USING PRIMITIVE PYTHAGOREAN TRIPLES AND THE BLOM'S SCHEME IN THE 4-WAY HANDSHAKE WIRELESS SECURITY PROTOCOL

By

ANTONY AKSHAY

Bachelor of Science in Computer Science

Oral Roberts University

Tulsa, Oklahoma

2008

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2014

USING PRIMITIVE PYTHAGOREAN TRIPLES AND THE BLOM'S SCHEME IN THE 4-WAY HANDSHAKE WIRELESS SECURITY PROTOCOL

Thesis Approved:

Thesis Adviser

Dr. Subhash Kak

Dr. David Cline

Dr. Michel Toulouse

Name: ANTONY AKSHAY

Date of Degree: MAY 2014

Title of Study: USING PRIMITIVE PYTHAGOREAN TRIPLES AND THE BLOM'S SCHEME IN THE 4-WAY HANDSHAKE WIRELESS SECURITY PROTOCOL

Major Field: COMPUTER SCIENCE

Abstract: The current standards for wireless security are WPA and its revised version WPA2 (IEEE 802.11i). At the basis of both of these is the WEP protocol that has been broken and automated software can crack it in under a minute. In order to put wireless security on a strong theoretical footing, this thesis proposes a novel way of using Pythagorean triples along with Blom's scheme to perform raw key exchange and authentication by using a 2 stage process to do the 4-way handshake similar to the one described in IEEE 802.11i. Primitive Pythagorean Triples (PPT's) are infinite and they display randomness that makes them good candidates for cryptographic key. We analyze the cryptographic strength of random keys generated by Primitive Pythagorean Triples and determine whether or not they can be used for wireless authentication and as raw keys for encryption in wireless security.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

PURPOSE AND REQUIREMENTS OF WIRELESS SECURITY

Wireless networks are increasingly used to connect to the internet for social and business applications. Hence establishing a secure connection is an important issue. Unlike physical wires where the signals travel though metal to transfer information and cannot be accessed unless the physical wire is broken into, wireless signals travel through air and these signals are carrying information packets that can be read or changed by anyone who can access them. These packets contain information on what is being accessed by the users which could be harmless information like just surfing the web or movie segments, or it could be something that is sensitive like credit card information. Since these packets are traveling through the air and can be picked up by anyone with wireless access, it becomes a security risk.

Wireless security comprises mainly of two components, encryption and authentication. Authentication is the process of determining a person or thing is actually who or it claims to be, whereas encryption is the process of convert text into cipher text, cipher text can only be read by authorized entities. There is a need to authenticate to who the wireless network connects and to encrypt the information being transferred so that prying eyes may not have access to it. Wireless security is provided by WEP (Wired Equivalent Privacy), WPA (Wi-Fi Protected Access), WPA2 (Wi-Fi Protected Access 2), with WPA2 being the latest. WPA2 is the currently accepted standard for wireless security. These technologies work by first authenticating a connection and then encrypting the information passed between the authenticated sender and authenticated receiver.

This thesis proposes a way to exchange raw keys and to achieve authentication using PPT’s. The authentication model in which we use PPT’s is the 4-way handshake which is also used by the WPA2 for authentication. We want to use PPT’s because there are infinite many of them and they have excellent randomness properties. We will investigate the conditions under which their use will potentially strengthen the WPA2 protocol.

ENCRYPTION AND ITS PURPOSE

Encryption is the process of encoding messages (or information) in such a way that third parties cannot read it, but only authorized parties can, Encryption doesn't prevent hacking but it prevents the hacker from reading the data that is encrypted.[17] Human readable information is called plaintext and scrambled information is called cipher text. We can convert plain text to cipher text using a key and an encryption scheme; similarly this cipher text can be recovered by using a key to decipher the cipher text to plain text. The process of converting plain text to cipher text is called encryption, and the study of converting plain text to cipher text is called cryptography. The process of converting cipher text to plain text is called decryption, and the study of breaking cipher text to reveal the plain text is called cryptanalysis. The driving force behind cryptography is algorithms, whereas the driving force of cryptanalysis is exploiting weakness in the cryptographic algorithms and exploiting incorrect implementation of the cryptographic algorithms.

An example of cryptanalysis is monitoring timing measurements of a CPU while running the cryptographic algorithm; this can help in determining the size of the secret key, this type of attack falls under the category of timings attacks and there are various other techniques of timing attacks. As to why we need encryption is simple; to keep information a secret. Today encryption is everywhere from login into a website to the protection of state secrets.

4-WAY HANDSHAKE

The 4-way handshake is both a key management and authentication protocol used in WPA2 described by the IEEE 802.11i. Some notations used in the 802.11i protocol are listed below.

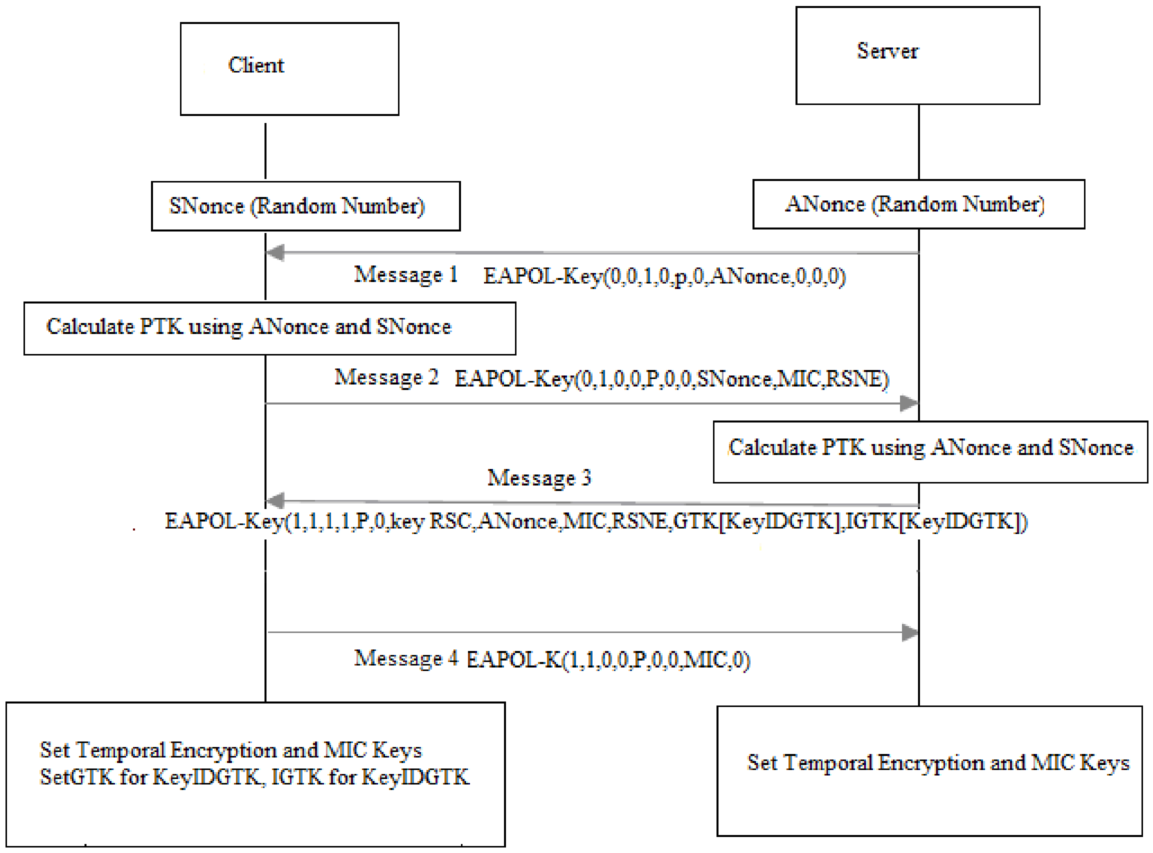

Client
Server
SNonce (Random Number)
ANonce (Random Number)
Message 1 EAPOL-Key(0,0,1,0,p,0,ANonce,0,0,0)
Calculate PTK using ANonce and SNonce
Message 2 EAPOL-Key(0,1,0,0,P,0,0,SNonce,MIC,RSNE)
Calculate PTK using ANonce and SNonce
Message 3
EAPOL-Key(1,1,1,1,P,0,key RSC,ANonce,MIC,RSNE,GTK[KeyIDGTK],IGTK[KeyIDGTK])
Message 4 EAPOL-K(1,1,0,0,P,0,0,MIC,0)
Set Temporal Encryption and MIC Keys
SetGTK for KeyIDGTK, IGTK for KeyIDGTK
Set Temporal Encryption and MIC Keys

Figure 1: 4-way Handshake as implemented in WPA2.

The formula to generate Pairwise Transient Key is:

PTK = PRF (PMK + ANonce + SNonce + AA + SPA), where

PRF - pseudo random function,

PMK - Pairwise Master Key,

ANonce – This is a random number know only to the server

SNonce –.This is a random number know only to the client.

AA – Server Mac Address (access point)

SPA – Client Mac Address (wireless client)

During the 4-way handshake there are four messages exchanged between the parties involved. Each message has a purpose that will be described below, which is a breakdown of the figure 9.

Message 1

This is sent from the server to the client, the message sent contains the ANonce along with the authenticator's MAC address. [13]

Message 2

The client creates its nonce, called SNonce. The client can now calculate the PTK. The client sends the SNonce to the authenticator. The client also sends the security parameters (RSN) information. The entire message gets an authentication check using the (KCK/MIC) from the pairwise key hierarchy. [13]

Message 3

The server derives the GTK key from the GMK key, an ANonce, RSN information element info, and a MIC. This information is then sent to the client in an EAPOL-Key frame. This is kept secret by encrypting it with the PTK.

Message 4

This message is a confirmation message it notifies the server that the temporal keys are installed on the client.

PROPOSED 4-WAY HANDSHAKE.

This thesis uses a novel idea of using PPT's and Blom's scheme in raw key exchange and authentication. Using the indexing property of PPT's (explained under literature review) we can generate infinite many string sequence of infinite length. Hence we can use these string sequences for authentication and also as keys (but not the same sequence.) We need a way to authenticate these string sequence and for that we use the Blom's scheme (explained under literature review) to exchange a secret number between client and server. Using this secret number the client and server can authenticate to each other and also agree upon a key. The toy example explained below will give us a better picture.

Example:

1) Using Blom's scheme the public matrix (S`) is sent from server to client. While at the same time the server sends the client a string "stringA" of size n. This string is part of a massive PPT's table.

2) The client upon receiving S` does a multiplication with its own private matrix to get the secret number x. The client now knows where to looks for x^{th} occurrence of "stringA" from which it can find "stringB" (it is also on length n). The client has now complete "string1" (string1 = stringAstringB and it is of size 2n), because they have the correct starting location of "string1". The client can respond with the string1 which also turns out to be the starting location of the raw key.

3) Upon receiving "string1" from the client, this if wrong will be ignored. This is part of the authentication process, because an incorrect string means that the server received the message from the wrong person. If it is correct then the server will pick a new string "stringC" of size n and transmit it to the client. This "stringC" represents the end location of the raw key.

4) Now that the client has “stringC”, it can respond with “string2” (string2 = stringCstringD and it is of size 2n). Since “string2” marked the end location of the raw key the client now too has the raw key.

Example: Suppose x = 4 (i.e. the 4th occurrence), if stringA = ABD, then stringB = DFE and string1 = ABDDFE. If stringC = CCC, then stringD = DFD and string2 = DFD.

The correct raw key:

EDADFECBDFEADEADCDFECBDFEADDEADCDEADFCCDFADCBDEADCEFD

CADEADFCDA.

Few other possibilities are (in these cases, incorrect):

string1 = ABDADE, string2 = CCCDEA

Raw key:

DEADEDADEADFBBFBFDEDEFADEABDFFDEADFBBBB

string1 = ABDFFB, string2 = CCCDEA

Raw key:

DEADFBBBB

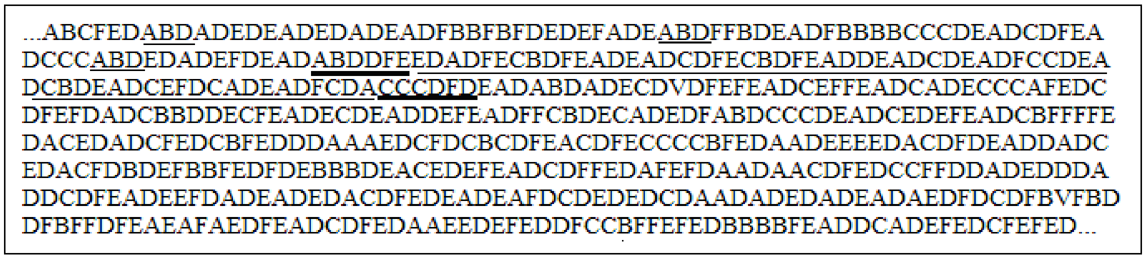

...ABCFEDABDADEDEADEDADEADFBBFBFDEDEFADEABDFFBDEADFBBBBCCCDEADCDFEA
DCCCABDEDADEFDEADABDDFEEDADFECBDFEADEADCDFECBDFEADDEADCDEADFCCDEA
DCBDEADCEFDCADEADFCDACCCDFDEADABDADECDVDFEFEADCEFFEADCADECCCAFEDC
DFEFDADCBBDDECFEADECDEADDEFEADFFCBDECADEDFABDCCCDEADCEDEFEADCBFFFFE
DACEDADCFEDCBFEDDDAAAEDCFDCBCDFEACDFECCCCBFEDAADEEEEDACDFDEADDADC
EDACFDBDEFBBFEDFDEBBBDEACEDEFEADCDFFEDAFEFDAADAACDFEDCCFFDDADEDDDA
DDCDFEADEEFDADEADEDACDFEDEADEAFDCDEDEDCDAADADEDADEADAEDFDCDFBVFBD
DFBFFDFEAEAFAEDFEADCDFEDAAEEDEFEDDFCCBFFEFEDBBBBFEADDCADEFEDCFEFED...

Figure 2: An indexed PPT’s table

CHAPTER II

WIRED EQUIVALENT PRIVACY (WEP.)

Wired Equivalent Privacy (WEP) was a security algorithm for wireless networks that was established by the IEEE in 1997. The IEEE document that describes this algorithm is 802.11-1997. This algorithm was found to be unsecured and has been superseded by WPA and WPA2. WEP has two stages, an authentication stage in which parties involved were authenticated and key where exchanged, the second step was transmission of data that was encrypted using the exchanged key. WEP natively supported two authentication mechanisms, shared-key authentication and open authentication. Both of these mechanisms were weak. In a shared-key authentication the access point (server) and the client would use a four way handshake. [5][6][7]

- The client sends an authentication request to the access point**.**
- The access point sends the client a pseudo-random number (typically referred to as a nonce value) or a plain text challenge.
- The client encrypts the nonce value using the WEP key and sends it back to the access point.
- The access point encrypts the same nonce value with the WEP key and compares it to what the client sent. If the values match, the client has the correct WEP key and the access point acknowledges the authentication attempt.

