International Journal on Advanced Science Engineering Information Technology

Vol.8 (2018) No. 5
ISSN: 2088-5334

Implementing a Secure Key Exchange Protocol for OpenSSL

Janaka Alawatugoda[#1], Seralathan Vivekaanathan[#], Nishen Peiris[#], Chamitha Wickramasinghe[#], Chuah Chai Wen[*]

[#] *Department of Computer Engineering, Faculty of Engineering, University of Peradeniya, Sri Lanka*
E-mail: [1]alawatugoda@eng.pdn.ac.lk

[*] *Information Security Interest Group, Faculty Computer Science and Information Technology*
University Tun Hussein Onn Malaysia, Malaysia
E-mail: cwchuah@uthm.edu.my

***Abstract*— Security models have been developed over time to examine the security of two-party authenticated key exchange protocols. In 2007, a reasonably strong security model for key exchange protocols has been proposed, namely extended Canetti-Krawczyk model (eCK model), addressing wide range of real-world attack scenarios. They constructed a protocol called NAXOS, that is proven secure in the eCK model. In order to satisfy the eCK security, NAXOS protocol uses a hash function to combine the ephemeral key with the long-term secret key, which is often called as "NAXOS trick". However, for the NAXOS trick based protocols, the way of leakage modelled in the eCK model leads to an unnatural assumption of leak-free computation of the hash function. In 2015, Alawatugoda, Stebila and Boyd presented a secure and NAXOS trick key exchange protocol, namely protocol P1. In this work, we implement the protocol P1 to be used with the widely-used OpenSSL cryptographic library. OpenSSL implementations are widely used with the real-world security protocol suites, particularly Security Socket Layer and Transport Layer Security. According to our knowledge, this is the first implementation of an eCK-secure protocol for the OpenSSL library. Thus, we open up the direction to use the recent advancements of cryptography for real-world Internet communication.**

***Keywords*— authenticated key exchange; eCK mode; OpenSSL; secure key; security models.**

I. INTRODUCTION

In symmetric key cryptography (SKC), a single key is used for encryption and decryption, which is often called as the secret key. Therefore, prior to message communication, both the sender and the receiver need to share the secret key, in such a way that no eavesdropper can learn the secret key. Hence, a trusted channel is needed to share the secret key. Historically, this trusted channel was accomplished by means of trusted couriers, diplomatic bags etc. This mechanism is undesirable, because of the necessity of a third party, and the entire confidentiality relies on the trustworthiness of this third party. With the ground breaking work of Diffie and Hellman [1] on public-key cryptography (PKC) and key distribution, cryptographers realized a more sophisticated solution for securely exchanging the secret key. Thus, the idea of public-key cryptography eliminates the need of a trusted third party to create a secure channel.

A. Diffie-Hellman Protocol

A key exchange protocol defines a method of securely exchanging secret session keys over a public channel. Diffie and Hellman, researchers at Stanford University, presented about public-key cryptography and key distribution in their ground breaking paper, "New Directions in Cryptography" [1]. The key exchange primitive presented in their paper is widely known as the Diffie-Hellman (DH) key exchange primitive and is considered to be the earliest practical authenticated key exchange protocol.

We simply explain the Diffie-Hellman protocol as follows: Users **A** and **B** negotiate on a group G of prime order q and a generator g. The values G, g, q are released as public values. In the protocol execution, **A** chooses a random element $a \in G$, computes $A \leftarrow g^a$ and sends it to **B**. Upon receiving A, **B** chooses a random element $b \in G$, computes $B \leftarrow g^b$ and sends it to **A**. Finally, **A** computes $K \leftarrow B^a = (g^b)^a$, and **B** computes $K \leftarrow A^b = (g^a)^b$. Thus, both **A** and **B** compute the same value $K \in G$.

