Malaysian Journal of Computer Science, Vol. 15 No. 1, June 2002, pp. 1-14

THE DESIGN OF AUTHENTICATED TELNET PROTOCOL TO ENHANCE CRYPTOGRAPHY AND SECURITY

Yi-Shiung Yeh and Wei-Shen Lai
Department of Computer Science and Information Engineering
National Chiao Tung University
Hsinchu, Taiwan 30043, R.O.C.
Phone: 886-3-5731813
Fax: 886-3-5724176
email: ysyeh@csie.nctu.edu.tw
wslai@csie.nctu.edu.tw

ABSTRACT

In this paper, an authenticated TELNET protocol is proposed to establish TELNET connections with authentication and security. To achieve this goal, specific cryptographic algorithms and certificates are used to resist attacks such as eavesdropping, tampering, forgery and personating.

The concept of designing this proposed protocol is to negotiate one cipher spec (one suite of cryptographic algorithm) and to generate the shared secret. Then, apply them to protect the communication securely. As for extensibility, this system also provides a framework by means of which new cryptographic algorithms can be incorporated if necessary. In consideration of compatibility issue, we define the command names and codes following the rules of the TELNET protocol.

Overall, this system enhances the security of TELNET protocol with NMAC, and it is compatible with the current TELNET protocol.

Keywords: ***TELNET protocol, SSL, TLS, Network protocol, NMAC***

1.0 INTRODUCTION

1.1 Motivation

The primary goal of the TELNET [1], [2] protocol is to regulate a set of standardised specifications for terminal devices and terminal-oriented processes to interface with each other. In a like manner, it can be seen that the TELNET protocol may also be used for terminal-terminal communication ("linking") and process-process communication (distributed computation). With this protocol, users can log on a remote server to extract a service from any terminal. However, it is to be emphasised that any data transmitted through such connection is in an insecure format. In other words, the TELNET applications following RFC 854 provide no security measures. For example, an intruder can use an eavesdropping tool to obtain a password or confidential information effortlessly.

Since SSL [3] and the TLS [4] have been widely implemented in network applications, we refer to these concepts to develop the authenticated TELNET protocol and reinforce it by using non-deterministic-MAC method in providing the users with a secure communication environment.

1.2 Objects

The goals of the authenticated TELNET protocol include the following:

Peer Authentication	Both ends (the client and the server) of the connection can authenticate each other. One side of the connection may reject the connecting request if the other side cannot be authenticated.
Data Confidentiality	All data sent and received by both sides of the connection are encrypted so that any intruder listening to the communication is unable to retrieve the original data.
Extensibility	Whenever the cryptographic algorithms used in this study are no longer secure or it can be easily breakable, there is no need to propose a new standard. Any existing and new algorithms can be incorporated into the authenticated TELNET protocol as required.

Compatibility — This authenticated system complies with the TELNET protocol, i.e., there will be no inconvenience caused if a common TELNET program is used to log on a typical TELNET server.

2.0 BACKGROUND

2.1 TELNET Protocol

The purpose of the TELNET Protocol is to provide a fairly general, bi-directional, byte (eight-bit) oriented communication facility. It permits users to utilise services on a remote machine. A TELNET connection is a Transmission Control Protocol (TCP) connection used to transmit data with interspersed TELNET control information.

The TELNET Protocol is established based on two main ideas:
1) the concept of a "Network Virtual Terminal",
2) the principle of negotiated options.

2.1.1 Network Virtual Terminal (NVT) [1, 2]

As we establish a TELNET connection, each end is assumed to commence and complete the communication at NVT. NVT is an imaginary device that provides a standard, network-wide, and intermediate representation of a canonical terminal. With this advantage, it eliminates the need for all hosts to retain information regarding characteristics of each terminal and their associated convention of terminal handling. Terminals of the client and the server can correspond to each other through the NVT to obtain the proper transformation.

2.1.2 Command and Control Negotiation [1, 2]

Interpret As Command, IAC, is a special character inserted in the data stream when both sides of the connection decide to send a command sequence. Basically, All TELNET commands consist of at least a two-byte sequence - *IAC code*. In addition, commands dealing with option negotiation can contain three-byte sequences - *IAC code option,* which can further minimise collisions if more comprehensive use of data space is needed.

The principle of negotiated options arises from the fact that many hosts want to provide additional services over and above those that are available within an NVT. Various independent and structured options may be added and can be used in conjunction with the "WILL, WONT, DO, DONT" structure to allow the client and the server to agree to a more sophisticated set of conventions as their TELNET connection.

The basic strategy of setting up the usage of options is to permit either side (or both) to initiate a request for some options to take effect, whereas the other side may either accept or reject the request. Certain command options require one or more parameters before effects can take place. In fact, the specification of TELNET protocol supports this by using the sub-negotiation commands: SB and SE. The SB and SE commands will be utilised to satisfy this requirement with their format being written in the form of: *IAC SB code sub-negotiation IAC SE*.