The WEP key was composed of numbers 0-9 and alphabets A-Z; these were very small set symbols. The key then was created using these symbols and the length of the key was based on the type of WEP encryption.

Bit size	Number of Digits	IV Size
64	10	24
128	26	24
256	58	24

Table 1: Key size and Number of Digits based on bit size in WEP

IV is the initiation vector, it is a 24-bit field sent in the clear text portion of a message. This 24-bit string was used to initialize the key stream generated by the RC4 algorithm, Reuse of the same IV produces identical key streams for the protection of data, and because the IV is short, it guaranteed repetitions, which lead to the key being found out, if an attack was performed on WEP.

WEP is considered unsecured for a few reasons such as the key management is not specified in the WEP standard and, therefore, a weaknesses because without interoperable key management, keys will tend to be long-lived (re-used a lot) and of poor quality (not random enough or too short.) RC4 in its implementation in WEP has been found to be weak, because the first three bytes of the key are taken from the IV that is sent unencrypted in each packet.

The authentication process itself is a weakness. By using the authentication, actually reduce the total security of the network and make it easier for the attacker to guess the WEP key, because if the WEP is using shared key authentication, then the attacker can monitoring the challenge and the encrypted response. Using this information the attacker can determine the RC4 stream used to encrypt the response, and use that stream to encrypt any challenge received in the future. So by monitoring a successful authentication, the attacker could later forge an authentication. [17][18]

One of the attacks performed on WEP is the Chop-Chop attack. In the Chop-Chop attack the last byte of a packet is chopped off and a guess is made at the plain text value of the byte, and a correction is made to the Integrity Check Value (ICV). This exploits the message modification vulnerability in WEP. If the guess for the chopped byte is correct then the packet will be a valid WEP packet and will be accepted by the server. If it is invalid then the packet will be discarded. In this approach one byte is found at a time, until the entire packet is reconstructed. When this process is complete, the attacker has a decrypted data packet from which they can extract the IP address of the network. Now they can inject malicious packets into the network from which they can extract the IV value of the key. [19]

WI-FI PROTECTED ACCESS (WPA/WPA2)

WPA (Wi-Fi Protected Access) was the next step in wireless security once WEP's unreliability became well known and documented. WPA2 is the next installment of WPA, and covers some of the weakness in WPA. WPA was implemented in 2003 as an intermediate solution when the weakness of WEP became unreliable. WPA2 was released in 2004. The critical changes made in WPA were; the change in key while WEP used a static 40 or 104 bit key WPA uses Temporal Key Integrity Protocol (TKIP) [11]. This is 128 bit key, also each packet had its own key, and this rendered the attack method used against WEP useless, which is collecting data to find the key. Another critical change was the removal of cyclic redundancy check (CRC), which was used for integrity checks.

CRC is a good algorithm to check for error caused by noise during transmission; it is easy to implement and is light weight. Unfortunately it is not good enough for cryptography, since it cannot detect tampering. CRC was replace with a message integrity check algorithm called Michael that was stronger than CRC, but Michael was not strong enough and was replace in WPA2, making WPA2 unlike WEP and WPA is a lot more process intensive. There where issue with WPA, since TKIP used the underlying mechanisms of WEP, it inherited some of the vulnerabilities of WEP. [19]

WPA is vulnerable to Beck-Tews attack which is an extension of the chop-chop attack that worked on WEP. Beck-Tews attack waits 60 second when a server rejects a modified packet. The reason for the wait is because in WPA when the server receives two incorrect packets within a time frame of 60 second; the server will implement counter measure by changing the key used.. By waiting 60 seconds the attack can bypass this counter measure. Ohigashi-Morii attack is an extension of Beck-Tews attack; while the Beck-Tews attack take about 15 minutes to work Ohigashi-Morii attack works in under a minute, The Ohigashi-Morii is a combination of man-in-the-middle attack combined with Beck-Tews attack. WPA2 was an improvement on WPA. WPA2 was established in 2004. Critical changes include replacing Counter Cipher Mode with Block Chaining Message Authentication Code Protocol (CCMP) a new Advance Encryption Standard (AES) based encryption instead of TKIP. WPA2 uses the 4-way handshake to accomplish two purposes, the access point (AP) authenticate itself to the client station (STA), and determining the key for encryption. [7]

IMPORTANCE OF KEYS IN DIFFERENT TYPES OF ENCRYPTION.

There are 3 methods of encryption symmetric, asymmetric, and hashing. There are many types of encryption schemes such as substitution, transposition, poly-alphabetic, one time pad etc. Below we will discuss how a key, encryption and the cipher text interacted with each other for different schemes.

In a substitution cipher the plain text units was replaced with cipher text units. A unit could be a single character or a group of character. To decipher the message the inverse substitution of the unit was applied to the cipher text to obtain the plain text. The substitution cipher itself had many variations like single substitution, poly-graphic, mono-alphabetic, and poly-alphabetic [9]. The Caesar and Atbash code are both example of the substitution cipher. For example consider a passage with 6000 alphabets plotted on a graph, since the English language is not random, some alphabet would appear more times than others. This lack of randomness can be seen in figure 3.

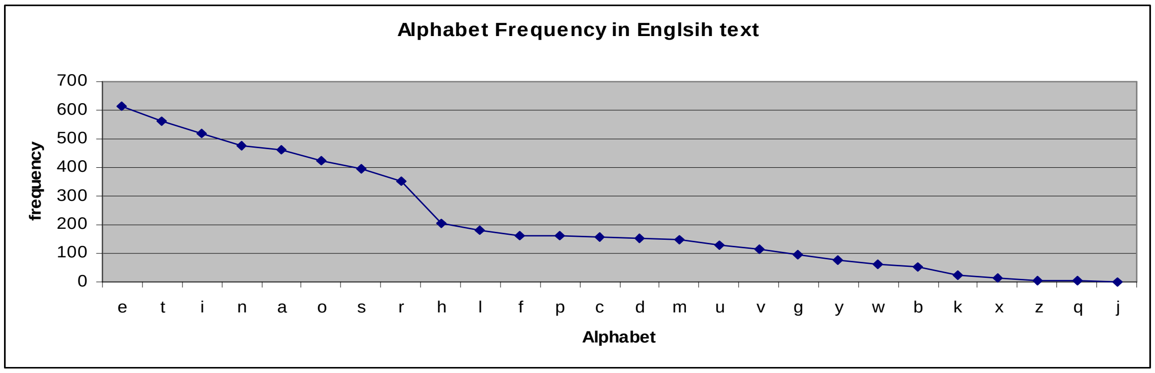


Figure 3: Alphabet frequency in a random English text.

Caesar cipher can be show mathematically for encryption as:

$E_n(x) = (x + n) \bmod 26$

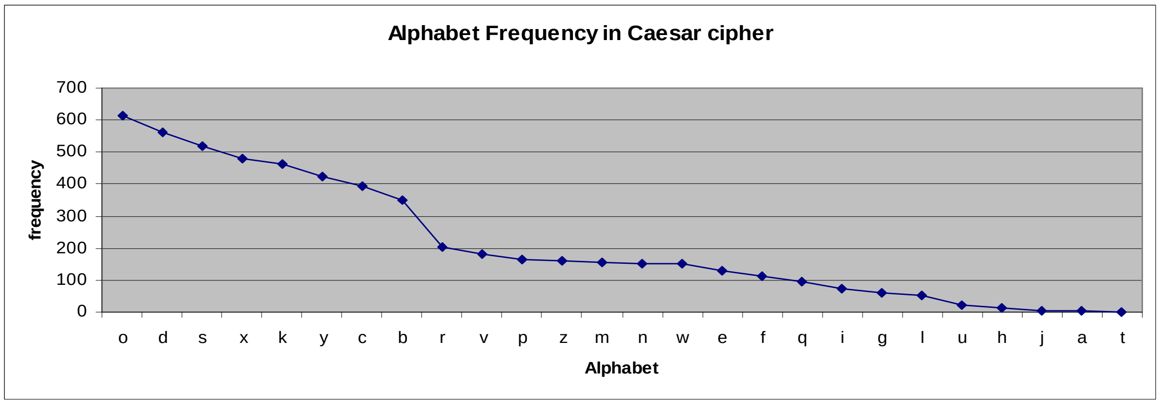


Figure 4: Alphabet frequency in a cipher text encrypted using Caesar cipher.

And for decryption as

$D_n(x) = (x – n) \bmod 26$

where x, n are between 0 and 25. If a passage of about 6000 alphabets in English was taken and the Caesar's cipher applied to the sample passage with a shift of 10. It will yield a graph as shown below.

A couple of observations can be made from the graph; the most obvious one is that the distribution is not random. If one was to compare figure 4 to figure 3, one would observe that the graph identical with only the alphabet label changed, the frequency still remains the same. For example ‘e’ in figure 3 is has the same frequency as ‘o’ in figure 4. This is a weakness in the ciphers itself; since the cipher just masks the text in a form that reader cannot comprehend. The cipher text still retains the property of plain text; that is lack of randomness displayed by the plain text; some alphabets appear more often than others. Two of the most occurring alphabets in the English language are E and T. So if someone wanted to crack this cipher text and they observe that some alphabets occur more often than others, it would be a safe bet for them to guess those high occurring alphabets are either E or T.

The lack of randomness is a weakness that can be exploited. Randomness removes patterns and makes it harder to guess. Another weakness is the spacing of words. It is easy to guess that three letter words might be ‘and’, or ‘the’, or ‘but’. Stream and block ciphers remove this weakness of word size giving away the word by either converting the text to one stream or of fixed block sizes

Atbash cipher is another substitution cipher. It replaces the first alphabet with the last alphabet, the second alphabet with the second to last alphabet. Mathematically the Atbash cipher can be represented by the following formula:

$$E(x) = D(x) = ((m – 1)x + (m - 1) \bmod m$$

Both the encryption and decryption use the same formula. If the sample passage used for the Caesar cipher is encrypted with the Atbash cipher. We would get a totally new cipher text.

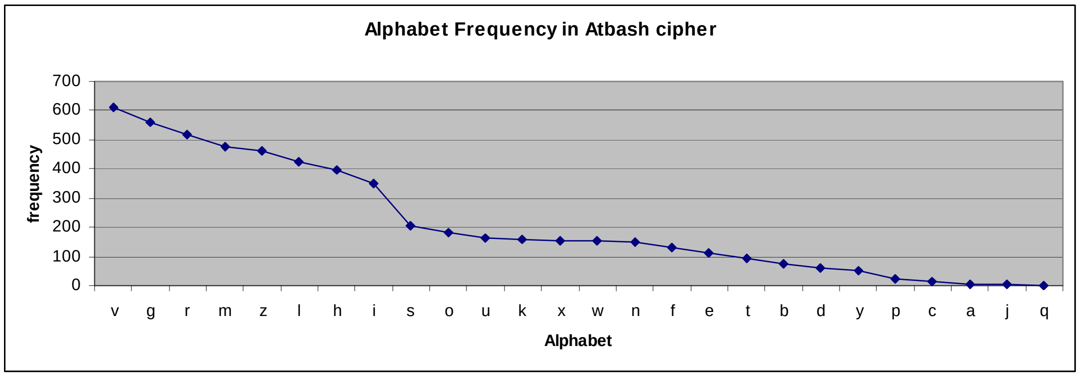


Figure 5: Alphabet frequency in a cipher text encrypted using Atbash cipher.

The observation can be made that certain alphabets yet once again appear more often that other and the plotted graph is the same. In other words the text is still not random and higher occurring alphabets can be guessed to be E or T. This cipher scheme suffers from not making the text random; this is due to the fact that key use to encrypt the text is not random. The Atbash cipher and the Caesar cipher are special cases of the affine cipher. They inherit the weakness of affine cipher, that is; they break Kerckhoffs Principle with state that “A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.” instead they use obscurity for security. And it takes only a simultaneous equation to encrypt and decrypt since the keys used are not random.

The Vigenere cipher is a poly-alphabetic substitution cipher [9][10]. The encryption is done by applying a series of different Caesar ciphers. It was considered uncrack able for three hundred years. Mathematically the encryption can be representing by the formula:

$$C_i = E_k (M_i) = (M_i + K_i) \bmod 26$$

And the decryption is represented by the formula:

$$M_i = D_k(C_i) = (C_i – K_i) \bmod 26$$

Where K is an alphabet of the key, and M is the alphabet of the plain text and C is the alphabet of the encrypted text.

For example suppose we wanted to encrypt "hello world" with the key "lemon" we would get "sixzb hsdzq". In other words,

> h=7, l=11.
>
> [7+11 mod 26]=18.
>
> s=18.

The number 18 is s on the alphabet table if a starts at 0. Similarly to decode this message, we have:

> i=8, e=4.
>
> [8-4 mod 26]=4.
>
> e=4.

Since 4 is e on the alphabet table and so "he" would be "si" when using the key word lemon.

One of the more famous methods used to crack Vigenere Cipher is the Kasiski examination. The test takes advantage that there might be a slight chance that some alphabets might be encrypted using the same key alphabets. From these repetitions some of the key alphabets could be guess by trial and error. Once part of the key was guess it could be cracked by statistical methods like frequency analysis.

The weakness of the Vigenere cipher was its key length, once the key length was guess the cipher, just becomes a very complicated Caesar cipher. As with our previous 6000 alphabet sample text, when encrypted with the keyword "lemon" and "honey" and then plotting on a graph based on the frequency of alphabets occurring. Even though the graphs are not identical, they are similar enough; but they look very different from figure 3, 4, 5.

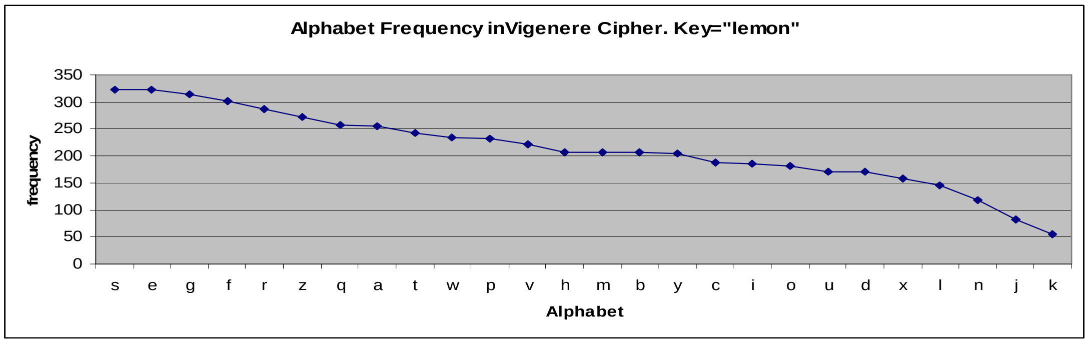


Figure 6: Alphabet frequency in a cipher text using Vigenere cipher using “lemon” as key.

