encryption scheme, the information or message to protect, referred to as plaintext, is encrypted using an encryption algorithm, generating cipher text that can only be read or understood if decrypted. Encryption can be used for data that is stored in disks or data that must be sent through an insecure network. In the former case, the data is encrypted before being stored and the authorized parties must decrypt the data when accessing them. In the later case, the data must be encrypted by the sender of the message and decrypt by the receiver.

Message encryption enforces confidentiality. However, other techniques must be used together with encryption to achieve integrity, authentication, and non-repudiation of messages, for example digital signature or verification of a message authentication code (MAC). Integrity is a way to guarantee that the information has not been modified, that is, any alteration of the original information will be detected. Authentication is a mechanism of determining whether someone or something is, in fact, who or what it is claimed to be. Many times, authentication system is based on credentials (e.g., username and password) provided are compared to those of a database of authorized users. If the credentials match, the process is completed and the user is authenticated. Typically, non-repudiation is a process in which a party cannot deny something, for example in a communication system, the sending of a message that it originated.

The most important part of encryption is choosing the right way to do it. Encryption can be divided in the following 3 main types [9]: symmetric encryption, or secret key encryption, uses the same key to encrypt and decrypt, and the authorized parts of the communication must agree on the key that will be used; asymmetric encryption, or public key encryption, where each party has 2 keys, one public key that is distributed to any recipient and is used to encrypt, and one private key that is used for decryption and must be kept safe by the owner so nobody can access it; finally, hybrid encryption, where public key encryption is used to share a private key for a session, and symmetric encryption is used during that session.

There are many symmetric encryption algorithms that have been proposed by the community. In a communication system where sensitive data are stored and transmitted, it is crucial to make a good selection. Despite the large number of available algorithms, 3 algorithms are being wildly accepted worldwide: DES [10] (Data Encryption Standard), 3DES [11] (Triple DES) and AES [12] (Advanced Encryption Standard). AES is the most used for sensitive payment information. AES is a standard created in 2001 by Rijmen-Daemen from Belgium, and was proposed as an alternative to DES (nowadays susceptible to brute-force attacks since its effective key length is 56 bits) and 3DES (that generates speed problems). This algorithm can be used with keys of 128, 192, or 256 bits, and handles blocks of 128 bits of data. It is simple but resistant to diverse attacks. Its operation consists of: processing the data in blocks of 4 columns of 4 bytes (this is called the state matrix), has 10, 12 or 14 rounds and uses the operator “exclusive or” (XOR) to merge the columns between the key and the state array.

Finally, to ensure that security conditions are the best, systems must follow the Payment Card Industry (PCI) standards. PCI DSS (Payment Card Industry Data Security Standard) is a standard for data security in the payment card industry. It emerges as a need to standardize security standards in the processing, storage, and transmission of payment transactions involving cards, and was developed by a committee composed of the most important credit and debit card companies: PCI SCC (Payment Card Industry Security Standards Council). Companies that process, store, or transmit card data must meet the standard or risk losing their permits to process credit and debit cards, face rigorous audits, or pay fines. Merchants and providers of credit and debit card services must validate their compliance with the standard periodically. The current version specifies 12 requirements that are called “control objectives” and must be enforced by solutions that are PCI certified.

4. AUTHENTICATION AND TOKENIZATION

Authentication is the process of verifying the identity of the participants in a communication, and it is generally done when there is a request of connection [13]. It is simply a way to ensure that the users are who they say they are. This process works hand in hand with encryption since they complement each other: encryption helps authentication by ensuring that messages sent between sender and receiver were not interfered with or understood by third parties, while authentication helps encryption by ensuring that the source and destination of protected messages are who they claim to be.

Authentication can be done based on 3 different factors: (1) based on something known, the most used of all, makes authentication using credentials that the user knows, for example, a username and password; (2) based on something owned, using something that user has, for example a card; and (3) biometric based, user authenticates trough a physical characteristic or unconscious gesture.

