CHAPTER-5

INTEGRATED DIFFIE-HELLMAN DIGITAL SIGNATURE (IDHDS) SECURITY ALGORITHM

This chapter deals with the detailed description of the security scenario using the cryptographic techniques and the designing and implementation of the Integrated Diffie-Hellman Digital Signature (IDHDS) with RSA for authentication and encryption of messages. Basic issues in management of key for proposed IDHDS are:

- All cryptographic systems have the problem of how to securely and reliably distribute the keys used
- In many cases, failures in a secure system are not only due to breaking the algorithm but also breaking the key distribution scheme
- Ideally the distribution protocol should be formally verified, recent advances make this more achievable
- Possible key distribution techniques include:
  
  #Physical delivery by secure courier
  - e.g. code-books used submarines
  - one-time pads used by diplomatic missions
  - registration name and password for computers

  #Authentication key server (private key, e.g. Kerberos)
  - have an on-line server trusted by all clients
  - server has a unique secret key shared with each client
  - server negotiates keys on behalf of clients

  #Public notary (public key, e.g. SPX)
• have an off-line server trusted by all clients
• server has a well known public key
• server signs public key certificates for each client

The challenges for implementing security to network is to address the major issues related to it like privacy, data security, confidentiality and authentication. A combination of authentication and encryption has been used to achieve this target. Managing keys for implementation of the security in network systems in an effective and efficient manner is the need of the hour. But computation and distribution of cryptographic keys was a problem for a long. Diffie-Hellman gave the practical solution to compute and exchange cryptographic keys. It is designed to provide users to share a secret key that can be used for encryption of messages between them securely. Three way mechanisms to ensure all three protection schemes of authentication, data security and verification to provide multilevel security has been proposed in this part of the work done and it proposes to make use of digital signature and Diffie Hellman key exchange blended with RSA encryption algorithm to provide data confidentiality. If the key is hacked during transmission then the key is used to decrypt the message thereby breaking the confidentiality of the message. This combination provides a multilevel security model which is very efficient.

During information and data transfer there can be attacks in the form of interruption, interception, modification or fabrication. These attacks are due to some action which compromises the information security. Security services enhance the security of the data processing and transferring. Security mechanisms are for detecting, preventing and recovering from a security attack. Important features of security are confidentiality, authentication, integrity, availability and non-repudiation etc. Cryptography is the study of secret writing which is related to

i) Developing algorithms to conceal the context of some message from all except the sender and the recipient to invoke privacy or secrecy
ii) Verify the correctness of the message and the sender to the recipient which is called authentication.

These two form the basis of many technological solutions to network security Problems. So simply speaking it is the art of transforming an intelligible message into the one which is unintelligible and vice versa. Generally speaking, there are two types of encryption techniques:

1) Techniques based on asymmetric (public key) algorithms, and
2) Techniques based on symmetric (secret key) algorithms. However, hybrid techniques are also commonly used, whereby public key techniques are used to establish symmetric (secret) key encryption keys, which are then used to establish other symmetric (secret) keys. In symmetric cryptography, also called classical cryptography, parties share the same encryption/decryption key. Therefore, before using a symmetric cryptography system, the users must somehow come to an agreement on a key to use. An obvious problem arises when the parties are separated by large distances which are commonplace in today’s worldwide digital communications. If the parties did not meet prior to their separation, how do they agree on the common key to use in their cryptosystem without a secure channel? They could send a trusted courier to exchange keys, but that is not feasible, if time is a critical factor in their communication [5].

5.1 Diffie-Hellman Key exchange Algorithm

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values. The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms.

Discrete Logarithm Problem

Briefly, we can define the discrete logarithm in the following way. Primitive root
of a prime number $p$ is one whose powers modulo $p$ generate all the integers from 1 to $p-1$. That is, if $a$ is a primitive root of the prime number $p$, then the numbers $a \mod p$, $a^2 \mod p$, .... $a^{(p-1)} \mod p$ are distinct and consist of the integers from 1 through $p-1$ in some permutation. For any integer $b$ and a primitive root ‘$a$’ of prime number $p$, we can find a unique exponent $i$ such that $b \equiv a^i \pmod{p}$ where $0 \leq i \leq (p-1)$. The exponent $i$ is referred to as the discrete logarithm of $b$ for the base $a \mod p$. We express this value as $dlog_{a,p}(b)[5]$. For this scheme, there are two publicly known numbers: a prime number $q$ and an integer $\alpha$ that is a primitive root of $q$. Suppose the users A and B wish to exchange a key. User A selects a random integer $X_A$ and computes $Y_A = a^{X_A} \pmod q$. Similarly, user B independently selects a random integer $X_B$ and computes $Y_B = a^{X_B} \pmod q$. Each side keeps the $X$ value private and makes the $Y$ value available publicly to the other side. User A computes the key as $K = (Y_B)^{X_A} \pmod q$ and user B computes the key as $K = (Y_A)^{X_B} \pmod q$. These two calculations produce identical results:

$$
K = (Y_B)^{X_A} \pmod q \\
= (a^{X_B} \pmod q)^{X_A} \pmod q \\
= (a^{X_B})^{X_A} \pmod q \\
= (a)^{X_BX_A} \pmod q \\
= (a)^{X_A}^{X_B} \pmod q \\
= (a^{X_A} \pmod q)^{X_B} \pmod q \\
= (Y_A)^{X_B} \pmod q
$$

by the rules of modular arithmetic. The result is that the two sides have exchanged a secret value. Furthermore, because $X_A$ and $X_B$ are private, an adversary only has the following ingredients to work with: $q$, $a$, $Y_A$, and $Y_B$. Thus, the adversary is forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute $X_B = dlog_{a,q}(Y_B)$. The adversary can then calculate the key $K$ in the same manner as user B calculates it.
Global Public Elements