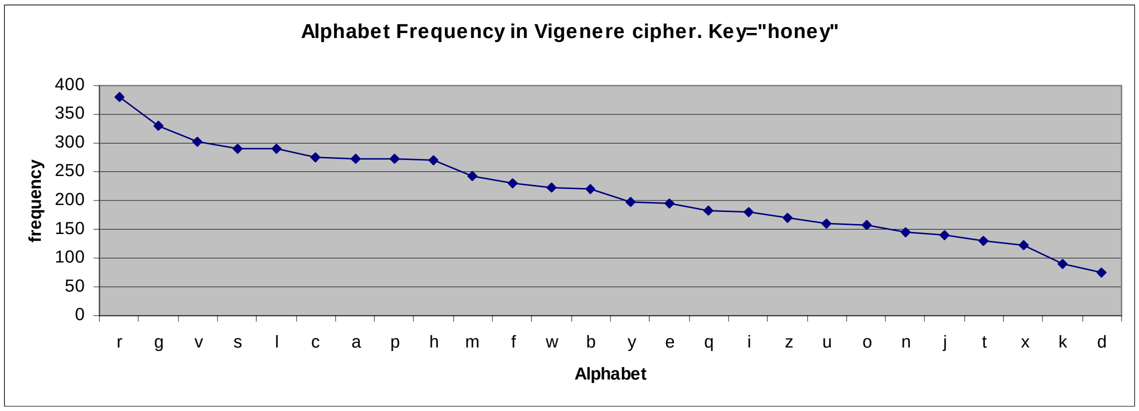


Figure 7: Alphabet frequency in a cipher text using Vigenere cipher using “honey” as key.

In fact figure 3, 4, 5 are identical; since the only thing that happened in the graphs was a substitution of one alphabet for another. Whereas in figure 6, 7 while there was substitution, the substitution kept changing based on the alphabet key being used, but still there is repetitions of the substitution, that is the reason for the similarities between figure 6 and figure 7. When looking at figure 6 and figure 7 higher frequency alphabets can be guess to be E or T, from which the key can be guesses.

When the key is random and is exactly the same size as the message that is to be encrypted, we get the one time pad encryption scheme. The one time pad has been mathematically proven to be un-crack able. But unfortunately for day to day practical application it is near impossible to implement due to its very rigid requirement. Should any of the requirements for a one time pad be compromised, the encryption is no longer secure and could be possibly cracked. For the one time pad to work as intended these are all the properties that need to hold true:

- A perfectly random one time pad (key) is need. This is a non-trivial software issue.
- The keys need to be as long as the plain text message. The key can be longer that the plain text, the excess key will be omitted.
- The key remains to be a secret, once use the key can never be reused and must be destroyed.

Practical application of the one-time pad is small, due to the third requirement, which conflict with human requirements like convenience and re usability.

When a text is encrypted with a one-time pad with correct implementation, should the cipher text be cracked it can never be guaranteed that the cracked text was the messages. A simple example to demonstrate this is using the message “hello” and the key “xmckl” would generate the cipher text “eqnvz”. Now if “eqnvz” was to be cracked using the key “xmckl” we would get “hello” back, but the key “ tquri” produce the plain text “later”. Should someone crack the cipher they will have no way of telling which one of the two generated plain text is actually the correct one.

Plain text	H	E	L	L	O	
	7	4	77	77	14	
						KEY:
+	23	12	2	10	11	XMCKL
=	30	16	13	21	25	
MOD(26)	4	16	13	21	25	
Cipher text	E	Q	N	V	Z	
Cipher text	E	Q	N	V	Z	
	4	16	13	21	25	

						KEY:
-	23	12	2	10	11	XMCKL
=	19	4	11	11	14	
MOD(26)	7	4	11	11	14	
Cipher text	H	E	L	L	O	
Cipher text	E	Q	N	V	Z	
	4	16	13	21	25	
						KEY:
-	19	16	20	17	8	TQURI
=	-15	0	-7	4	17	
MOD(26)	11	0	19	4	17	
Plain text	L	A	T	E	R	

Table 2: Encryption and decryption process of the one-time pad using different keys.

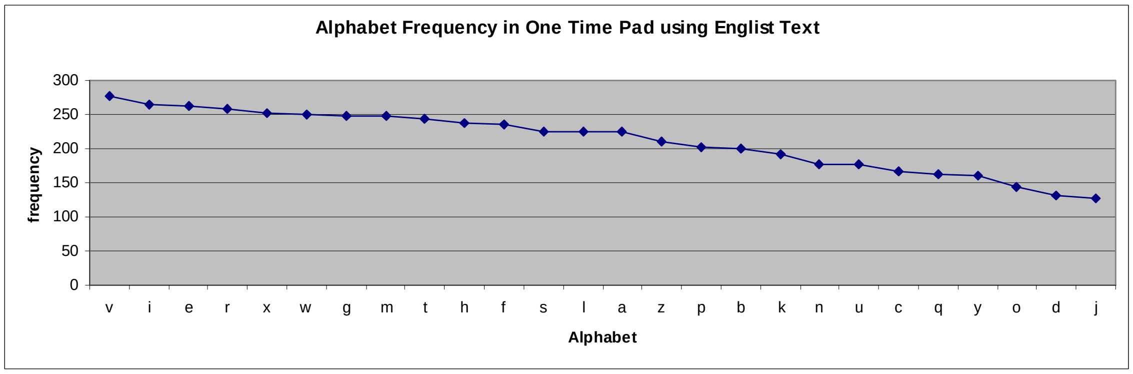


Figure 8: Alphabet frequency in a cipher text using one time pad with random English text as key.

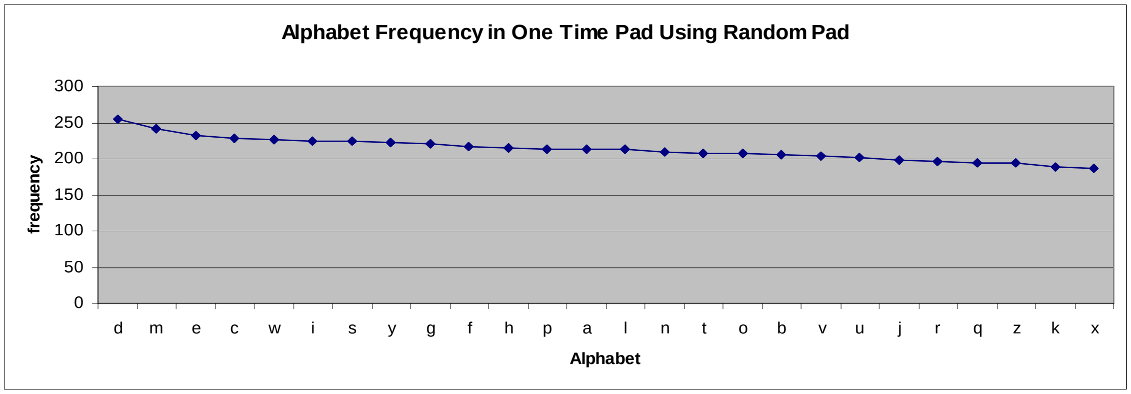


Figure 9: Alphabet frequency in a cipher text using one time pad with random pad as key.

Using the sample 6000 alphabet passage and applying the one time pad to it using an pseudo random strings of equal length as the passage (figure 9) and English text (figure 8). Observer the distribution, using English text as the pad causes a rapid fall in alphabet frequency as opposed to using assumed random strings. The pad is pseudorandom because the string used was using a pseudorandom generator, probably a truly random string would give a near straight line in the graph.

GENERATION OF PRIMITIVE PYTHAGOREAN TRIPLES (PPTs).

PPT's are three positive integers a, b, c; such that $a^2 + b^2 = c^2$, where c is the hypotenuses of a right angle triangle and a, b being the other 2 sides. They derive their name from the Pythagorean Theorem. PPT's are often written as (a, b, c). A primitive PPT's is one in which a, b, c are co-primes to each other. The existence of PPT's far back as 800 B.C. Many ancient texts around the world show the awareness of the strange property display by right angle triangle that is now called the Pythagorean Theorem. This theorem is one of the most widely known theorem in mathematics, with over 75 different proofs. The first documented proof is credited to the Greek mathematician Pythagoras. PPT's comes in two forms of a, b, c; where $a^2 + b^2 = c^2$ (primitive PPT's) and $na^2+nb^2=nc^2$ (n is the scaling factor.), It is the second form from which many of the practical application derive their usefulness in the fields of physics, computer science and social networks.

For example any involvement with a squared number in physics, when dealing with kinetic energy, say we have two energy source that can accelerate a bullets to 300mph and 400mph respectively, we can find out how fast a bullet will be when exposed to the both energy source simultaneously. The answer being 500mph, since the formula for this calculation involves square numbers. Simple application of Pythagorean Theorem provided the solution. PPT's have some interesting proprieties.

- Exactly one of a, b is odd, c is odd.
- Exactly one of a, b is divisible by 3 Exactly one of a, b is divisible by 4
- Exactly one of a, b, c is divisible by 5
- At most one of a, b is a square.
- The hypotenuse, c, is an odd number.
- Every integer greater than 2 is part of a PPT's.
- For any PPT's, the product of the two non-hypotenuse legs is always divisible by 12, and the product of all three sides is divisible by 60.
- There exist infinitely many PPT's whose hypotenuses are squares of natural numbers.
- There exist infinitely many PPT's of which one of the arms is the square of a natural number.
- For each natural number n, there exist n Pythagorean triples with different hypotenuses and the same area.
- For each natural number n, there exist at least n different Pythagorean triples with the same arm a, where: 'a' is some natural number.
- For each natural number n, there exist at least n different triangles with the same hypotenuse.
- In every Pythagorean triple, the radius of the in-circle and the radii of the three ex-circles are natural numbers.

- There is no Pythagorean Triple of which the hypotenuse and one arm are the arms of another Pythagorean Triple.

Since generating key for cryptographic systems is hard, due to the requirement of keys being long and completely random, to make guessing impossible. PPT's are can be good candidates for generating cryptographic keys, but before they can be used as key generators it is necessary that they are easy to generate, be infinite, and random. There are many ways to generate Pythagorean Triples; there is the Euclid's method, Fibonacci method, Dickson's method and many more. Some of these methods produce a subset of all Pythagorean Triples, whereas others produce a mixture of Pythagorean Triples and Primitive PPT's and some produce all Pythagorean Triple.

<u>Euclid's method for generating Pythagorean triples</u> [3]

To generate a, b, c using Euclid's method, the formula is give below.

$a = m^2 - n^2$, $b = 2mn$, $c = m^2 + n^2$.

Where: A seed number is used which is decomposed into m, n; such that m, n are positive integers, and $m > n$. Also if m, n are co-primes (i.e. GCD (m, n) = 1) and $m - n$ is odd then the generated triples are primitive.

Example:

$m = 4, n = 3$

$a = 16 - 9 \Rightarrow 5$

$b = 2*4*3 \Rightarrow 24$

$c = 16 + 9 \Rightarrow 25$

(a, b, c) = (5, 24, 25)

<u>Dickson's method for generating Pythagorean triples</u>

To find the solution to $x^2 + y^2 = z^2$.

Let r, s be to positive integers such that $r^2 = 2st$ is a square. Then:

$x = r + s$, $y = r + t$, $z = r + s + t$.

When r, s are co-primes (i.e. GCD (m, n) = 1) the triple will be primitive. For example:

Let r = 4

So $r^2/2 = 8$

Factors of 8 are 1, 2, 4, 8. Hence (1, 8) (2, 4)

When

s = 1, t = 8. x = 4 + 1 => 5; y = 4 + 8 => 12; z = 4 + 1 + 8 => 13

s = 2, t = 4. x = 4 + 2 => 6; y = 4 + 4 => 8; z = 4 + 2 + 4 => 10

Dickson's method generates all Pythagorean triples.

<u>Fibonacci method for generating Pythagorean triples</u>

For generating primitive triples use the sequence of consecutive odd integers 1, 3, 5, 7 … and the fact that the sum of the first n terms of this sequence is n^2. Fibonacci method does not give all Pythagorean triples.

If *k* is the nth member of the sequence then:

n = (K +1)/2.

<u>Example</u>: Let k = 49 = $7^2 = a^2$.

Then n = (49 + 1) / 2 => 25; the sum of the previous 24 terms is $b^2 = 24^2$, and $c^2 = 25^2$.

The PPT is (7, 24, 25).

Now that we have seen some method to generate Pythagorean triples, let us discuss how PPT's will be used in the key generation. We need keys to be long and we will need many PPT's, in fact there are infinite many PPT's and this has been proved numerous times by many mathematicians over the centuries. The intuitive proof is that if $a^2+b^2=c^2$, then it can factored up on *n*, where *n* is a positive integer, hence infinite. Thus $na^2+nb^2=nc^2$, which turns out to be a Pythagorean triple. A more mathematical proof would be Euclid's proof on infinite many Pythagorean triples.

INDEXING PROPERTY OF PYTHAGOREAN TRIPLES.

Primitive Pythagorean triples come in 6 classes based on the divisibility of a, b, c, by 3, 4, and 5. [2]

Proof: We can ignore divisibility by 4, for that always characterizes b. Now assume 3 divides a, then 5 can divide either a, b, or c. Next assume that 3 divides b, then 5 can divide a, b, or c. This enumerates all the six possibilities. The 6 classes are listed out as follows. [2]

1. Class A: a is divisible by 3 and c is divisible by 5.

Div	a	b	c
3	x		
5			x

Table 3: Classification of Class A

Examples: (3, 4, 5), (33, 56, 65)

2. Class B: a is divisible by 5 and b is divisible by 3.

Div	a	b	c
3		x	
5	x		

Table 4: Classification of Class B

Examples: (5, 12, 13), (35, 12, 37)

3. Class C: a is divisible by 3 and 5.

Div	a	b	c
3	x		
5	x		

Table 5: Classification of Class C

Examples: (15, 8, 17), (45, 28, 53)

4. Class D: b is divisible by 3 and c is divisible by 5.

Div	a	b	c
3		x	
5			x

Table 6: Classification of Class D

Examples: (7, 24, 25), (13, 84, 85)

5. Class E: a is divisible by 3 and b is divisible by 5.

Div	a	b	c
3	x		
5		x	

Table 7: Classification of Class E

Examples: (21, 20, 29), (9, 40, 41)

6. Class F: b is divisible by 3 and 5.

Div	a	b	c
3		x	
5		x	

Table 8: Classification of Class F

Examples: (11, 60, 61), (91, 60, 109)

With these six classification (indexing) of PPT's are possible to generate a key composed of A's, B's, C's, D's, E's and F's or an alphabet mapping where an alphabet is mapped to combination of two indexes.

It is good that there are infinite many PPT's as assists in providing protection against attacks that uses statistical data; this will be discussed later on.

Theorem: There are infinite many Pythagorean Triples.