2.2 Cryptographic Algorithm [5]

2.2.1 Secret-Key Cryptography

Secret-key cryptography, also known as symmetric cryptography, uses the same key to encrypt and decrypt the messages. Therefore, the sender and the receiver of a message must share a secret; namely, the key. In this work, it is intended to design the protocol in such a way that the secret-key cryptography can be used to encrypt and decrypt the message to exchange data stream between the client and the server.

2.2.2 Public-Key Cryptography

Public-key cryptography, also known as asymmetric cryptography, comprises two keys: one key to encrypt the message and the other one to decrypt the message. These two keys are mathematically interrelated so that data encrypted with either key can only be decrypted using the other. Each user (the client or the server) possesses two keys: a public key and a private key. The owner that holds two keys can distribute the public key using the

certificate. Because of the correlation existing between these two keys, anyone receiving the public key can be assured that data encrypted with the public key and sent to the receiver can only be decrypted by the owner when applying the private key. In this proposed protocol, the client encrypts the shared secret first using the public key distributed by the server and sends the message to the server. Afterwards, the server decrypts the message with its private key and gets the shared secret.

2.2.3 Digital Signature

Digital signatures are capable of providing two important features. One of them is to prove the information to be generated by the owner, and the other one is to assure that the information is original. The certificate will use these functions to verify the identification. Here, we apply these capabilities to our proposed system to permit the client and the server to authenticate each other.

2.2.4 One-Way Hash Function

A cryptographic hash function is a mathematical transformation that takes information of arbitrary length and converts it into a fixed-length number. It is infeasible to computationally find two chunks of message that hash to the same. We apply the hash function to the data stream concatenating with secret-key cryptography to generate a message authentication code - MAC.

3.0 PROPOSED DESIGN

3.1 Authenticated TELNET Protocol Overview

The authenticated TELNET protocol is composed of two phases - negotiation phase and application phase. In the first phase, the client and the server can authenticate each other, negotiate the cryptographic parameters (e.g., the public-key cryptography, the secret-key cryptography, plus the hash function), and generate a shared secret. In the second phase, application phase, data stream is protected using the secret-key cryptography and the MAC algorithm defined in the negotiation phase.

The following procedures are necessary as the client and the server first start their communication: selecting cryptographic algorithms, authenticating each other, and using public-key cryptography to exchange the shared secrets. These processes are performed in the negotiation phase, which can be described as follows.

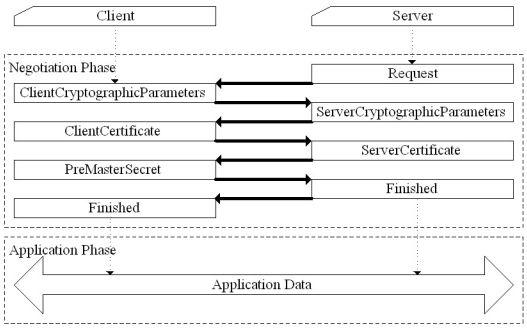

Fig. 1: Option of sub-negotiation in accordance with authenticated TELNET

As shown in Fig. 1, the negotiation phase starts with a request message. The request message sent by the server is a simple notification, indicating that the client should begin the negotiation phase by sending a message containing client cryptographic parameters. The messages of cryptographic parameters are used to establish the cipher spec (including the public-key cryptography, the secret-key cryptography, and the hash function). Additionally, two partial secrets are generated and exchanged.

Following the transmission of cryptographic parameters, the client and the server must exchange certificate with each other by using the certificate messages, and explicitly verify their certificate. Thus they can confirm the identification of whom they expect to connect with.

At this point, the client and the server have finished negotiating a cipher spec and authenticating to each other. They must further generate the secret used in the application phase to protect the data stream. The client generates a pre-master secret and encrypts it using the public-key cryptography as defined in the cipher spec to couple with the server's certificate. And then the client sends the encrypted pre-master secret to the server. Until now, the client and the server have applied the same algorithm (described in Section 3.2.2) to compute the master secret to be derived from the pre-master secret and two partial secrets. This master secret is then used to derive a secret block to be generated as keys or MAC-secrets used in the application phase.

By far, the client and the server have determined the cipher spec, the keys, and the secrets. They send the finished messages to notify that the negotiation phase is accomplished. The finished message is protected with the negotiation digests (described in Section 3.2.2). After sending the finished message, the negotiation phase is complete and the client and the server may start to enter the application phase.

In the application phase, all of the data streams will be exchanged under the protection of the secret-key cryptography and the MAC algorithm. As discussed in the previous sections, the secret-key cryptography and the hash function used in the MAC algorithm were determined in the negotiation phase. That is, both keys and MAC-secrets essential to the secret-key cryptography and the MAC algorithm respectively have already been derived in the negotiation phase.