- $q$ prime number
- $\alpha < q$ and $\alpha$ a primitive root of $q$

User A Key Generation

- $X_A < q$
- Select private $X_A$
- Calculate public $Y_A$
  \[ Y_A = \alpha^{X_A} \mod q \]

**Figure 5.1** shows the key generation scheme for the D-H algorithm[5].

<table>
<thead>
<tr>
<th>Global Public Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$ prime number</td>
</tr>
<tr>
<td>$\alpha &lt; q$ and $\alpha$ a primitive root of $q$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User A Key Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select private $X_A$</td>
</tr>
<tr>
<td>Calculate public $Y_A$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User B Key Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select private $X_B$</td>
</tr>
<tr>
<td>Calculate public $Y_B$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generation of Secret Key by User A</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K = (Y_B)^{X_A} \mod q$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generation of Secret Key by User B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K = (Y_A)^{X_B} \mod q$</td>
</tr>
</tbody>
</table>

**Figure 5.1** Key generation scheme of Diffie-Hellman Algorithm[5]

User B Key Generation

- Select private $X_B$  
  
  $X_B < q$

[120]
Calculate public $Y_B \equiv \alpha^{XB} \mod q$

Calculation of Secret Key by User A
$K = (Y_B)^{XA} \mod q$

Calculation of Secret Key by User B
$K = (Y_A)^{XB} \mod q$

Figure 5.2 shows the key exchange protocol[5]

The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

The Computational Diffie-Hellman Problem is defined as follows: Let $p$ be a prime and let $\alpha$ be a primitive root mod $p$. Given $\alpha^x \mod p$ and $\alpha^y \mod p$, find $\alpha^{xy} \equiv \beta \mod p$. Recall that Eve has access to both $\alpha^x$ and $\alpha^y$ as they are both made
public during the exchange. It is not currently known whether or not this problem is easier than computing discrete logs. A related problem, known as the Decisional Diffie-Hellman Problem, is defined as follows: Let $p$ be a prime and let $\alpha$ be a primitive root mod $p$. Given $\alpha^x \pmod{p}$ and $\alpha^y \pmod{p}$, and $\beta \neq 0 \pmod{p}$, decide whether or not $K \equiv \alpha^{xy} \pmod{p}$. In other words, if someone offers a number to Eve and claims it is $K$, can Eve decide whether or not that person is telling the truth with the information captured in the open channel? Solving these problems Eve can attack the Diffie-Hellman Key Exchange protocol. It may either pretend to be sender or it may alter the messages between the two clients also, it may simply hear to the conversation and compromise the privacy of the communication.[4]

**Man-in-the-Middle Attack:**
Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows.
1. Darth prepares for the attack by generating two random private keys $X_{D1}$ and $X_{D2}$ and then computing the corresponding public keys $Y_{D1}$ and $Y_{D2}$.
2. Alice transmits $Y_A$ to Bob.
3. Darth intercepts $Y_A$ and transmits $Y_{D1}$ to Bob. Darth also calculates $K_2 = (Y_A)^{X_{D2}} \pmod{q}$.
4. Bob receives $Y_{D1}$ and calculates $K_1 = (Y_{D1})^{X_B} \pmod{q}$.
5. Bob transmits $Y_B$ to Alice.
6. Darth intercepts $Y_B$ and transmits $Y_{D2}$ to Alice. Darth calculates $K_1 = (Y_B)^{X_{D1}} \pmod{q}$.
7. Alice receives $Y_{D2}$ and calculates $K_2 = (Y_{D2})^{X_A} \pmod{q}$.

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key $K_1$ and Alice and Darth share secret key $K_2$. All future communication between Bob and Alice is compromised in the following way.
1. Alice sends an encrypted message $M$: $E(K_2, M)$.
2. Darth intercepts the encrypted message and decrypts it to recover \( M \).

3. Darth sends Bob \( E(K_1, M) \) or \( E(K_1, M') \), where \( M' \) is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob. The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates [5].

### 5.2 Authentication and Encryption Algorithms

Authentication and encryption algorithms have been used for design of the proposed integrated: In the digital signature schemes we have public key signature schemes where the private-key signs (creates) signatures, and the public-key verifies signatures. Only the owner (of the private-key) can create the digital signature, hence it can be used to verify who created a message. Anyone knowing the public key can verify the signature, provided they are confident of the identity of the owner of the public key - the key distribution problem. These schemes usually don't sign the whole message (doubling the size of information exchanged), but just a hash of the message. Digital signatures can provide non-repudiation of message origin, since an asymmetric algorithm is used in their creation, provided suitable timestamps and redundancies are incorporated in the signature.