Proof: Consider the identity $n^2 + 2n + 1 = (n+1)^2$. Whenever $2n+1$ is a square, this forms a Pythagorean Triple. But $2n+1$ comprise *all* the odd numbers; every other square numbers is odd; there are an infinite number of odd squares; hence there are an infinite number of Pythagorean triples. Pythagorean triples constructed from Euclid's proof are show in table 9. There are an infinite number of Pythagorean triples not of this form, too. We can use the same technique on $n^2 + 4n + 4 = (n+2)^2$. Whenever 4n+4 = 4(n+1) is a square, we get a Pythagorean Triple. Now we are looking for squares of all numbers divisible by 4, hence squares of even numbers. (However, if m is divisible by 2 but not by 4, if m=2(2k+1), then the new Triple is simply a multiple of a 'Euclidian' one. So we consider only multiples of 4.)

[m, where m=2k+1]	[n=(m*m-1)/2]	[n+1]	[m, where m=4k]	[n=(m*m)/4 - 1]	[n+2]
3	4	5	4	3	5
5	12	13	8	15	17
7	24	25	12	35	37
9	40	41	16	63	65
11	60	61	20	99	101
13	84	85	24	143	145
15	112	113	28	195	197
17	144	145	32	255	257
Etc…	Etc…	Etc..	Etc…	Etc…	Etc…

Table 9: Generating infinite Pythagorean Triples.

Since there are infinite many Pythagorean Triple factoring out the n will give us the PPT's, therefore there are infinity many PPT's, the length of the key made from PPT's can be long; the issue of length of key has been resolved. The next issue that needs to be addressed for key generation is randomness. To show that the indexing of PPT's is random we subject it to the Diehard test, the Auto correlation test, Frequency stability test.

CHAPTER III

UNDERSTANDING RANDOMNESS.

The success and strength of a cryptographic system must not be its algorithm, i.e. keeping the algorithm a secret. If a cryptographic scheme relies on the secrecy of its algorithm, it cannot be claimed as a secure cryptographic system, since it relies on obscurity and has never been peer reviewed, for a cryptographic to be accepted as a feasibly secure system it needs to be peer reviewed. Many, if not all feasible secure system have one thing in common, and that is they use keys to encrypt and decrypt data. Keys come in various forms, i.e. mechanical, biometrics or digital keys. Digital keys rely on the fact that they cannot be guess, i.e. they are random.

The definition of randomness changes upon perspective, philosophical it mean lack of purpose, in the realm of religion it is embodied as free will, in cryptography it is the inability to guess a bit even with prior knowledge of the previous bit. Currently all known encryption schemes except for the one time pad can be cracked given infinite time and computing power. So when what exactly makes a system secure?

Modern secure systems rely on computational hardness for security. A modern computer can execute a few billion instructions per second which translates into a few million calculations per second. A cryptographic scheme that requires a hundred million calculations (just under 2^{27}) might appearing daunting to a human, but a computer can do that many calculation under a few minutes. Hence modern cryptography relies on making the number of calculation required to solve the encryption daunting even to a day modern computer. This is done by using keys that are hard for a computer to generate; thereby slowing down the speed at which the computer can run the decryption algorithm (the decryption algorithm in itself is slow). The key can be long like a 128 bit or 256 bit key, but the key will be useless if it can be easily guessed. In fact any mathematical function that can be applied to generate the key is bad, because then it can be guessed (i.e. the occurrence of the next number is based on the occurrence of the previous number.).

When a key consist of totally randomly generated numbers, then computer need to generate all possible permutations of the key in order to able to crack it (i.e. brute force.) If the key was of length 128 and consist of 0's and 1's then it the number of permutation is 2^{128} which roughly equals about the number 34 followed by 37 zeroes, just to show the largeness of this number, let’s consider the oldest object we know to exists, our very own universe, it is about 14 billion years old, this is the number 14 followed by 9 zeroes. Hence 2^{128} is a daunting number even to the modern computer, couple this with the slowness of the decryption algorithm, we can say that the problem is computationally hard and that the encryption scheme is reasonably secure.

The latest trend in brute force attack (i.e. calculating all possible keys, 2^{128} possibilities for a key of size 128) has shifted from CPU (Central processing unit) processing to GPU (Graphical processing unit) processing. The reason it that even though the number of calculation is large the calculation itself is relatively light enough, such that a GPU can handle it. Unlike the CPU that is one massive chip, GPU consist of high number of microchips that, with GPU Decryption programs a keys that used 128 bits may not consider reliably secure. Hence the advent of 256 bit keys. No matter the size of the key if the key can be guessed, then the encryption can be broken, this is the reason for randomness being essential for encryption.

Now that we have seen how important randomness is to a cryptographic system, let us see how easy it is for us to implement randomness. A computer is a deterministic machine, hence algorithmically it is impossible for a computer to generate random events. Often people mistake that the random number generator function found in the math libraries of a high level language as random, in truth these are pseudo random generators, they appear random to humans and are not safe to be used in cryptographic system. This is due to the fact they use a seed value that is expanded or transformed to generate the random number. If a cryptosystem used these function to generate its random keys, these keys can be guesses using timing attacks [4].

Humans find the concept of randomness very hard to understand, and this can get us into big trouble. There are many real life examples that support the previous statement, the gamblers fallacy that has plagued gambler for ages. If a gambler suffering from gambler fallacy learned that a slot machine had a one in twenty (1/20) probability to pay out, the gamble would expect that in twenty tries surely they would win at least once, or that if the machine has not made a pay out in a while it must be nearing a time where the machine is due to pay out. This in fact is wrong because the machine does not keep track on the attempts. Each attempt has a 1/20 chance to pay out and that chance is random.

A more recent example would be with the shuffle feature in the iPod, when the iPod was first released to the general public, there was a common complaint that the shuffle feature of the iPod was not working correctly, and people felt that too often songs from the same artist or album played back to back. In response the code for the shuffle program was changed so that the song played did not come from the same artist or album as the last song, the programmers made the shuffle feature less random to make it feel more random to the user.

Randomness is counter-intuitive, it tends to cluster events together and this is one of reasons humans have a hard time being random. Fortunately there are many other events in the universe that are random, and these random event can be measured and fed as input to a computer that require true randomness. The decay of radioactive materials is a random event that can be measured by a Geiger counter; events taking place at a quantum level are random. So if human cannot accurately say if something is random or not, how then is something random?

There are tests and properties that define randomness. The Diehard test is a collection of test to check randomness, if it is random it need to pass the Diehard test. Frequency stability is a randomness property and it mean that in a sufficiently long string all substring in that sting will appear equal number of times, if something is random it need to display frequency stability.

TESTS FOR RANDOMNESS.

We want to test if PPT's indexing are random so that we may use them in cryptography as key generators. The criteria to qualify for randomness for our purpose are lack of pattern, outcome of next event is independent of current events, and equal probability for each event. The Auto correlation test will help us determine if there are any patterns in the PPT's indexing and if each occurrence of an index is dependent on the previous index. The Overlapping permutation test will show the probability of events (events is these cases are the A, B, C, D, E, and F indexing classification.) The Diehard test; which is a series of statically test that measure the quality of random number generators, the tests the Diehard test [8] are:

Birthday spacing test: Choose random points on a large interval. The spacing between the points should be asymptotically exponentially distributed.

Overlapping permutation test: Analyze sequences of five consecutive random numbers. The 120 possible orderings should occur with statistically equal probability.

Ranks of matrices: Select some number of bits from some number of random numbers to form a matrix over {0, 1}, then determine the rank of the matrix. Count the ranks.

Monkey test: Treat sequences of some number of bits as "words". Count the overlapping words in a stream. The number of "words" that don't appear should follow a known distribution. The name is based on the infinite monkey theorem.

Count the 1s: Count the 1 bit in each of either successive or chosen bytes. Convert the counts to "letters", and count the occurrences of five-letter "words".

Parking lot test: Randomly place unit circles in a 100 x 100 square. If the circle overlaps an existing one, try again. After 12,000 tries, the number of successfully "parked" circles should follow a certain normal distribution.

Minimum distance test: Randomly place 8,000 points in a 10,000 × 10,000 square, and then find the minimum distance between the pairs. The square of this distance should be exponentially with a certain mean.

Random spheres test: Randomly choose 4,000 points in a cube of edge 1,000. Center a sphere on each point, whose radius is the minimum distance to another point. The smallest sphere's volume should be exponentially distributed with a certain mean.

The squeeze test: Multiply 2^{31} by random floats on [0, 1) until you reach 1. Repeat this 100,000 times. The number of floats needed to reach 1 should follow a certain distribution.

Overlapping sums test: Generate a long sequence of random floats on [0, 1). Add sequences of 100 consecutive floats. The sums should be normally distributed with characteristic mean and sigma.

Runs test: Generate a long sequence of random floats on [0, 1). Count ascending and descending runs. The counts should follow a certain distribution.

The craps test: Play 200,000 games of craps, counting the wins and the number of throws per game. Each count should follow a certain distribution.

Although there are 12 tests in Diehard test, only some of the tests will be applied to PPT's. Only the birthday spacing test, overlapping permutation, and monkey test are relevant. The birthday spacing test evaluates the randomness of groups of n sequences showing frequency stability. Overlapping permutation test can show uniform distribution of randomness. The infinity money test will show all indexed sequence of size *n* occurring within the PPT's; unfortunate the infinite money test though relevant the implementation is beyond the scope of this thesis.

Test for randomness come in two kinds, empirical and theoretical. Empirical test use the generated data to perform the test, Auto correlation and overlapping permutation test are example of these kinds of test. Theoretical tests, which are better when they exist because they are logic tests in the sense that they require knowledge of the structure of the generator. Theoretical test include the chi-square test and The Kolmogorov-Smirnov test. Chi Square are used to test the differences between two or more actual samples sets. The Kolmogorov-Smirnov test (KS-test) tries to determine if two data sets differ significantly, for example in this case one set will be the index sequence generated by PPT's and the other set can be a dice roll, since they both have six possible values.

CHAPTER IV

PROPOSED 4-WAY HANDSHAKE USING PRIMITIVE PYTHAGOREAN TRIPLES AND BLOM'S SCHEME.

In the proposed 4-way handshake we use both PPT's and Blom's scheme to achieve key exchange and authentication. We use PPT's for keys because there is infinity many PPT's and they are random [20] (see also [21]-[23] for other random sequences). We use Blom's scheme because we need a way to authenticate many client to server. The relationship between client and server is many to many, hence we need a scheme that can handle large user base, be secure, easy to use and easy to reset, Blom's scheme fit all these parameters. This is a 2 stage process in the first stage we establish initial data like private and public matrix for both the server and client, also other miscellaneous information like time stamp format, seed generation from time stamp, and if any classes of the PPT's will be left out in the index table. This initial step is done though a trusted organization. We know that there are six classifications for PPT's which is discussed in chapter II of this thesis. We can generate a string composed on A, B, C, D, E, F. The proposed idea for using PPT's will be explained step by step below. Suppose the key is embedded in the authentication message, and the message is say 10 digits long.

Message 1: Server send a PPT's string of *n* digits long, its public matrix to the client, and a time stamp.

Message 2: The client can calculate the secret number from the server's public matrix and then generate the seed from the time stamp and secret number. Now that the client knows the seed, it can generate the index table. Client concatenates to the received string another *n* digit for a total length of 2*n*, and resend the sequence to the server along with its public key (client's public key.) At this point the client knows the starting position of the raw key.

Message 3: The server can compute the secret number from the client public matrix, and then it can generate the seed from which the PPT's table is created and confirm the clients reply. At this point both the server and the client know the starting position of the raw key. Server sends a new PPT's string of *n* digits long

Message 4: The client can look up the first occurrence of the received string after the starting position of the raw key is found. Once this position is found the client concatenates to the received string another *n* digit for a total length of 2*n*, and resend the sequence to the server. The server can confirm this string by check if it is valid, this is the authentication. Once verified the server and the client now know the starting and ending position of the raw key, from which they can derive the raw key.

Example: Suppose the generated indexed PPT's table has this particular sequence of indexes. "ACECBDDEBFACDDDFCEEFDBCFFAAEF"

Step 1: Server sends ECBDD to client.

Step 2: Client replies with ECBDDEBFAC to server.

Step 3: Sever sends EEFDB to client.

Step 4: Client replies with EEFDBCFFAA.

The raw key will be DDDFC. And the server and client have authenticated themselves to each other. This is just an example with a small sequence, in the real world application; the string can be even longer since PPT's are infinite. However the term "key" may or may not be used literally. It could be used as a key. It could also be used as the section of the PPT table that will be used for various tasks.

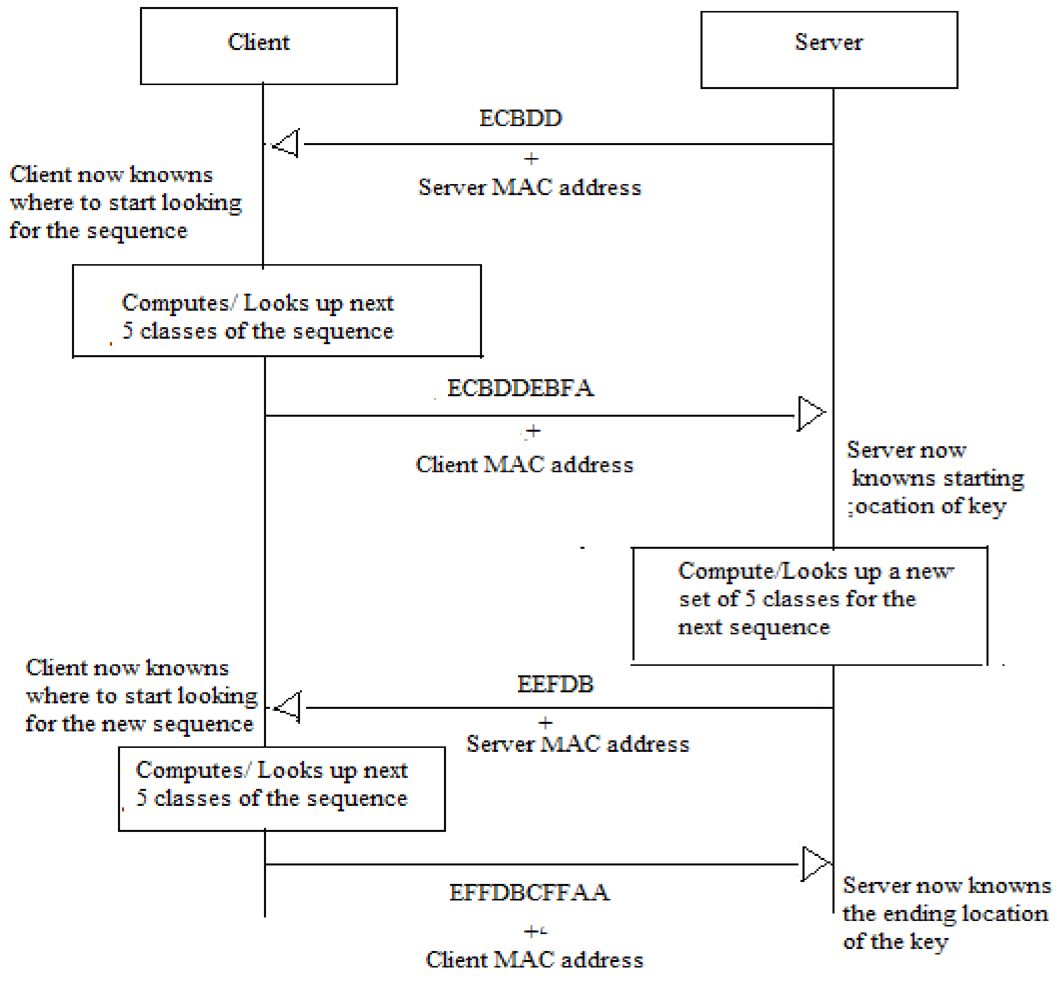

Client
Server
ECBDD
+
Server MAC address
Client now knowns where to start looking for the sequence
Computes/ Looks up next 5 classes of the sequence
ECBDDEBFA
+
Client MAC address
Server now knowns starting ;ocation of key
Compute/Looks up a new set of 5 classes for the next sequence
Client now knowns where to start looking for the new sequence
EEFDB
+
Server MAC address
Computes/ Looks up next 5 classes of the sequence
EFFDBCFFAA
+
Client MAC address
Server now knowns the ending location of the key

Figure 10: Proposed 4-way handshake.

