DYNAMIC MULTI-FACTOR SECURITY

An Undergraduate Research Scholars Thesis

by

JOHN LESLIE, SHAYOK DUTTA and AMNAY AMIMEUR

Submitted to Honors and Undergraduate Research
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Research Advisor: Dr. Sunil P. Khatri

May 2015

Major: Computer Engineering (Electrical Engineering Track)

TABLE OF CONTENTS

ABSTRACT

Dynamic Multi-Factor Security (May 2015)

John Leslie, Shayok Dutta, Amnay Amimeur
Department of Electrical and Computer Engineering
Texas A&M University

Research Advisor: Dr. Sunil P. Khatri
Department of Electrical and Computer Engineering

This thesis identifies the current limitations of electronic remote authentication systems and presents a new remote authentication system that addresses these limitations. Examples of these limitations can be easily observed in everyday life. Some more common examples include: credit card theft, identity theft, insurance fraud and hacking of private computer networks. Our proposed solution includes a multi-factor protocol which has two key features. First, it dynamically updates private ID numbers such that no two iterations of the authentication protocol use the same set if private IDs for each involved party, using a True Random Number Generator (TRNG). This prevents any unauthorized access of private information, and even if this information is compromised, the authentication protocol is not compromised, since the subsequent iteration of authentication uses new IDs. Second, the protocol uses multiple authentication factors (two in our implementation), to further enhance security. These additional authentication factors are also dynamically updated after each iteration of the protocol. The protocol was implemented in a system which simulates a credit card transaction, highlighting the usefulness of our protocol in real world remote authentication. We expect this new electronic remote authentication system to solve many of the current failings of modern electronic authentication schemes.

DEDICATION

I, John Leslie, dedicate this to my grandfather Paul A. Wade. Even though lost to us, a source of inspiration that I feel privileged to sit in the shadow of. And to my father, in the face of cancer he has inspired me to press on, life waits for no one.

I, Amnay Amimeur, dedicate this work to Mohand and Haffida Amimeur. On their shoulders I stand, and for that, I am forever grateful.

I, Shayok Dutta, dedicate this to my mother, Keya Dutta, my father Nripendu Dutta, and my brother Soumak Dutta. Through there guidance, adversity has transcended to understanding.

ACKNOWLEDGEMENTS

We would like to thank our advisor, Dr. Sunil P. Khatri for guidance, support, and technical advice.

We would like to thank Dr. Laszlo B. Kish for his valuable advice.

NOMENCLATURE

DMFS	Dynamic Multi-Factor Security, the principal name for this project and research
Card	The user's Credit Card
POS	The Point Of Sale (such as a store)
XOR / $\oplus$	A logical operation involving two operands and yielding a result
N1	The secret Card number generated and held on the Card
N2	The new secret Card number generated to replace N1 and held on the Card
M1	The secret POS number generated and held on the POS
M2	The new secret POS number generated to replace M1 and held on the POS
SF_C1	The secret Second Factor generated on the Card
SF_C2	The secret updated Second Factor generated to replace SF_C1
SF_P1	The secret Second Factor generated on the POS unit
SF_P2	The secret updated Second Factor generated to replace SF_P1
PubID_C	The public identification number of the Card
PubID_P	The public identification number of the POS
Bank / Visa	The bank or Credit Card granting institution
$	The dollar amount required to purchase an item during a credit card transaction
Y/N	A plain text approval or disapproval message
TRNG	True Random Number Generator
PRNG	Pseudo Random Number Generator
LNA	Low Noise Amplifier
OpAmp	Operational Amplifier

CHAPTER I

INTRODUCTION

In today's economy, the need to access resources through the internet is growing increasingly common. Whether these resources are information from a place of business, medical records, bank access, or perhaps an online purchase, the need to properly authenticate users has become a very important issue. There are many recent examples of unauthorized access to private and public businesses. For example in 2013, Target Corporation was compromised ("hacked") by an unauthorized agent, resulting in the unauthorized release of thousands of credit card account numbers. This is one example out of hundreds, all resulting in the unauthorized access of thousands of individuals' confidential information. These events are harmful to both the victimized customers and to the institutions, which are held responsible for the legal and financial repercussions of the security breach. Another example where additional security measures would be critical is the case of a medical doctor working from home. The doctor needs access to private medical records of patients. If that doctor's link to this information is compromised, private and highly sensitive medical information can become public.

In the past, mechanisms have been researched and developed that require more than one authentication factor to obtain access to a resource. The concept of *multifactor security* is not a new idea. A simple example is a home with an alarm system. In this example, a malicious individual needs a key to enter your house (the first factor) and a code to disable the alarm system (the second factor). This concept is equally applicable for online and computer network based access control as it is for physical security. Recent studies have focused on dynamic ID

based remote authentication for specific applications such as wireless payment over a web-based architecture, and mobile client-server environments. Other work conducted has focused on biometrics and smart cards as an implementation of multiple factors[1], [2], [6], [9], [10], [11], [12].

One such research study was conducted by Tiwari et al 2011[10]. This study developed a multi-factor approach to wireless web transactions conducted over mobile devices. The authors implemented a multi-factor based security protocol in Java[10]. This decision was made due to the portability of Java, which is architecture agnostic. The authors also made recommendations on how to achieve two-way authentication, in order to authenticate both parties involved. Their implementation was based in an application layer solution. As a result, they made use of HTTP, since it is a stateless protocol. To overcome the barriers associated with session tracking, they used cookies to assist the back end portion of their implementation[10]. The results of their research were encouraging due to the ease of implementation and the scalability of their recommended solution. The cost of implementation is dependent on the protocols and policies in use at the adopting financial institution. In contrast to the work described in this thesis, Tiwari et al 2011[10] does not regenerate the first factor after each transaction. Also, their second factor is not dynamic.

Another study conducted by Chen et al 2011[1] investigated the use of smart cards and mobile devices as authentication tools, introducing a fingerprint biometric as the second-factor. An added feature is that their proposed scheme uses only hash functions as the method of authentication. The authors endeavored to improve upon one-way authentication schemes by devising a three phase authentication process. The first phase is called registration phase. This

phase is where a user selects their credentials such as, user ID, and password. Through the use of the XOR function and hashing, the server and the user compute various message digests to be used in the next phase. The second phase is called the login phase. This is where the user contacts the server and submits their credentials. The third phase is the authentication phase. The user and server compute each other's credentials and decide whether or not the other is whom they claim to be. Their results mainly focused on the cryptographic strength and performance metrics of their implementation when compared to previous implementations. Overall, their resilience and cryptographic features are superior to the methods that they compared their work with. In addition, the performance in terms of required computation and required transmission were significantly lower than those protocols that they compared their scheme with. Unlike the work described in this thesis, the work of Chen et al 2011[1] does not use dynamic first and second authentication factors.

In a study performed by Sun et al 2012[9], the authors identify and address the inherent weakness of the (common) situation in which a server knows all of the user's private keys. The concept of user anonymity is addressed at the implementation level of their proposed scheme, as well as how to achieve perfect forward secrecy[1]. Their system relies on the concept of computational infeasibility through the Elliptic Curve Discrete Logarithm Problem (ECDL) and the Computational Diffie-Hellman Problem (CDH)[9]. This is a common method by which to achieve strongly secure systems with lasting cryptographic integrity. They went on to prove their security method mathematically. The results of their study showed that a scheme to address both user

[1] Perfect Forward Secrecy is the property of a key agreement based protocol in which the session key being compromised does not imply the private keys are also compromised. This means future session keys are not implicitly exposed, even when keys in the past have been exposed.

anonymity and perfect forward secrecy is possible, and directly addressed by their proposed protocol[9]. The work presented in this thesis is qualitatively different, since we explore dynamic two-factor security, instead of focusing on a case where a server stores every user's private keys.

The objective of our research is

1) To develop both the theory and prototype (proof of concept) for a new authentication protocol.
2) This protocol provides security through dynamic IDs (e.g. credit card number), and a secure method by which to update both the authenticator (e.g. credit card corporation) and the user equipment with the new ID.
3) The change of IDs is realized through a True Random Number Generator (TRNG), instead of a Pseudo-Random Number Generator (PRNG). TRNGs derive their randomness from the randomness of physical phenomena, and are hence considered significantly more secure than PRNGs.
4) The second factor of our multi-factor approach is also dynamic, enhancing the security of the authentication process.

Our system implements multi-factor security using a pre-shared secret. This pre-shared secret is used to generate a second factor key simultaneously on the Card (or POS) and at Visa.

Using the improvements made by Tiwari et al 2011[10], we implemented our protocol using Java. We also incorporated their suggestions on two-way authentication, and included that in our implementation. From the work conducted by Chen et al 2011[1], we decided to make use of XOR and hashing functions in order to reduce the system loads. This decision was made on the

basis that a credit card scheme would need to allow for high traffic and low resource consumption. Their idea for hashing as a method of authentication is implemented in our protocol as well. The concerns addressed by Sun et al 2012[10] are only partially addressed in our scheme. We do not address the user anonymity concern as we assume that Visa would need to know who is charging what amount, and whom to bill for that amount. We do however implement a system that we believe possesses the property of perfect forward secrecy.

The addition of multiple factors in our approach provides the ability to verify the identity of all transaction members to each other. This allows each transaction to begin in a "No Trust" state. This means that for any member of a given transaction (the members are Card, POS and Visa), there initially exists no assumption of trust for all other members involved. As the authentication protocol proceeds, trust is securely established between the members. The details of this are described in Chapter II.

Our scheme presents a means of highly secure, dynamic, multi-factor remote authentication through the use of the Credit Card example. It is very important to observe, however, that our scheme is highly flexible, and can be applied to general remote authentication scenarios as well.

CHAPTER II

METHODOLOGY

2.1 Overview of DMFS System

This chapter will describe the architecture and implementation of our secure remote authentication protocol, and its organization is as follows. In the remainder of this section, we will provide a high level description of the protocol, breaking it down into a series of *processes*. Each process will be further explored in subsequent sections that detail the steps and information transmitted during that process. Finally, an in-depth description of the implementation of the protocol as an embedded system will be discussed in Section 2.3.

The protocol is comprised of two processes. The first process of the protocol is termed the *three-party authentication process*. It is used to verify that all parties in the transaction are exactly who they say they are, preventing the protocol from being executed with any malicious intent. The second process of the protocol is the *card approval process*. Its purpose is to approve a charge to the cardholder's account and update the card and bank with the cardholder's new credit card number. The three-party authentication process must happen before the card approval process for the transaction to be considered secure.

The three parties taking part in the credit card transaction are the bank or credit card institution (denoted VISA), the point of sale (denoted POS), and the credit card (denoted Card). Each party has a strictly defined collection of information with which to work. The card stores at least two

items at any given time, a public ID (denoted PubID_C) and a private credit card number (denoted N1). During a transaction, the card produces two additional items: a second private ID (denoted N2) which will serve as the new credit card number for the next transaction, and a second factor (denoted SF_C), which is used to encrypt and decrypt private IDs. In a similar way, the point of sale will also have its own public ID (denoted PubID_P) and private ID (denoted M1) and will also produce a second factor (denoted SF_P) and another new private ID (denoted M2) to replace the private ID used during the transaction. It should be stated that the second factor is seeded by the most recent private ID. The specifics of second factor generation are discussed in section 2.3.1 The bank, the third party in the transaction, will hold the public IDs and private IDs of both the card and the point of sale and will also be capable of producing the same second factors used by the card and point of sale.

