Literature Survey: An investigation into the field of cryptography and cryptographic protocols

Bradley Cowie and Barry Irwin (Supervisor)

July 28, 2009

Abstract

The field of Information Security and the subfield of cryptographic protocols are both vast and continually evolving and expanding fields. The use of cryptographic protocols as a means to provide security to web servers and services at the transport layer, by providing both encryption and authentication to data transfer, has become increasingly popular. I intend to discuss the need for research into cryptography and to look at existing cryptographic algorithms, cryptographic protocols and related concepts. Finally I intend to look at some related work in detecting encrypted applications.

1 Introduction

This literature review introduces and defines concepts relating to cryptography, cryptographic protocols, issues relating to cryptography and the development of software frameworks. Cryptography is the discipline, art and science of ensuring that messages are secure from possible "attacks", whether these "attacks" be eavesdropping, impersonation or corruption. Cryptography provides security through a number of mathematical transformations that can be proven to be mathematically secure provided some optimum conditions [15]. We however need to cognizant that cryptography on its own is insufficient to ensure a high level of security within an organization, that is to say that cryptography is not the silver bullet to solve all information security issues and should be used in conjunction with good security practices [17]. Cryptography, like the Information Security field itself, is an incredibly broad field involving many existing disciplines such as abstract algebra to provide mathematical proofs for the guaranteed correctness of an algorithm, statistics for analysis of cryptographic algorithms and quantum physics for quantum based random number generation

for quantum cryptography [17]. In this literature review I intend to discuss some cryptographic principles, cryptographic algorithms and the related processing and security costs of employing these algorithms.

Cryptographic protocols are a vital component of Information Security [15] as a means of securing modern networks against would-be attackers by providing data integrity, encryption and authentication to network traffic at the transport layer [19]. Sensitive information, such as banking details, that transverses networks will most likely do so through an encrypted tunnel provided by the cryptographic protocol; it is thus imperative that both the protocol itself is secure and the applications use of the protocol is correct and sensible. A recent paper by Lee *et al.* shows that in a study of over 19000 web servers, 98.36% of the servers provided support for TLS and 97.92% provided support for SSLv3.0 and 85.37% provided support for SSLv2.0 [7]. These statistics serve to show the prevalence of SSL/TLS and the need to support these protocols.

2 Motivation for such research

Due to the upsurge in the demand for secure transactions over the Internet, constant evaluation and research into the field of Information Security particularly in the fields of cryptography and cryptanalysis is vital. To further emphasis this I will outline some of the potential applications of this type of research.

2.1 Prevalence of Web Based Transactions

HTTPS has become prevalent as a means to communicate with a web server securely; however if an attacker were to use HTTPS as a means to perform an attack, it becomes difficult to detect such an attack due to the encrypted nature of the traffic. It would be useful if a system existed to decrypt this traffic and then perform analysis. This is highlighted by work done by Marklinspike [9] in developing a tool, SSLStripper, that removes the secure components of a connection allowing for a new form of MITM (man in the middle) attack where the user believes that his connection is secured (using HTTPS) but in reality messages are passed through HTTP, and are intercepted by a third-party. Furthermore the SANS institute announced "Increasingly Sophisticated Web Site Attacks That Exploit Browser Vulnerabilities - Especially On Trusted Web Sites" as the top security menace in the "Top Ten Cyber Security Menaces for 2008" with "Web Application Security Exploits" in 8th position [11].

2.2 Software Development Habits

Wang *et al.* [20] comment that in the long term, software development cannot afford to consider implementing security only after the application has been developed or late in the development

cycle as irreparable security compromises may already exist and that attempts to correct them would require significant resources. Further we consider that security is one of the core metrics in McCall's Software Quality Checklist [1]. However, software development is notorious for being over budget and far exceeding its expected completion date; as a result we often find that security is left until late in the development cycle and sometimes even after the application has been built [20]. Often this causes poorly implemented security and this only serves to degrade the quality of the system built as it provides the user with a false sense of security; further an insecure application that passes and receives sensitive information is as equally unusable as an application that fails to meet its specifications in terms of correctness [20]. We could argue that the reason why security is not part of many development cycles in earlier stages is due to the difficulty and tedium of checking the correctness of security [16, 18]. To put this in context, if we consider that between January 2004 and December 2008, there have been 26139 reported security vulnerabilities [10]. It would be useful if there existed a framework that decrypted data and then provided some analysis on issues pertaining to the implemented security.

2.3 Detection of encrypted applications

The use of libraries such as openSSL provides a means to add encryption to generic traffic; this creates a problem for the analysis of network traffic as the traffic is now encrypted. For example, most common torrent clients provide a means to encrypt traffic or by means of using an encrypted tunnel provided through SSH as a means to avoid the content blocking of p2p applications. This makes it difficult to block or limit certain types of traffic which may be the goal of a network administrator. Bernaille and Teixeira [2] suggest a system for the early recognition of encrypted applications is outlined and developed with a high degree of success in terms of identification of applications within an SSL connection. They take the approach of using specific parts of the TCP payload to identify the SSL connection by studying said traffic in detail and then producing patterns to be used in detection methods [2] . A similar methodology of analyzing the TCP payloads could be incorporated into the research topic.