However there is one critical issue and that is there can be infinite many sequence similar to the sequence passed in message 2 and message 4 which can lead to having multiple keys. We resolve this issue but using a secret number, the number is only known between the client and server. The number is decided by both the server and the client and the exchanged technique used here is Blom's scheme described in the paper "An optimal class of symmetric key generation systems." [14]. The idea is to use symmetric matrices to locate the number in the matrix by computing it using a public and private matrix [14] [15]. The matrix operations can be speeded up by the use of multilayered architectures [24],[25].

Key management stage (Blom’s scheme)

Blom’s scheme is a symmetric threshold key exchange protocol. It enables any two parties to independently create a shared key for communication. In our implementation we will follow the steps below to generate the public and private keys (in our case matrixes).

1) Chose a very large prime number. (pre-deployment)

2) Generate a random square symmetric matrix K, This matrix is to be secret. (pre-deployment)

3) Compute private and public matrix for server and client.

Example: Let our prime (P) be 23

Let our square symmetric K be q×q, here q = 5

$$K=\begin{pmatrix} 6 & 7 & 9 & 12 & 16 \\ 7 & 8 & 10 & 13 & 17 \\ 9 & 10 & 11 & 14 & 18 \\ 12 & 13 & 14 & 15 & 19 \\ 16 & 17 & 18 & 19 & 20 \end{pmatrix}$$

Determine random private matrix for server (X) and client(Y).

$$X=\begin{pmatrix} 3 \\ 5 \\ 12 \\ 1 \\ 31 \end{pmatrix} \quad Y=\begin{pmatrix} 13 \\ 51 \\ 2 \\ 10 \\ 3 \end{pmatrix}$$

Compute the public matrix for server and client.

$$X` = (KX) \bmod P,\ Y` = (KY) \bmod P$$

$$X` = \begin{pmatrix} 2 \\ 8 \\ 22 \\ 22 \\ 22 \end{pmatrix} \quad Y` = \begin{pmatrix} 0 \\ 10 \\ 15 \\ 19 \\ 4 \end{pmatrix}$$

The client sends its public matrix (Y`) to the server and the server likewise will send its public matrix (X`) to the client. To decide the secret number the server and client will multiple their private matrix with their counterpart's public matrix.

Secret Number = $X^TY`$ mod P= $Y^TX`$ mod P, where T is the transpose.

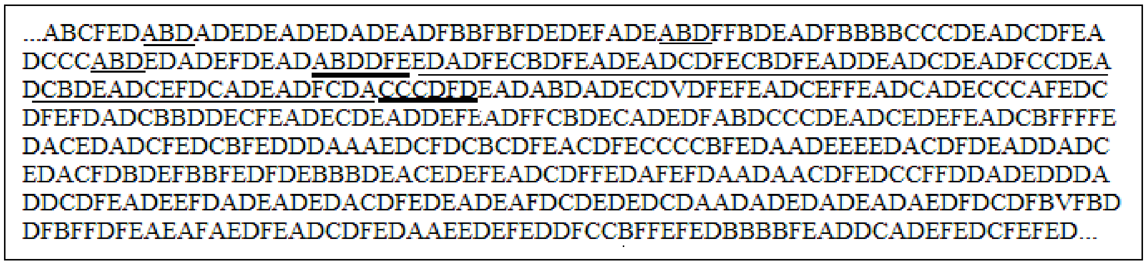
$X^TY`$ mod 23 = 764 mod 23 = 5 (server side)

$Y^TX`$ mod 23 = 373 mod 23 = 5. (client side)

Therefore the shared secret number is 5.

The security strength of the above mentioned key management system is; assuming that K is of size q x q, then for an attacker to compromise this system, they need to compromise at least q client and server connections. The private matrix must not be linearly dependent on each other; otherwise a group attacker can determine the secret number of client. To ensure this does not happen a Maximum Distance Separable matrix (MDS) must be used. A MDS is a matrix that has certain diffusion properties.

The figures below describe the protocol in whole. Suppose the secret number is 4. Look in the index for the 4th occurrence of the string ABD.



```
...ABCFEDABDADEDEADEDADEADFBBFBFDEDEFADEABDFFBDEADFBBBBCCCDEADCDFEA
DCCCABDEDADEFDEADABDDFEEDADFECBDFEADEADCDFECBDFEADDEADCDEADFCCDEA
DCBDEADCEFDCADEADFCDACCCDFDEADABDADECDVDFEFEADCEFFEADCADECCCAFEDC
DFEFDADCBBDDECFEADECDEADDEFEADFFCBDECADEDFABDCCCDEADCEDEFEADCBFFFFE
DACEDADCFEDCBFEDDDAAAEDCFDCBCDFEACDFECCCCBFEDAADEEEEDACDFDEADDADC
EDACFDBDEFBBFEDFDEBBBDEACEDEFEADCDFFEDAFEFDAADAACDFEDCCFFDDADEDDDA
DDCDFEADEEFDADEADEDACDFEDEADEAFDCDEDEDCDAADADEDADEADAEDFDCDFBVFBD
DFBFFDFEAEAFAEDFEADCDFEDAAEEDEFEDDFCCBFFEFEDBBBBFEADDCADEFEDCFEFED...
```

Figure 11: Proposed 4-way handshake using the indexed PPT's table.

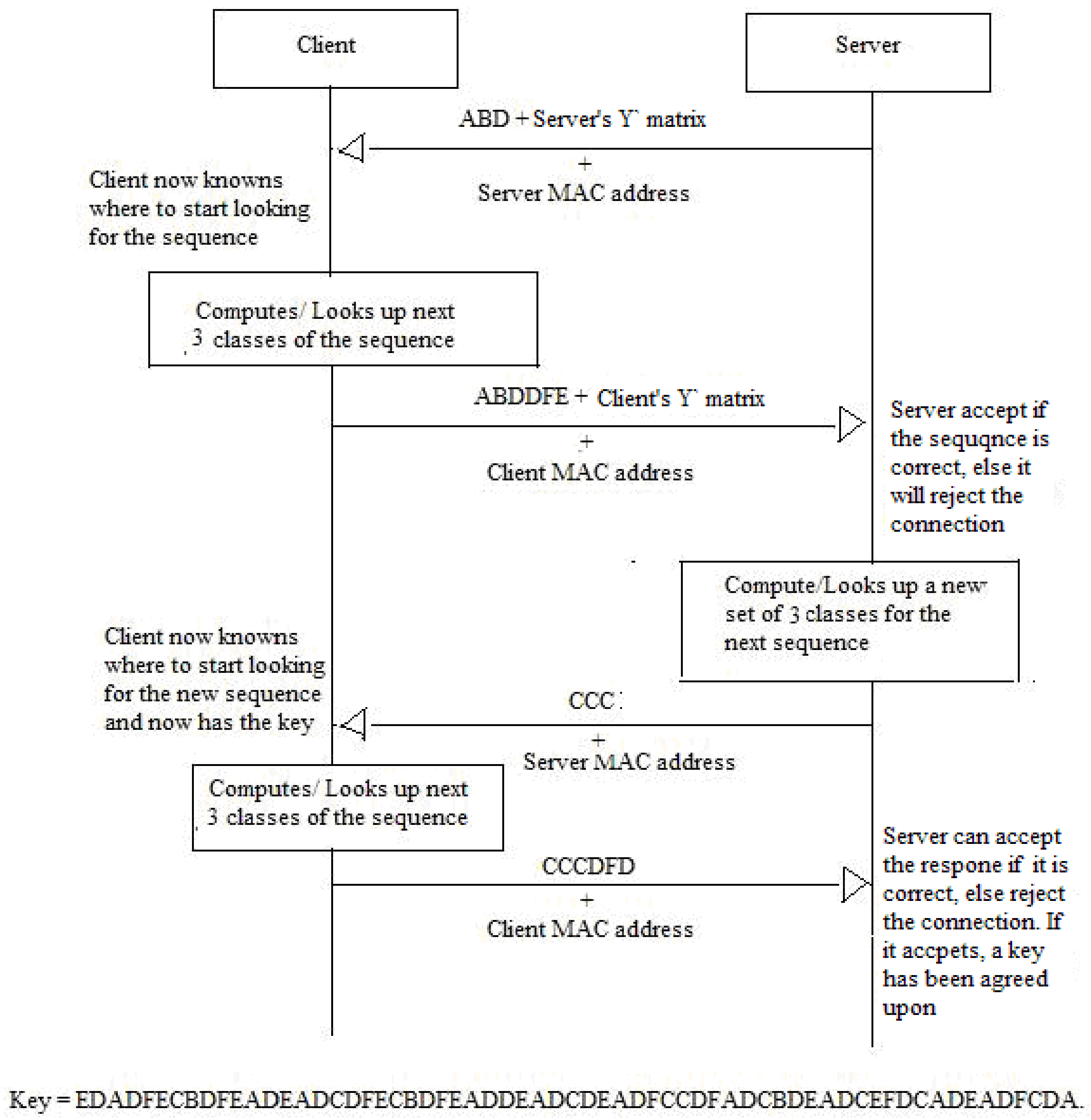

Client
Server
ABD + Server's Y` matrix
+
Server MAC address
Client now knowns where to start looking for the sequence
Computes/ Looks up next 3 classes of the sequence
ABDDFE + Client's Y` matrix
+
Client MAC address
Server accept if the sequqnce is correct, else it will reject the connection
Compute/Looks up a new set of 3 classes for the next sequence
Client now knowns where to start looking for the new sequence and now has the key
CCC
+
Server MAC address
Computes/ Looks up next 3 classes of the sequence
CCCDFD
+
Client MAC address
Server can accept the respone if it is correct, else reject the connection. If it accpets, a key has been agreed upon
Key = EDADFECBDFEADEADCDFECBDFEADDEADCDEADFCCDFADCBDEADCEFDCADEADFCDA.

Figure 12: Step by step pictorial representation of the proposed 4-way handshake.

CHAPTER V

EXPERIMENTS AND RESULT

The auto-correlation function has been frequently used to determine the randomness of data {21]-[23]. Some general considerations related to randomness are given in [26]-[31]. It takes two variables in the series and then finds out if there is a relationship between these two variables in the series (i.e. do they influence each other in any kind of way.) The equation for auto-correlation is as shown below:

$$C(k)=\frac{1}{N}\sum_{n=0}^{n=N-1}a(n)a(n+k)$$

where:

a(n): value of the data point in the n^{th} position.

a(n+k): value of the data point in the $(n+K)^{th}$ position.

N: total number of data point in the series.

k: the auto-correlation distance,

In the case of PPT's, the experiment was run with different data sets on different variable separated by a distance of k. Since we are checking through the series, we need to make sure at least a reasonable number of variables are checked before we achieve an overflow comparison. Overflow can be described as going out of bounded. We handle overflow by treating the series as a circular series that is; when we hit overflow we continue from the start. Since the equation uses mathematical multiplication we need to represent our classes as number. We can either chose to represent our classes with balanced weights on such as A=3, B=2, C=1, D=-1, E=-2, F=-3 or unbalanced weights such value; A=1, B=2, C=3, D=4, E=5, F=6.

PPT's are random when arranged in order of a, b, or c [1]. We shall show this by generating ~100,000 PPT's order them with respect to a, b, or c and use balanced and unbalanced weights to show they are random when order in a, b, or c.

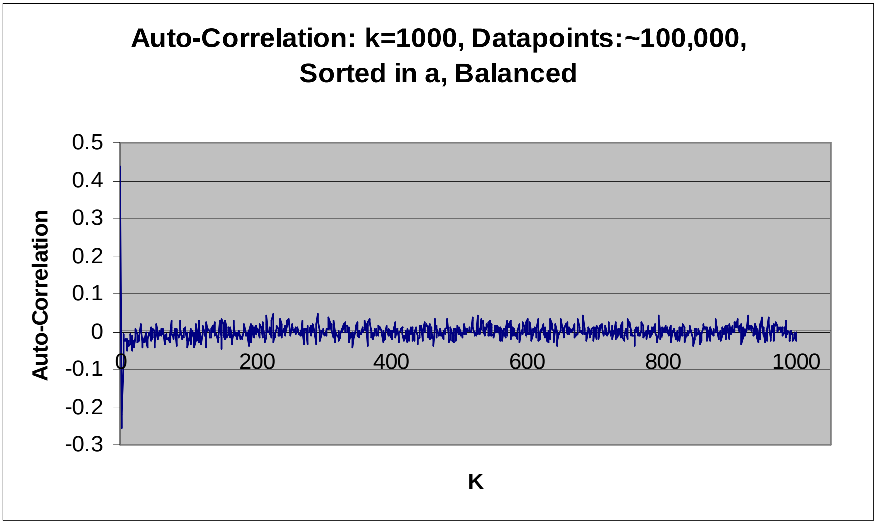


Figure 13: Auto-Correlation function on a balanced scale with k=1000 and 100,000 data points sorted with respect to a.