3.2 Negotiation Phase

3.2.1 Command Name and Codes

Table 1: Command names and codes

Name	Code
ATELNET	43 (provisional)
REQUEST	0
CLIENT_CRYPTOGRAPHIC_PARAMETERS	1
SERVER_CRYPTOGRAPHIC_PARAMETTERS	2
CLIENT_CETIFICATE	3
SERVER_CERTIFICATE	4
PRE_MASTER_SECRET	5
FINISHED	6
ERROR	7

Table 2: The definition of cipher suite

Name	Code
RSA_3DES-EDE-CBC_MD5	0x010101
RSA_3DES-EDE-CBC_SHA-256	0x010102
RSA_3DES-EDE-CBC_RIPE-MD	0x010103
.....	
ECC_AES-128-CBC_SHA-256	0x020302
ECC_AES-128-CBC_RIPE-MD	0x020303

Note: A cipher suite comprises three cryptographic algorithms: the public-key cryptography, the secret-key cryptography, and the hash function. In this paper, we have defined several types of cipher suite (e.g., RSA_3DES-EDE-CBC_MD5, RSA_3DES-EDE-CBC_SHA-256, etc.) and presented it in the appendix. Any additional cipher suite can be incorporated as needed. As a result, it allows the users to define new cipher suites or cryptographic algorithms (e.g. ElGamal [6], IDEA [7] or MD5 [8], etc.) if necessary.

3.2.2 Command Meanings

This paper makes reference to a 'server' and a 'client'. For the convenience of writing this paper the 'server' is defined as the part of the connection that makes the passive TCP open (TCP LISTEN state), and the 'client' is defined as the part of the connection that makes the active TCP open.

IAC WILL ATELNET

The client end of the connection sends this command to indicate the willingness of sending and receiving the authentication information and to start negotiating the cipher suite.

IAC DO ATELNET

The server end of the connection sends this command to indicate the willingness of sending and receiving the authentication information and to start negotiating the cipher suite.

IAC WONT ATELNET

The client end of the connection sends this command to indicate the refusal of sending and receiving the authentication information in negotiating the cipher suite; the server side sends this command if a DO ATELNET command is received.

IAC DONT ATELNET

The server side of the connection sends this command to indicate the refusal of sending and receiving the authentication information in negotiating the cipher suite; the client side sends this command if a WILL ATELNET command is received.

IAC SB ATELNET REQUEST IAC SE

The server side of the connection sends this command to request the client to start the negotiation phase. Only the server side (DO ATELNET) is permitted to send this command. However, if the authenticated TELNET option of sub-negotiation is still in process after this command has been sent from the server, then the client end should ignore it.

IAC SB ATELNET CLIENT_CRYPTOGRAPHIC_PARAMETERS <partial secret> <cipher suite list> IAC SE

The client side of the connection sends this command to request the server to make a final selection from a cipher suite list sent to the server earlier. This cipher suite list is a sorted list of cipher suites, which are sorted according to the client's preferences. Only the client side is allowed to send this command.

This command has another parameter, partial secret, located prior to the cipher suite list. This partial secret is 64-byte in length, which is long enough to be an input for most of the hash function. Besides, this partial secret must be large and random enough to be different from that of the server. It may comprise the pseudorandom number, the time stamp, and the nonce, etc. The data structure, described as XDR [9], of this parameter may be defined as follows:

```
struct {
        opaque pseudorandom[32];
        opaque timestamp[4];
        …..;
} PartialSecret;
//determined by the implementer
```

It is recommended that the length of the partial secret is larger than the input size of the hash function used to compute the master secret and the secret block.

IAC SB ATELNET SERVER_CRYPTOGRAPHIC_PARAMETERS <partial secret><cipher suite> IAC SE

The server side of the connection sends this command to indicate the completion of selecting the cipher suite. However, the following rule must be kept in mind: The public-key cryptography as defined in the cipher suite must be capable of fitting the server's certificate. The server must comply with this constraint because the public-key cryptography in association with the server's certificate is used to encrypt the pre-master secret in the PER_MASTER_SECRET command. If no suitable cipher suite is available, the server should return an error message of no cipher suite and close the connection. Only the server side is allowed to send this command.

IAC SB ATELNET CLIENT_CERTIFICATE <certificate list> IAC SE

After both sides of the connection have determined the cipher spec, the client side of the connection sends a certificate list about the local use to the server upon receiving the SERVER_CRYPTOGRAPHIC_PARAMETERS command. The server will explicitly verify the certificate and authenticate the local user. The certificate list is a sequence of the X.509.v3 certificates, which is set in order with the local user's certificate first and the root certificate authority last. It is not necessary to have the feature of the public-key cryptography as far as the local user's certificate is concerned. If no certificate is available, the client should send an error message of no certificate and close this connection. Only the client side is allowed to send this command.

IAC SB ATELNET SERVER_CERTIFICATE <certificate list> IAC SE

Upon receiving the CLIENT_CERTIFICATE command, the server will verify the client's certificate and authenticate the client's identification. If the client's certificate is invalid (e.g., corrupt, revoked, and expired, etc.), the server should return an error message of invalid certificate to client and close this connection. If the client's certificate is valid, and the server will send a certificate list to the client. This certificate needs to match with public-key cryptography as defined in the cipher spec. Technically, the public-key cryptography together with the server's certificate is used to encrypt the pre-master secret in PRE_MASTER_SECRET command. Only the server side is allowed to send this command.