2.2 Protocol Description

2.2.1 Three Party Authentication Process

The three party authentication process is a collection of six steps, four of which involve transmitting information across communication channels between the three parties. These steps are described next.

$$0.\ Card \rightarrow Visa: PubID\_C$$

$$0a.\ POS \rightarrow Visa: PubID\_P$$

$$0b.\ POS \rightarrow Visa: H(PubID\_P \oplus M1 \oplus SF\_P1)$$

The Card and POS initiate communication with Visa by sending PubID_C and PubID_P. POS then creates a packet containing $PubID\_P \oplus M1 \oplus SF\_P1$. The POS sends the hash of this packet to Visa. The hash serves as a request to update SF_P.

$$1.\ POS \rightarrow\ Card: PubID\_P$$

$$1a.\ Card \rightarrow\ POS: PubID\_C$$

$$1b.\ POS\ \rightarrow\ Card: \$$$

The Card and POS exchange public IDs, and POS sends the Card the amount of money it expects to charge the card holder.

$$2.\ POS\ \rightarrow\ Visa: (M1 \oplus SF\_P2 \oplus M2, PubID\_P, PubID\_C, \$)$$

$$2a.\ POS\ updates\ M1\ to\ M2$$

The POS sends Visa four values concatenated together. The first value is $M1 \oplus SF\_P2 \oplus M2$. The second and third values are PubID_P and PubID_C (which was received from the Card in step 1a). The final value is the dollar amount the POS expects to charge the cardholder. Once the POS sends Visa M2, the POS is able to update its current private ID to M2.

$$3.\ Visa\ extracts\ M2\ from\ step\ three$$

$$3a.\ Visa\ stores\ PubID\_C\ and\ \$\ from\ step\ three$$

In this step, Visa extracts M2 received from POS in step three. Visa can do this because it is currently storing M1 and can produce the same SF_P used to encrypt the message sent in step three. M2 can be computed by:

$$M2 = (M1 \oplus SF\_P2 \oplus M2) \oplus (M1 \oplus SF\_P2)$$

Once M2 is extracted, Visa stores it as the new private ID for the POS.

$$4.\ Visa \rightarrow Card: H(PubID\_P \oplus N1 \oplus SF\_C1)$$

Visa sends card a hash of a packet containing $PubID\_P \oplus N1 \oplus SF\_C1$. Visa can do this because PubID_P was received in step 2 and stored in step 3a, and Visa is storing N1 and can generate SF_C1.

$$5.\ Card\ compares\ the\ hash\ from\ Visa\ to\ its\ own\ hash\ of\ what\ it\ believes\ to\ be:$$

$$PubID\_P \oplus N1 \oplus SF\_C1$$

The last step of the three party authentication process is the Card's comparison of the hash it received in step five with its own hash of the PubID_P received in step two. This is because the card has access to all three elements used in the hash, so it can independently compute the hash of these three items and compare them with the hash value it received in step 4. Should the hashes match, it means that the PubID_P verified by Visa matches the PubID_P POS sent to the Card, and the Card can trust that the POS is approved by Visa. It simultaneously proves that Visa can be trusted because it knows the Card's private ID. This process is outlined in Figure 1 below. In this figure, the numerical step described above is indicated, along with the direction of communication for each numerical step.

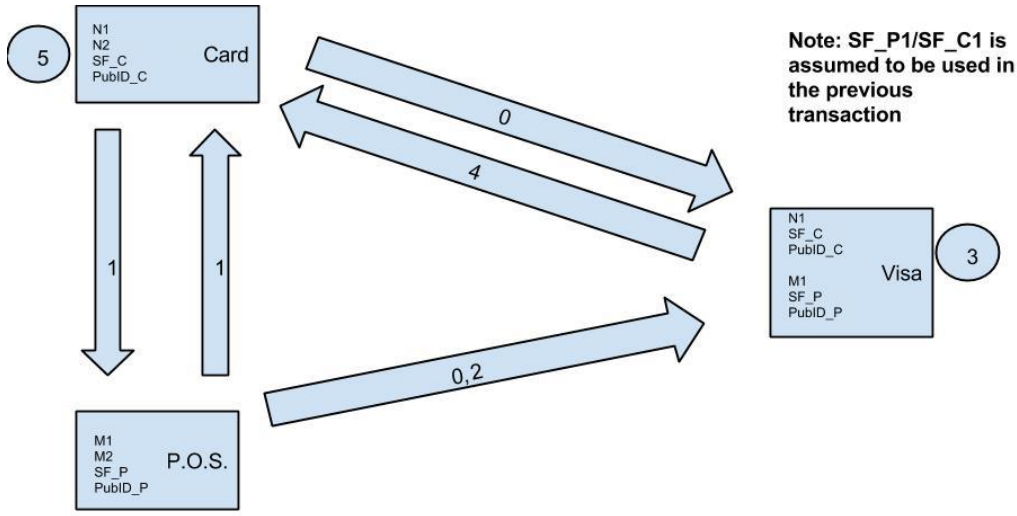

Figure 1*: Diagram of Three Party Authentication Process*

2.2.2 Card Approval Process

The card approval process is a series of five steps which serve to update the private ID of the cardholder both on the card and at Visa.

$$6.\ Card \rightarrow Visa\text{: }\$, N1 \oplus SF\_C2 \oplus N2$$

Card sends Visa two values: the dollar amount it received during the three party authentication process and the new private ID it generated on its own encrypted (XOR'd) with the current private ID and second factor. Card also sends Visa the dollar amount it received during the three party authentication process. Sending the dollar amount is necessary because it prevents any party, both involved with or outside of the transaction, from modifying the dollar amount to be charged to the cardholder's account. Visa now extracts the new private ID of the card (N2). It can do this because it has SF_C2 and N1, so it can XOR them with the second value that it received from card, and extract N2.

$$\text{N2} = (N1 \oplus SF\_C2 \oplus N2) \oplus (N1 \oplus SF\_C2).$$

Once Visa extracts the new private ID, N2, N1 is replaced with N2 in the Visa database.

$$7.\ Visa\text{: } Approve\ or\ deny\ transaction$$

Visa now must verify that the Card is holding the correct private ID and public ID by checking its database to see if N1 extracted in step 6 and PubID_C from step 0 match N1 and PubID_C stored in its database. Visa also must verify that the dollar amount in step 6 has not changed throughout the transaction. This is done by comparing the dollar amount in step 6 to the dollar amount received in step 2.

$$8.\ Visa\ \rightarrow\ POS\text{: } H(M1 \oplus SF\_P2 \oplus Y/N), \$$$

Visa sends POS a hash of the approval or denial, encrypted (XOR'd) with the POS private ID and second factor, along with the dollar amount to be charged to the cardholder.

$$9.\ POS\ determines\ approval\ or\ denial\ status$$

$$9a.\ POS\ verifies\ \$\ from\ step\ 8\ is\ equal\ to\ \$\ from\ step\ 1$$

$$9b.\ POS\ \rightarrow\ Card\text{: } \$$$

In step 9, POS must determine whether the transaction was approved or denied by Visa. The POS computes two hashes – H(M1⊕SF_P2⊕Y) and H(M1⊕SF_P2⊕N), and compares each with the message digest received in step 8. Should H(M1⊕SF_P2⊕Y) match the message from Visa in step 8, then the transaction was approved, but if H(M1⊕SF_P2⊕N) matches, then the transaction was denied. In step 9a, POS can now verify that the dollar amount to be charged to the cardholder has remained the same throughout the entire transaction and send the dollar amount to Card (step 9b).

$$10. Card\ verifies\ \$\ from\ step\ 9\ is\ equal\ to\ \$\ from\ step\ 1$$

$$10a.\ Card \rightarrow POS: Final\ Approval$$

Card must verify that the dollar amount received in step 9 matches the dollar amount received in step 1 of the three party authentication process. If they match, the Card must approve the transaction, and once approval has been made, the transaction is complete. The Card sends a final message to Visa to indicate its decision.

$$11. Card \rightarrow Visa: H(N2 \oplus SF\_C2)$$

Visa then computes a similar hash, if the received hash is the same, Visa proceeds with the charge. If the Card declines, instead of the hash being sent, a transmission of all "0's" is sent. If any bits are modified it is considered an insecure transaction. Each of the steps discussed in this section are illustrated in Figure 2.

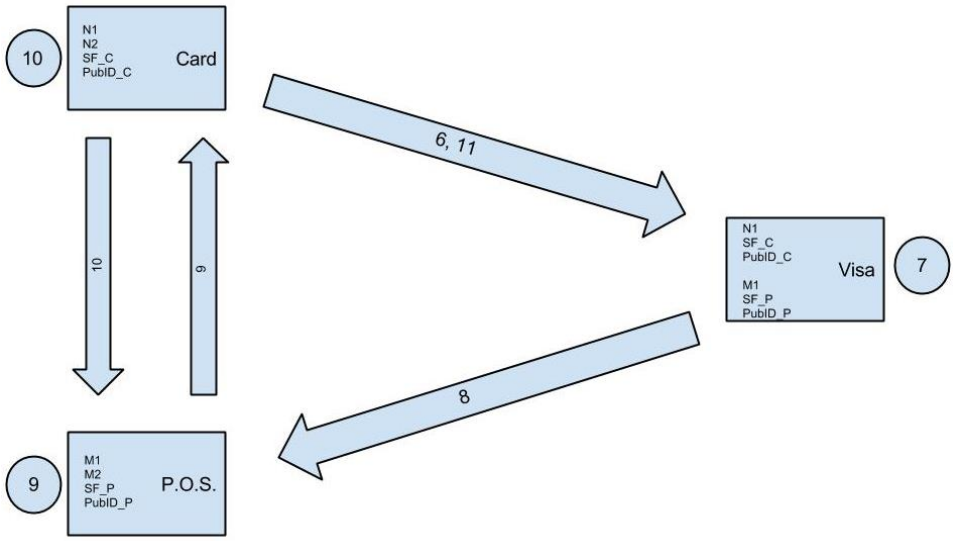

Figure 2*: Diagram of Approval Process*

2.3 Implementation

To start, there will a brief discussion on a prototypical implementation of such a protocol. This would include hardware representations of the three entities (Card, POS, and Visa). Each of these three parties must securely communicate with each other, sending authentication information along with monetary transaction information, for a credit card transaction to take place with the protocol described above. For this secure communication to take place, Card, POS, and Visa must have a processor as well as memory storage. Additionally, the Card and POS must both contain a true random number generator (TRNG) for the generation of new IDs. Lastly, the Card and POS must both contain a second factor generator as the protocol involves multiple factors of security. The blocks that make up a Card, POS, and Bank are shown in Figures 3, 4, and **7**, respectively.