An eavesdropping attacker observes the communication above may see A and B values. Given $A = g^a$, $B = g^b$ computing $K = g^{ab}$ is "computational Diffie-Hellman problem (CDH)". There is no known efficient algorithm,

which is capable of computing $K = g^{ab}$ given that $A = g^a$, $B = g^b$.

Several key exchange protocols are designed on top of the Diffie-Hellman protocol [2]–[7]. The Diffie-Hellman protocol is applied to many security protocol suits including Security Sockets Layer (SSL), Transport Layer Security (TLS), Secure Shell (SSH), and IP Sec.

B. NAXOS Trick and eCK-secure Key Exchange

Generally, a security model is a scheme that is designed to enforce security policies. Extended Canetti-Krawczyk (eCK) security model [8] is considered as a widely used security model for key exchange protocols, as it addresses wide range of real-world attack scenarios. We discuss about the security models in detail in Section II.

The motivation of constructing the eCK model was to make the adversary obtain both the ephemeral and long-term secret keys of a party to compute the session key of that particular communication. In the NAXOS protocol [8], the session key is computed using the so-called NAXOS trick. In this trick, the "psuedo" ephemeral key *psk* is calculated as a hash value of the ephemeral key *esk* and the long-term secret key *lsk*, such that *psk* = H(*esk*, *lsk*). This psuedo ephemeral key is not stored. Hence, the adversary have to get both the *lsk* and *esk* to calculate the psuedo ephemeral key *psk*. Following the NAXOS protocol, few NAXOS trick based authenticated key exchange protocols are presented [4]. However, in NAXOS trick based protocols, the leakage of ephemeral key modelled in the eCK model seems not natural, as the eCK model leaks *esk*, while *psk* remains safe. In a work of Alawatugoda et al. [9], a NAXOS trick free eCK-secure AKE protocol is presented, namely the protocol P1.

C. OpenSSL Cryptographic Library

OpenSSL is an open source project which provides a commercial grade toolkit for the TLS and SSL protocol suits. It is a general purpose cryptographic library. This cryptographic library is implemented in C programing language, but it has wrappers in Java and some other languages as well. There are three variants of Diffie-Hellman primitive used in the OpenSSL cryptographic library.

- Anonymous DH: Diffie-Hellman primitive is used without authentication.
- Fixed DH: Embeds the server's public parameter in the certificate.
- Ephemeral DH: Uses one-time public keys. The authenticity is verified using signatures. Since the public keys are one-time values, compromising the long-term signing keys does not reveal the old session keys. Therefore, this protocol provides perfect forward secrecy.

According to statistics from various cryptography related sources, a considerable large number of websites (more than 4,545,900) rely on the security measures of the OpenSSL cryptographic library. Apparently, OpenSSL cryptographic library has a steady usage as well. One important factor to highlight here is that the highest number of websites among them, fall under the vertical of business related websites. Therefore, it is obvious that the OpenSSL cryptographic library has a significant place in today's Internet.

D. Our Contribution

In this work we implement the eCK-secure protocol P1 of Alawatugoda et al. [9] to be used with the widely using cryptographic library, the OpenSSL library. The OpenSSL implementations are widely used for the real-world security protocol suites, particularly Security Socket Layer protocol and Transport Layer Security protocol. As per our knowledge, this is the first implementation of an eCK-securet key exchange protocol to be used for real-world communications. Thus, we open up the direction to use the recent advancements of cryptography for real-world Internet communications.

II. Material and Method

In this section, key exchange security models and extended canetti-krawczyk models are presented.

A. Key Exchange Security Models

Researchers keep on constructing authenticated key exchange protocols, strengthening them against various attacks and vulnerabilities. The continuous growth of key exchange protocols created a necessity of a formal methodology to analyze their security. Therefore, in order to fulfill this necessity, computer scientists have introduced key exchange security models.

Security models are designed in a way that they reflect present real-world attacks, and of course potential attacks that can exist in future.