A digital signature (not to be confused with a digital certificate) is a mathematical technique used to validate the authenticity and integrity of a message, software, or digital document. The digital equivalent of a handwritten signature or stamped seal, but offering far more inherent security, a digital signature is intended to solve the problem of tampering and impersonation in digital communications. Digital signatures can provide the added assurances of evidence to origin, identity and status of an electronic document, transaction or
message, as well as acknowledging informed consent by the signer. In many
countries, including the United States, digital signatures have the same legal
significance as the more traditional forms of signed documents. The United States
Government Printing Office publishes electronic versions of the budget, public
and private laws, and congressional bills with digital signatures.

Digital signatures are based on public key cryptography, also known as
asymmetric cryptography[92]. Using a public key algorithm such as RSA, one
can generate two keys that are mathematically linked: one private and one public.
To create a digital signature, signing software (such as an email program) creates a
one-way hash of the electronic data to be signed. The private key is then used to
encrypt the hash. The encrypted hash along with other information, such as the
hashing algorithm is the digital signature. The reason for encrypting the hash
instead of the entire message or document is that a hash function can convert an
arbitrary input into a fixed length value, which is usually much shorter. This saves
time since hashing is much faster than signing. The value of the hash is unique to
the hashed data. Any change in the data, even changing or deleting a single
character, results in a different value. This attribute enables others to validate the
integrity of the data by using the signer's public key to decrypt the hash. If the
decrypted hash matches a second computed hash of the same data, it proves that
the data hasn't changed since it was signed. If the two hashes don't match, the data
has either been tampered with in some way (integrity) or the signature was created
with a private key that doesn't correspond to the public key presented by the signer
(authentication).

A digital signature can be used with any kind of message whether it is
encrypted or not simply so the receiver can be sure of the sender's identity and that
the message arrived intact. Digital signatures make it difficult for the signer to
deny having signed something (non-repudiation) assuming their private key has
not been compromised as the digital signature is unique to both the document and
the signer, and it binds them together. A digital certificate, an electronic document that contains the digital signature of the certificate-issuing authority, binds together a public key with an identity and can be used to verify a public key belongs to a particular person or entity.

Most modern email programs support the use of digital signatures and digital certificates, making it easy to sign any outgoing emails and validate digitally signed incoming messages. Digital signatures are also used extensively to provide proof of authenticity, data integrity and non-repudiation of communications and transactions conducted over the Internet.

5.2.1 Digital Signature Algorithm (DSA):

DSA takes three parameters called Global Public Key Components:

\[ p = \text{prime number where } 2^{1-1} < p < 2^L \text{ for } 512 < L < 1024 \text{ and } L \text{ is a multiple of 64} \]

\[ \text{and } 1024 \text{ bits in increments of 64 bit} \]

\[ q = \text{prime divisor of } (p-1), \text{ where } 2^{159} < q < 2^{260}; \text{ i.e bit length of 160 bits} \]

\[ g = h^{(p-1)/q} \mod p, \text{ where } h \text{ is any integer with } 1 < h < (p-1) \text{ such that } h^{(p-1)} \mod p > 1 \]

User’s Private Key

\[ x = \text{Random or pseudo random integer with } 0 < x < q \]

User’s Public Key

\[ y = g^x \mod p \]

User’s per message Secret Number

\[ k = \text{Random or pseudo random integer with } 0 < k < q \]

To create a signature a user calculates two quantities, ‘r’ and ‘s’, that are functions of the public key components (p,q,g), the users private key(x), the Hash code of the message H(M) and an additional integer K that should be generated randomly or pseudo randomly and be unique for each signing. At the receiving end verification is performed using the formulas as shown below:-

[125]
### Signing:

\[ r = (g^k \mod p) \mod q \]
\[ s = [ k^{-1} (H(M) + x^r) ] \mod q \]

Signature = (r, s)

Figure 5.3 explains the application of digital signature

![Application of digital signature](image)

### Verifying:

\[ w = (S')^{-1} \mod q \]
\[ u_1 = [(H(M')w)] \mod q \]
\[ u_2 = (r') w \mod q \]
\[ v = [(g^{u_1} y^{u_2}) \mod p] \mod q \]

### Test

\[ M = \text{Message to be signed} \]
\[ H(M) = \text{Hash of } M \]
\[ M', r', s' = \text{Received versions of } M, r, s \]

The receiver generates a quantity ‘v’ that is a function of the public key
components, the senders’ public key, and the Hash code of the incoming message. If this quantity matches the ‘r’ component of the signature, then the signature is validated. The test at the end is on the value ‘r’, which does not depend on the message at all. Instead, ‘r’ is a function of K and the three global public key components. The multiplicative inverse of k(mod q) is passed to the function that also has as input the message Hash code and user’s private key[6]. The structure of this function is such that the receiver can recover ‘r’ using the incoming message and signature, the public key of the use and the global public key. Given the difficulty of taking discrete logarithms, it is infeasible for an opponent to recover k from ‘r’ or to recover x from ‘s’. The only computationally demanding task in signature generation is the exponential calculation \( g^k \mod p \), because this value does not depend upon the message to be signed and it can be calculated ahead of time. A user could precalculate a number of values of ‘r’ to be used to sign documents as needed. Similarly determination of the multiplicative inverse \( k^{-1} \) is another demanding task and number of these values can be precalculated.