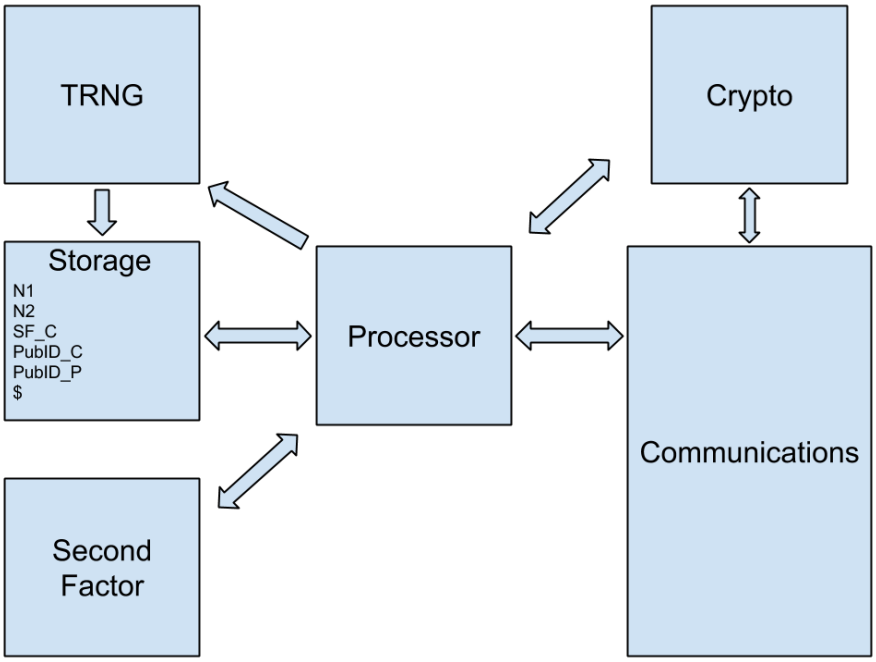

Figure 3: *Card block diagram*

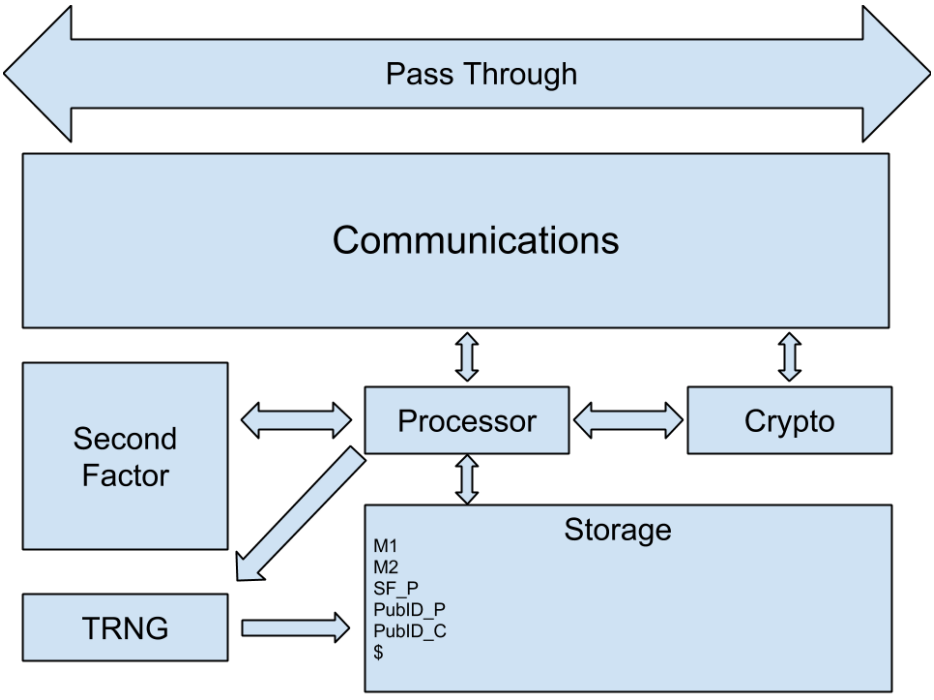

Figure 4: *POS block diagram*

Our implementation is discussed in two steps – the communication sub-system, and the True Random Number Generator (TRNG). These are discussed in the next two sub-sections.

2.3.1 Communication Sub-system

The implementation of the POS and Card, although similar, have specific distinctions which are a function of their intended uses. Each node (POS or Card) was made up of the same basic components however the typical usage scenarios for each have implied several design characteristics that when implemented will make each distinct for its intended function.

The POS contained two major components, the TRNG and a processor. The TRNG was attached to the processor for use throughout the transaction process. The processor itself contained software code for communication, memory and storage, as well as cryptographic functions such as hashing.

The Card also contained all of the aforementioned components. Each of these components will be discussed separately, later on. It is important to note that the card can function in one of two modes. The first mode occurs when the Card is used at a physical "brick and mortar" shop or point of sale. This mode is referred to as the "Walmart Case". The second mode occurs when the Card is used over a network such as the internet for purchases at a distant point of sale. This mode is referred to as the "Amazon Case". Each of these cases is what makes the architecture of the Card slightly different than that of the POS. The POS tends to be a static implementation, unlike the Card.

For the “Walmart Case”, the Card made use of the POS’s network connection to authenticate with Visa. A potential implementation of this would be to plug the Card into a socket at the point of sale similar to the function of an ATM machine. The “Amazon Case” would be similar except that the point of sale could be thousands of miles away (on the servers of Amazon). In this case the Card would make use of the card owner’s network connection via personal computer, home network or even through the user’s smartphone, to connect to the Amazon servers. Unlike the card, the POS will generally be installed in a static location with power and network connection provided. This is the only distinguishing characteristic between the Card and POS.

The protocol was implemented using Java 1.7 to create a client/server based communication application. In the discussion below, “server” refers to Visa, and “nodes” refer to either the Card or the POS. This design decision was made for several reasons. First, Java is architecture-agnostic. This enabled the implementation of lightweight embedded solutions which have the ability to run our scheme. Second, Java has memory management and eliminates different types of attacks such as Buffer Overflow Attacks. Datagram (UDP) based communication was chosen due to the connectionless nature of UDP. This prevents a socket from being permanently open to any given node, and creates much less overhead in the packets themselves. Because we wanted the Card and POS to be as lightweight as possible, this approach seemed to be the most viable. Visa being the server and the Card/POS being the clients, it was clearly advantageous that the server have vastly more compute power than the clients. Nearly all of the protocol management is handled by Visa. We determined that the best practices allow for a UDP packet’s payload to be 512 bytes. This was where all of the information required for our protocol to manage itself was located. We defined all of our primitives inside the payload of UDP packets. This decision was

made so that our implementation would be routable over the internet. Each step was wholly contained in a single UDP packet, this prevented packet fragmentation over lossy networks such as wireless or cell networks. A graphic of the UDP Payload and our protocol's packet structure is included below in Figure 5.

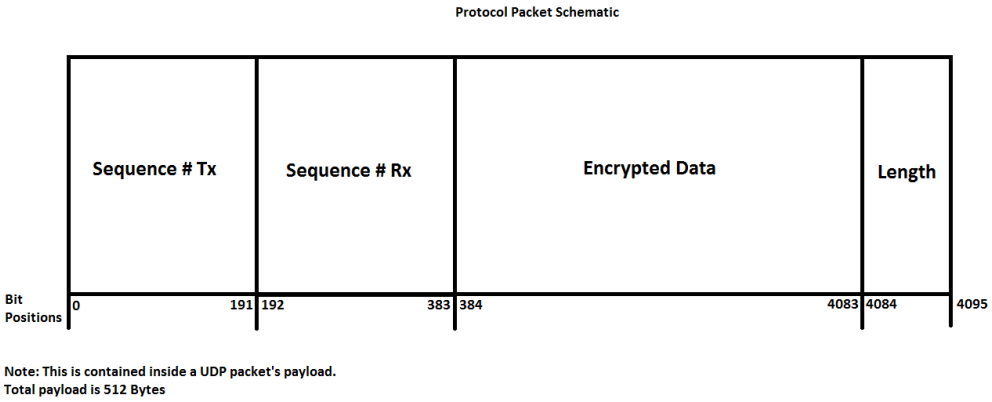

Figure 5: *Protocol Packet Schematic*

The sequence numbers were used to track who the packet belonged to, and what step the contents of the encrypted data are appropriate for, as well as what step was being requested next. The encrypted data section was simply the payload for the given step number. The length section described how long (in bits) the encrypted data was. Each packet received was unpacked and processed individually to prevent one node from jumping ahead of another node in the process of a transaction. The system's status was maintained in lists on the server and used to verify where each node was in the algorithm throughout the transaction. Due to the nature of asynchronous communications it was necessary to sometimes throttle the speed of the process to ensure events happened in the proper order and correctly. This was accomplished by using acknowledgement (ACK) packets. These packets were sent by the server to the nodes to acknowledge that their information was received and processed. ACK packets were generated and sent at the end of specific steps in the protocol to ensure proper flow and prevent any race conditions. The hash

function mentioned above was implemented as SHA-512 for both the Card and POS. The SHA family of hash functions are widely used as cryptographic functions, and SHA-512 was chosen specifically to enhance the cryptographic integrity of our protocol. The second factor was used throughout the communication process to ensure cryptographic integrity and to assist in authentication. Each node came pre-loaded with a random image generated by Visa. This image had a high resolution and was geometrically randomized. The image itself was stored locally on the POS and the Card for use by the system to generate the second factor. The way the second factors (SF_P and SF_C) were generated is as follows. The node received an update request. An algorithm picked at random a range of pixels from the stored image. The random range was derived by seeding the algorithm with the respective N1 or M1 value. Because N1 and M1 were generated at random by the TRNG (discussed later), the range chosen by the algorithm was also random. These pixels were resolved down to binary 1's and 0's and sent to the hash function which produced its output (also referred to as the message digest of the binary message). The output of the hash was the new second factor. This process is outlined in the figure below.