3 Cryptography

Cryptography is a common component of any Information Security infrastructure, whether it be for the encryption of large files for secure long term storage or ensuring that communication lines are safe for the transfer of confidential information [17]. In this section I discuss two basic schemes of cryptography, symmetric cryptography and public key cryptography, also outlining cryptographic hash functions.

3.1 Symmetric Cryptography

Symmetric cryptography, also known as secret key cryptography, has been in use since ancient times and has a wide variety of different implementations ranging from simple substitution ciphers such as Caesars Cipher to complex and supposedly "mathematically unbreakable" algorithms such as AES [8]. Symmetric key encryption makes use of a single key that must be kept secret, this key is used for both the encryption and decryption of messages to be sent or stored. I will outline some of these functions, how they work and the relative amount of work required to perform each.

3.1.1 The Data Encryption Standard (DES)

The Data Encryption Standard was developed by IBM and was selected in 1976 as an official Federal Information Processing Standard for the United States [8]. The original DES algorithm used a 64-bit key, of which 8-bits are used for parity and the remaining 56-bits are used to encrypt the plain-text. The required computations for brute forcing a DES key would be 2^{55} operations, given a 64-bit plain-text and 64-bit DES key. While the DES algorithm itself is considered to be resistant to cryptanalysis, the actual keys used for encryption are considered to be fairly weak [15, 5]. The DES algorithm consists of three phases.

Phase 1

The first 64-bits of plain-text, which we will call collectively, x , run through an Initial Permutation function, which we shall denote as IP, returning 64-bits of output, which we will call x_0. We can mathematically represent this as $x_0 = IP(x)$.

The output is separated into equal length sections, obviously consisting of 32-bits each. We will represent this separation as L_0R_0, where L_0 represents the first 32-bits and R_0 represents the remaining 32-bits. Further we define an inverse function of the Initial Permutation function, which we call IIP [5].

Phase 2

The output then undergoes 16 repetitions of a computation that is key dependent using some cipher function, which we shall call f, making use of a key scheduling function which we shall call KS. A key scheduler calculates all the sub-keys for each round or iteration. The output of each iteration or round can be represented as $x_i = L_iR_i$ with $1 \leq i \leq 16$ with $L_i = R_i - 1$ and $R_i = L_i \otimes f(R_i - 1, K_i)$. The K_i's are 48-bit blocks that can be derived from the original 56-bit string using KS [5].

Phase 3

In the final phase, IP is applied to $x_{16}$to give another 64-bit cipher block which we will call C, i.e. $C = IIP(x_{16}) = IIP(R_{16}L_{16})$. We note the inverse property applies, that is $IIP(IP(x)) = x$ [5].

The Cryptographic Hash function, f

Firstly, this function will expand the R_i's from their 32-bit block to a 48-bit block through an expansion permutation. Essentially this function increases the bit length by reusing some of the bits in the $R_i's$, and also re-ordering them making use of a lookup table. We then exclusive-or this output together with K_i[5].

This result is then broken up in 8 blocks of 6-bits each. These 6-bit blocks are then passed through an S-box giving an output of 4-bits. The S-box takes the first bit and the last bit of the input forming a 2-bit binary number. The $base_{10}$ value of this 2-bit number is used to select a row [5]. The remaining inner 4-bits are used to select a column number. These row and column values are used to index a value from the S-box . The 4-bit output of each of these 8 boxes is then concatenated to yield a 32-bit output which is finally given to the permutation function P which gives a result of 32-bits [5].

Key Scheduling

The key scheduling function, KS, is used to make the 48-bit K_i's from the original 56-bit key . We note that while DES keys are 64-bit, only 56-bits are actually used to seed the random functions as 8-bits are used for error checking. Every 8th bit (i.e 8th, 16, 24 ... 64) is used for parity. The key scheduling functions consist of two permutation functions, $PC_1$and PC_2, where PC stands for Permutation Choice.

To select the K_i's we apply the following algorithm. Given a 64-bit key K, we discard the 8-bits used for parity and apply PC_1 to the remainder of the key. This can be represented as $PC_1(K) = C_0D_0$ where C_0 represents the first 28-bits and D_0 represents the remainder. PC itself has two components, with the first half determining C_i and the second half determining D_i. To calculate the individual C_iD_i we apply a LS_i function, which represents the number of left cylindrical shifts, this is a value which is either 1 or 2, by whichC_i or D_i is to be shifted. That is $C_i = LS_i(C_i - 1)$ and $D_i = LS_i(D_i - 1)$.

TheLi function is yet another look up table function. The bits of C_i and D_i are then concatenated together and PC_2 is applied to the output of the concatenation, that is $K_i = PC_2(C_i, D_i)$. For decryption the same key is used, but the order of functions applied is reversed.