IAC SB ATELNET PREMASTER_SECRET <encrypted pre-master secret> IAC SE

Upon receiving SERVER_CERTIFICATE command, the client will verify the server's certificate and authenticate the server's identification. If the server's certificate is invalid, the client will return an error message of invalid certificate to the server and close this connection. If the server's certificate is valid, this means that the client and the server have already authenticated each other by now. The client will generate a pre-master secret of which both sides need it to compute the master secret. The pre-master secret must be encrypted with the public-key cryptography in association with the server's certificate before being transmitted. Only the client side is allowed to send this command. The pre-master is 64-byte in length and the pre-master secret may comprise the pseudorandom number, the time stamp, and the nonce, etc. In addition, the partial secret must be sufficiently large and random, and the data structure of the partial secret may be defined as follows:

```
struct {
    opaque pseudorandom[32];
    opaque timestamp[4];
    …..;
} PreMasterSecret;
//determined by the implementer
```

The client and the server compute the master secret and the secret block using the following algorithms (the client and the serve must perform the same algorithm.):

```
master_secret =
    Hash(pre-master_secret + Hash(0x00 + pre-master_secret + client's partial_secret
        + server's partial_secret))
    + Hash(pre-master_secret + Hash(0x01 + pre-master_secret + client's partial_secret
        + server's partial_secret))
    …..
    + Hash(pre-master_secret + Hash(iteration + pre-master_secret + client's partial_secret
        + server'spartial_secret));
```

Note: The meaning of iteration is defined in Section 4.0.

The pre-master secret should be deleted from memory immediately after the master secret has been computed.

The purpose of the master secret is to generate the keys and the secrets to be used in the application phase. The procedure, a fixed method, of computing the keys and the secrets is shown as follow:

```
secret_block =
    Hash(master_secret + Hash(0x00+master_secret + client's partial_secret + server's partial_secret))
    + Hash(master_secret + Hash(0x01+master_secret + client's partial_secret + server's partial_secret))
    + Hash(master_secret + Hash(0x02+master_secret + client's partial_secret + server's partial_secret))
    + […];
```

This procedure must generate enough output in terms of its size for both the keys and the secrets. The secret block is partitioned as follows:

client_MAC_secret[hash function's input_size]
server_MAC_secret[hash function's input_size]
client_key[secret-key cryptography's key_size]
server_key[secret-key cryptography's key_size]
client_PAD[1]
server_PAD[1]
client_IV[secret-key cryptography's IV_size]
server_IV[secret-key cryptography's IV_size]

The hash function and the secret-key cryptography used in the above procedure were determined in the cipher spec beforehand. These procedures used by the client and the server must be identical.

IAC SB ATELNET FINISHED <negotiation digest>IAC SE

First, using the just-negotiated algorithms, keys, and secrets to protect the finished message. By computing the negotiation digest, the finished command is protected as well. The finished command requires no acknowledgement, and it is just an indication that the negotiation phase is completed. Upon receiving the finished command, the receiver must confirm whether the contents are correct or not. After validating the finished command, the receiver may begin sending the confidential data immediately. As to the server side, this finished command can be sent immediately after receiving the pre-master secret command from the client. As for the client side, this finished command can be sent immediately after receiving the finished command from the server.

The negotiation digest is defined for the client and the server respectively as follows:

negotiation_digest =
Encryption$_{client_key}$(random$_{client}$ + Hash(random$_{client}$ + master_secret
+ IAC SB ATELNET CLIENT_CRYPTOGRAPHIC_PARAMETERS<partial secret> <cipher suite list> IAC SE
+ IAC SB ATELNET CLIENT_CERTIFICATE <certificate list> IAC SE));
// the client's negotiation digest.
negotiation_digest =
Encryption$_{server_key}$(random$_{server}$ + Hash(random$_{server}$ + master_secret
+ IAC SB ATELNET REQUEST IAC SE
+ IAC SB ATELNET SERVER_CRYPTOGRAPHIC_PARAMETERS <partial secret><cipher suite> IAC SE
+ IAC SB ATELNET SERVER_CERTIFICATE <certificate list> IAC SE));
// the server's negotiation digest

The negotiation digest is computed using the MAC algorithm proposed by Yeh [10]. The hash function was determined in the cipher spec.

IAC SB ATELNET ERROR <error message> IAC SE

The procedure of handling error in the negotiation phase is very simple. When an error is detected, the side that detects it can send an error command to the other side. Upon sending or receiving the error command, both sides close the connection immediately. The types of error messages are defined as follows:

no cipher suite	No cipher suites suitable can be accepted.
no certificate	No certificates available can be provided.
invalid certificate	The certificate may be invalid, corrupted, revoked, and expired, etc.
invalid negotiation digest	The negotiation digest is not correct.

3.2.3 Example

An example is provided to illustrate how the command sequences are used, how both sides of the connection negotiate the cryptographic algorithms, and how the shared secret is generated.