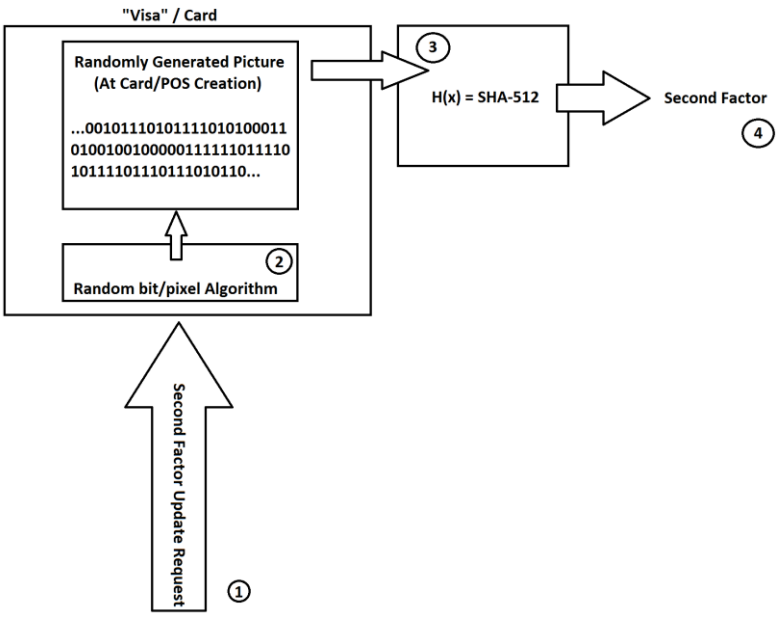

Figure 6: *Hash Architecture*

The final piece of the communication sub-system is Visa. Visa (also referred to as Bank) is considered the authorizing body and is responsible for maintaining a database for all users of the system. This includes information pertaining to all the Card users and the POS users. It should be mentioned that some assumptions about the Bank have been made to establish a firm scope for the overall project. The first assumption is that the Bank will set up and maintain their own system, network, databases and subsystems as required for their purposes. The second assumption is that the Bank is a secure entity with its own intrusion prevention and security protocols to prevent data loss and data compromise. For each user (whether a Card or POS) the Bank maintains current versions of the following information: the current Card Number/POS Number (N1 or M1 respectively), the current second factor for the Card or POS (SF_C or SF_P respectively) and the public ID for each user (denoted as PubID_C for the Card user and PubID_P for the POS user). The Bank will maintain authority across all transactions and will administer the authentication process and the approval process for each transaction. An overview

of the Bank's responsibilities can be seen in Figures 1 and 2. An illustration of the major components of the Bank can be seen in Figure 7 below.

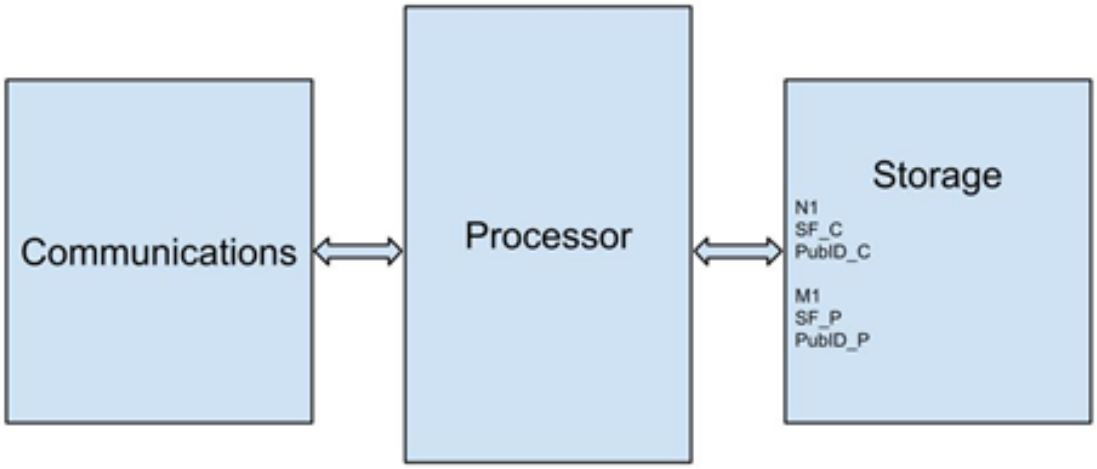

Figure 7: *High-level Bank/Visa block diagram*

The Bank performs its authorization role by analyzing the incoming communications from each node in a transaction and comparing stored information (secret and otherwise) to what has been received. This process requires the use of a hash function identical to the one used and described for the Card and POS. This function will be used to verify the contents of cryptographic transmission during the authorization and approval process. The Bank's main responsibility in each transaction is to confirm the identity of all parties in a transaction and to verify available cash or credit to a Card. The Bank does not however approve or disapprove the transaction, which is reserved to the Card itself. The overall process in all of its steps can be seen in Section 2.2.1 (Three Party Authentication) and Section 2.2.2 (Card Approval Process).

2.3.2 True Random Number Generator (TRNG)

We use the TRNG to generate the Card ID numbers and the POS ID numbers. These outputs of the TRNG are used to generate the Second-Factors (N1, M1 and SF_C, SF_P respectively).

The TRNG is based on the physicality of the bipolar junction transistor (BJT) circuit element. Every transistor (due to manufacturing limitations) will have minute variances in its electrical properties and therefore possesses a unique and random response when its p-n junction is properly reverse biased. The key to obtain a random output is to leave the reverse biased npn transistor's collector node floating. This draws a random amount of current to the collector from its emitter node. This process is known as the avalanche noise effect[3], [13], [14]. Essentially, we monitor the voltage generated by the random electrical current output by a reverse biased transistor near its breakdown voltage and amplify that signal to populate a memory element. This memory element is then accessed when a new random number is required. This process happens at a very high frequency (about 10MHz in our implementation) and hence the TRNG is capable of generating large random numbers very quickly.

The TRNG design is modified from Rob Seward's random number generator[16]. We chose this implementation of a TRNG due to its simplicity, since it uses only three transistors and a few basic circuit elements. The total power consumption for our implementation was 22 mW. We constructed the circuit using a 12VDC supply to ensure that the transistors would operate at the breakdown voltage. This was required to reverse bias Q1. The specific BJTs used only required (10.82 VDC). We anticipate that this design could be reduced in size and power consumption

considerably. Additionally, the method of amplifying and sampling the output of our TRNG is fairly straightforward. To amplify the output, we simply used another BJT transistor.

In our TRNG design, shown in Figure 8, transistor Q1 is the reverse biased transistor with its collector node floating. Because the collector node is floating, Q1 is drawing random amounts of current from its emitter. The bases of Q1 and Q2 are connected, so the intrinsic resistance of Q1 in conjunction with random amounts of current drawn results in random voltages at the base of Q2. This places transistor Q2 to operate in the linear region, thereby passing the random signal generated by the reverse biased transistor, Q1, to transistor Q3 where it is then amplified. This generates the output signal (Vout) from our TRNG which is then sampled and converted into a bit string.

To sample our output signal we used a modified version of Von-Neumann Entropy Extraction Method[4]. We measure the average period of our signal and sample at three times that frequency. Each sample is actually constructed of two readings of the voltage of the amplifier, taken very close to each other. In practice, these samples are separated by 100 microseconds as limited by our hardware. The difference between these two readings can be a representative of the derivative of the output signal. If the polarity of the derivative is negative, we assign a logical '0' value to the corresponding TRNG bit. If the polarity of the derivative is positive, we assign a logical '1' value to the corresponding TRNG bit. Because our sampling does not use any reference voltages or currents, it is very robust to environmental conditions such as temperature.

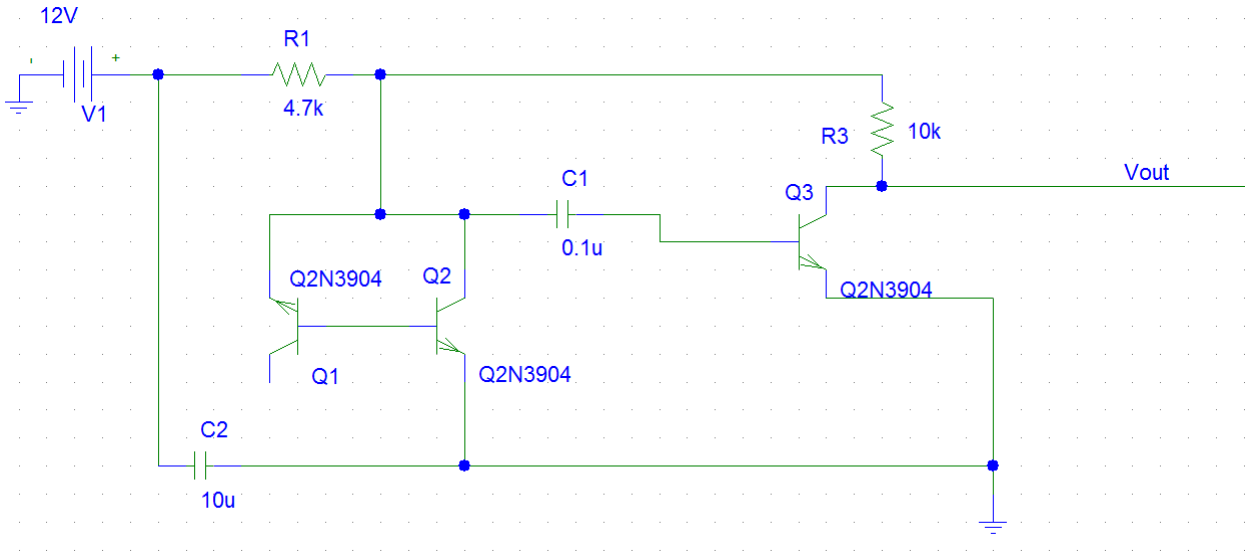

Figure 8: *True Random Number Generator Circuit Schematic*

CHAPTER III

RESULTS

3.1 Protocol Verification

As mentioned earlier, the XOR operation serves as the primary means of encryption in our protocol. Once multiple numeric entities are XOR'd a message is created. To extract an individual entity from a message, all the entities except for the one that is to be extracted, must be XOR'd with the message. For example, consider a message that is the XOR of two entities A and B (denoted $A \oplus B$). Suppose B is known. Then, if A is to be extracted from this message, this is done by computing $A \oplus B \oplus B$. In the various steps of our protocol (described in Sections 2.2 and 2.3), private IDs as well as other numeric information stored on the Card, POS, and Visa are XOR'd together (and/or hashed) and are transmitted to the intended recipient. During the transmission process, the encrypted message is exposed to the public and can potentially be intercepted or stored by a malicious attacker. Due to the nature of the XOR operation, the ability it provides to extract individual entities in the operation, the attacker could potentially extract sensitive information such as the private IDs or the second factors of the Card or POS. As a result, the protocol must be put through a verification process that tracks and stores all the transmitted information of the protocol, in an attempt to extract private, sensitive information.

To verify the security of our protocol, a C program (we will refer to this program as our *Verification Algorithm*) was generated to store all exposed communications between the three

entities (Card, POS, and Visa) for two consecutive iterations of the protocol. The exposed communications comprise any information that is transmitted along the arrows in Figures 1 and 2. From these stored values, the program attempts to extract any private or sensitive information. To accomplish this, every possible XOR combination of the exposed encrypted messages are generated.

The *Verification Algorithm* takes the universe of transmitted messages as input. If the universe of transmitted messages is of size N, our program generates the XOR of all possible combinations of 2, 3, 4, … N messages (terms).. The total number of generated terms is $2^N - 1 - N$. After this generation step, we have a total of $2^N - 1$ messages in a single list. In the second step, this list is uniquified (only one copy of every item in the list is retained). After uniquification, we scan all these messages, and flag any messages which expose any private or sensitive information (such as N1, N2, M1, M2, SF_P1, SF_P2, SF_C1 or SF_C2).