3.1.2 Triple DES (3DES)

In 1999 NIST set 3DES as the interim encryption standard for 1999. While 3DES is considered to be more secure than DES, it is also far more computationally intense. We can describe the algorithm as follows [17].

Let $E_k(x)$ and $D_k(x)$ represent the encryption and decryption functions respectively for a given key k. The variable x will represent the 64-bit bit-string that we wish to secure. We can obtain the cipher text from $C = E_{k_3}(D_{k_2}(Ek1(x)))$ and we can obtain the original bit-string by applying the inverse functions in the following way $x = D_{k_1}(E_{k_2}(D_{k_3}(c)))$. For optimal security the three keys should be unique, this corresponds to an actual key strength of 168-bits. We can choose to make two of the keys, $K = K_j$ but $i \neq j$, this reduces the actual key length to only 112-bits [5].

3DES is considered to be the slowest of the 64-bit ciphers in a software implementation, however it is thought to also be the most secure. Hardware accelerators may be used to improve performance. 3DES suffers from potential MITM attacks which allow the keys to manipulated allowing for only 112-bits keys, as two keys are identical. 3DES, like DES, is potentially susceptible to chose and known plain-texts type attacks. However due to the increase in key length it is far more resistant to brute force type cryptanalysis [5].

3.1.3 AES

The AES accepted candidate, Rijndael, was designed by John Daemen and Vincent Rijmen from Belgium and was published in 1998, it is an iterated block cipher allowing for variable key length and allows for a choice from a number of different block size. Rinjndael supports block sizes of 128-bits, 192-bits and 256-bits. Rijndael is byte orientated, compared to the bit orientated nature of DES. The number of rounds or iterations applied is dependent on the sizes of the block and the key used. For example if the block size is 128-bits and if we let m be the size of key and r the number of rounds is given by $r = k/32 + 6$. At the start a 128-bit block of plain-text is used as the initial state. This initial state will be passed through a number of key-dependent transformations, finally returning a 128-bit block of plain-text. A state is treated as a 4x4 matrix, where $A_{i,j}$ will represent a single byte with $0 \leq i, j \leq 3$, i referring to the rows and j referring to the columns. For example $A_{0,0}$ is the first byte and $A_{1,0}$ is the 5th byte. Rijndael makes use of four basics operators to allow for transformation from one state, say $A = (A_{i,j})$, to another state, say. The set of operators used by Rijndael include the following four operators [5].

Operator 1 : Byte Substitution

This is a non-linear permutation that operates on each byte in the current state independently, allowing for parallelism. In this phase we take 8-bytes of the 16-byte phase a multiply them an 8 x 8 matrix, i.e. matrix multiplication of an 8 x 8 matrix by a 8x1 column vector resulting in a 8x1 column vector. This can be efficiently implemented by making use of a 256-bit lookup table or an S-box [5].

Operator 2 : Shift Row

This is a cyclic shift of the bytes in a state. This could be represented as say $B_{i,j} = A_{i,(j+1)} mod 4$. The first row will undergo no changes, however the second row will shift one column, the third row shifts two columns and the third row will shift three columns [5].

Operator 3 : Mix column

Each of the columns A_i undergoes a linear transformation. A transformation is applied to a column at a time and is equivalent to multiplying the columns contents by a 4 x 4 matrix, that is matrix multiplication of a 4 x 4 matrix with a 4 x 1 column matrix containing the columns values [5].

Operator 4 : Round Key Addition

For every round a round key, RK, is generated from the cipher key via the key scheduling function. The round key is the same length as the encryption block and are represented in a 4 x 4 matrix, similar to how the plain-text is represented. We then perform exclusive or the round key with the current state [5].

Key Scheduler

The key scheduler consists of two sections, the key expansion function and key round key selection. A key expansion function is used to expand the cipher key to produce the required number of bits for the round keys. The required number of key bits is equal to $N(R+1)$ where $N$is the required block size and R is the number of rounds to be completed. In Round Key Selection if we assume a block size of 128-bits, after the Key Expansion has taken place, the most significant 128-bits and used for the first round, the next most significant 128-bits are then used for the next round and so forth [5].

The Rijndael Encryption Algorithm

As already mentioned the Rijndael encryption algorithm takes as input a state and produces a state that contains the cipher-text. The algorithm can be described as below

Algorithm 1 Rinjdael Encryption Algorithm

```
RijndaelEncrypt(state, key[0, ..., 4K-1])
//Essentially, take a state containing the plain-text to be encrypted and a K-word cipher stored
in an array called key

InverseKeyExpansion(key[0, ..., K-1], W[0, ..., N(R+1) - 1])
//The first k words of W contain 4k bytes of the key array

AddRoundKey(state, W[0,...,3N])
//Adds the first round key to the state

for( int i = r-2 ; i> 0 ; i–)
{
    InverseByteSubstitution(state)
    InverseShiftRow(state)
    InverseMixColumn(state)
    AddRoundKey(state,W[i,... 3+i])
}

//Do the Final Round

ByteSubstitution(state)
ShiftRow(state)
ByteSubstitution(state)
AddRoundKey(state, W[N(R+1) - 4,...,N(R+1) - 1])

//End of Encryption Algorithm
```

The Rijndael Decryption Algorithm

The decryption of encrypted data is achieved by applying the inverse functions to those used in the encryption phase in the same order.