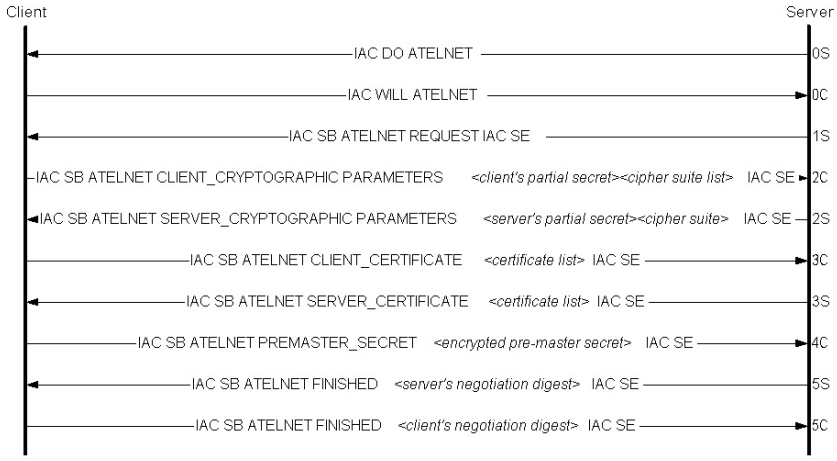

Fig. 2: A ECC_AES-128-CBC_RIPE-MD example

The steps contained in Fig. 2 are described as follows:

(0S) and (0C)

The client and the server both agree to perform the authenticated TELNET option.

(1S)

The server sends a request command to request the client to start the negotiation phase.

(2C) and (2S)

The client and the server exchange their partial secret. They also determine a cipher spec consisting of three cryptographic algorithms: the public-key cryptography, the secret-key cryptography, and the hash function. For example, ECC_AES-128-CBC_RIPE-MD means that:

public-key cryptography = ECC
secret-key cryptography = AES-128-CBC
key size = 16 bytes
IV size=16 bytes
block size = 8 bytes
hash function = RIPE-MD
input size = 64 bytes
output size=16 bytes
iteration = 4

(3C) and (3S)

The client and the server authenticate each other by exchanging and verifying their certificate.

(4C)

The client generates the pre-master secret. By using the public-key cryptography coupled with the server's certificate this pre-master secret is encrypted from the transmission point of view. The pre-master secret was used to compute the master secret from which it can be used to compute the keys of secret-key cryptography and the secrets of the MAC algorithm.

master_secret =
RIPE-MD(pre-master_secret + RIPE-MD(0x00 + pre-master_secret + client's partial_secret
+ server'spartial_secret))
+ RIPE-MD(pre-master_secret + RIPE-MD(0x01 + pre-master_secret + client's partial_secret
+ server'spartial_secret))
+ RIPE-MD(pre-master_secret + RIPE-MD(0x02 + pre-master_secret + client's partial_secret
+ server'spartial_secret))
+ RIPE-MD(pre-master_secret + RIPE-MD(0x03 + pre-master_secret + client's partial_secret
+ server's partial_secret));
secret_block =
RIPE-MD(master_secret + RIPE-MD(0x00 + master_secret+client's partial_secret

```
        + server's partial_secret))
    + RIPE-MD(master_secret + RIPE-MD(0x01 + master_secret + client's partial_secret
        + server's partial_secret))
    + RIPE-MD(master_secret + RIPE-MD(0x02 + master_secret + client's partial_secret
        + server's partial_secret))
    + ….
    + RIPE-MD(master_secret + RIPE-MD(0x0c + master_secret + client's partial_secret
        + server's partial_secret));
// In this example, the iteration is four.  Thus, it needs two cipher keys of 16-bytes, two MAC secrets of
64-bytes, two padding byte, and two IV of 16-bytes.

The secret block was partitioned as follows:
    client_MAC_secret = secret_block[0..63]
    server_MAC_secret = secret_block[64..127]
    client_key = secret_block[128..143]
    server_key = secret_block[144..159]
    client_pad = secret_block[160]
    server_pad = secret_block[161]
    client_IV = secret_block[162..177]
    server_IV = secret_block[178..193]
```

(5S) and (5C)

The sender of the finished commands declares that the negotiation phase was finished. The finished commands now contain the negotiation digest. The client and the server confirm that the other side had finished the negotiation phase.

3.3 Application Phase

Once the negotiation phase is complete, the cipher spec and the shared secret can be determined. Both sides of the connection possess the shared secrets that are needed to encrypt the contents and compute the MAC on their contents. Cipher spec offers the techniques used to perform the encryption and the MAC operations.

In the application phase, the sender computes the MAC using the secret-key cryptography and the hash function. After computing the MAC, the sender generates the structures of the temporary block using the length of the content, the content, and the MAC including the padding and the length of the padding. The sender then encrypts the temporary block using the secret-key cryptography and forwards it to the lower layers such as the socket layer or the network layer in a non-empty block.