5.2.2 Secure Hashing Algorithms
# Message Authentication: Message authentication is concerned with, protecting the integrity of a message, validating identity of originator, non-repudiation of origin (dispute resolution) electronic equivalent of a signature on a message, an authenticator, signature, or message authentication code (MAC) is sent along with the message. The MAC is generated via some algorithm which depends on both the message and some (public or private) key known only to the sender and receiver, the message may be of any length, the MAC may be of any length, but more often is some fixed size, requiring the use of some hash function to condense the message to the required size if this is not achieved by the authentication scheme, need to consider replay problems with message and MAC. This require a message sequence number, timestamp or negotiated random values.

# Authentication using Private-key Ciphers: If a message is being encrypted using a session key known only to the sender and receiver, then the message may also be authenticated, since only sender or receiver could have created it. Any interference will corrupt the message (provided it includes sufficient redundancy to detect change) but this does not provide non-repudiation since it is impossible to prove who created the message. Message authentication may also be done using the standard modes of use of a block cipher. If sometimes you do not want to send encrypted messages, one can use either CBC or CFB modes and send final block, since this will depend on all previous bits of the message. No hash function is required, since this method accepts arbitrary length input and produces a fixed output, usually use a fixed known IV. This is the approached used in Australian EFT standards AS8205. Its major disadvantage is small size of resulting MAC since 64-bits is probably too small.

# Hashing Functions: Hashing functions are used to condense an arbitrary length message to a fixed size, usually for subsequent signature by a digital signature algorithm. Good cryptographic hash function $h$ should have the following properties:
• It should destroy all homomorphic structures in the underlying public key cryptosystem (be unable to compute hash value of 2 messages combined given their individual hash values)
• It should be computed on the entire message
• It should be a one-way function so that messages are not disclosed by their signatures
• It should be computationally infeasible given a message and its hash value to compute another message with the same hash value
• It should resist birthday attacks (finding any 2 messages with the same hash value, perhaps by iterating through minor permutations of 2 messages.
• It is usually assumed that the hash function is public and not keyed. Traditional CRCs do not satisfy the above requirements and its length should be large enough to resist birthday attacks (64-bits is now regarded as too small, 128-512 proposed)

# SHA (Secure Hash Algorithms): SHA was designed by NIST and it produces 160-bit hash values. It pad message so its length is a multiple of 512 bits and it initialise the 5-word (160-bit) buffer (A,B,C,D,E) to (67452301, efcdab89,98badcfe,10325476,c3d2e1f0). It process the message in 16-word (512-bit) chunks, using 4 rounds of 20 bit operations each on the chunk & buffer and output hash value is the final buffer value. SHA is a close relative of MD5, sharing much common design, but each having differences.

# SHA-256 SHA-384 Algorithms : SHA-1 produces a hash value of 160 bits. In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512, respectively. Collectively, these hash algorithms are known as SHA-2. These new versions have the same underlying structure and use the same types of modular arithmetic and logical
binary operations as SHA-1. SHA-2 is also specified in RFC 4634, which essentially duplicates the material in FIPS 180-3 but adds a C code implementation. In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on SHA-2 by 2010. Shortly thereafter, a research team described an attack in which two separate messages could be found that deliver the same SHA-1 hash using 2^69 operations, far fewer than the 2^80 operations previously thought needed to find a collision with an SHA-1 hash. This result should hasten the transition to SHA-2. In this section, we provide a description of SHA-512. The other versions are quite similar.

# SHA-512 Logic : The algorithm takes as input a message with a maximum length of less than 2^128 bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks[94]

# Message Digest Algorithms MD5 : The MD5 algorithm is designed to be quite fast on 32-bit machines. In addition, the MD5 algorithm does not require any large substitution tables; the algorithm can be coded quite compactly. The MD5 algorithm is an extension of the MD4 message-digest algorithm. MD5 is slightly slower than MD4, but is more "conservative" in design. MD5 was designed because it was felt that MD4 was perhaps being adopted for use more quickly than justified by the existing critical review; because MD4 was designed to be exceptionally fast, it is "at the edge" in terms of risking successful cryptanalytic attack. MD5 backs off a bit, giving up a little in speed for a much greater likelihood of ultimate security. It incorporates some suggestions made by various reviewers, and contains additional optimizations[93].

# The Blowfish Algorithm: Blowfish is a symmetric encryption algorithm, meaning that it uses the same secret key to both encrypt and decrypt messages. Blowfish is also a block cipher, meaning that it divides a message up into fixed
length blocks during encryption and decryption. The block length for Blowfish is 64 bits; Blowfish is public domain, and was designed by Bruce Schneier[92] expressly for use in performance-constrained environments such as embedded systems. It has been extensively analyzed and deemed "reasonably secure" by the cryptographic community. Encryption and Decryption function of the Blowfish algorithm is as given below.