Algorithm 2 Rinjdael Encryption Algorithm

```
RijndaelDecrypt(state, key[0, ..., 4K-1])
//Essentially, take a state containing the cipher-text a K-word cipher stored in an array called key

InverseKeyExpansion(key[0, ..., K-1], W[0, ..., N(R+1) - 1])
//The first k words of W contain 4k bytes of the key array

AddRoundKey(state, W[0,...,3N])
//Adds the first round key to the state

for( int i = 0 ; i < r - 2; i++)
{
    InverseByteSubstitution(state)
    InverseShiftRow(state)
    InverseMixColumn(state)
    InverseAddRoundKey(state,W[i,... 3+i])
}

//Do the Final Round

InverseByteSubstitution(state)
InverseShiftRow(state)
InverseByteSubstitution(state)
AddRoundKey(state, W[N(R+1) - 4,...,N(R+1) - 1])

//End of Encryption Algorithm
```

3.2 Public Key Cryptography

As mentioned in symmetric key encryption there is difficulty in the distribution of symmetric keys due to the nature of symmetric key encryption i.e. if you have the secret key you can encrypt/decrypt messages, so if the key is stolen through some means the encryption becomes useless. Further for each pair of people who wish to communicate, a key is required to encrypt the communication, this creates a logistical nightmare when trying to manage all the keys that a system may need to communicate. Public key encryption was designed to solve this problem by having a key-pair for each user, a public key that is given out to those who are to receive messages from the system and a private key used to encrypt the message, which is kept secret to the system. Given the public key it should not be computationally feasible to compute the private key, in this way the private key and public key should be related in such a way that it should not be easy to derive the private key from the public key; this usually entails some sort of "unsolved problem" such as the factorization of large numbers or the discrete logarithm problem [17].

3.2.1 Mathematics primer

In order to understand some of the concepts used in public key cryptography, we need a basic understanding of some mathematical concepts, especially abstract algebraic concepts such as groups, co-primes and relatively prime numbers.

Groups

A group is a set of mathematical elements together with a binary operation, that is an operation that takes two inputs and produces a single output, that together satisfy the following four properties. Let G be a group and $a, b, c \in G$ with $*$ the binary operator of G.

1. Closure : If there are two elements $a, b \in G$ then the product $a * b \in G$.
2. Associativity: The defined binary operation, $*$, is associative, that is for $\forall a, b, c \in G$ then $a * (b * c) = (a * b) * c$
3. Identity: There is an identity element $a \in G$ such that $\forall b \in G$, $a * b = b$ and $b * a = b$
4. Inverse: For each element there must exist an inverse. Let $b \in G$ then there must $\exists b^{-1} \in G$ such that$b * b^{-1} = a$ and $b^{-1} * b = a$ where a is the identity of G.

An example group would be say $\mathbb{Z}_{10^*}$, that is the set of integers modulo 10 under the action of integer multiplication [14].

Greatest Common Divisor (GCD)

The greatest common divisor of two positive integers, say $a, b \in G$, is the largest positive integer that divides both integers a and b.The greatest common divisor of a and b is commonly represented as $GCD(a, b)$ [21].

Relatively Prime

Two integers are said to be relatively prime to each other if the largest and thus only positive divisor of the two is the integer one. That is, if $a, b \in G$ then it follows that a and b are relatively prime to each other if and only if $GCD(a, b) = 1$. We note that the terminology "relatively prime" and "co-prime" are effectively equivalent for our purposes [22].

Congruency in Algebra

Two integers are said to be congruent if the two numbers are equivalent modulo n. For example 5 and 11 are congruent modulo 3. [23].

Prime Roots

Let m and p be integers, m is said to be a prime root of p if any integer co-prime to p is congruent to the power of $g \ mod \ n$, if we consider the set of integers under the operation of multiplication modulo 14, then 3 and 5 are the only prime roots modulo 14 [3].

3.2.2 Diffie-Hellman Key exchange

The Diffie-Hellman key exchange algorithm was the first public-key cryptographic scheme to be published and was published in 1976. The scheme exploits the difficulty of the discrete logarithm problem for the field of the multiplicative integers modulo n. The Diffie-Hellman Key exchange protocol allows for the exchange of cryptographic keys through an insecure channel, this provides a solution to the key distribution problem experienced with symmetric key encryption [5].

We can illustrate how this key exchange algorithm works through an example. Lets assume that Alice and Bob wish to share a cryptographic key over an insecure channel. The following series of steps would allow for this [5].