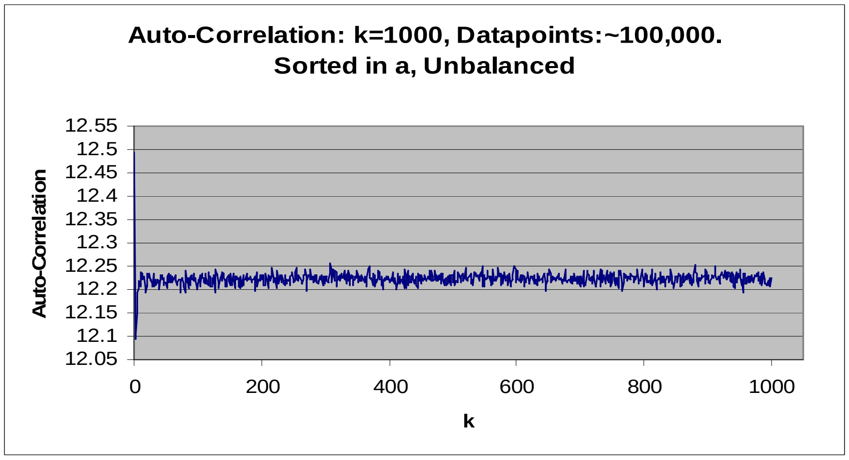


Figure 14: Auto-Correlation function on an unbalanced scale with k=1000 and 100,000 data points sorted with respect to a.

This is a plot of the auto-correlation function applied on a PPT's series that has 100,000+ data point sorted in a. The graphs show strong correlation between the classes. When we plot it on a balance graph we see that the distance between classes less than .01. Where are when viewed on an unbalanced graph the distance is between 12.2 and 12.25. The unbalanced graph shows that the distance grows only by the range of the unbalance (6-1=5).

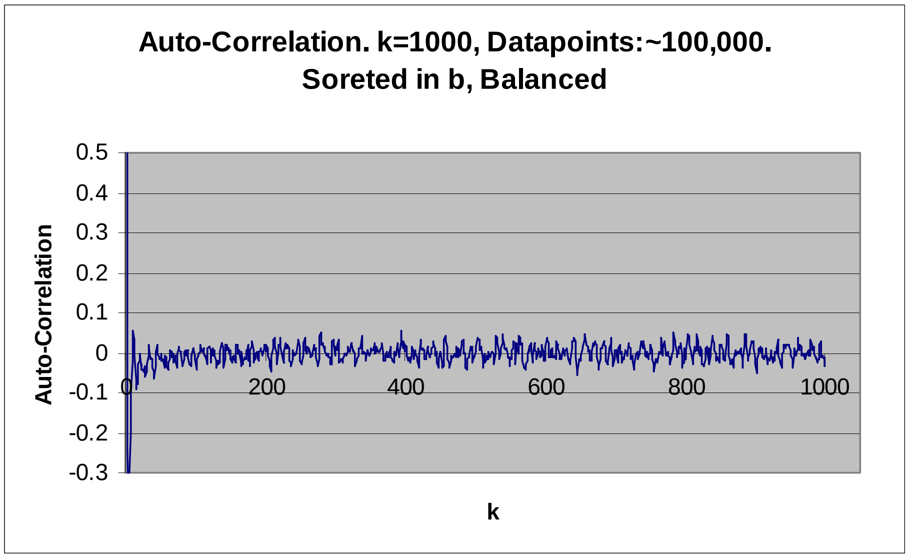


Figure 15: Auto-Correlation function on a balanced scale with k=1000 and 100,000 data points sorted with respect to b.

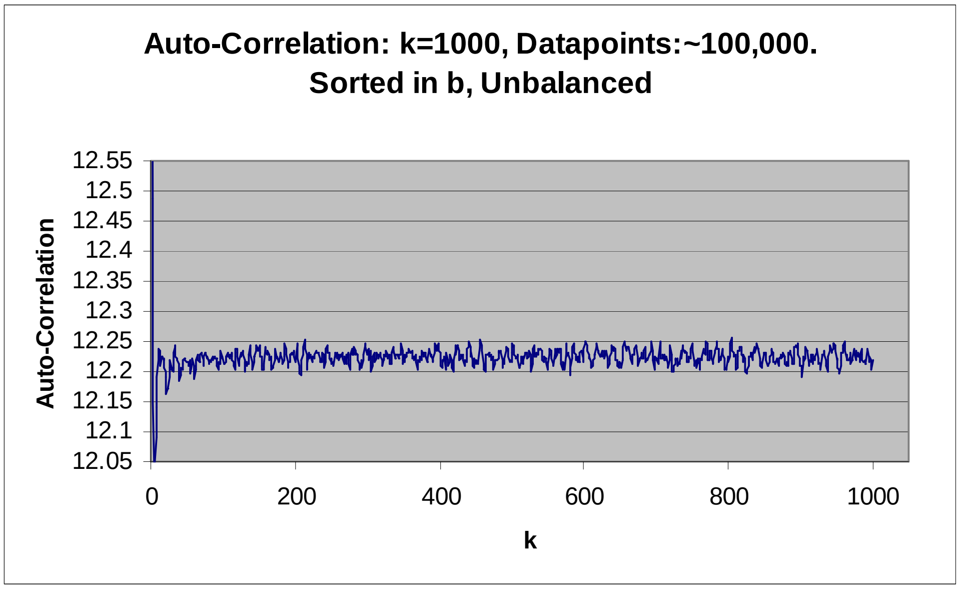


Figure 16: Auto-Correlation function on an unbalanced scale with k=1000 and 100,000 data points sorted with respect to b

The results when plotted when sorted in ‘b’ is similar to that of when sorted in ‘a’.

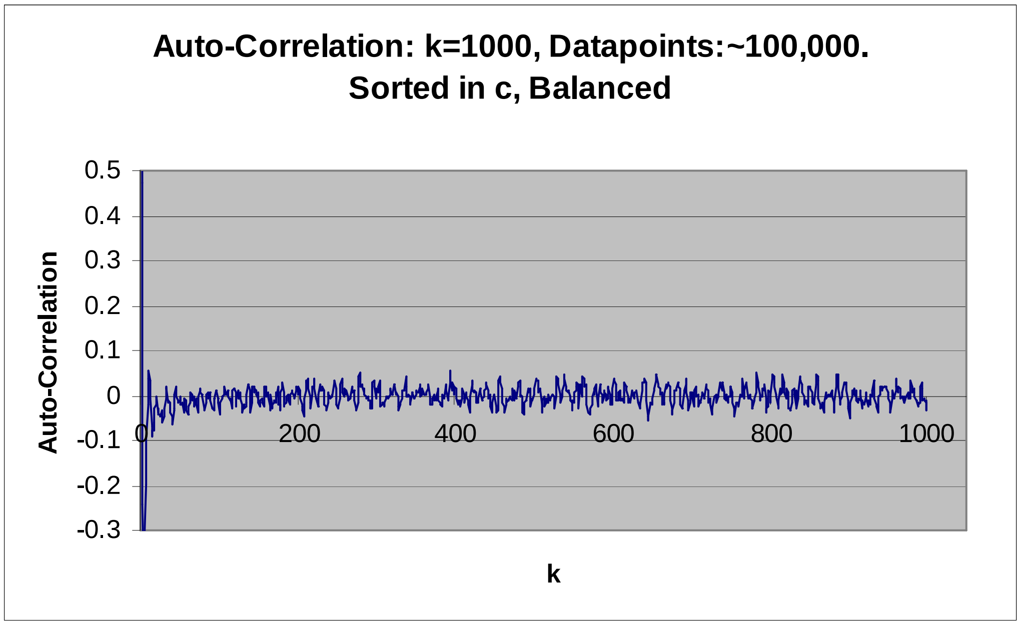


Figure 17: Auto-Correlation function on a balanced scale with k=1000 and 100,000 data points sorted with respect to c

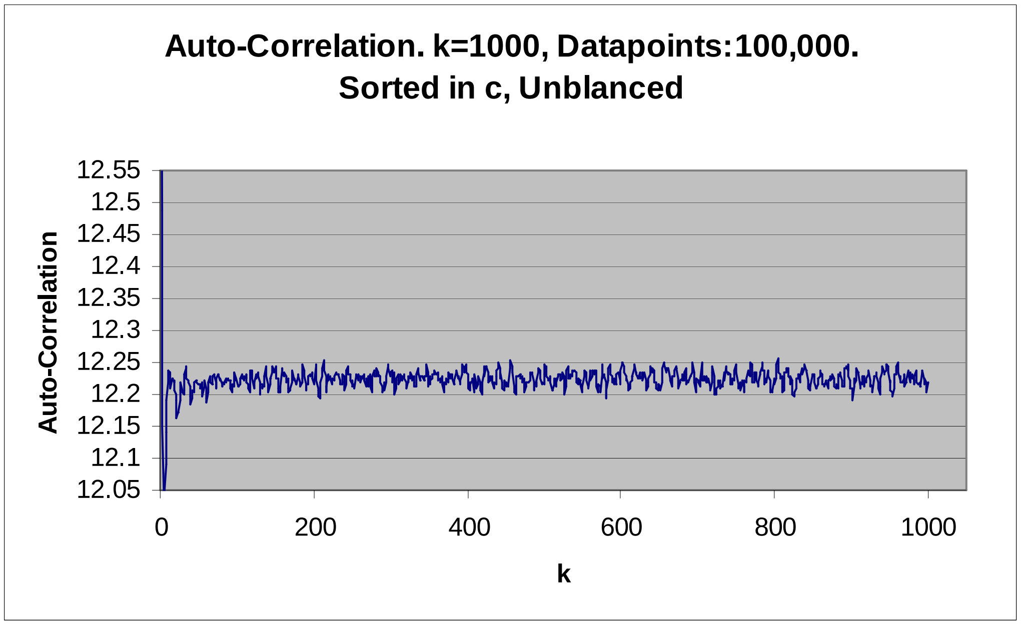


Figure 18: Auto-Correlation function on an unbalanced scale with k=1000 and 100,000 data points sorted with respect to c

The results when plotted when sorted in c is similar to that of when sorted in a and b. Looking at the graphs we can see that there is hardly any deviation that stand out, all the value are between .01 and -.01 for balanced and 12.2 and 12.25 for unbalanced. Since there is not much deviation we can safely say that there is strong correlation between the data points in the series, thereby making them random enough for our use.

Even though the auto-correlation function tells us if the series is random it does not tell us if the series is uniformly or non-uniformly random. For that we will be using the Overlapping permutation test.

The overlapping permutation test is part of the Diehard test, which is a battery of statistical test to determine the quality of random number generators. The overlapping permutation test can be described “Analyze sequences of five consecutive random numbers. The 120 possible orderings should occur with statistically equal probability.”[16]

In our case we chose at random five consecutive data points, generate the 120 possible orderings (5!), and then check how many times each ordering appear in the series. Below are 5 graphs of different sequence.

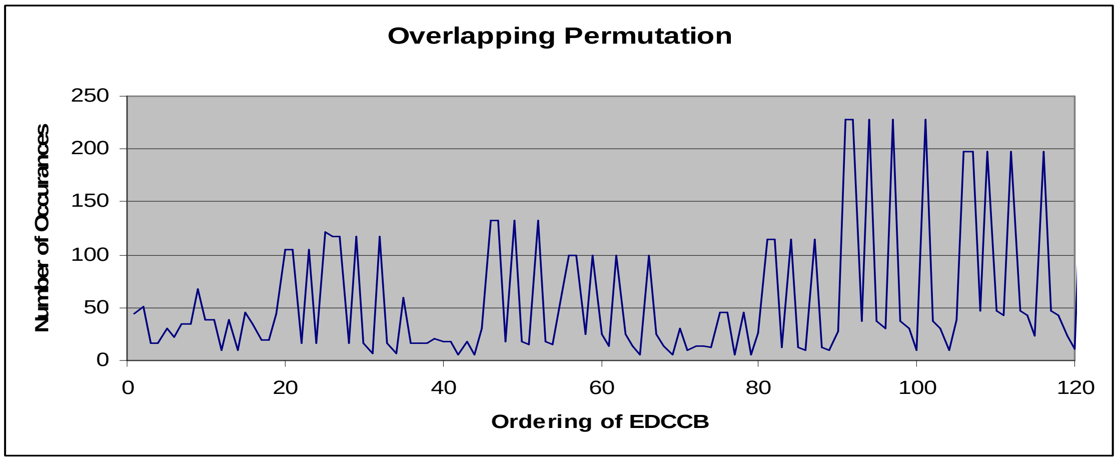


Figure 19: Plot of overlapping permutation with different ordering's of EDCCB

Statistical Data on EDCCB Ordering

Average	Median	SD	Max	Min
56.48	31	60.01	228	5

Table 10: Statistical data on overlapping permutation with different ordering's of EDCCB

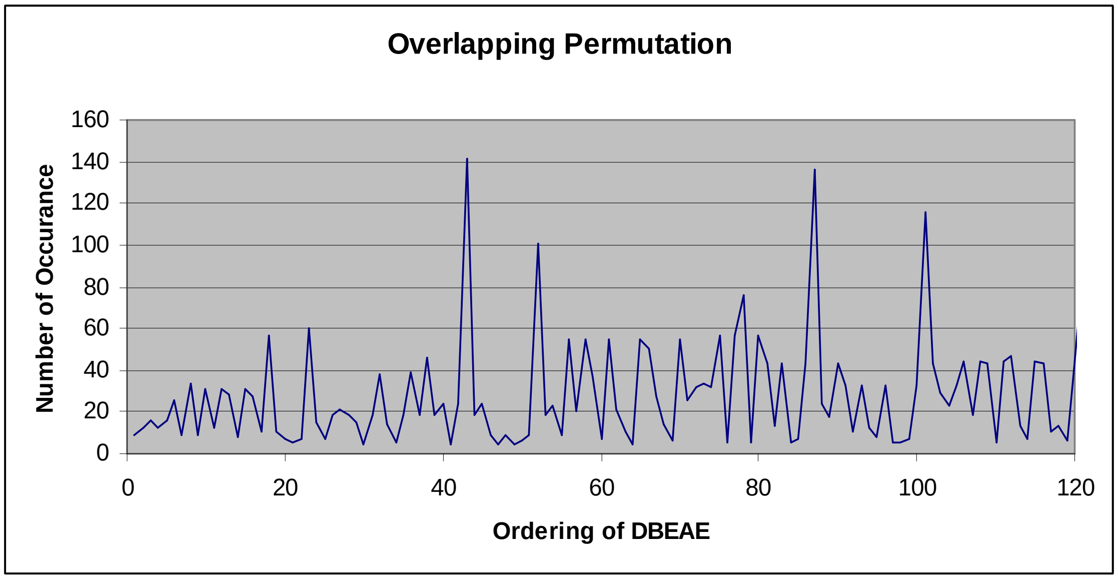


Figure 20: Plot of overlapping permutation with different ordering's of DBEAE

Statistical Data on DBEAE Ordering

Average	Median	SD	Max	Min
28.37	22	24.83	141	4

Table 11: Statistical data on overlapping permutation with different ordering's of DBEAE