For example, suppose the universe of transmitted messages in the two consecutive iterations of our protocol are the messages $(A \oplus B)$, $(A \oplus C)$, $(A \oplus D)$, and $(B \oplus C)$, as shown in Figure 9 (Column 1). The C program generates the XOR of every possible combination of the XOR of 2, 3, 4, … N of these messages, as shown in Column 2 of Figure 9. This list is uniquified and the resulting output is shown in Column 3 of Figure 9. Note that since Column 3 does not have a singleton variable A, B, C or D in it, the messages communicated in this example do not leak the values of any of these variables.

Universe of Transmitted Messages	Universe and All Generated Messages	Uniquified List of Extracted Messages
$A \oplus B$	$A \oplus B$	$A \oplus B$
$A \oplus C$	$A \oplus C$	$A \oplus C$
$A \oplus \mathrm{D}$	$A \oplus \mathrm{D}$	$A \oplus \mathrm{D}$
$B \oplus C$	$B \oplus C$	$B \oplus C$
	$B \oplus C$	removed
	$B \oplus D$	$B \oplus D$
	$A \oplus \mathrm{C}$	removed
	$C \oplus \mathrm{D}$	$C \oplus \mathrm{D}$
	$A \oplus \mathrm{B}$	removed
	$A \oplus \mathrm{B} \oplus \mathrm{C} \oplus D$	$A \oplus \mathrm{B} \oplus \mathrm{C} \oplus D$
	$B \oplus \mathrm{D}$	removed
	$C \oplus D$	removed
	0	0
	$A \oplus B \oplus \mathrm{C} \oplus \mathrm{D}$	removed
	$A \oplus \mathrm{D}$	removed

Figure 9*: Transmitted and Extracted Message Table*

For our protocol, the results of this program indicate that public information, such as public ID numbers and price of an item, can be recovered. However, no private information stored on Card, POS, or Visa can be extracted by a malicious attacker snooping the communication channels. An abridged version of these results are presented in Appendix A.

3.2 TRNG Verification

The verification process for the TRNG was performed in both a quantitative and qualitative manner. Qualitative results were analyzed by extracting a large string of output bits from the TRNG. This output string was converted into a bitmap image, and displayed. This process and its results will be discussed in Section 3.3.

Quantitative results were obtained using the NIST[8] analysis suite. The NIST suite is distributed by the National Institute of Standards and Technology, and is widely considered the gold standard in testing if a bit string is truly random. The NIST suite conducts several tests on the bit string, and if the bit string passes all or most of these tests, it is considered random. We produced multiple bit strings (specifically, we produced 20 sample bit strings, each of length 1 million) from our TRNG and fed them into the NIST suite for statistical analysis. There were 15 tests performed in total. Our TRNG passes all NIST tests, which is a very strong indication that the output of our TRNG module is cryptographically viable, and that the TRNG produces random numbers. These results are depicted below in Figures 10-12. A full account of the results can be found in Appendix B. Figure 10 reports the proportion of passing samples among our 20 sample bit strings. Note that for each test, this proportion is above the minimum passing proportion. Figure 11 reports the p-values for each of the 15 tests, which are all above the passing p-values. Finally, Figure 12 presents a description of each of the 15 NIST tests that were run.

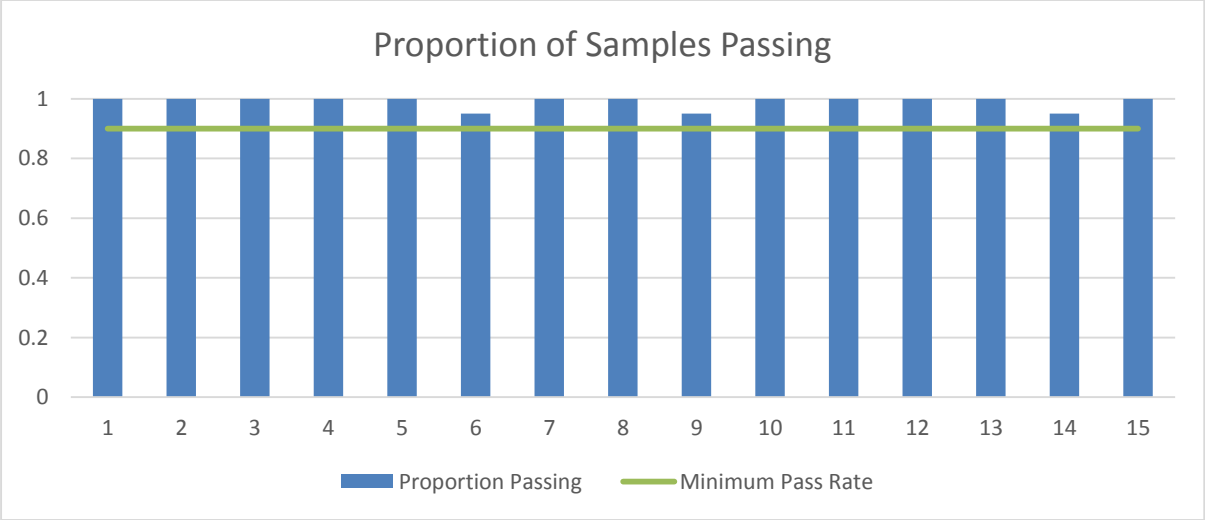

Figure 10: *Proportion of samples passing each NIST test*

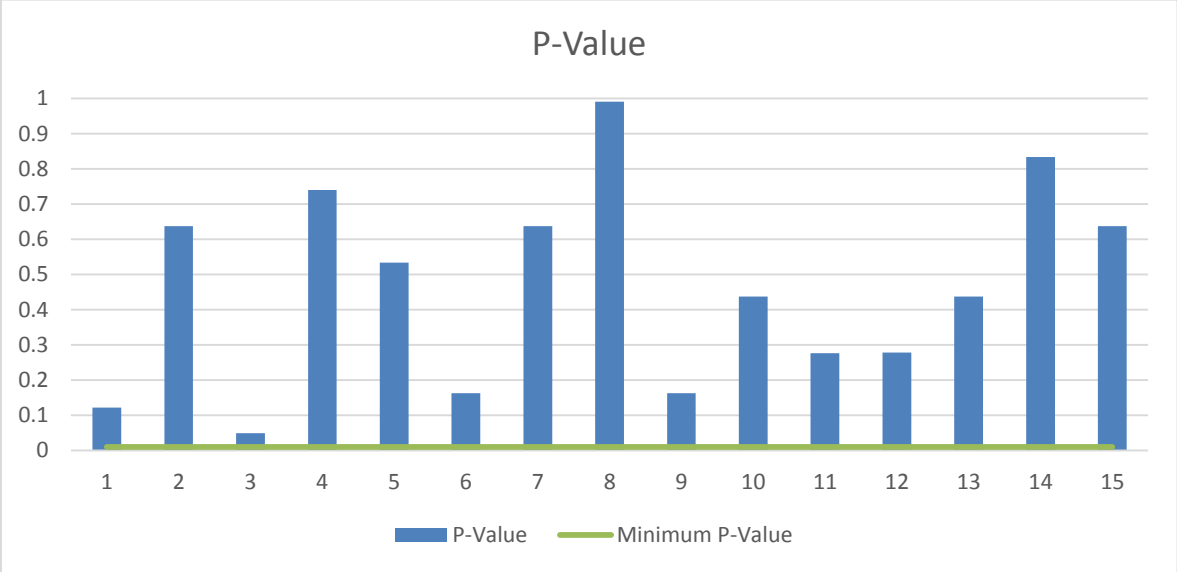

Figure 11: *P-Value for each NIST test*

Test Number	Test Name
1	Frequency
2	Block Frequency
3	Cumulative Sums
4	Runs
5	Longest Run
6	Rank
7	FFT
8	Non Overlapping Template Matching Test
9	Overlapping Template Matching Test
10	Universal Statistical Test
11	Approximate Entropy
12	Random Excursions Test
13	Random Excursions Variant Test
14	Serial
15	Linear Complexity

Figure 12: *List of test numbers and test names*

3.3 Bitmap Analysis

Bitmap images are a simple image format. This format is particularly simple to generate given a bit string. Given the bits of a bit string, successive clusters of k bits form the grey-scale value of a particular pixel in the bitmap image. In our case we used a bit string generated by our TRNG. For comparison, we also generated the same number of bits from the Java random number generator. We produced two bitmap images one from each bit string. The comparison of these two images serves as our qualitative evaluation for the TRNG output. Each bit string was 20 million bits in length. This ensured that a large enough sample from each source (Java and our TRNG) was produced to provide a reliable comparison of the performance of each RNG. Figures 13 and 14 display the resulting bitmap images for our TRNG (labeled TRNG) and Java RNG (labeled JRNG). Open source code written by Philipp C. Heckel, under the GNU License was modified in order to generate the images[15]. We then ran each image through a filter to equalize the average darkness of the output (to make the comparison between the images easier). Note that the TRNG image is smoother and less granulated when compared to the JRNG image.

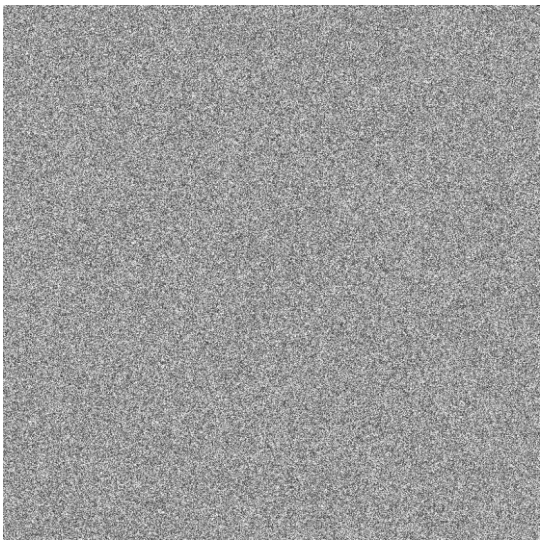

Figure 13: *TRNG*

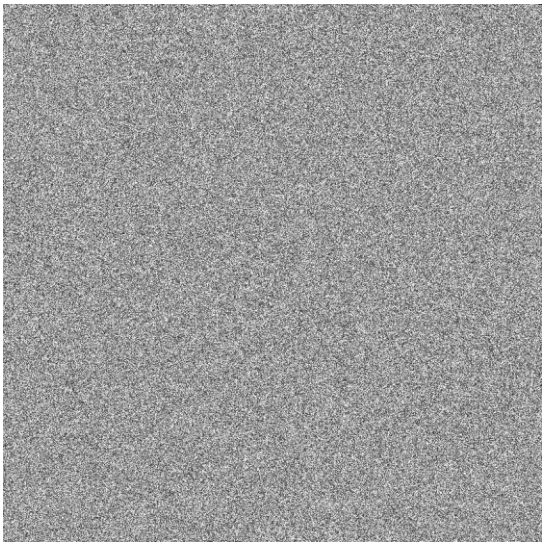

Figure 14: *JRNG*

3.4 DMFS Physical Implementation

We implemented the Card and POS as a client and Visa as a server. The clients were installed on Intel Edison development boards, and the server was installed on an Intel Next Unit of Computing (NUC). The client and server were developed in Eclipse IDE on Java 1.7. To verify the protocol both clients and the server were connected over WiFi (IEEE 802.11g). A photo of our overall implementation is included in Figure 15. The Edisons are outlined in black and red while the NUC is outlined in blue.