Tokenization is the process where an account or card number is replaced by a unique identifier called token [14]. Each token is associated with a card or account number, so there is also a detokenization process that consists of retrieving the original number of the card or account, by supplying the token. The security of this system is based on the complexity of obtaining the actual number of the card or account if the only information known is the token. The origin and motivation of tokenization is based on the fact that the information of the cards/accounts and their respective holders cannot be exposed, leading to very complex systems where it is required to take care of many aspects of security. Storing a token is a safe process, since the token alone, without the key, does not reveal any sensitive information, making it much easier to store information and comply with the PCI standards.

The storage of multiple tokenized credit cards is called card vault. A digital credit card vault is a system where tokenized information is stored for each card. The vaults must comply with various security mechanisms, including PCI, when handling sensitive information. The most important security mechanism is that the complete information of a card can only be obtained through a token from a valid and reliable applicant.

5. TOOLS, LANGUAGES, AND STANDARDS FOR THE APPLICATION

In this section, we introduce the tools, languages, and standards for the message format that are usually used in this kind of transactions and those that were used for our application.

Online payment transactions use 3 different formats for messages: ISO 8583, XML, and JSON. ISO 8583 is a standard from the International Organization for Standardization (ISO) for financial transactions that involve debit or credit cards. This standard defines a message format and a communication flow so that remote systems can exchange these transaction requests and replies. An XML (Extensible Markup Language) [18] message is a hierarchy of tags that represent the structured information on the web (all types of documents), so that this information can be stored, transmitted, processed, visualized, and printed, by diverse types of applications and devices. JSON (JavaScript Object Notation) is a light data exchange format that is easy to read, write, generate, and interpret, by both humans and computers. JSON is composed of two main structures: (1) a parameter-value combination, representing an object or record; and (2) an ordered list of values, which is known in most languages as an array.

For a complex system, like a vault application, many tools and languages have to be used. In the following subsections, we present the most popular ones.

5.1. Java

Java is a general-purpose, concurrent, class-based, and object-oriented language [15]. It is a strongly typed static language, that is, each variable must be assigned a data type and that data type can only be used, like another type, by a conversion.

5.2 Java EE (Enterprise Edition)

Java EE is a platform that streamlines and facilitates the quick development of Java-based enterprise and robust applications. It aims to provide developers with a set of APIs in order to reduce the time and complexity of development and improve application performance [16].

5.3 MySQL

MySQL is an open-source relational database management system (RDBMS) [17]. This system is known in the community for having high performance and availability, which makes it a good option to be integrated in applications where many transactions are made on the database and short response times are needed.

5.4 The javax.crypto Packet

Java provides a very useful packet called javax.crypto that has a set of classes that facilitate the development of any cryptographic scheme.

5.5 Java Database Connectivity (JDBC)

JDBC is an API for the Java programming language, which defines access to relational databases from Java applications. It provides methods to query, update, and remove data in a relational database. In can be used to access databases managed by the major RDBMSs, such as MySQL, PostgreSQL, and Oracle.

5.6 RESTful Services

REST (Representational State Transfer) refers to a set of principles that forms a style of web architecture. In this architectural style, data and functionality are considered resources and accessed through URI (Uniform Resource Identifiers) [18].

5.7 HTTP

HTTP (Hypertext Transfer Protocol) is a stateless protocol for the transfer of data and information in the Internet network and is known mainly for performing communication between web servers and web browsers [19]. In the HTTP protocol, the POST method is mostly used to send data filled into a form in a web browser, from an application to a server. In this method, the parameters of the request are not stored anywhere, and are sent in the body of the request.

6. ARCHITECTURE AND DEVELOPMENT OF OUR SOLUTION

Our solution for the mentioned problem was to develop a vault application that could be used by different merchants and customers. To reach that objective we define the following guidelines: the application data is divided into different vaults, a vault is associated with one merchant; tokens are associated to customers and to merchants; only registered merchants have access to the application; a merchant can only see the data of his own vault. The application must be modular, since this helps in having a better control of security levels and it will be much easier to fix bugs. Therefore, we decided to make a system composed by different web services with their own functions. All these different services look like one for external applications and payment systems. Services will be connected to a centralized database. The architecture of the application is shown in Fig. 1.