The data structures of the temporary block for either the block cipher or the stream cipher are defined as follows:

```
struct{
    opaquedata_length[2];
    opaquecontent[data_length];
    opaqueMAC[cipher_spec.hash_function.input_size];
    opaquePADDING_LENGTH[2];
    opaquePADDING[PADDING_LENGTH];
}BlockCipher_TemporaryBlock;
struct{
    opaquedata_length[2];
    opaquecontent[data_length];
    opaqueMAC[cipher_spec.hash_function.input_size];
}StreamCipher_TemporaryBlock;
```

The MAC of the content is computed by using the algorithm proposed by Yeh [10]:

$MAC = Encryption_{key}$ (MAC_secret+increment+Hash (MAC_secret+increment+packet_no+length+content));

encryption	The secret-key cryptography was determined and established in cipher spec.
hash	The hash function was determined and established in the cipher spec.
key	The key of the receiver for the secret-key cryptography was computed in the negotiation phase.
MAC_secret	The MAC_secret was computed in the negotiation phase.

Increment	At the start, the increment must be zero. The increment must be increased one at a time.

The temporary block must be encrypted with the corresponding secret-key cryptography, and then the encrypted temporary block will be transmitted.

The purpose of the padding added in the block cipher data structure is to make the length of the structure to be multiple for the block length in the block cipher. The initialisation vector (IV as abbreviation) and the padding byte used in block cipher are computed in the negotiation phase.

4.0 DESIGNING CIPHER SUITE

In this section, we would like to describe how to specify the cryptographic algorithm and how to define the cipher suite respectively by extending the cryptographic algorithms and incorporating them with the cipher suites.

When defining the cipher suite, it involves three types of cryptographic algorithms: the public-key cryptography, the secret-key cryptography, and the hash function. Most importantly, it is necessary to specify any cryptographic algorithms that have not been designated. We describe the format of specification as follows:

public-key cryptography

public-key-cryptography code

public-key-cryptography	It specifies the name of the public-key cryptography.
code	It specifies the code of the public-key cryptography, and it cannot be iterant.

secret-key cryptography

secret-key-cryptography code stream/block key_size expanded_key_size effective_key_bit IV_size blocke_size

secret-key-cryptography	It specifies the name of the secret-key cryptography.
code	It specifies the code of the secret-key cryptography, and it cannot be iterant.
stream/block	It specifies which kind of the secret-key cryptography it is.
key_size	It specifies the number of bytes from the secret_block to be used for generating the keys.
expanded_key_size	It specifies the number of bytes actually used with regard to the secret-key cryptography.
effective_key_bit	It specifies the number of bits of the key actually being fed into the encryption routines.
IV_size	It specifies the number of bytes from the secret_block to be used as the initialization vector for the cipher algorithm.
block_size	It specifies the number of bytes actually being fed into and generated by the encryption routines.

hash function

hash-function input_size output_size iteration

hash-function	It specifies the name of the hash function.
code	It specifies the code of the hash function, and it cannot be iterant.
input_size	It specifies the number of the bytes being fed into the hash function for each loop.
output_size	It specifies the number of bytes generated by the hash function for each loop.
Iteration	It specifies how many times the sub-procedure will be used during the generation of the master secret. It is expected that the number of iteration should be higher than the quotient obtained by dividing the input size with the output size.

Based on the above-mentioned format, any cryptographic algorithms can be specified by users. And these specific algorithms can be used to define the cipher suite. In this proposed protocol, we have specified the cryptographic algorithms: RSA [11], ECC [12], 3DES [13], RC6 [14], AES [15], MD5, SHA [16] and RIPE-MD [17] (as shown in the appendix) and defined the cipher suites: RSA_3DES-EDE-CBC_MD5, RSA_AES-128-CBC_RIPE-MD, etc. However, cryptographic algorithms or cipher suites that are unspecified or undefined in this proposed protocol could be designed following the illustrations demonstrated in the subsequent sections. We present individually three

examples to describe how to specify the cryptographic algorithm, and then use them to define a new cipher suite.

4.1 Cipher Suite Comprising Public-Key Cryptography

The steps are:

1. Specify the public-key cryptography
2. Define the Cipher Suite Code

If a user needs to use the public-key cryptography unspecified in this protocol to encrypt the secret, he must specify a new public-key cryptography, define a new cipher suite, and assign a new cipher suite code. For example, the ElGamal will be used as the public-key cryptography. User must specify 0x03 as the code of the ElGamal in defining the ElGamal_3DES-EDE-CBC_MD5 as 0x030101. To use this cipher suite, the server's certificate must be concatenating with the ElGamal, and then the client and the server can exchange the secret using the ElGamal.

4.2 Cipher Suite Comprising Secret-Key Cryptography

The steps are:

1. Specify the secret-key cryptography
2. Define the Cipher Suite Code

If a user needs to use the secret-key cryptography unspecified in this protocol to encrypt the data stream. The procedure is similar to that in Section 4.1. Again, user must specify the information regarding both key and block size. For example, the IDEA will be used as the secret-key cryptography. User must specify 0x04 for code, block for type, 16 for key_size, 16 for expanded_key_size, 128 for effective_key_size, 8 for IV_size, and 8 for block_size. User defines the RSA_IDEA-CBC_MD5 as 0x010401. With this cipher suite, the client and the server can encrypt and decrypt the data stream exchanged between them.