1. Both Alice and Bob decide upon a suitable prime$p$ and an integer m with the properties that m is a prime root of p and that both m and p can be made public.
2. Alice then selects some integer m_a. She then computes $y_a = m^{m_a} \ mod \ p$, and sends this value of y_a to Bob.
3. Bob then selects some integer m_b. He then computes $y_b = m^{m_b} \, mod \, p$, and sends this value of y_b to Alice. y_a and y_b are commonly called Diffie-Hellman public values.
4. Alice then computes K, the secret key, by calculating the value $L = y_b^{m_a}$.
5. Bob then computes K, the secret key, by calculating the value $L = y_a^{m_b}$[5, 15].

We can mathematically prove that both Alice and Bob will arrive to the same value for K. The crux of this protocol lies in the fact that is computationally difficult to calculate m_a or m_b from y_a or y_b respectively. Being able to easily calculate these values would be equivalent to producing a solution or algorithm for solving the discrete logarithm problem.

3.2.3 RSA

While the Diffie-Hellman Key exchange protocol provides a solution to the key distribution problem, it does not provide a practical public key cryptographic system. In 1978 Ronald Rivest, Adi Shamir and Len Adlemar created the first public key cryptographic system. We can describe the processes followed in RSA as below [5].

1. Generate two large primes, which we shall call p and q. The choice of p and q should be uniformly random and they should be of a similar bit length [5].
2. Calculate the product of these two primes, which we will call n i.e.$n = pq$ [5].
3. We then calculate the number of integers that are less than n and are relatively prime to n. This can be calculated making use of the Euler Phi functions, that is $\varphi(n) = (p-1)(q-1)$ where$\varphi(n)$ is the number of integers less than n and relatively prime to n [5].
4. A random number, which we shall call b, is selected with $1 < b < \varphi(n)$ with b relatively prime to $\varphi(n)$. This ensures the existence of a multiplicative inverse [5].
5. Calculate $a = b - 1 \ mod \ \varphi(n)$ [5].
6. We then keep a, p and q secret while making n and b publicly available [5].

Encryption of the plain-text occurs in blocks with each block less than $log_2 n$ bits in length. We can generate the cipher text making use of b and n in the following relation, $c = x^b$. We can regenerate the plain-text by calculating $x = c^a \ mod \ n$.

The crux of the this scheme is the difficulty in factoring large numbers efficiently and task of finding the e_{th} roots modulo a composite number n whose factors are not known, also known as the RSA problem [5].

4 Hash Functions

Cryptographic hashes take a message of arbitrary length and produce a fixed length output which is a called a fingerprint, hash or message digest. Hash functions are used to verify the integrity of messages or files that have been transfered. A good hash function is one that is resistant to collisions. A collision occurs when two messages, x and y with $x \neq y$ but $h(x) = h(y)$. Popular hash functions include SHA-1, MD5, RIPEMD-160 and Tiger [15]. I will consider MD5 as an examples of cryptographic hashes.

4.1 MD5

The MD5 hash function was developed by Ronald Rivest at MIT as an improvement to the existing MD4 hash. The MD5 hash function takes a message, which we will name x, of an arbitrary length and produces a 128-bit hash, which we will represent as $H(x)$. MD5 algorithm consists of the following five phases [5].

1. Appending padding bits : The message is padded with a single 1 bit and a number of 0 bits such that length of the message is a multiple of 512. Mathematically we can represent this requirement as $\|x\| = 448 \ mod \ 512$. Padded bits are always added even if the original message is 64-bits, implying that the number of bits padded on is between 1 and 512 [5].

2. Append the length : The 64-bit representation of the original message is appended to the end of the new padded message. If the length exceeds 2^{64} then only the lower order bits of the message are appended. At this point the message will be exactly divisible by 512 and hence divisible by 16 [5].

3. Initialize the Message Digest Buffer : The buffer used to compute the hash is 128-bit longs. This buffer is formatted as four 32-bit registers labeled A,B,C and D. These registers are initialized to the following values [15].

   A : 01 23 45 67

   B : 89 ab cd ef

   C : fe dc ba 98

   D : 76 54 32 10

4. Process the message : The message is processed as 16 word blocks of 32-bits each. Let X and M denote a word blocks and the message and X[i] an element of that word block. The algorithm that describes the phase is represented below [5].

Algorithm 3 Process message phase of MD5

```
for (int i = 0 ; i < n/16; i++)
{
  for( int j = 0 ; j < 15; j++)
  {
     X[j] = M[i * 16 + j]
  }
  A ∧ A = A //bitwise and of A with itself
  B ∧ B = B
  C ∧ C = C
  D ∧ D = D
  Round1
  Round2
  Round3
  Round4
  A = (A + AA) mod 2^32
  B = (B + BB) mod 2^32
  C = (C + CC) mod 2^32
  D = (D + DD) mod 2^32
}
```

Where Round1 through to Round4 are auxiliary functions that make use of a 64-bit element table T[1...64] where $T[i] = 2^{32} \times abs(sin(i))$. The exact details of these rounds has been omitted but may be found in [5].