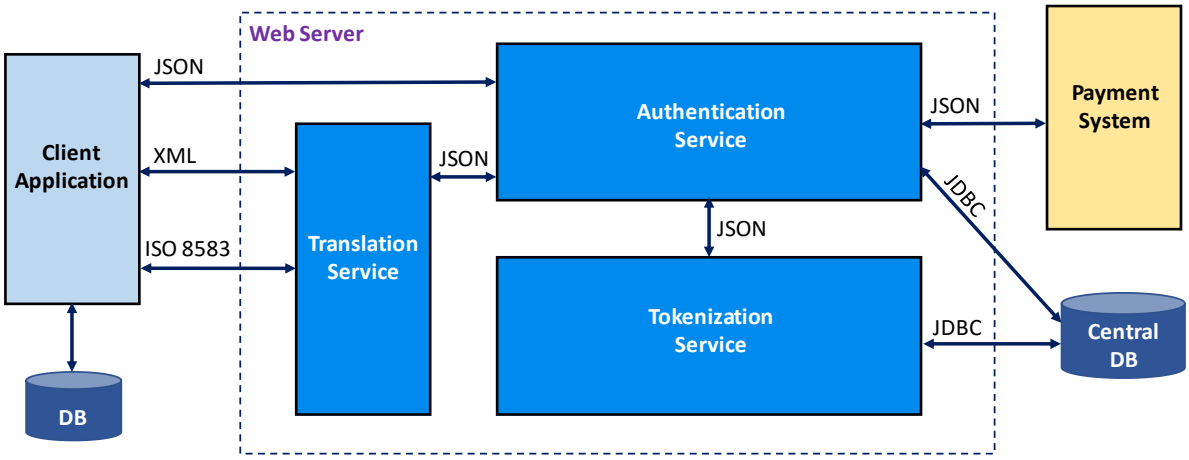

Fig -1: Architecture of the Proposed Solution

The natural flow of our application would be the following: the administrator of a commerce (by using the client application) registers it through the authentication service, and it is returned to him that the registration was successful, the ID of the vault, and token of the merchant. The token of the merchant is stored in the local database (the commerce database). A customer of the commerce creates a payment option with the associated vault by using the client application and by entering its credit card information (name of customer, credit card number, expiration date, and CVV number). The client application sends the required information to the authentication service that generates a token through the tokenization service. As a response, the web service returns the token to the client application, where it is stored locally in the commerce database. From now on, the client application has the token (which gives some information regarding the card of the customer) as a payment option for this customer. To make a payment, the customer uses the client application and selects the card, that invokes the authentication service, which is in charge of performing validations, detokenization and the sending of the information of the card to the payment system; The payment system takes a decision about approving or not the payment and returns the result of the transaction to the authentication service, that finally forwards it to the client application.

The central database is a very important part of this system. In our architecture, it is allocated in a remote safe server and it communicates with the web services by using JDBC [20]. The databases stored in this server are encrypted, so if a hacker makes a copy of them, he will not be able to decrypt the information. The database has a set of tables that allow the secure storage of information of customer cards (random key, token, and additional information), vaults (ID), merchants (random key and name), and associations between customer cards, merchants and vaults.

The application was developed in Java, and we created RESTful web services that communicate between each other through JSON messages. As mentioned previously, the core of the architecture is based on three main web services, that we describe next:

Tokenization Service: this service is intended to generate a token associated with a customer card or a merchant. Its structure must be quite simple, and to maintain a high level of security, it only has access to the database to insert the record of a new generated tokens. This service has a main class that handles requests, a class that does the encryption, a service configuration class, and a class that handles connection to the database using the JDBC API. For the development of this service, we used packets such as: javax.crypto.*, javax.ws.rs.*, org.json.*, among others. In the vault system, this service will be invoked by the authentication service. However, a separate route to it was created as an alternative, to allow future expansion of the application, so it can be reused in another project or for another purpose.