# Encryption

```c
void Blowfish_encipher(blf_ctx *c,unsigned long *xl,unsigned long *xr)
{
    unsigned long Xl;
    unsigned long Xr;
    unsigned long temp;
    short i;
    Xl=*xl;
    Xr=*xr;
    for(i=0;i<N;++i)
    {
        Xl=Xl^c->P[i];
        Xr=F(c,Xl)^Xr;
        temp=Xl;
        Xr=Xl;
        Xr=temp;
    }
    temp=Xl;
    Xl=Xr;
    Xr=temp;
    Xr=Xr^c->P[N];
    Xl=Xl^c->P[N+1];
    *xl=Xl;
    *xr=Xr;
}
```

# Decryption

```c
void Blowfish_decipher(blf_ctx *c,unsigned long *xl,unsigned long *xr)
{
    unsigned long Xl;
    unsigned long Xr;
    unsigned long temp;
    short i;
    Xl=*xl;
    Xr=*xr;
    for(i=0;i<N;++i)
    {
        Xl=Xl^c->P[i];
        Xr=F(c,Xl)^Xr;
        temp=Xl;
        Xr=Xl;
        Xr=temp;
    }
    temp=Xl;
    Xl=Xr;
    Xr=temp;
    Xr=Xr^c->P[N];
    Xl=Xl^c->P[N+1];
    *xl=Xl;
    *xr=Xr;
}
```
unsigned long Xr;
unsigned long temp;
short i;
Xl=*xl;
Xr=*xr;
for(i=N+1;i>1;--i)
{
    Xl=Xl^c->P[i];
    Xr=F(c,Xl)^Xr;
    temp=Xl;
    Xl=Xr;
    Xr=temp;
}

In this description, a 64-bit plaintext message is first divided into 32 bits. The "left" 32 bits are XORed with the first element of a P-array to create a value I'll call P', run through a transformation function called F, then XORed with the "right" 32 bits of the message to produce a new value F'. F' then replaces the "left" half of the message and P' replaces the "right" half, and the process is repeated 15 more times with successive members of the P-array. The resulting P' and F' are then XORed with the last two entries in the P-array (entries 17 and 18), and recombined to produce the 64-bit ciphertext. Blowfish requires about 5KB of memory. A careful implementation on a 32-bit processor can encrypt or decrypt a 64-bit message in approximately 12 clock cycles. Blowfish works with keys up to 448 bits in length.

5.3 Implementation of IDHDS - The Proposed Algorithm
The idea of applying Digital Signature using RSA is that ‘f’ is function that is known to everyone, but only you know your decryption function. As shown in
Figure 5.4, in order for Alice to sign a message, $m$, sends $g_A(m)$ together with an indication that the message is from Alice. When Bob gets it he sees that the message is from Alice & applies his public encryption function, $f_A$ to $g_A(m)$ and will get $m$ purportedly from Alice to Bob, he has no way of being successful since he does not know $g_A$. That means that you can sign a message without encrypting it. In the scheme described in this section, anyone can intercept Alice’s signed message and read it because her public key is known. Applying encryption in addition to signing a message is quite simple, If Alice wants to sign and encrypt a message, he can do it in the sequence as shown in Figure 5.4. This implies that when Bob applies Alice’s public key to what is received, the result is

$$f_A(g_A(f_B(m))) = f_B(m)$$

then when Bob applies his private key he sees

$$g_B(f_B(m)) = m$$

The plaintext message is received from Alice. The public key of Bob is applied to encrypt the message.

![Figure 5.4 Correct Ordering For Authenticity](image)

Then private key of Alice is used to sign the message. The signed message is then sent to Bob. Bob applies the public key of Alice to authenticate the message and then uses his private key to decrypt the message.
The order of operations for the simultaneous encryption and signing is:

Encrypt $\rightarrow$ Sign $\rightarrow$ Send $\rightarrow$ Authenticate $\rightarrow$ Decrypt

The sequence required to generate the messages for transmission are as follows:

If Alice wants to transmit information to Bob then Alice will use Bob's public key to encrypt the message. This encrypted message may be compressed using gzip. The compressed message is further encrypted with the private key of Alice. Thus the message is digitally signed. This digitally signed message along with digital certificate is transmitted.