Components of a security model:

- **Definition:** abstraction of the algorithm.
- **Powers of the Adversary:** Queries which the adversary is capable of performing to reveal information of protocol instances.
- **Security game (challenge):** Order that the adversary uses the queries.
- **Security definition:** Condition to the adversary to win the challenge.

There is a set of queries which an adversary is allowed to use (powers of the adversary). The queries leak various secret values to the adversary. A protocol to be secure within the security model, the adversary's advantage of identifying the real session key of the target session from a random value (from the same distribution) has to be negligible.

Researchers aim to construct security models which are capable of modeling maximum of possible known attacks mentioned below:

- Implicit Key Authentication: assures that nobody other than the protocol participants calculates the session key. Such protocols are known as *authenticated key exchange* protocols.
- Key Confirmation: assures that all the other protocol participants compute the session key.
- Unknown Key Share (UKS) Security: no party *P* shares a session key with a party *Q*, believing that it is shared with some other party *R*.
- Key Compromise Impersonation (KCI) Security: having the long-term secret key of a party *P* does not

make the adversary capable of impersonating other parties to *P*.

- Forward Secrecy: an adversary who knows the long-term secret keys of two parties cannot compute the past session keys between those two parties.

Bellare-Rogaway (BR) [10], [11], Canetti-Krawczyk (CK) [12] and extended Canetti-Krawczyk (eCK) [8] are few of the widely used key exchange security models. Table I shows a comparison between the above three security models with respect to security features addressed by the security models.

TABLE I [9]
SECURITY FEATURES SUPPORTED BY BR, CK AND ECK SECURITY MODELS

Security Features	BR	CK	eCK
Implicit key authentication	yes	Yes	yes
Key confirmation	yes	Yes	yes
UKS	yes	Yes	yes
KCI	no	No	yes
Forward secrecy (FS)	no	Yes	Weak-FS

B. eCK Model

We now look at the eCK model of LaMacchia et al. [8]. Wcall *principal* to a party which involves in the protocol run. Each party has a pair of long-term public/secret keys, which are generated before the protocol run. Each instance of the protocol between any two principals is defined as a *session*.

We model a session belongs to the principal as an oracle. $\prod^{s}_{U,V}$ is an oracle which represents the s^{th} protocol instance between owner principal *U* and a partner principal *V*. When $\prod^{s}_{U,V}$ computes a session key it come to the accepted state. Any time a session can terminate without computing a session key. The information of whether acceptance or not publicly available.

1) Partnering: Legitimate protocol execution between principals V and U makes two partner sessions owned by V and U respectively. In order to define the partnership, we have used the notation of a session identifier (SID). The SID is the concatenation of messages exchanged between two intended principals with their identities. The SID of $\prod^{s}_{U,V}$ is denoted by $SID^{s}_{U,V} = (comm_1 \| \dots \| comm_n)$. Two oracles $\prod^{s}_{U,V}$ and $\prod^{s'}_{V,U}$ are partners if:

- $\prod^{s}_{U,V}$ and $\prod^{s'}_{V,U}$ compute session keys (both oracles come to accepted state) $SID^{s}_{U,V} = SID^{s'}_{V,U}$ (here $s = s'$).
- $\prod^{s}_{U,V}$ and $\prod^{s'}_{V,U}$ agree on the initiator of the protocol.

2) Powers of the Adversary: The adversary A is an efficient algorithm that adaptively asks below queries.

- Send (*U*,*V*,*s*,*m*): This query sends a message *m* to the oracle $\prod^{s}_{U,V}$ as sending from the oracle $\prod^{s}_{V,U}$. $\prod^{s}_{U,V}$ will send to A the next message according to the protocol conversation. This query is also used to initiate a new protocol instance.
- Session-Key reveal (*U*,*V*,*s*): If an oracle $\prod^{s}_{U,V}$ hold a session key, A gets to know it.
- Ephemeral-Key reveal (*U*,*V*,*s*): Reveals the ephemeral keys of the oracle $\prod^{s}_{U,V}$ to A.
- Corrupt (*U*): A get all the long-term secrets of *U*. This query captures the UKS attacks, KCI attacks and forward secrecy.
- Test (*U*,*s*): When A asks this query, the oracle $\prod^{s}_{U,V}$ chooses a random bit $b \in \{0,1\}$ and if $b = 1$, the real session key is revealed to A, otherwise returns a random string.