4.2 Security and performance of MD5

MD5 is considered to be the fastest hash function when compared to SHA-1, RIPEMD-160 and Tiger. It is approximately three times as RIPEMD-160 and approximately 2.8 times faster than SHA-1 and Tiger. MD5 produces only a 128-bit hash compared to the larger hash sizes produced by the other algorithms and thus is considered the least secure [5].

Table 1: Feature comparision of cryptographic hashes

Algorithm Name	Performance	Hash Length	Messages Required to find Collision
MD5	1	128-bit	2^{64}
SHA-1	2.8	160-bit	2^{80}
RIPEMD-160	3	160-bit	2^{80}
Tiger	2.8	192-bit	2^{96}

5 Cryptographic Protocols

We may define a protocol as a series of steps taken in order to achieve some goal. In the case of cryptographic protocols, the goal is to allow for the secure communication of parties by agreeing

upon some standards that are to be used to encrypt/decrypt the messages sent.

5.1 Architecture of TLS/SSL

It is important to understand the underlying architecture for each of cryptographic protocols. We will consider the architecture of TLS focusing solely on the Handshake Phase, as it is the most significant to the development of the framework. Firstly, we consider some of the goals of SSL/TLS as these goals dictate the structure of TLS [19]. TLS aims to provide a secure connection between two parties with interoperability, extensibility, allowing for incorporation of encryption algorithms or hashing functions and efficiency provided by caching . We will consider basic architecture of TLS as it is very similar to the architecture of SSL 3.0. For our purposes, we need only to consider the Handshake phase of SSL

5.1.1 The Handshake

During this phase decisions are made as to what cryptographic parameters are to be used for the actual TLS connection. This include deciding on the protocol version, selecting a cipher suite and performing some secret key exchange.

The client sends a client hello message to the server. The server then possibly responds with a server hello message. If there is no response then a fatal error occurs and the connection is closed. These hello messages establish: the protocol version to be used, session ID, cipher suite to be used, compression algorithm to use, clientHello.random and ServerHello.random. The actual key exchange may consist of up to four messages containing: the Server Certificate, the Client Certificate, the Server Key exchange and the Client Key exchange. If the Server Certificate is to be authenticated it is sent after the hello messages phase. Following that the Server Key exchange message may be sent if necessary. If the server passes the authentication, it may request the Client Certificate (if the client has one and if it is required by the cipher suite). The server then sends a Hello Done message back to the client indicating the end of the Hello Message part of the handshake is complete. The server then waits for a for a client response. If the certificate request message was sent then the client needs to respond with a certificate. The client will then send its Client Key exchange message with the contents dependent on the public key encryption algorithm chosen. After the exchanges have taken place a Change Cipher Suite Message is sent from the client to server. The client then sends new messages containing the new algorithms and keys. The server responds by sending a Change Cipher Suite Message back with the new keys and algorithms. The handshake is then complete [19].

5.1.2 Practices in SSL/TLS

It has already been mentioned that cryptographic protocols are a popular method of securing web servers. We need to consider that simply providing support for cryptographic protocols is not sufficient to provide adequate security. Lee *et al.* [7]produce a tool, the PSST (probing SSL Security Tool), to perform analysis of over 19000 web servers employing SSL/TLS. They conclude from their results that in 2006, 85.37% of the over 19000 web servers still provided support for SSLv2.0, a fundamentally flawed protocol due to weakness to Man in the Middle (MITM) attacks, while 66.55% of servers still supported DES-40 encryption even though the US export laws limiting the key length of DES to 40-bits is no longer in effect. It is unwise to still provide support for SSLv.2.0 as its well documented that MITM attacks can force the adoption of a weak encryption protocol like DES-40 creating a large and exploitable vulnerability for brute force attacks. While adaption of new algorithms such as AES, is prevalent, the rate at which old standards are no longer being supported is not sufficiently rapid; it is, therefore, important that these issues are highlighted when performing analysis of a systems security.

5.2 SSH

SSH1 and the SSH-1 protocol were developed in 1995 by Tatu Ylönen, a researcher at the Helsinki University of Technology in Finland. Its the logical successor of Telnet, providing encryption to the communications made. Like Telnet, SSH's primary use is to make use of a command line interface on a remote machine, however it can also be used for file transfer and secure RDP. In this way it allows for transparent encryption i.e. the user is unaware of the encryption/decryption occurring in the background. It is critical to realize that SSH is a protocol and not a product, and as such has a number of different implementations. SSH provides its users with three basic security features : authentication, encryption and integrity. SSH provides support for secure remote logins, secure file transfer, secure remote command execution and port forwarding The core of SSH is the Binary Packet Protocol (BPP) performs the underlying symmetric encryption and authentication [4].

5.2.1 Components of SSH

Server and Client

A program on the host machines that handles incoming SSH connection dealing with the authentication and authorization of users. In UNIX this is usually done a program named SSHD but

there are windows implementation such as Bitvise WinSSHD. A client is a program that makes requests for secure remote logins and secure file copy. Typical SSH clients are putty, scp, sftp and Bitvise Tunnelier [4].

Session

A session is a persistent connection made between client and server. The session begins when server authenticates the client and ends once the connection is closed [4].