4.3 Cipher Suite Comprising Hash Function

The steps are:

1. Specify the hash function
2. Define the Cipher Suite Code

If a user needs to use the hash function unspecified in this protocol to compute the MAC, the procedure is similar to that in Section 4.1. User must specify code, the size of input, the size of output, and the iteration when specifying the hash function. For example, the MD5 will be used as the hash function. User must specify 0x04 for code, 64 for input_size, 16 for output_size, and 4 for the iteration. User defines the RSA_3DES-EDE-CBC_MD5 as 0x010104. With this cipher suite, the client and the server can use the MD5 to compute the MAC, the master secret, and the secret block.

5.0 DISCUSSION AND SECURITY ANALYSIS

The main purpose of designing the authenticated TELNET option is to provide a secure and authenticated environment for TELNET connection. It is preferable to exercise authenticated TELNET option immediately after the connection. After completing the negotiation phase, all the transmitted data are secured. Several secure mechanisms are proposed in this work to offer guarded capabilities to resist various kinds of attacks even if attackers have the ability to capture, modify, delete, insert, reply, and forge the transmitted data. These secure mechanisms are described in detail as follows:

5.1 Authentication

In the negotiation phase, the client and the server must present their certificate to the other party for authenticating the identification. It is worth mentioning that both sides of the connection must verify each other's certificate to confirm that it is valid; in other words, it has not expired or it has not been revoked yet. The authority of certificate plays an important role in this proposed protocol. In this system, it is assumed that the authority of certificate is trustworthy. Nevertheless, a dishonest authority of certificate may deceive and damage the system.

5.2 Detecting Attacks in Negotiation Phase via MAC

Any attacks may occur at any time as the TELNET communication is underway. In negotiation phase, the negotiation digest is used to detect the intrusion. Each sender computes the negotiation digest of the command sequences and sends it to the receiver. Then the finished command including the result is sent to the receiver. Upon receiving, the recipient must possess the command sequences received from the sender prior to the finished command and verify the negotiation digest. Therefore, the client and the server can detect the protocol for any attacks.

5.3 Protecting Transmitted Data

After the negotiation phase is complete, the client and the server can exchange data under the protection of both secret-key cryptography and MAC. In this protocol, we use the public-key cryptography to exchange the shared secret. With shared secret, we can finally derive the key and the mac_secret. The MAC used in the data stream is to check the integrity. Any data before their transmission will be encapsulated in a data stream to contain the length of content, the content, the MAC, and the padding if a block cipher is used. The data stream will be encrypted and then transmitted.

If an attacker intercepts the transmitted data stream, the original text cannot be deduced because it is encrypted. The usage of the increment when computing the MAC is to prevent data replay, deletion, and insertion. This increment can detect the replay attack. On the other hand, the MAC can resist against any forgery and tampering. In this protocol, the MAC is much more secured when applying the Yeh's algorithm.

6.0 CONCLUSION

In this study, an authenticated TELNET command option is proposed to provide security and authentication for the TELNET communication. The advantages of this proposed protocol are that it is compatible with the traditional TELNET program, and it gives the flexibility and extensibility to the use of cryptographic algorithm.

In the negotiation phase, the negotiation digest is used to detect any attacks for the protocol. Moreover, the secret-key cryptography and the MAC are generated to establish a secure connection between a client and a server to communicate through an insecure channel.

Based on the concept of SSL and TLS protocol, we also apply the cipher suite to negotiate the cipher spec, and give the implementer the capability to define more cipher suites. To be specific, this proposed system has been designed as a main framework capable of creating options of sub-negotiation in accordance with the authenticated TELNET protocol. Whenever a new cryptographic algorithm is planned or an existing one is no longer secure, it is relatively easy for those designers to incorporate new cipher suites into the authenticated TELNET option effortlessly.

REFERENCES

[1] J. Postel and J. Reynolds, "Telnet Protocol Specification". Internet Engineering Task Force, *RFC 854*, May 1983.

[2] J. Postel and J. Reynolds, "Telnet Option Specifications". Internet Engineering Task Force, *RFC 855*, May 1983.

[3] A. Frier, P. Karlton and P. Kocher, *The SSL 3.0 Protocol*. Netscape Communications Corp., Nov 1996.

[4] TLS T. Dierks, C. Allen, "The TLS Protocol Version 1.0," Internet Engineering Task Force, *RFC 2246*, Jan 1999.

[5] B. Schneier, *Applied Cryptography*. 2nd ed., John Wiley & Sons, 1996.

[6] T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". *Advances in Cryptology: Proceedings of CRYPTO 84*. Springer-Verlag, 1985, pp. 10-18.

[7] X. Lai and J. Massey, "A Proposal for a New Block Encryption Standard". *Advances in Cryptology-EUROCRYPT '90 Proceedings*. Springer-Verlag, 1991, pp. 157-165.

[8] B. Schneier, "One-Way Hash Functions". *Dr. Dobb' Journal*, V. 16, N. 9, Sept. 1991, pp. 148-151.

[9] R. Srinivansan, "XDR External Data Representation Standard". *RFC 1832*, Aug. 1995.