Plaintext $\rightarrow$ Encrypt $\rightarrow$ Compress $\rightarrow$ Sign $\rightarrow$ Cipher

\[ M \rightarrow E_{kpB}[M] \rightarrow Z[E_{kpB}[M]] \rightarrow E_{kqA}[Z[E_{kpB}[M]]] \]

where kpB=public key of Bob

kqA=private key of Alice

E=encryption

Z=compression

Cipher $\rightarrow$ Authenticate $\rightarrow$ Decompress $\rightarrow$ Decryption $\rightarrow$ Plaintext

When Bob receives the message Bob applies public key of Alice to authenticate the received message. Then the authenticated version of the message is decompressed using gunzip. The decompressed message is then used for application of private key of Bob to decrypt to the plaintext of the message.

\[ D_{kpA}[E_{kqA}[Z[E_{kpB}[M]]] \rightarrow Z[E_{kpB}[M]] \rightarrow Z^{-1}[E_{kpB}[M]] \rightarrow E_{kpB}[M] \rightarrow D_{kqB}[E_{kpB}[M]] \rightarrow M \]

where kpA=public key of Alice

kqB=private key of Bob

D=Decryption

$Z^{-1}$=decompression
5.3.1 Integrated DH-DS Algorithm (IDHDS)

The integrated Diffie-Hellman Digital Signature algorithm begins with an assumed prime number. Function primitive is used to compute the primitive root of the given prime number. There can be more than one primitive roots of the given prime number. Given coprimes of prime number \( g^n \mod p \) where \( n \) varies from 1 to \( (p-1) \). If the remainders are unique then \( g \) is the primitive root of \( p \). Given \( x \) as the secret key of user A compute \( g^x \mod p \). The user B selects the secret key \( y \). \( y \) is then used to compute \( B \) to \( g^y \mod p \). User A transmits \( A \) to user B and user B transmits \( B \) to A. With this received value \( A \) computes \( k1 = B^x \mod p \). Then user B uses A to compute \( k2 = A^y \mod p \). The computation of \( K1 \) comes the same as computation of \( K2 \) i.e the key computed for application in the encryption process. Once the key is generated we begin to digitally sign the message. Digital signature if applied to a document the sender cannot nonrepudiate. Signature generation begins by selection of public key component p such that 

\[
2^{L-1} < p < 2^L (512 \leq L \leq 1024), \quad L \text{ is a multiple of 64 bits.}
\]

Then select \( q \) such that it is prime divisor of \( (p-1) \). Initialize \( i \) to 2. Start the while loop to check if remainder of \( (p-1)/q \) is 0 then set \( q=1 \) else increment \( i \). Choose \( h \) such that it is an integer in the range 1 and \( (p-1) \). Compute \( t \) such that \( h^{(p-1)/q} \mod p \).

Choose \( x1 \) as pseudorandom number between 0 and \( q \) and compute \( y1 \) as \( (t^{x1} \mod p) \). Then choose a pseudorandom value for \( k \) between 0 and \( q \). To compute \( r \) component of digital signature we apply \( (t^k \mod p) \mod q \). The value of \( r \) is independent of either the input message or hash of the message. It is a function of \( k, p, q \) and \( t \). The value computed for \( r \) is secret since it involves the use of \( k \) which is a one time secret number between 0 and \( q \). Then we input the integer or alphabetic message in \( \text{msg} \). The \( \text{msg} \) in textual form is liable to spoofing so hash of the \( \text{msg}(\text{digest1}) \) is computed using \( \text{md5\_hex}(\text{msg}) \). This function accepts as input variable size message and generates a hash value of 128 bits. This hash value is then utilized in the computation of \( s \) value of digital signature as \( S1 = [k^{\text{digest1}} \mod p] \mod q \).
\[ P \text{digest1} + x1 \times r \] \% q where \( x1 \) is a random number between 0 and q. The value of \( S1 \) computed is then transmitted to the receiver. Thereafter \( \text{sha1}_\text{hex}(\text{msg}) \) is applied to generate hash of the message using \( \text{sha1} (\text{digest2}) \). This function takes as input a message of any length less than \( 2^{64} \) and produces as output hash value of 160 bits. The variable \( \text{digest2} \) is then used to compute \( S2 \) as \( [k^{-1} \times \text{digest2} + x1 \times r] \% q \). After this, \( \text{sha256}_\text{hex}(\text{msg}) \) is used. This function takes as input message of length less than \( 2^{64} \) and produces hash value of 256 bits. This generates hash value \( \text{digest3} \) which is used to compute \( S3 \) as \( [k^{-1} \times \text{digest3} + x1 \times r] \% q \). Then \( \text{sha384}_\text{hex}(\text{msg}) \) is applied. This function accepts a message of any length less than \( 2^{128} \) and produces a hash value of length 384 bits. \( \text{sha384}_\text{hex}(\text{msg}) \) operates on 64 bit words which computes \( \text{digest4} \) to be used for computation of \( S4 \) as \( [k^{-1} \times \text{digest4} + x1 \times r] \% q \). Lastly we apply \( \text{sha512}_\text{hex} \). This function takes as input a message of length less than \( 2^{128} \) and produces a hash value of length 512 bits. The hash produced is \( \text{digest5} \) to be further used in \( S5 \) computation as \([k^{-1} \times \text{digest5} + x1 \times r] \% q \). Blowfish algorithm is then applied to encrypt the message.