Our implementation of the protocol handles a variety of reasonable operating conditions and error conditions. To perform our tests, we ran transactions through our system described above and observed for errors (both logical and runtime) to validate functionality. The test cases used we run over two transactions in order to provide the same exposure as our *Verification Algorithm* from Section 3.1. Correct operation (authentication of the card, update of the new card ID, and granting of funds for purchase) were tested for two consecutive transactions.

Below is a list of exceptions and errors that were successfully handled by our implementation of this protocol.

1) Incorrect Pub_ID
2) Incorrect Private ID (i.e. N1, N2, M1, and M2)
3) Incorrect Dollar Amount
4) Not Enough Funds Available
5) Out Of Sync Car/POS

6) Malicious Approval Packet Detection

7) Authentication Error

This list is to be considered a minimum for acceptable error handling for proper protocol function. Conceivably, there are additional error cases that will be dependent upon the policies and specific protocols of the implementing organization.

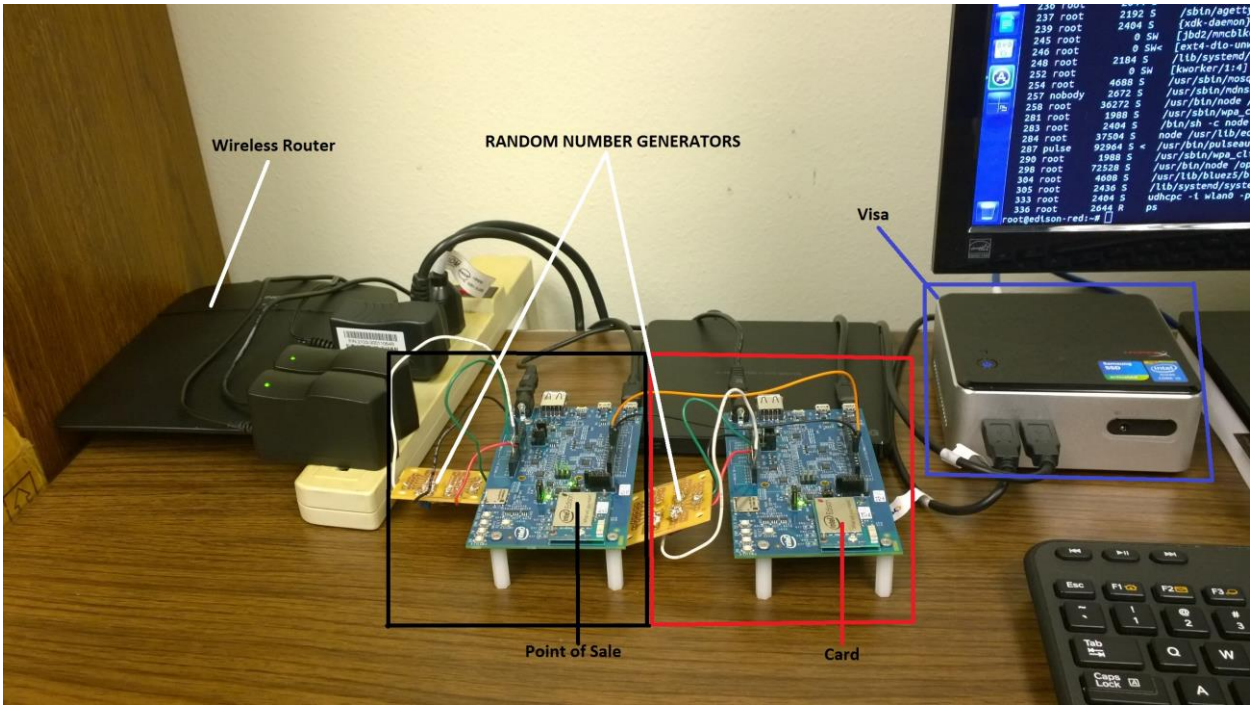

Figure 15: DMFS Physical Implementation

3.5 Discussion

Our protocol implemented an authentication scheme that verifies all parties in a transaction. This authentication process assumes that no node trusts any other node at the beginning of the transaction, and by the end of it, all nodes agree that everyone is whom they claim to be. This addresses several currently failings. Our scheme is able to thwart *spoofing* attacks by using

cryptographic hashing, by detecting bit manipulation and rejecting malicious packets. Furthermore, *Man-In-The-Middle* eavesdropping attacks are mitigated through the use of sophisticated encryption that rotates credentials every transaction cycle. We were able to implement our system in such a way that an eavesdropper would need intimate knowledge of the pre-shared (secret) second factors to gain access to the unencrypted messages. Unauthorized access is mitigated by the mechanism in our protocol that authenticated every node in the system.

Other shortcomings of current systems are handled by our implementation as well. Specifically, the lack of a second factor, the lack of standardization and the lack of an authentication layer are all addressed as design parameters to our algorithm. They comprise the foundation of how our protocol operates; as such, all of these concerns are addressed. The key strength of our scheme lies in the fact that for both the POS and the Card, both the IDs and the second factors are changed with every iteration.

We verified the cryptographic integrity of our protocol by evaluating each combination of data transmitted, and every permutation was XOR'd recursively until no new information was generated. We evaluated the results to determine if any critical information was exposed over two successive iterations. From this result, by using mathematical induction, we can show that this results in no information leakage over an infinite number of protocol iterations. This was accomplished by writing a C program to permute and compute all combinations of transmitted data symbolically. Treating each message as a symbol allowed us to quickly validate that no leakage had occurred. The benefit is that we can say, if implemented properly, and the second

factor is unknown, the likelihood for an attacker to glean any sensitive information is practically zero. In other words, a brute force attack would be computationally infeasible.

One of the strengths of our system is the TRNG. It is important to verify how random our random number generator is, and this was done by generating large bit streams from the device and passing them into a series of standardized tests included in a test suite developed by the National Institute of Standards and Technology (NIST). The NIST suite is a battery of tests which implement sophisticated statistical analysis of large bit streams (millions of bits) that were generated by our TRNG. Our TRNG passed all 15 of these tests.

If our scheme was in use, the Target credit card breach would be benign, since the hackers would have obtained stale IDs and second factors. These IDs and second factors would not be valid for the following transaction, protecting both the card owners and the corporations involved.

CHAPTER IV

CONCLUSION

Current remote authentication systems have frequently exhibited significant weaknesses. Nearly all modern remote authentication technologies are built upon older systems and protocols. These older implementations were not designed to be used in today's e-commerce, mobile and web-rich environments. New threats are usually addressed by adding additional layers (patches) onto existing systems. The overall performance of these systems in terms of security has proven to be lacking time and again.

Our protocol addresses many concerns raised by recent cyber-attacks. With the features outlined in this thesis, we address *Spoofing Attacks*, *Man-In-The-Middle Attacks*, *Extension Attacks*, *Forgery Attacks*, *Skimming Attacks*, *Buffer Overflow Attacks* and many others. Our scheme was designed from the ground up as a general purpose remote authentication protocol with applications in e-commerce and as such, can be easily modified to accommodate many other use cases such as: secure VPN login, remote access to patient records or legal documents, remote sensing and nearly any other application requiring three parties to authenticate and exchange sensitive information.

What makes our approach novel is the use of dynamic ID's and a dynamic second factor in concert with reliable hashing algorithms to encrypt all traffic natively inside the protocol. We built in a three party authentication procedure and ways to detect malformed and malicious packets. The protocol is also resistant to packet fragmentation and works well in asynchronous

transmission media. There would be no need to implement additional layers of authentication or security with our design.

We also showed our scheme to be theoretically secure using tools such as NIST, Bitmap analysis and a sound theoretical analysis (*Verification Algorithm*) that ensures that our scheme does not leak sensitive information. The protocol was implemented on real hardware and over real networks and tested for functionality. We have designed, tested, verified; and implemented the scheme described in this thesis successfully.

As we look toward the future and the advancement of the Internet of Things (IoT), biometrics as second factor or even as an additional factor would be an easy addition to our system. Fingerprint scanners could be added to the Card and to the POS devices to prevent unauthorized access and thereby provide a layer of physical security to the system[5]. We anticipate that scaling our design down to fit onto a real card or even a mobile device is well within reach.

REFERENCES

[1] C. Chen, C. Lee and C. Hsu, "'Mobile device integration of a fingerprint biometric remote authentication scheme," International Journal of Communication Systems, vol. 25, no. 5, pp. 585-597.

[2] M.L. Das, A. Saxena and V.P. Gulati, "'A dynamic ID-based remote user authentication scheme," Consumer Electronics, IEEE Transactions on, vol. 50, no. 2, pp. 629-631.

[3] J.B. Johnson, "'Thermal agitation of electricity in conductors," Nature, vol. 119, no. 2984, pp. 50-51.

[4] B. Jun and P. Kocher, "'The Intel random number generator," Cryptography Research Inc.white paper.

[5] M.K. Khan, "'Fingerprint biometric-based self-authentication and deniable authentication schemes for the electronic world," IETE Tech.Rev., vol. 26, no. 3, pp. 191-195.

[6] M.K. Khan, S. Kim and K. Alghathbar, "'Cryptanalysis and security enhancement of a 'more efficient & secure dynamic ID-based remote user authentication scheme'," Comput.Commun., vol. 34, no. 3, pp. 305-309.

[7] L.B. Kish, "'Totally secure classical communication utilizing Johnson (-like) noise and Kirchoff's law," Physics Letters A, vol. 352, no. 3, pp. 178-182.

[8] A. Rukhin, J. Soto and J. Nechvatal, A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications, 1997.

[9] H. Sun, Q. Wen, H. Zhang and Z. Jin, "'A novel remote user authentication and key agreement scheme for mobile client-server environment,"Appl.Math, vol. 7, no. 4, pp. 1365-1374.

[10] A. Tiwari, S. Sanyal, A. Abraham, S.J. Knapskog and S. Sanyal, "'A multi-factor security protocol for wireless payment-secure web authentication using mobile devices," arXiv preprint.

[11] Y. Wang, J. Liu, F. Xiao and J. Dan, "'A more efficient and secure dynamic ID-based remote user authentication scheme," Comput.Commun., vol. 32, no. 4, pp. 583-585.

[12] F. Wen and X. Li, "'An improved dynamic ID-based remote user authentication with key agreement scheme," Comput.Electr.Eng., vol. 38, no. 2, pp. 381-387.

[13] B.M. Wilamowski and J.D. Irwin, "'Fundamentals of industrial electronics,", 2011.

[14] H. Zhun and C. Hongyi, "'A truly random number generator based on thermal noise,", pp. 862-864.