In the case of the tokenization of a customer card, the service is invoked by the authentication service by sending a JSON message that contains the customer card's data (name of customer, credit card number, expiration date, and CVV

number) to the tokenization service. Since a random key is required for the generation of the token, we use the Java javax.crypto.KeyGenerator, specifying the type of encryption algorithm to be used (which is AES in our application), the encoding to be used (which is UTF-8 in our application), and the size of the key (we chose 128 bits in our application), to generate the key. This key will only be generated the first time a customer card is inserted and associated with a merchant, and it will be stored in the central database by the tokenization service.

Subsequently, the tokenization service will take the card number and divide it into 3 parts which are: the first 4 digits, the 8 intermediaries, and the last 4. An important point is that since it is a credit card vault only, it is assumed that the card number will always have 16 digits. The 8 intermediate digits will be encrypted using the methods of the encryption class and later concatenated again with the initial 4 digits and 4 final digits, thus generating the token. It is worth to clarify that the first 4 digits of a card are well known and permit to identify the bank issuer. Hence, it is not necessary to encrypt them. Also, for customers that register several cards with the same merchant, the last four digits of their cards will be the method that they will use to identify each card. Hence, the 4 last digits are not encrypted for this reason. The token is also stored in the central database by the tokenization service. Note that the 8 intermediate digits that were encrypted, become 32 alphanumeric digits after encryption.

After the generation of the token for the customer card, the tokenization service will use the previous generated random key to encrypt the rest of the data of the customer card (name of customer, expiration date, and CVV number). The resulting information will also be stored in the central database by the tokenization service. Finally, the tokenization service ends its work by returning the generated token in JSON format to the authentication service.

In the case of the tokenization of a merchant, a random key is also generated and used to encrypt the commerce name with the same encryption mechanism of the customer cards. Both, the random key and the merchant name are stored in the central database by the tokenization service. The created token is sent back to the authentication service that will forward it to the client application, where it will be stored in the local database for future use.

Authentication Service: this web service contains all the logic and administration of the vault system itself. It is responsible for interacting directly with the client applications and receiving their requests. It is subdivided into two large parts.

The first part of this service consists of the administration of the information of merchants that are associated to the vault system, so they can be created, updated, eliminated, or associated to the vaults. In the approach of the solution, it was assumed that each merchant will have its particular vault, however, it was left open the option that a vault can be associated with more than one merchants.

The second part of this service is focused on the administration of customer cards. In this part, there are two options, the first is to invoke the tokenization service for the tokenization of the information of a customer card and the second is to retrieve the data of a customer card (plain text), identified by a received token, and send these data to the payment system.

Translation Service: this third and last web service has as main objective to handle the interaction between the vault system and client applications that do not support the JSON format messages. As mentioned in Section 5, the most common formats for this type of transaction are ISO 8583, JSON, and XML. Hence, this service will receive requests in ISO 8583 or XML format, and will translate them into JSON format, that can be understood by the authentication service as shown in Fig. 1. The objective of the translation service is to make our vault system compatible with the bigger number of possible client applications.

Additionally, since security in this system is a key point, it was covered by three main strategies and a lot of validations. First, the environment where the system was located has its own security mechanisms and access control, so sensitive information is in a more secure environment. Second, the design of this system is made to be used with SSL certificates, which means that instead of using the HTTP protocol for the exchange of information, HTTPS [28] is used. Third, when a merchant associates its client application with the vault system, a merchant token is generated and stored in the local database of the client application. For the authentication of the merchant by the vault application, this token has to be sent by the client application in the headers of all HTTP requests. When a service receives a request from a client application, it will decrypt the received merchant token with its associated random key (stored in the central database), and if the result matches the name of the merchant (also stored in the central database), access will be allowed, otherwise it will be denied. It was decided to use a fixed merchant token and not one per session since the sessions are very short in this system. In almost all cases, sessions will be transactions of short times and little data to exchange.

7. VALIDATION OF THE PROPOSED SYSTEM

The validation of the proposed application must be done at several levels. For example, the PCI requirements [29] were achieved since: (1) the system is on a secure, protected and isolated server, (2) the proposed application architecture and flow, and (3) the security measures implemented.

Another aspect that guarantees a higher level of security in this system, is the fact that each time a customer card is associated with a vault, the token will be different. That is, it is guaranteed that the same customer card in different vaults will generate different card tokens.