After the \( r \) and \( s \) components are computed, this forms the digital signature generation, then the digital signatures are to verified. The value of \( w1 \) is calculated as \((1/S1') \% q\) where \( S1' \) is the received value of \( S1 \). The value of \( \text{digest1}' \) is used to calculate \( u1 \) as \((\text{digest1}' \times w1) \% q\) where \( \text{digest1}' \) is the received value if \( \text{digest1} \). Next use \( r' \) value to compute \( u2 \) as \((r' \times w1) \% q\) where \( r' \) is the received component of \( r \). After \( u1 \) and \( u2 \) gets computed we finally calculate \( v \) as \( \left( (g^{u1} \times y \times u2) \% p \right) \% q \). If the calculated \( v \) value comes equal to \( r' \) value then the signature is verified.

\begin{verbatim}
Begin
Prime \( p \)
int g = sub primitive(p)
x \downarrow secret key of user A
A \downarrow g^x \% p
Y \downarrow secret key of user B
B \downarrow g^y \% p
k1 \downarrow B^x \% p
k2 \downarrow A^y \% p
\end{verbatim}
k1=k2=\text{key for encryption}(k)\\ i=2\\ \text{while}(i<p-2)\\ \hspace{1em}\text{start}\\ \hspace{2em}\text{if}((p-1)\%i==0) \text{ then}\\ \hspace{3em}q=i\\ \hspace{2em}\text{else}\\ \hspace{3em}i=i+1\\ \hspace{1em}\text{end}\\ q\uparrow \text{prime divisor of } (p-1)\\ h\uparrow \text{integer between } 1 \text{ and } p-1\\ t\uparrow h^{((p-1)/q)}\%p\\ x_1\uparrow \text{random number between } 0 \text{ and } q\\ y_1\uparrow (t^{x_1})\%p\\ k\uparrow \text{random number with } 0<k<q\\ r\uparrow (t^k)\%q\%q\\ \text{msg}\uparrow \text{input message}\\ \text{digest1}\uparrow \text{md5\_hex}(\text{msg})\\ S1\uparrow (k^{-1} \ast \text{digest } 1 + x_1 \ast r) \% q\\ \text{digest2}\uparrow \text{sha1\_hex}(\text{msg})\\ S2\uparrow (k^{-1} \ast \text{digest } 2 + x_1 \ast r) \% q\\ \text{digest3}\uparrow \text{sha256\_hex}(\text{msg})\\ S3\uparrow (k^{-1} \ast \text{digest } 3 + x_1 \ast r) \% q\\ \text{digest4}\uparrow \text{sha384\_hex}(\text{msg})\\ S4\uparrow (k^{-1} \ast \text{digest } 4 + x_1 \ast r) \% q\\ \text{digest5}\uparrow \text{sha512\_hex}(\text{msg})\\ S5\uparrow (k^{-1} \ast \text{digest } 5 + x_1 \ast r) \% q\\ E=\text{blowfish}(\text{msg});\\ W\uparrow (1/S1') \% q\\ u_1\uparrow (\text{digest1'*w}) \% q\\ u_2\uparrow (r'*w) \% q\\ v\uparrow (\text{g}^u_1 \ast (y_1 \ast u_2) \% p)\%q\\ v= r' \{ \text{signature verified}\}

\text{end}\\ \text{sub primitive}(p)\\ \hspace{1em}\text{for } n=1 \text{ to } p-1\\ \hspace{2em}\text{do}\\ \hspace{3em}y=\text{g}^n \% q\\ \hspace{4em}\text{if } y \text{ unique}\\ \hspace{4em}\text{return } y

[137]
5.4 Results

Table 5.1 shows the computed key and r components of the digital signature for given prime numbers. DSA involves modular exponentiations therefore RSA based signature with RSA key will includes 1024 bit value. More the key lengths more is the security of the system. The computed key value is a function of primitive root of prime number and also depends on chosen secret. This computed key can be further used in encryption. The ‘r’ component increases with prime numbers generally with exceptions.

<table>
<thead>
<tr>
<th>Prime Number</th>
<th>Computed key</th>
<th>h value</th>
<th>‘r’ Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>4</td>
<td>23</td>
<td>9</td>
</tr>
<tr>
<td>37</td>
<td>33</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>43</td>
<td>27</td>
<td>37</td>
<td>7</td>
</tr>
<tr>
<td>53</td>
<td>43</td>
<td>47</td>
<td>1</td>
</tr>
<tr>
<td>103</td>
<td>66</td>
<td>91</td>
<td>18</td>
</tr>
<tr>
<td>293</td>
<td>61</td>
<td>219</td>
<td>132</td>
</tr>
<tr>
<td>337</td>
<td>273</td>
<td>331</td>
<td>79</td>
</tr>
<tr>
<td>557</td>
<td>471</td>
<td>551</td>
<td>113</td>
</tr>
<tr>
<td>997</td>
<td>963</td>
<td>991</td>
<td>248</td>
</tr>
<tr>
<td>65537</td>
<td>12365</td>
<td>65531</td>
<td>21949</td>
</tr>
</tbody>
</table>

It depicts that as the prime number is increased the h value increases but the r component shows a decrease in value at some places but the computed key increases accordingly. The Graph 5.1 shows that with the prime number increase
in the range of 37% to 43% computed key shows an increase from 50% to 67%.