Keys

Keys are used as the random component to initialize the cryptographic functions. Keys used by SSH are the user key, host key and session key. The user key is the persistent asymmetric key used by the clients used by a server as a way to identify the client. The host key is also an asymmetric key that is used to prove the identity of the server to the client. The session key is a randomly generated, symmetric key for encrypting the communication between an SSH client and server. It is shared by the host and client in a secure manner during the SSH connection setup so that an eavesdropper can't discover it. Both sides then have the session key, which they use to encrypt their communications. When the SSH session ends, the key is destroyed [4].

Key generator

A program that creates persistent keys, for both users and hosts, for SSH. OpenSSH makes use of ssh-keygen [4].

Known-hosts database

A collection of host keys. Clients and servers refer to this database to authenticate one another [4].

Agent

A program that caches user keys in memory, so users do not have to keep retyping their pass phrases. The agent responds to requests for key-related operations, such as signing an authenticator, but it doesn't disclose the keys themselves. It is a convenience feature. OpenSSH and Tectia have the agent ssh-agent, and the program ssh-add loads and unloads the key cache [4].

Architecture of SSH

The SSH protocol consists of four independent protocols listed below

- SSH Transport Layer Protocol (SSH-TRANS) : This is the core component of the protocol allowing for the initial connection to be made, server authentication,basic encryption and integrity services. Once a SSH Transport Layer Protocol connection is made, a ssh client has a full-duplex byte stream connection to an authenticated host [4].
- SSH Authentication Protocol (SSH-AUTH) : Following a successful SSH-TRANS connection, the client may use the SSH-AUTH protocol, using the SSH-TRANS connection, to authenticate with the server. SSH-AUTH defines an abstraction in which many different implementations of authentication could potentially be used, only specifying the format and order of authentication, requirements for success or failure and how a client learns of the available methods [4].
- SSH Connection Protocol (SSH-CONN) : After authentication has occurred SSH client may call the SSH-CONN protocol to provides additional services using the SSH-TRANS connection. These services include support for multiple interactive and non-interactive sessions, terminal handling; data compression; and remote program execution [4].
- SSH File Transfer Protocol (SSH-SFTP) : A client application may use SSH-SFTP over an SSH-CONN channel to allow for secure file transfer for file manipulation [4].

SSH is designed to be modular and extensible. All of the core protocols mentioned above provide abstract services that ensure a minimum level of functionality provide and requirements they must meet, but allow multiple mechanisms for doing so, as well as a way of easily adding new mechanisms. All the critical parameters of an SSH connection are negotiable, including the methods and algorithms used in [4]:

- Session key exchange
- Server authentication, also known as
- Data privacy and integrity
- User authentication
- Data compression

6 Related tools

6.1 SSLdump

SSLdump [12] is an SSL/TLS network protocol analyzer which identifies TCP connections on the chosen network interface and attempts to interpret them as SSL/TLS traffic. When it identifies SSL/TLS traffic it decodes the records and displays them in a textual form to stdout. If given the cryptographic keys involved it can be used to decrypt the traffic passing through.

6.2 SSLsniffer

SSLSniffer [6] provides similar functionality as SSLDump with the exception that it can act as a SSLv3/TLS and SSLv2 proxy server. The issue with these sorts of tools is two-fold, they don't provide any security analysis and further they are protocol specific. I should consider talking about frameworks and development in PHP as well

7 Related Works

7.1 Analysis of traffic and security

The shaping of network traffic is a vital task in modern networks, preventing users from abusing application by the use of p2p, especially in a university type environment. This is usually achieved by making use of a database of known signatures and then comparing against outbound traffic and then "throttling" traffic dependent on user defined rules. We note that the old technique of blocking certain "bad" ports by packet inspection is so longer an accurate means for detection of what application is associated to that packet as most application servers can change the port, or offer a range of ports on which they will accept connections. A similar task is done by NIDS in checking incoming packets for possible viral or hacker type attacks. These scenario's become further complicated by the availability of libraries such as openSSL which allow, with relative ease, the ability to encrypt the communications of an application, this applies both to say adding encryption features to a new p2p type application, or adding that sort of functionality to existing applications such as many Bittorrent clients such as uTorrent which off SSL encryption, or to attackers who choose an attack vector where there packets will be encrypted, such as HTTPS for example. existing NIDS and Traffic Shaper utilities, such as say PFSense, untangle, zeroshell, cannot match these encrypted signatures against anything in there databases and thus cannot

make an intelligent decision as to what to do with these packets. The solution is to build an encrypted network traffic analyzer [2].