[15] GNU General Public License (June 29, 2007) Version 3. Free Software Foundation. URL: http://www.gnu.org/licenses/gpl.html

[16] CC Creative Commons License Attribution-NonCommercial 4.0 International URL:

http://creativecommons.org/licenses/by-nc/2.5/legalcode

APPENDIX

Appendix A: XOR Blaster Abridged Results

Eve's Vector...All communication wires known. 2 Transactions. Rev8.
PubID_C
HashPubID_PXORM1XORSF_P1
PubID_P
$
H(M2^SF_P2^Y/N)
Y/N
HashPubID_PXORM2XORSF_P2
$2
H(N2^SF_C2)
H(N3^SF_C3)
H(M2^SF_P3^Y/N2)
Y/N2
N1 ^ N2
PubID_C ^ HashPubID_PXORM1XORSF_P1
PubID_C ^ PubID_P
PubID_P ^ HashPubID_PXORM1XORSF_P1
PubID_C ^ $
$ ^ HashPubID_PXORM1XORSF_P1
PubID_P ^ $
PubID_C ^ H(M2^SF_P2^Y/N)
HashPubID_PXORM1XORSF_P1 ^ H(M2^SF_P2^Y/N)
PubID_P ^ H(M2^SF_P2^Y/N)
$ ^ H(M2^SF_P2^Y/N)
PubID_C ^ Y/N
Y/N ^ HashPubID_PXORM1XORSF_P1
PubID_P ^ Y/N
Y/N ^ $
Y/N ^ H(M2^SF_P2^Y/N)
PubID_C ^ HashPubID_PXORM2XORSF_P2
HashPubID_PXORM1XORSF_P1 ^ HashPubID_PXORM2XORSF_P2
PubID_P ^ HashPubID_PXORM2XORSF_P2
$ ^ HashPubID_PXORM2XORSF_P2
H(M2^SF_P2^Y/N) ^ HashPubID_PXORM2XORSF_P2
Y/N ^ HashPubID_PXORM2XORSF_P2
PubID_C ^ $2
HashPubID_PXORM1XORSF_P1 ^ $2
PubID_P ^ $2
$ ^ $2
H(M2^SF_P2^Y/N) ^ $2
Y/N ^ $2

HashPubID_PXORM2XORSF_P2 ^ $2
PubID_C ^ H(M2^SF_P3^Y/N2)
HashPubID_PXORM1XORSF_P1 ^ H(M2^SF_P3^Y/N2)
PubID_P ^ H(M2^SF_P3^Y/N2)
$ ^ H(M2^SF_P3^Y/N2)
H(M2^SF_P2^Y/N) ^ H(M2^SF_P3^Y/N2)
Y/N ^ H(M2^SF_P3^Y/N2)
HashPubID_PXORM2XORSF_P2 ^ H(M2^SF_P3^Y/N2)
H(M2^SF_P3^Y/N2) ^ $2
PubID_C ^ Y/N2
HashPubID_PXORM1XORSF_P1 ^ Y/N2
PubID_P ^ Y/N2
$ ^ Y/N2
H(M2^SF_P2^Y/N) ^ Y/N2
Y/N ^ Y/N2
Y/N2 ^ HashPubID_PXORM2XORSF_P2
Y/N2 ^ $2
Y/N2 ^ H(M2^SF_P3^Y/N2)

Appendix B: NIST Results

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
---
generator is <data/20M.pi>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0	1	0	1	5	2	1	4	2	4	0.122325	20/20	Frequency
1	4	2	2	2	2	2	1	0	4	0.637119	20/20	BlockFrequency
0	0	1	0	2	1	1	4	4	7	0.004301	20/20	CumulativeSums
0	1	1	1	1	4	0	3	3	6	0.048716	20/20	CumulativeSums
2	2	4	1	2	0	3	1	2	3	0.739918	20/20	Runs
3	4	0	0	2	3	2	3	2	1	0.534146	20/20	LongestRun
6	2	2	0	2	0	2	1	2	3	0.162606	19/20	Rank
3	1	1	0	3	2	2	1	3	4	0.637119	20/20	FFT
4	2	1	5	1	2	0	1	2	2	0.350485	20/20	NonOverlappingTemplate
5	1	2	0	1	2	2	2	1	4	0.350485	20/20	NonOverlappingTemplate
3	0	3	2	1	2	2	1	4	2	0.739918	19/20	NonOverlappingTemplate
2	0	3	1	1	3	3	1	4	2	0.637119	20/20	NonOverlappingTemplate
2	3	4	0	2	1	2	3	0	3	0.534146	20/20	NonOverlappingTemplate
1	0	2	3	0	4	2	6	0	2	0.048716	20/20	NonOverlappingTemplate
2	2	0	2	4	4	1	2	1	2	0.637119	20/20	NonOverlappingTemplate
1	0	5	2	1	2	1	2	2	4	0.350485	20/20	NonOverlappingTemplate
4	1	0	0	1	3	5	1	5	0	0.025193	19/20	NonOverlappingTemplate

1	2	1	1	4	2	3	2	4	0	0.534146	20/20	NonOverlappingTemplate
2	3	4	2	1	1	0	1	1	5	0.275709	20/20	NonOverlappingTemplate
0	3	2	2	4	2	1	0	2	4	0.437274	20/20	NonOverlappingTemplate
4	0	1	1	0	2	2	6	1	3	0.066882	20/20	NonOverlappingTemplate
1	3	6	2	1	1	2	0	3	1	0.162606	20/20	NonOverlappingTemplate
0	1	2	3	2	2	0	4	4	2	0.437274	20/20	NonOverlappingTemplate
1	3	1	3	2	1	1	3	2	3	0.911413	19/20	NonOverlappingTemplate
2	2	6	0	1	1	3	2	3	0	0.122325	20/20	NonOverlappingTemplate
5	1	2	3	1	1	1	1	3	2	0.534146	19/20	NonOverlappingTemplate
1	2	2	1	1	1	2	3	2	5	0.637119	20/20	NonOverlappingTemplate
0	1	0	1	8	2	1	0	2	5	0.000439	20/20	NonOverlappingTemplate
1	2	1	2	6	2	2	1	1	2	0.350485	20/20	NonOverlappingTemplate
1	3	0	4	4	3	3	1	1	0	0.275709	20/20	NonOverlappingTemplate
3	0	0	4	1	1	2	1	5	3	0.162606	19/20	NonOverlappingTemplate
7	3	1	2	2	0	2	0	2	1	0.035174	20/20	NonOverlappingTemplate
4	2	1	0	2	3	2	1	2	3	0.739918	20/20	NonOverlappingTemplate
6	3	1	1	1	0	2	2	3	1	0.162606	19/20	NonOverlappingTemplate
0	3	1	2	3	0	7	2	1	1	0.025193	20/20	NonOverlappingTemplate
2	1	2	0	4	1	4	3	2	1	0.534146	20/20	NonOverlappingTemplate
1	2	1	0	1	1	7	1	3	3	0.035174	20/20	NonOverlappingTemplate
2	3	2	2	2	0	1	2	4	2	0.834308	20/20	NonOverlappingTemplate
4	2	3	1	2	3	2	2	0	1	0.739918	18/20	NonOverlappingTemplate
4	3	5	3	0	2	1	0	1	1	0.162606	19/20	NonOverlappingTemplate
4	0	3	0	2	2	2	1	2	4	0.437274	19/20	NonOverlappingTemplate
2	2	3	2	2	2	3	1	2	1	0.991468	20/20	NonOverlappingTemplate
1	2	0	4	4	3	3	2	0	1	0.350485	19/20	NonOverlappingTemplate
3	2	2	3	4	1	1	1	3	0	0.637119	20/20	NonOverlappingTemplate
1	1	2	3	2	0	1	3	2	5	0.437274	20/20	NonOverlappingTemplate
1	2	3	0	2	4	2	3	1	2	0.739918	20/20	NonOverlappingTemplate
2	4	1	4	1	0	3	1	1	3	0.437274	20/20	NonOverlappingTemplate
2	1	1	2	3	2	4	2	1	2	0.911413	20/20	NonOverlappingTemplate
0	3	2	2	3	3	1	1	2	3	0.834308	20/20	NonOverlappingTemplate
2	3	1	1	2	1	3	4	1	2	0.834308	20/20	NonOverlappingTemplate
0	2	2	0	2	6	2	1	2	3	0.162606	20/20	NonOverlappingTemplate
4	0	4	1	4	1	2	1	3	0	0.213309	20/20	NonOverlappingTemplate
2	1	4	3	1	1	1	5	0	2	0.275709	20/20	NonOverlappingTemplate
4	2	2	1	2	0	2	3	4	0	0.437274	19/20	NonOverlappingTemplate
4	3	2	2	3	1	3	2	0	0	0.534146	20/20	NonOverlappingTemplate
2	0	3	2	4	0	1	3	4	1	0.350485	20/20	NonOverlappingTemplate
1	1	5	1	0	5	3	1	1	2	0.122325	19/20	NonOverlappingTemplate
3	3	2	2	2	0	3	0	4	1	0.534146	20/20	NonOverlappingTemplate
4	2	0	2	2	4	1	1	3	1	0.534146	19/20	NonOverlappingTemplate
1	3	0	3	4	1	4	2	2	0	0.350485	20/20	NonOverlappingTemplate
1	2	3	1	1	3	4	2	0	3	0.637119	20/20	NonOverlappingTemplate
3	2	3	2	1	1	1	0	4	3	0.637119	18/20	NonOverlappingTemplate
1	3	1	2	0	6	4	2	1	0	0.066882	20/20	NonOverlappingTemplate