We performed simple merchant authentication tests that consisted in requesting services through the authentication service with a modified (partially or completely modified) or

missing merchant token, and in all cases, access was denied. Similarly, we did simple customer card authentication tests, and all the illegal accesses were denied.

At the code level, some additional restrictions were developed, almost all as an extra step for verification and access to data. For example, we always verify that the merchant has a vault before allowing it to generate or retrieve a customer card token. Also, when a customer card token is retrieved, the system validates that the merchant that requests it is associated with the vault where it is stored.

Finally, we did some tests about the response times of the application. For the registration and management functions of cards and merchants, the response times are practically immediate. Longer waiting times can only be generated if there is a problem in the device making the request to the vault system or a failure in the network through which the request is made. On the other hand, for the recovery of a token and the realization of a payment, the waiting times can be much more variable since they not only depend on the devices, the network and the system, but also on the response time from the bank. Bank transactions usually have a response time lower than 1 second; however, there may be failures or delays in transmission, both at the time of the request or the reply. To measure an average of the system response time for a payment, tests were performed using simulated payment systems offered by two different Venezuelan national banks: Banesco and Banplus. It is worth to mention that these simulated payment systems are provided by the banks to certified system payments for developing and testing purposes before they put a new developed application in service. For the tests, a universe of 500 transaction was taken that consisted in four types of scenarios: (1) Banesco payment system using Banesco cards, (2) Banesco payment system using Banplus cards, (3) Banplus payment system using Banplus cards, and (4) Banplus payment system using Banesco cards. Fig. 2 depicts the results obtained. As expected, we can observe that transactions made with a bank using their own credit cards are faster, since the bank can take a decision about approving or not the payment without contacting another bank.

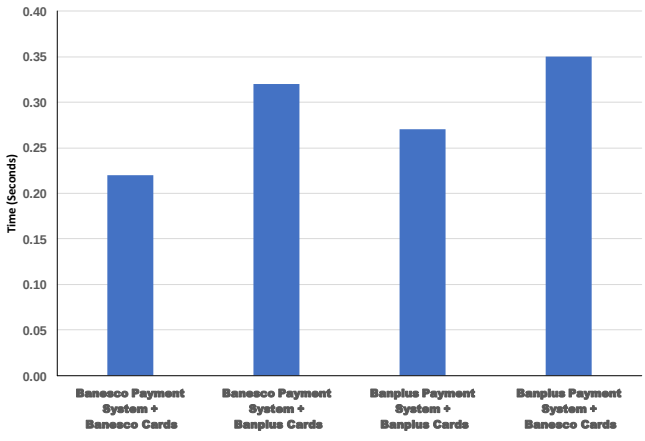

Fig -2: Response Times in Different Scenarios

Although the system will be running in a server that has access control that will be in charge of limiting the number of requests that it receives per second, we decided to make some stress tests. For that, we use the same simulated payment system offered by Banesco and Banplus, and made several payment requests per second, to observe how the response times is affected when the vault system receives multiple requests in a short period of time. Seven types of tests were performed to study the variations of the response times when the system is loaded. From the first to the seventh type of test, the number of requests per second were: 1, 25, 100, 200, 300, 400, and 500, respectively. Unlike the previous experiments, we did not enforce the selection of a particular bank for these experiments. That is, we used customer cards from both banks (Banesco and Banplus), and the system payment will choose the bank autonomously. Fig. 3 depicts the results obtained. As the number of concurrent requests increases, so does the response time of each request. However, in all cases, the requests for the transactions were answered satisfactorily, with an acceptable response time.