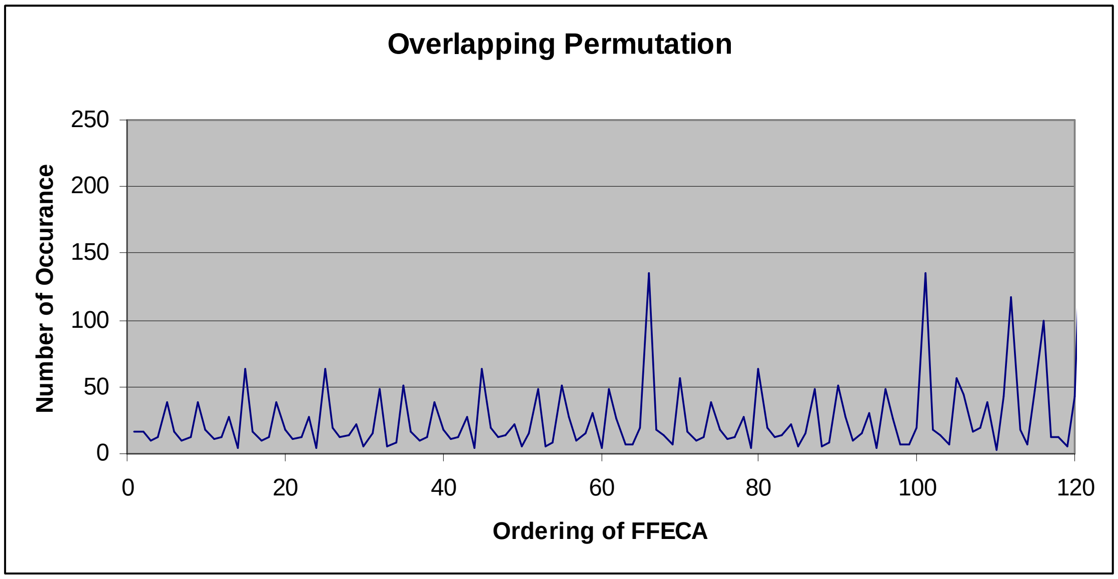


Figure 21: Plot of overlapping permutation with different ordering's of FFECA

Statistical Data on FFECA Ordering

Average	Median	SD	Max	Min
26.33	16	28.77	198	3

Table 12: Statistical data on overlapping permutation with different ordering's of FFECA

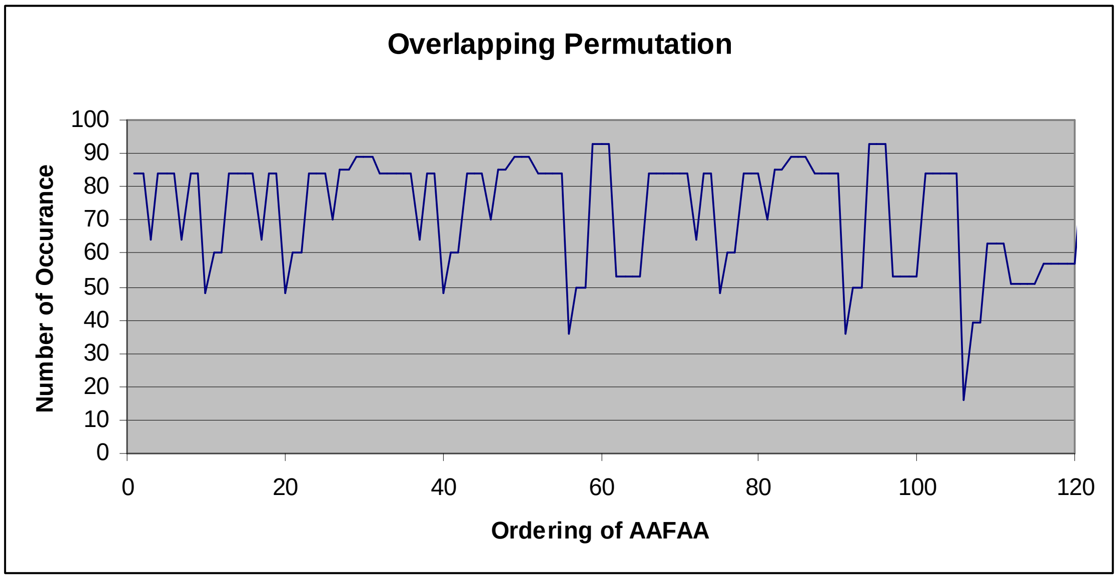


Figure 22: Plot of overlapping permutation with different ordering's of AAFAA

Statistical Data on AAFAA Ordering

Average	Median	SD	Max	Min
73.36	84	16.43	93	16

Table 13: Statistical data on overlapping permutation with different ordering's of AAFAA

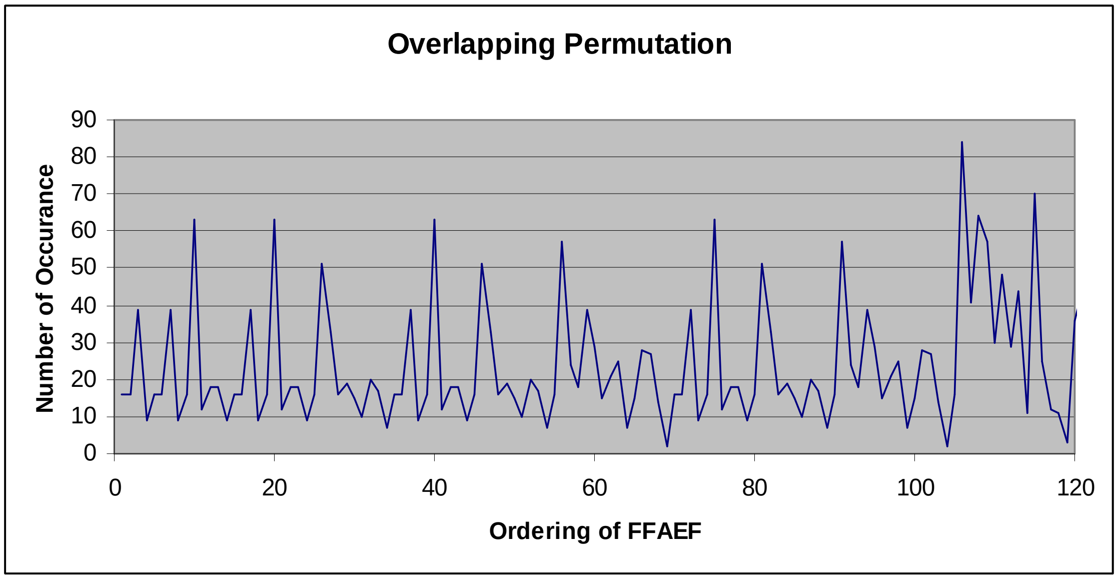


Figure 23: Plot of overlapping permutation with different ordering's of FFAEF

Statistical Data on FFAEF Ordering

Average	Median	SD	Max	Min
23.56	17	16.28	84	2

Table 14: Statistical data on overlapping permutation with different ordering's of FFAEF

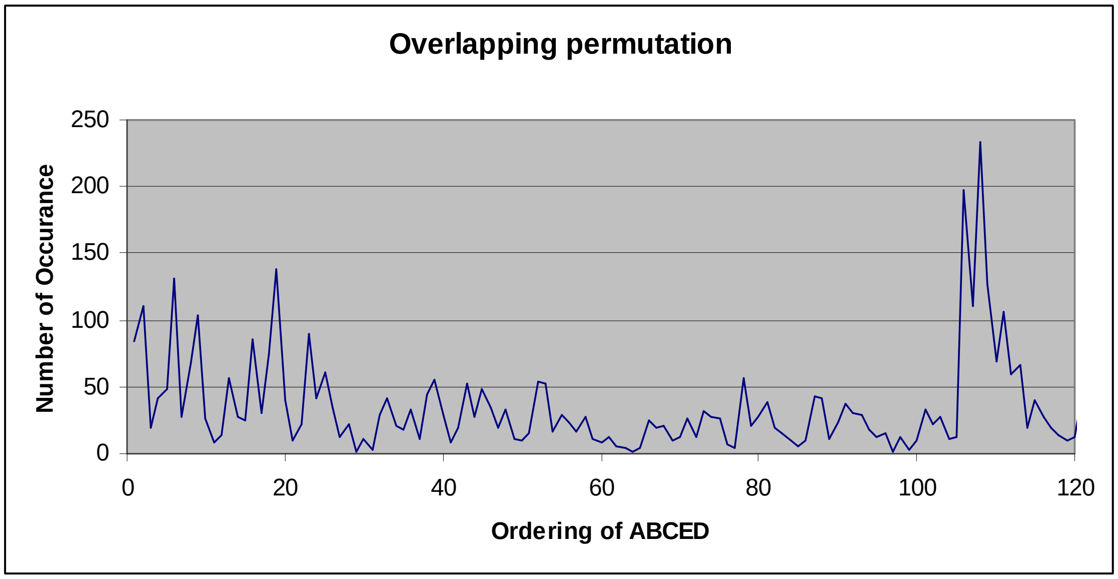


Figure 24: Plot of overlapping permutation with different ordering's of ABCED

Statistical Data on ABCED Ordering

Average	Median	SD	Max	Min
34.93	25	36.55	234	2

Table 15: Statistical data on overlapping permutation with different ordering's of ABCED

Looking at graphs and statistical data it is safe to say that the PPT's are not uniformly distributed. There is a possibility that there could be other 5 character sequences that display uniform distribution. So this test by itself cannot guarantee non-uniform distribution. We need to do a frequency stability test to ensure our results.

Frequency Stability Test: is a very simple test described by checking the occurrence of strings of lengths 1, 2, 3… N. The Overlapping Permutation test is a special case of this test. With string length of 1, we count the number of A, B, C, D, E, F. With string length 2 we count the number of AA, AB, AC…FF. Once we plotted the graphs and if we observer that the graphs as similar to each other then we can safely say the PPT's are uniformly distributed, otherwise they are not uniformly distributed.

Strings	Frequency	Percent
F	16636	16.58178
D	16658	16.60371
C	16702	16.64756
A	16757	16.70238
E	16772	16.71733
B	16802	16.74724

Table 16: Frequency of the 6 classes in an indexed PPT's table that has 100,000 data points.

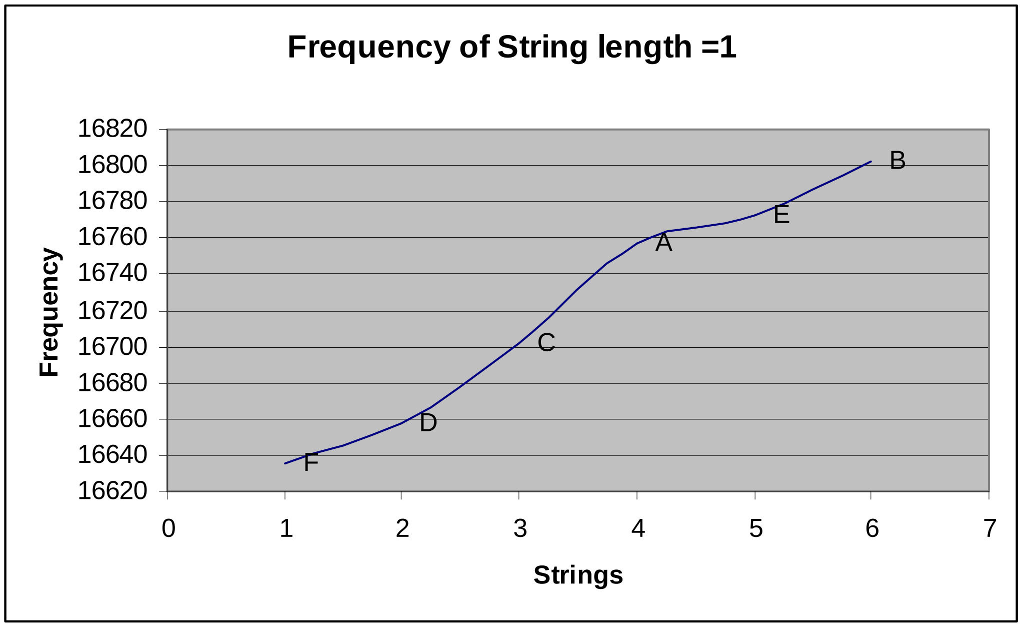


Figure 25: Frequency of string length 1 an indexed PPT's table that has 100,000 data points

When the frequencies are sorted in ascending order and then plotted on a graph; we can see that the distribution is almost uniform. As the length of the string increases, the series losses it uniform distribution, as in the case of string length = 2 displayed below.

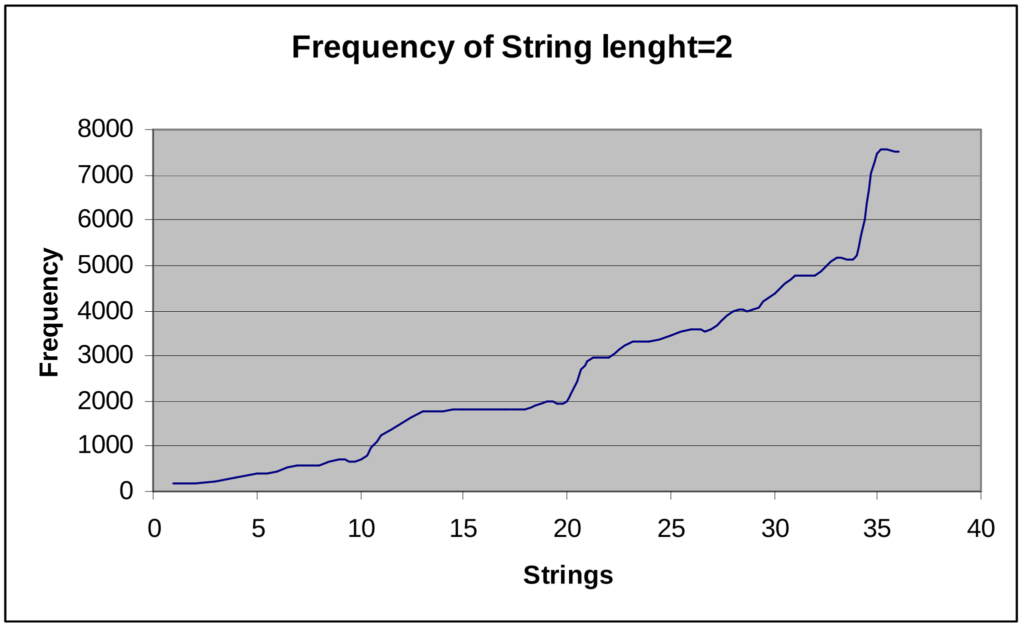


Figure 26: Frequency of string length 2 an indexed PPT's table with 100,000 data points.

In this graph we have string has occur less than 1000 time as opposed to strings that occur more than 7000 rimes. All sense of uniform distribute is gone. This mean we have to take care in selecting the size of the string used in message 1 and 3. If we make it too large we might run the risk of choosing a string that is easy for the attacker to home in on.

In order to avoid this we can string sizes that are small. This would mean more guesses will have to be made by the attacker. A string size of one would be ideal, because from the auto correlation function we know the series is random, and when using length one we make the series uniformly distributed (i.e. the attacker needs to make at least (1/6)N guess to get a correct hit.) Combine this with the fact that the attacker does not know what the table looks like (because there can be infinite many tables) the odds of a correct guess is astronomical.