Graph 5.1 Computed Keys and ‘r’ for given Prime Numbers

**MD5 in IDHDS (r and s):** Table 5.2 shows the ‘r’ and ‘s’ components of the digital signature using MD5 hash algorithm in IDHDS

**Table 5.2  ‘r’ and ‘s’ components of DS with MD5 in IDHDS**

<table>
<thead>
<tr>
<th>‘r’ component of the digital signature</th>
<th>‘s’ component of the digital signature using MD5 in IDHDS algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>value of ‘r’</td>
<td>MD5</td>
</tr>
<tr>
<td>58</td>
<td>27</td>
</tr>
<tr>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>88</td>
<td>56</td>
</tr>
<tr>
<td>60</td>
<td>66</td>
</tr>
<tr>
<td>155</td>
<td>52</td>
</tr>
<tr>
<td>107</td>
<td>31</td>
</tr>
<tr>
<td>54</td>
<td>140</td>
</tr>
<tr>
<td>84</td>
<td>6</td>
</tr>
</tbody>
</table>
Graph 5.2 'r' and 's' components of digital signature with MD5 in IDHDS

Table 5.2 and Graph 5.2 show the computed ‘r’ and ‘s’ values for the proposed IDHDS algorithm using MD5 hash algorithm. The values of ‘r’ and ‘s’ shall be checked to determine if r=0 or s=0. If either r=0 or s=0 a new value of k will be generated and the signature have to be recalculated, however it extremely unlikely that r=0 or s=0. Values of r and s computed provide valid signature which is transmitted along with the message to the verifier.

SHA in IDHDS (r and s): Table 5.3 shows the ‘r’ and ‘s’ components of digital signature using SHA hashing algorithms in IDHDS Algorithm

Table 5.3  'r' and 's' components of digital signature with SHA in IDHDS

<table>
<thead>
<tr>
<th>'r' component of the digital signature</th>
<th>'s' component of the digital signature using IDHDS Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SHA1</td>
</tr>
<tr>
<td>58</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>54</td>
</tr>
<tr>
<td>88</td>
<td>47</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>155</td>
<td>43</td>
</tr>
<tr>
<td>107</td>
<td>25</td>
</tr>
<tr>
<td>54</td>
<td>134</td>
</tr>
<tr>
<td>84</td>
<td>154</td>
</tr>
</tbody>
</table>
Graph 5.3 shows the variation of ‘r’ and ‘s’ components of digital signature with SHA in IDHDS. The values of ‘r’ and ‘s’ shall be checked to determine if r=0 or s=0. If either r=0 or s=0 a new value of k will be generated and the signature have to be recalculated, however it extremely unlikely that r=0 or s=0. Values of r and s computed provide valid signature which is send along with message to the verifier.

Execution Time: Table 5.4 shows the execution time of Diffie-Hellman, Digital Signature Algorithm and IDHDS for different number of rounds.

Table 5.4 Execution Time of the D-H, DS, IDHDS
Graph 5.4 Comparative Execution Time of the DH, DS, IDHDS

Graph 5.4 shows the comparative execution times for 30 rounds of the proposed IDHDS algorithm, D-H and DS. Execution time of the proposed algorithm is much less as compared to other two algorithms.

Performance Analysis of IDHDS

The time difference required for execution of integrated DHDS algorithm textual messages and integer messages is about 5-6%. Time complexity of the algorithm is $O(\log k \ M(n))$ where $M(n)$ is complexity of multiplying two n bit integers. Signature generation requires one modular exponentiation so its complexity is $O(\log^3 n)$ and signature verification requires two modular exponentiation so its complexity is $2O(\log^3 n)$ since here $k=O(n)$ and time complexity of multiplying two n bit integers is $O(\log^2 n)$.

5.5 Proposed Hypothetical Security Model

In the end hypothetical security model has been proposed by combining the smart snort and the IDHDS. It incorporates the implementation of the proposed encryption and authentication algorithm IDHDS with smart Snort which has been discussed in this and the previous chapter. Basic block diagram of this proposed model is shown in the Figure 5.5:
1. Suppose a packet arrives and is logged in event logs. It is encrypted using IDHDS algorithm and its pattern is matched with encrypted pattern database (signature database) already stored in the system. Signature database is specifically known patterns of unauthorized behaviour. If it matches with the signature database that means it is an attack packet so an alert is generated.

2. If the pattern does not match it is sent for a second stage check Anomaly Detection which is statistical behaviour analysis. If it is not the normal behaviour the signature database is updated to include this new signature.

3. If it shows a normal behaviour then packet data is decrypted and if it does not match then system alerts for deny access.
4. This way evasion of IDS may be reduced to some extent at important checkpoints. The model include IDHDS algorithm for Encryption and smart snort for pattern matching.

5.6 Conclusion

This chapter includes the proposed algorithm IDHDS for key generation, Digital signature generation and Encryption using Blowfish Algorithm. The sequence for authenticity is proposed to include encryption, compression, Digital signature and transmission. The computation of s component of Digital signature is done using MD5 and SHA (various methods) algorithms for hash computation. The results show variations of r and s values of Digital signature using hashing algorithms. Execution time of IDHDS is less than Diffie Hellman and Digital signature. In the end a hypothetical security model is proposed which includes IDHDS for Digital signature generation and verification and Encryption with smart snort for fast pattern matching.