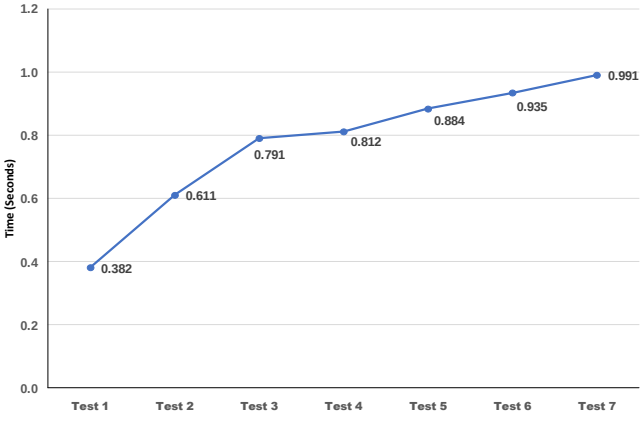

Fig -3: Response Time when Making Concurrent Requests

8. CONCLUSIONS AND FUTURE WORK

The need of merchants to adapt to technological advances is increasing, so they can accommodate to new tendencies such as offering their products and services via the web and mobile applications. To support these new tendencies of the commerce, a large number of electronic payment systems has emerged. However, at the moment of payments, the information of the credit card is entered manually and repetitively when the customer is making several purchases, which can be stressful and unpractical. For this reason, in this research work, we proposed a new application to achieve a safe and efficient payment system. For each merchant, our system creates a vault where the information of the customer cards is kept. For each card, a card token is generated and used as a replacement of the card number. The complete information of a customer card (name of customer, credit card number, expiration date, and CVV number) can be rapidly and safely recovered through this

token. Also, our solution is compatible with most of the existing platforms since it accepts petitions in ISO 8583, JSON, and XML format.

As future work, we propose to add the management of debit cards and bank accounts to our vault system. Also, we are interested in linking more payment options to our application. Finally, we also want to investigate the feasibility of making an application for digital wallet using card tokenization in Venezuela.

ACKNOWLEDGEMENT

We want to thank professor Antonio Russoniello, from the School of Computer Science of the Central University of Venezuela, for all his suggestions and feedbacks during the development of this research project.

REFERENCES

[1] PCI DSS – Documentation. Requirements and Security Assessment Procedures. April 2016.

[2] PCI Security Standards Council. PCI Security https://www.pcisecuritystandards.org/pci_security

[3] Internation Organization for Standardization. ISO 8583-1:2003. Financial Transaction Card Originated Messages – Interchange Message Specifications – Part 1: Messages, Data Elements and Code Values. June 2003.

[4] L. Bassett. Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON. O'Reilly Media. 1st edition. August 2015.

[5] PG Wizard Books, XML Crash Course: Step by Step Guide to Mastering XML Programming. CreateSpace Independent Publishing Platform. March 2017.

[6] Visa Token Service https://developer.visa.com/products/vts.

[7] VDEP http://www.visa.com.mx/asociandose-con-nosotros/tecnologia-de-pago/visa-token-service.html.

[8] Tokenization http://www.mastercard.com/gateway/implementation_guides/Tokenization.html.

[9] W. Stalling, Cryptography and Network Security. Pearson. 7th edition. March 2016.

[10] FIPS PUB 46-3. Data Encryption Standard. May 2005.

[11] J. Pattrick. Information Security and Criptology. June 2017.

[12] FIPS PUB 197. Advanced Encryption Standard. November 2001.

[13] R. Smith. Authentication: From Passwords to Public Keys. Addison-Wesley Professional. 1st edition. October 2001.

[14] Scoping SIG, Tokenization Taskforce PCI Security Standards Council. PCI DSS Tokenization Guidelines. Version 2.0. August 2011.

[15] H. Schildt. Java: The Complete Reference. McGraw-Hill Education. 9th edition. April 2014.

[16] D. Coward. Java EE 7: The Big Picture. McGraw-Hill Education. 1st edition. October 2014.

[17] M. Maclaughlin. MySQL Workbench: Data Modeling & Development. McGraw-Hill Education. 1st edition. April 2013.

[18] L. Richardson and S. Ruby. RESTful Web Services. O'Reilly. 1st edition. May 2007.

[19] D. Gourley, B. Totty, M. Sayer, A. Aggarwal, and S. Reddy. HTTP: The Definitive Guide. 1st edition. Octobre 2004.

[20] M. Brzustowicz. Data Science with Java: Practical Methods for Scientists and Engineers. 1st edition. June 2017.