7.2 Running Mode Analysis

Running Mode Analysis is a technique for the formal analysis of cryptographic protocols. It makes use of conclusions derived from model checking. The central component of Running Mode Analysis involves creating a system including an attacker, a protocol and two parties attempting communication and then discovering all of the possible modes the system can enter. For example, in a three-principal security system there are seven running modes; if we can show that these seven modes do not exist then the protocol is deemed to be safe within the system. When working with complex protocols, such as SSL, it is a matter of decomposing the more complex protocol into a number of smaller protocols and then performing Running Mode Analysis on each of the simpler protocols. This sort of analysis is often done by hand and provides an interesting means of the verification of the correctness of a protocol. In a by paper Zhang and Liu [24], running mode analysis is performed on the SSL Handshake protocol. While it may not be important to perform such an analysis, as such research already exists; it's important to understand that many protocols are fundamentally flawed and identification of such flaws when providing analysis of application security would be a useful addition.

8 Software framework

A software framework is a reusable design together with an implementation to solve some software problem. The framework represents a model of a problem or problem domain which defines at the least a partial implementation for this model. Framework provide an abstraction in which the code that provides generic functionality can be overridden allowing for specialization. Through the use of good design principles and code reuse, frameworks improve the overall productivity of developers, although frameworks do require a greater "upfront" cost during development in comparison to non-framework development. In this paper we shall consider some of the components that are part of a software framework [13].

9 Conclusion

I have considered some of the core concepts involved in cryptography and cryptographic protocols. Though, I have omitted a considerable amount of work done in this field, due to its vastness. It is clearly apparent that it is no longer possible to be an expert within Information Security but rather an expert in one of its subsidiary fields. Cryptography is a field of great interest both academically and economically and the intelligent use of cryptography will lead to improved user satisfaction and safety when using networks to perform confidential tasks.

10 Acknowledgment

The author would like to acknowledge the support of Telkom SA, Comverse, Tellabs, Stortech, Mars Technologies, Amatole Telecommunication Services, Bright Ideas Project 39, THRIP and the NRF through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

References

[1] Reesa E. Abrams. A checklist for developing software quality metrics. In *ACM 82: Proceedings of the ACM '82 conference*, pages 5–6, New York, NY, USA, 1982. ACM.

[2] Laurent Bernaille and Renata Teixeira. Early recognition of encrypted applications. pages 165–175. 2007.

[3] Karl Dahlke. Modular mathematics, primitive root. Online : `http://www.mathreference.com/num-mod,proot.html`, Accessed 25 June 2009, 2009.

[4] Richard E. Silverman Daniel J. Barrett, Robert G. Byrnes. *SSH, the secure shell, 2nd Edition.* O'Reilly, 2005.

[5] Carlton R. Davis. *IPSec, Securing VPNs.* McGraw Hill Publishers, 2001.

[6] Eu-Jin Goh. Sslsniffer. Online: `http://crypto.stanford.edu/~eujin/sslsniffer/index.html`, Accessed April 2009.

[7] Homin K. Lee, Tal Malkin, and Erich Nahum. Cryptographic strength of ssl/tls servers: current and recent practices. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 83–92, New York, NY, USA, 2007. ACM.

[8] Murdoch Mactaggart. Introduction to cryptography. Online : `http://www.ibm.com/developerworks/library/s-crypt02.html`, Accessed April 2009, 2001.

[9] Marlinkspike. New tricks for defeating ssl inpractice. 2009.

[10] NIST. National vunerability database. Online: `http://nvd.nist.gov/`, Accessed April 2009.

[11] Alan Paller. Top ten cyber security menaces for 2008. 2008.

[12] Eric Rescorla. Ssldump. Online: `http://www.rtfm.com/ssldump/`, Accessed April 2009, 2005.

[13] Dirk Riehle. Framework design - a role modeling approach. WISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH, 2000.

[14] Todd Rowland and Eric Weisstein. Groups, mathsworld. Online : `http://mathworld.wolfram.com/Group.html`, Accessed 25 July 2009.

[15] Schneier. *Applied Cryptography.* Wiley and Sons, 1996.

[16] B. Schneier. Security in the real world: How to evaluate security. *Computer Security Journal, v 15*, pages 1–14, 1999.

[17] B. Schneier and N. Ferguson. *Practical Cryptography.* Wiley Publishing, 2003.

[18] Bruce Schneier. Why cryptography is harder than it looks. Online: `http://www.schneier.com/essay-037.html`, Accessed April 2009, 1997.

[19] C. Allen T. Dierks. The tls protocol version 1.0. *RFC Editor*, 1999.

[20] Huaiqing Wang and Chen Wang. Taxonomy of security considerations and software quality. *Commun. ACM*, 46(6):75–78, 2003.

[21] Eric Weisstein. Greatest common divsor, mathsworld. Online : `http://mathworld.wolfram.com/GreatestCommonDivisor.html`, Accessed 25 July 2009.

[22] Eric Weisstein. Relatively prime, mathsworld. Online : `http://mathworld.wolfram.com/RelativelyPrime.html`, Accessed 25 July 2009.

[23] Eric W Weisstein. Congruence, mathsworld. Online : `http://mathworld.wolfram.com/Congruence.html`, Accessed 25 July 2009.

[24] Yuqing Zhang and Xiuying Liu. Running-mode analysis of the security socket layer protocol. *SIGOPS Oper. Syst. Rev.*, 38(2):34–40, 2004.