[10] Y. S. Yeh, C. C. Wang, "Construct Message Authentication Code with One-Way Hash Functions and Block Ciphers". *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, V. E82-A, No. 2, Feb. 1999, pp. 390-393.

[11] R. L. Rivest, A. Shamir, L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystem". *Communications of the ACM*, V. 21, N. 1, Feb. 1978, pp. 120-126.

[12] V. S. Miller, "Use of Elliptic Curves in Cryptography". *Advances in Cryptology - CRYPTO '85, Lecture Notes in Computer Science*, 218 (1986), Springer-Verlag, pp. 417-426.

[13] ANSI X9.17 (Revised), *American National Standard for Financial Institution Key Management (Wholesale).* American Bankers Association, 1985.

[14] R. L. Rivest, M. J. B. Robshaw R. Sidney and Y. L. Yin, "The RC6 Block Cipher.v1.1". Aug. 1998. Available at www.rsa.com/rsalabs/aes/.

[15] National Institute of Standards and Technology, NIST FIPS PUB 197, *Advanced Encryption Standard (AES).* U.S. Department of Commerce, Nov. 2001.

[16] National Institute of Standards and Technology, NIST Draft FIPS PUB 180-2, *Secure Hash Standard.* U.S. Department of Commerce, May 2001.

[17] Research and Development in Advanced Communication Technologies in Europe, "RIPE Integrity Primitives: Final Report of RACE Integrity Primitives Evaluation (R1040)". *RACE*, Jun. 1992.

BIOGRAPHY

Yi-Shiung Yeh is an Associate Professor. He obtained his PhD in Computer Science, Department of EE & CS, University of Wisconsin-Milwaukee, 1978. His research interest includes Cryptography and Information Security, Reliability and Performance, and DNA Computation.

Wei-Shen Lai is currently a student in the Philosophy Program in Computer Science and Information Engineering, Department of CSIE, National Chiao-Tung University. He obtained his MS in Computer Science and Information Engineering, Department of CSIE, National Chiao-Tung University, in 1995. His research interest includes Cryptography and Information Security, Network Security, Electronic Commerce, and Computer Architecture.

APPENDIX

Table 3: The definition of public-key cryptography

Name	Code
NULL	0x00
RSA	0x01
ECC	0x02

Table 4: The definition of secret-key cryptography

Name	Code	Type	Key Size (byte)	Expanded Key Size (byte)	Effective Key Size (bit)	IV Size (byte)	Block Size (byte)
NULL	0x00	N/A	N/A	N/A	N/A	N/A	N/A
3DES-EDE-CBC	0x01	Block	24	24	168	8	8
RC6-128	0x02	Stream	16	16	128	16	16
AES-128-CBC	0x03	Block	16	16	128	16	16

Table 5: The definition of hash function

Hash Function	Code	Input Size	Output Size	Iteration
NULL	0x00	N/A	N/A	N/A
MD5	0x01	64	16	4
SHA-256	0x02	64	32	2
RIPE-MD	0x03	64	16	4

Table 6: The definition of cipher suite

Name	Code	Public-Key Cryptography	Secret-key cryptography	Hash Function
NULL_NULL_NULL	0x000000	NULL	NULL	NULL
RSA_3DES-EDE-CBC_MD5	0x010101	RSA	3DES-EDE-CBC	MD5
RSA_3DES-EDE-CBC_SHA-256	0x010102	RSA	3DES-EDE-CBC	SHA-256
RSA_3DES-EDE-CBC_RIPE-MD	0x010103	RSA	3DES-EDE-CBC	RIPE-MD
RSA_RC6-128_MD5	0x010201	RSA	RC6-128	MD5
RSA_RC6-128_SHA-256	0x010202	RSA	RC6-128	SHA-256
RSA_RC6-128_RIPE-MD	0x010203	RSA	RC6-128	RIPE-MD
RSA_AES-128-CBC_MD5	0x010301	RSA	AES-128-CBC	MD5
RSA_AES-128-CBC_SHA-256	0x010302	RSA	AES-128-CBC	SHA-256
RSA_AES-128-CBC_RIPE-MD	0x010303	RSA	AES-128-CBC	RIPE-MD
ECC_3DES-EDE-CBC_MD5	0x020101	ECC	3DES-EDE-CBC	MD5
ECC_3DES-EDE-CBC_SHA-256	0x020102	ECC	3DES-EDE-CBC	SHA-256
ECC_3DES-EDE-CBC_RIPE-MD	0x020103	ECC	3DES-EDE-CBC	RIPE-MD
ECC_RC6-128_MD5	0x020201	ECC	RC6-128	MD5
ECC_RC6-128_SHA-256	0x020202	ECC	RC6-128	SHA-256
ECC_RC6-128_RIPE-MD	0x020203	ECC	RC6-128	RIPE-MD
ECC_AES-128-CBC_MD5	0x020301	ECC	AES-128-CBC	MD5
ECC_AES-128-CBC_SHA-256	0x020302	ECC	AES-128-CBC	SHA-256
ECC_AES-128-CBC_RIPE-MD	0x020303	ECC	AES-128-CBC	RIPE-MD