There is always the possibility that there exist an algorithm that would generate PPT in a uniform distributed fashion, but since the protocol has no say on which algorithms to use, we considered prudent to show that PPT's are no uniformly distributed, hence care should be take when choosing strings.

SECURITY ISSUE: BIRTHDAY ATTACK

Even though PPT's are random and the secret number kept secret, it is still possible for an attacker to guess the raw key being used. This is due to the fact that the raw key is obtained from a table that never changes. So because the table does not change, there can only be a fixed number of permutations the attacker needs to run before the either current key being used is guessed or the secret number being used. This is a case of the birthday attack. Though both of these scenarios are undesirable, the attacker guessing the secret number is more important.

Once the attacker can guess the secret number all raw keys become useless, since the start and end location of the raw key is transmitted over open channel. If the attacker is able to guess the raw key being used then all messages sent using that particular raw key is compromised. It would also seem that guess the key is a lot easier than guessing the secret number, since the secret number is never transmitted over open channels. Nevertheless it is a necessity that this be avoided at all cost. The obvious solution would be to make the table (series) containing the PPT's change frequently, if not possible for every message.

This can easily be done by using the time stamp in conjunction with the secret number as the seed for generating PPT's. The formula to generate a seed is as show below:

Seed = (Hour+1)(Minute+1)(Second+1)(Secret Number)

Example:

Suppose:

Time Stamp = 4.09.09

Secret Number = 11

Seed = 5 × 10 × 10 × 11 = 5500

Now using the seed of 5500 we can generate N PPT's. Since the secret number is never transmitted over the communication channel, the attacker only has 3 of the 4 components of the seed. Now the attacker cannot guess the table. If the table is changed often enough the attacker cannot guess the raw key either.

SECURITY THROUGH SCALABILITY ISSUE

Since we need to generate N PPT's, where N is a very large number (upward of a few million) and ideally we generate a table for every message. So long as both the client and server can generate the same table with the same elements, this is the only requirement. It is not important that the table be a proper set of all PPT's, as long as the client and server have the same triples. The Euclidian method is $O(n^4)$ is of polynomial time and is feasible. Most well know algorithms are O(n^4) or worse. To make this process even faster we could design hardware to generate the PPT's instead of software.

Let us discuss the scalability issues the attacker has to overcome in order to compromise the protocol. Firstly there the Blom's scheme secret number and the symmetric square matrix used to generate the private and public matrix. The symmetric square matrix is of size q. the secret number is x. The time stamp (S) is transmitted over open channel, so that is not a factor. Supposing that the attacker know the PPT generation algorithm, they still have to generate a minimum of $q \times P$ many tables, where P is the prime number used in Blom's scheme and q is the size of square symmetric matrix, both q and P is unknown to the attacker. On top of this there can be a possibility that the client and server are ignoring one or more classes; agreed upon the first stage of the protocol (5 being the maximum number of classes that can be ignored for a maximum of 20 different possibilities). Remember due to the way Blom's scheme interacts with the prime number P, the secret number will be less than or equal to P, which in turn affect the seed. Figure 27, shows when |P|=20 and q=50,000 the growth is polynomial to calculate n PPT's. Figure 28 show that the number of calculation need to generating n PPT's is polynomial, but this time the growth is nearly 3 time more, this is due to the fact we included Blom's scheme. Combine this with the fact that some classes (x) may or may not be ignored (20 at best, 6 chose 3), only further complicates the issue for the attacker, since the attacker does not know P, q, and x.

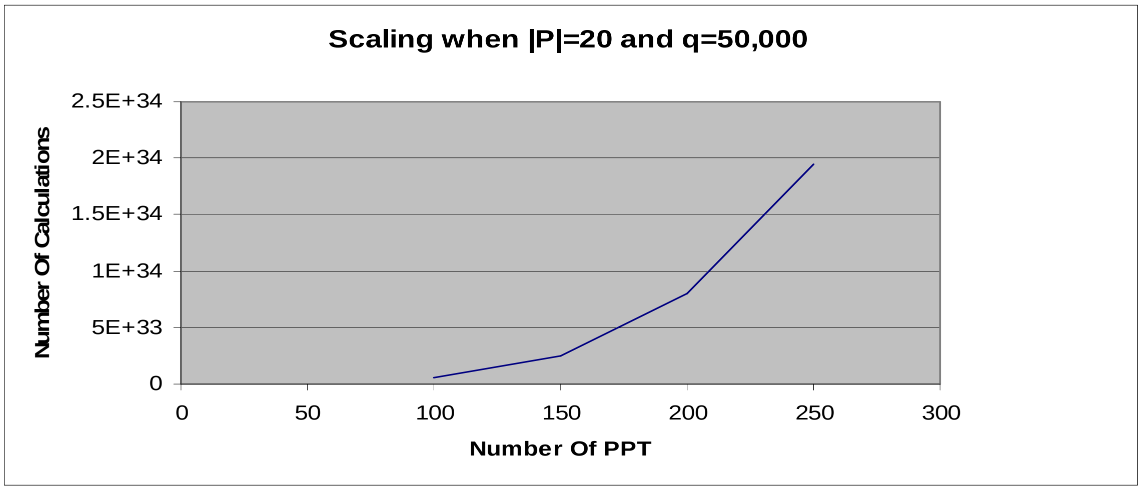


Figure 27: Scaling when |P|=20 and q=50,000

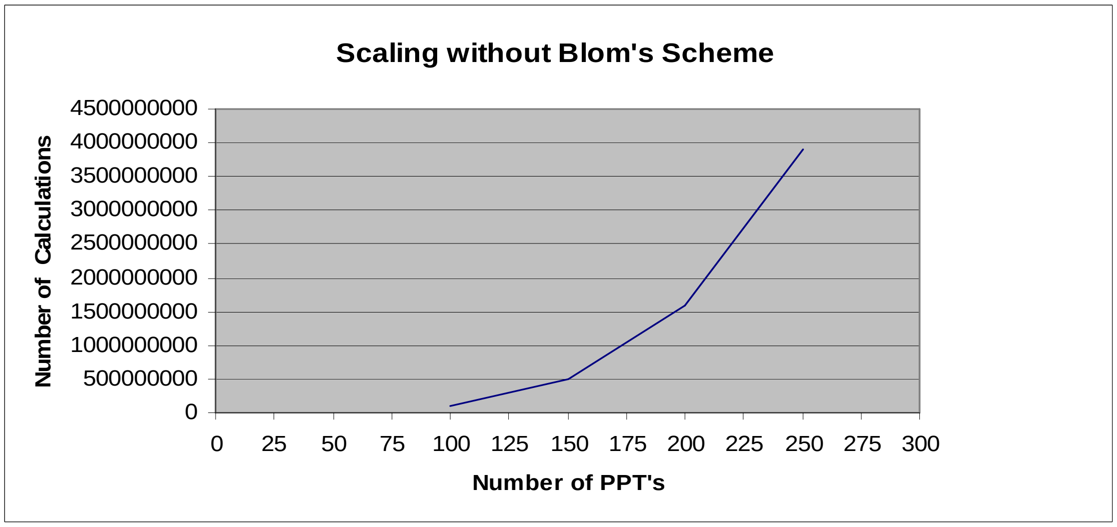


Figure 28: Number of Calculations for n PPT’s without Blom’s scheme.

A COMPLETE EXAMPLE.

This is a step by step toy example of the proposed 4-way handshake. Even before the client and server can start communicating, a few facts must be decided upon, like the classes that will be ignored, the algorithm that will be used for generating the PPT's and their individual private and public matrix. This is called the first stage can be handled either through a third party or by the server/client perhaps at eh point of manufacture. The second stage is where the 4-way handshake happens. Below are figures and expatiation of each handshake (message)

First Stage condition: The D class will be ignored; Euclid algorithm will be used to generate PPT's. Both client and server have their respective private and public matrix. The secret number is 5, this is unknown to the server and client just yet, but is revealed for the reader's sake.

Step 1: Server stores the time stamp and waits for the clients reply.

> Message 1: The server chooses a time stamp (1.02.03), a string "ABC" and its public matrix to the client.

Step 2: The client upon receiving the server public matrix; can calculate the secret number. Using this secret number and time stamp it can calculate seed, with which it can generate the index PPT table. Now knowing the indexed PPT table and the secret number the client can complete the string it received from the server. The starting position of the string marks the starting location of the raw key.

> Message 2:, The client sends the completed string to the server along with the client's public matrix to the server.

Step 3: The server can use the client's public matrix and calculate the secret number. With this new information along with the time stamp the server can find the seed and generate the index PPT table. Using the index table the server can verify the client by check if the string sent by the client is correct. If the string is correct the server continues, else the connection is terminated. When accepted the server now too has the starting position of the raw key.

Message 3: The server chooses another string, but this time from the index PPT generated and sends this string to the client. The starting position of this string marks the end of the raw key.

Step 4: The client looks up the index PPT table and finds the corresponding string and completes its. At this point the client knows the ending location of the raw key and has the raw key.

Message 4: Clients sends back the completed string to server.

Step 5: The server checks the received to see if it is correct. If it is then the server acknowledges it and now has the raw key too, since this string.

CHAPTER VI

CONCLUSIONS

In order to put wireless security on a strong theoretical footing, this thesis proposes a novel way of using PPT's along with Blom's scheme to perform raw key exchange by use of a 4-way handshake similar to the one described in IEEE 802.11i. Since PPT's constitute an infinite set and they display good randomness properties they make good candidates for cryptographic applications. We analyzed the cryptographic strength of random keys generated by PPT's and determined a way they can be used for wireless authentication and as raw keys for encryption in wireless security.

The proposed 4-way handshake provided both authentication and raw key exchange and it very much mirrors the current implementation of the 4-way handshake in WPA2. Since the PPT's are not uniformly distributed, the server must be careful when choosing a initial string to be transmitted in message 1 and 3. As occurrences of long string quickly decline, using long string can undermines the whole process of key agreement, by helping the attacker to guess the starting and ending location of the raw key in the table. So as long as this issue is avoided, the proposed 4-way handshake performs it purpose as a raw key exchange and authentication protocol.

The using of time stamps for generating PPT’s tables prevent the attacker from observing messages over a period of time and being able to guess both the raw key and the secret number. This is achieved by using a time stamp along with the secret number as the seed to generate different tables at different intervals, because we are using a secret number unknown to the attacker, there can be infinite seed per time stamp. Even if the attacker is able to guess the raw key they cannot guess the secret number because there can be infinite tables that contain raw the key. We discussed that to compromise the secret number the attack needs to compromise at least q users where q is the size of the symmetric matrix used in Blom’s scheme. When implemented correctly q can range in the millions, so as long as the secret number is safe the chance that the attacker can make a second correct guess is astronomical.

REFERENCES

1. M. Prabhu and S. Kak, Random Sequences from Primitive Pythagorean Triples. 2012. arXiv: 1211.2751.

2. S. Kak. Pythagorean Triples and Cryptographic Coding. 2010. arXiv:1004.3770.

3. W. J. Spezeski. Rethinking Pythagorean Triples. 2008. Application and Applied Mathematics: An International Journal (AAM) Vol, 3, Issue 1(June 2008), pp. 100 – 112

4. P. C. Kocher. Timing Attack on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. 1996.

5. T. Wrightson. Wireless Network Security A Beginner’s Guide. 2012.

6. N. Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: The insecurity y of 802.11. In The Seventh Annual International Confer enc e on Mobile Computing and Networking (MobiCom 2001), 2001.

7. IEEE Std 802.11i.2004. 2004. The Institute of Electrical and Electronics Engineers, Inc). ISBN 0-7381-4074-0.

8. The Marsaglia Random Number CDROM including the DieHard Battery of Test of Randomness. Florida State University.

9. D. Kahn. The Codebreakers: The Story of Secret Writing. Simon & Schuster. 1999 ISBN 0-684-83130-9l.

10. A. A. Bruen and M.A. Forcinito, Cryptography, Information Theory, and Error-Correction: A Handbook for the 21st Century. 2011.

11. J. Wright, Explaining WPA2 Wireless Security, 2006, Network World.

12. IEEE Std 802.11^{TM}-2012 (Revision of IEEE Standard 802.11-2007). The Institute of Electrical and Electronics Engineers, Inc). 2012

13. G. Stefanick . CWSP Journey, Chapter 5. 2010.

14. R. Blom, An optimal class of symmetric key generation systems, Report LiTH-ISY-I-0641, Linköping University, 1984

15. R. Blom, Non-public key distribution. In Proc. CRYPTO 82, pages 231–236, New York, 1983. Plenum Press.

16. G. Marsaglia, The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness. Department of Statistics, Florida State University, Tallahassee, FL, USA., 1995. <http://www.stat.fsu.edu/pub/diehard/>.

17. O. Goldreich, Foundations of Cryptography: Volume 2, Basic Applications. Vol. 2. Cambridge university press, 2004.

18. A. Bittau, M. Handley, J. Lackey. The Final Nail in WEP's Coffin. IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos, 2006.

19. K. Beaver and P. T. Davis, Hacking Wireless Networks for Dummies, Chapter "Understanding WEP Weaknesses", Wiley Publishing, 2005.

20. S. Kak, Residue Classes of the PPT Sequence, 2013, arXiv:1305.1900

21. S. Kak and A. Chatterjee, On decimal sequences. IEEE Transactions on Information Theory, IT-27: 647 – 652, 1981.

22. S. Kak, Encryption and error-correction coding using D sequences. IEEE Transactions on Computers, C-34: 803-809, 1985.

23. S. Kak, Classification of random binary sequences using Walsh-Fourier analysis. IEEE Trans, Electronic Compatibility EMC-13, 1971.

24. S. Kak, Multilayered array computing. Information Sciences 45, 347-365, 1988.

25. S. Kak, A two-layered mesh array for matrix multiplication. Parallel Computing 6, 383-385, 1988.

26. S. Kak, The cubic public-key transformation. Circuits Systems Signal Processing 26: 353-359, 2007.

27. S. Kak, Information, physics and computation. Foundations of Physics 26: 127-137, 1996.

28. A. Parakh and S. Kak, Online data storage using implicit security. Information Sciences 179: 3323-3331, 2009.

29. A. Parakh and S. Kak, Space efficient secret sharing for implicit data security. Information Sciences 181: 335-341, 2011.

30. S. Kak, The Nature of Physical Reality. Peter Lang, 1986, 2011.

31. S. Kak, The Architecture of Knowledge. CSC, New Delhi, 2004.

VITA

Antony Akshay

Candidate for the Degree of

Master of Science

Thesis: USING PRIMITIVE PYTHAGOREAN TRIPLES AND THE BLOM'S SCHEME IN THE 4-WAY HANDSHAKE WIRELESS SECURITY PROTOCOL

Major Field: Computer Science

Biographical:

Education: B.S. Oral Roberts University, Tulsa, 2008.

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in May 2014.