5	1	0	6	2	1	2	1	2	0	0.035174	18/20	NonOverlappingTemplate
1	3	4	3	2	1	1	4	0	1	0.437274	20/20	NonOverlappingTemplate
1	0	1	2	3	4	4	1	2	2	0.534146	20/20	NonOverlappingTemplate
1	2	6	2	2	0	1	0	2	4	0.090936	20/20	NonOverlappingTemplate
3	1	1	1	3	1	2	3	1	4	0.739918	20/20	NonOverlappingTemplate
2	0	5	3	1	1	2	4	2	0	0.213309	20/20	NonOverlappingTemplate
1	3	1	2	2	3	3	1	0	4	0.637119	20/20	NonOverlappingTemplate
0	4	4	0	1	1	4	2	1	3	0.213309	20/20	NonOverlappingTemplate
2	3	4	2	2	3	0	0	3	1	0.534146	20/20	NonOverlappingTemplate
2	1	2	3	1	2	5	1	2	1	0.637119	19/20	NonOverlappingTemplate
0	2	1	5	2	2	3	1	1	3	0.437274	20/20	NonOverlappingTemplate
0	3	2	1	2	3	3	2	2	2	0.911413	20/20	NonOverlappingTemplate
1	2	2	3	3	0	1	2	1	5	0.437274	20/20	NonOverlappingTemplate
7	0	2	1	3	2	2	2	1	0	0.035174	19/20	NonOverlappingTemplate
1	4	3	1	1	2	2	0	3	3	0.637119	19/20	NonOverlappingTemplate
1	1	1	1	3	2	2	2	3	4	0.834308	20/20	NonOverlappingTemplate
5	2	5	0	1	2	2	0	0	3	0.066882	20/20	NonOverlappingTemplate
1	0	2	4	4	1	2	3	2	1	0.534146	20/20	NonOverlappingTemplate
5	5	0	1	2	1	1	0	2	3	0.090936	19/20	NonOverlappingTemplate
4	2	1	5	1	2	0	1	2	2	0.350485	20/20	NonOverlappingTemplate
2	1	3	1	1	2	4	2	2	2	0.911413	19/20	NonOverlappingTemplate
3	1	2	1	1	4	3	3	0	2	0.637119	19/20	NonOverlappingTemplate
1	1	3	1	4	1	1	4	4	0	0.275709	19/20	NonOverlappingTemplate
3	2	1	1	2	3	4	2	2	0	0.739918	20/20	NonOverlappingTemplate
3	3	5	0	1	1	1	2	1	3	0.350485	20/20	NonOverlappingTemplate
3	3	1	1	1	0	3	3	2	3	0.739918	19/20	NonOverlappingTemplate
1	1	1	4	2	3	2	4	2	0	0.534146	20/20	NonOverlappingTemplate
3	2	2	4	2	2	1	2	2	0	0.834308	20/20	NonOverlappingTemplate
1	2	1	2	5	1	0	3	0	5	0.090936	20/20	NonOverlappingTemplate
3	2	4	3	3	2	0	0	1	2	0.534146	20/20	NonOverlappingTemplate
3	2	1	1	2	1	0	3	4	3	0.637119	19/20	NonOverlappingTemplate
2	2	1	1	3	2	3	1	2	3	0.964295	20/20	NonOverlappingTemplate
3	2	2	2	3	1	2	1	2	2	0.991468	20/20	NonOverlappingTemplate
3	2	3	1	6	2	0	1	0	2	0.122325	20/20	NonOverlappingTemplate
5	2	2	3	1	0	0	0	2	5	0.066882	20/20	NonOverlappingTemplate
1	3	1	4	2	3	2	1	2	1	0.834308	20/20	NonOverlappingTemplate
3	1	1	1	4	2	4	0	1	3	0.437274	20/20	NonOverlappingTemplate
2	0	3	4	2	2	3	1	2	1	0.739918	20/20	NonOverlappingTemplate
2	1	4	2	0	0	1	6	3	1	0.066882	20/20	NonOverlappingTemplate
1	1	2	3	4	3	1	1	2	2	0.834308	20/20	NonOverlappingTemplate
5	1	2	2	0	0	4	2	1	3	0.213309	19/20	NonOverlappingTemplate
2	2	0	1	2	5	2	2	1	3	0.534146	19/20	NonOverlappingTemplate
3	3	1	4	0	3	2	3	0	1	0.437274	20/20	NonOverlappingTemplate
3	1	1	3	1	4	4	1	0	2	0.437274	20/20	NonOverlappingTemplate
0	2	3	4	1	1	5	1	2	1	0.275709	20/20	NonOverlappingTemplate
2	2	4	1	2	1	2	4	0	2	0.637119	20/20	NonOverlappingTemplate

1	1	5	0	3	4	2	3	1	0	0.162606	20/20	NonOverlappingTemplate
0	4	3	2	4	0	3	1	2	1	0.350485	20/20	NonOverlappingTemplate
3	4	1	2	2	1	4	2	1	0	0.534146	20/20	NonOverlappingTemplate
5	1	3	1	3	1	3	0	2	1	0.350485	20/20	NonOverlappingTemplate
1	1	2	2	2	3	2	1	2	4	0.911413	20/20	NonOverlappingTemplate
1	2	1	3	1	0	5	2	5	0	0.090936	20/20	NonOverlappingTemplate
3	2	1	2	0	3	2	4	1	2	0.739918	20/20	NonOverlappingTemplate
2	3	2	4	1	0	3	3	2	0	0.534146	19/20	NonOverlappingTemplate
3	1	3	2	2	4	0	2	3	0	0.534146	20/20	NonOverlappingTemplate
2	1	4	1	0	2	4	2	1	3	0.534146	20/20	NonOverlappingTemplate
3	3	3	1	0	1	1	1	3	4	0.534146	19/20	NonOverlappingTemplate
2	4	1	1	2	1	1	3	2	3	0.834308	20/20	NonOverlappingTemplate
2	2	2	5	1	2	3	1	1	1	0.637119	20/20	NonOverlappingTemplate
4	0	0	3	2	1	3	2	4	1	0.350485	20/20	NonOverlappingTemplate
3	3	3	1	3	3	1	0	1	2	0.739918	19/20	NonOverlappingTemplate
2	2	1	2	1	5	1	1	4	1	0.437274	20/20	NonOverlappingTemplate
0	3	4	1	4	2	4	1	0	1	0.213309	20/20	NonOverlappingTemplate
0	2	2	1	3	1	4	2	0	5	0.213309	20/20	NonOverlappingTemplate
3	2	0	0	3	2	3	2	4	1	0.534146	20/20	NonOverlappingTemplate
0	2	2	1	1	3	1	2	5	3	0.437274	20/20	NonOverlappingTemplate
2	5	1	1	4	2	2	1	1	1	0.437274	19/20	NonOverlappingTemplate
2	4	4	1	1	2	0	3	2	1	0.534146	20/20	NonOverlappingTemplate
5	0	6	2	1	1	3	2	0	0	0.017912	19/20	NonOverlappingTemplate
4	1	2	3	2	1	1	3	1	2	0.834308	19/20	NonOverlappingTemplate
4	1	1	0	2	0	2	5	3	2	0.213309	20/20	NonOverlappingTemplate
2	4	1	1	2	5	1	4	0	0	0.122325	20/20	NonOverlappingTemplate
3	2	2	1	0	1	1	4	3	3	0.637119	20/20	NonOverlappingTemplate
4	1	3	1	1	1	2	0	4	3	0.437274	20/20	NonOverlappingTemplate
1	3	3	3	1	1	4	1	1	2	0.739918	20/20	NonOverlappingTemplate
3	3	3	2	1	0	4	1	3	0	0.437274	20/20	NonOverlappingTemplate
4	1	3	2	0	2	1	2	3	2	0.739918	19/20	NonOverlappingTemplate
0	3	4	2	4	0	5	2	0	0	0.048716	20/20	NonOverlappingTemplate
4	1	2	3	1	0	1	5	2	1	0.275709	20/20	NonOverlappingTemplate
1	2	4	3	3	1	1	1	2	2	0.834308	20/20	NonOverlappingTemplate
0	3	2	3	2	3	1	2	1	3	0.834308	20/20	NonOverlappingTemplate
2	1	1	4	4	1	4	0	1	2	0.350485	20/20	NonOverlappingTemplate
1	2	2	1	3	0	5	2	1	3	0.437274	20/20	NonOverlappingTemplate
3	0	1	2	3	2	0	2	3	4	0.534146	20/20	NonOverlappingTemplate
2	1	2	2	1	0	3	2	3	4	0.739918	19/20	NonOverlappingTemplate
3	3	1	1	3	0	0	2	5	2	0.275709	20/20	NonOverlappingTemplate
1	4	1	3	2	2	0	2	3	2	0.739918	20/20	NonOverlappingTemplate
2	3	1	1	0	2	0	4	4	3	0.350485	20/20	NonOverlappingTemplate
3	2	2	2	1	1	2	1	3	3	0.964295	20/20	NonOverlappingTemplate
2	2	1	2	1	2	2	2	4	2	0.964295	20/20	NonOverlappingTemplate
6	0	1	2	1	1	1	2	2	4	0.122325	20/20	NonOverlappingTemplate
1	1	2	2	2	3	4	2	1	2	0.911413	20/20	NonOverlappingTemplate

5	5	0	1	2	1	1	0	2	3	0.090936	19/20	NonOverlappingTemplate
3	3	1	4	5	0	0	2	1	1	0.162606	19/20	OverlappingTemplate
3	0	4	3	0	3	3	1	1	2	0.437274	20/20	Universal
5	4	3	1	1	0	2	2	1	1	0.275709	20/20	ApproximateEntropy
1	0	0	2	2	5	3	1	1	2	0.002971	17/17	RandomExcursions
3	2	0	2	1	0	2	2	2	3	0.090936	17/17	RandomExcursions
3	2	2	1	3	0	3	0	1	2	0.048716	17/17	RandomExcursions
3	1	0	2	3	2	2	1	0	3	0.048716	17/17	RandomExcursions
1	0	3	3	1	0	1	2	3	3	0.025193	17/17	RandomExcursions
2	1	1	1	1	4	0	3	3	1	0.025193	17/17	RandomExcursions
3	1	2	1	2	0	5	1	1	1	0.006196	17/17	RandomExcursions
3	2	2	2	1	1	1	1	3	1	0.275709	17/17	RandomExcursions
2	1	1	1	0	3	5	2	0	2	0.002971	17/17	RandomExcursionsVariant
2	0	1	1	2	2	5	0	4	0	0.000296	17/17	RandomExcursionsVariant
2	0	2	1	2	4	2	1	2	1	0.090936	17/17	RandomExcursionsVariant
2	2	2	2	2	0	2	2	1	2	0.437274	17/17	RandomExcursionsVariant
1	3	1	4	1	1	2	2	1	1	0.090936	17/17	RandomExcursionsVariant
0	3	2	0	1	2	3	4	1	1	0.012650	17/17	RandomExcursionsVariant
1	1	1	2	1	1	5	1	2	2	0.025193	17/17	RandomExcursionsVariant
1	1	2	1	3	0	3	4	1	1	0.025193	17/17	RandomExcursionsVariant
1	1	0	2	1	1	3	4	1	3	0.025193	17/17	RandomExcursionsVariant
0	2	3	1	3	1	4	1	2	0	0.012650	17/17	RandomExcursionsVariant
1	1	2	2	2	3	0	3	1	2	0.162606	17/17	RandomExcursionsVariant
3	2	0	3	1	2	1	1	3	1	0.090936	17/17	RandomExcursionsVariant
2	2	4	2	3	0	0	2	1	1	0.025193	17/17	RandomExcursionsVariant
2	1	6	2	1	3	1	0	0	1	0.000134	16/17	RandomExcursionsVariant
1	4	4	0	2	1	2	1	2	0	0.006196	16/17	RandomExcursionsVariant
1	4	5	0	1	1	2	1	0	2	0.000648	17/17	RandomExcursionsVariant
1	3	3	4	0	0	3	2	1	0	0.002971	17/17	RandomExcursionsVariant
0	3	3	2	1	0	3	0	4	1	0.002971	17/17	RandomExcursionsVariant
3	2	1	1	3	4	1	2	2	1	0.834308	19/20	Serial
3	1	2	4	0	2	5	2	1	0	0.213309	19/20	Serial
2	1	1	5	2	3	2	2	1	1	0.637119	20/20	LinearComplexity

- -

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 18 for a sample size = 20 binary sequences.

The minimum pass rate for the random excursion (variant) test is approximately = 15 for a sample size = 17 binary sequences.

For further guidelines construct a probability table using the MAPLE program provided in the addendum section of the documentation.

- -