3) Freshness: An oracle is fresh iff:

- The particular oracle or its partner (if exists), it has not been asked to reveal the session key.
- If partner exists, none of the following have been asked:
  a. Ephemeral-Key reveal (*U*,*V*,*s*) and Corrupt (*U*).
  b. Ephemeral-Key reveal (*V*,*U*,*s'*) and Corrupt (*V*).
- If partner does not exist, none of the following have been asked:
  a. Corrupt (*V*).
  b. Ephemeral-Key reveal (*U*,*V*,*s*) and Corrupt (*U*).

4) Challenge

- Stage 1: A is capable of asking any of Session-Key reveal, Send, Corrupt and Ephemeral-Key reveal queries at will.
- Stage 2: A asks a Test query to any oracle.
- Stage 3: A continues asking queries again without violating the test session's freshness.
- Stage 4: A outputs its guess on the value *b* (lets name it as b`). $A$ wins if the value $b` = b$.

5) Security Statement

$Succ_A$ be the instance that the A wins the eCK challenge. A protocol (let's say π) is secure in eCK model if there is no efficient A which is capable of winning the eCK challenge with non-negligible advantage.

C. eCK-secure and NAXOS Trick Free AKE Protocol

The protocol P1 of Alawatugoda et al. [9] is proven-secure in the eCK model.

Let *G* be a group of prime order *q* and generator *g*. After exchanging the public values both principals compute shared secrets (Z_1, Z_2, Z_3 and Z_4). Then the session key is computed as a random oracle combination of Z_1, Z_2, Z_3 and Z_4, which can be replaced by a hash function in the implementation. Table II shows the protocol diagram of protocol P1.

In the protocol execution, for each party, the PI protocol needs 5 exponentiations and one random oracle computation, whereas the NAXOS protocol needs 4 exponentiations and 3 random oracle computations.

TABLE II
PROTOCOL P1 [9]

Alice		Bob
$a \xleftarrow{\$} \mathbb{Z}^*_q, A \leftarrow g^a$		$b \xleftarrow{\$} \mathbb{Z}^*_q, B \leftarrow g^b$
	Protocol Execution	
$x \xleftarrow{\$} \mathbb{Z}^*_q, X \leftarrow g^a$	$\xrightarrow{Alice,X}$ $\xleftarrow{Bob,Y}$	$y \xleftarrow{\$} \mathbb{Z}^*_q, Y \leftarrow g^y$

$Z_1 \leftarrow B^a, Z_2 \leftarrow B^x$	$Z_1' \leftarrow A^b, Z_2' \leftarrow X^b$
$Z_3 \leftarrow Y^a, Z_4 \leftarrow Y^x$	$Z_3' \leftarrow A^y, Z_4' \leftarrow X^y$
$K \leftarrow H(Z_1, Z_2, Z_3, Z_4, Alice, X, Bob, Y)$ K	
$\leftarrow H(Z_1', Z_2', Z_3', Z_4', Alice, X, Bob, Y)$	
K is the session key	

1) Technical Preliminaries for security Proof:

Definition 1 (Decision Diffie-Hellman (DDH) Problem)

Given (g^a, g^b, g^c) for a, $b \xleftarrow{\$} \mathbb{Z}_q$, the DDH problem is to distinguish whether $c = ab$ or not.

Definition 2 (Gap Diffie-Hellman (GDH) Problem)

Given (g^a, g^b) for a, $b \xleftarrow{\$} \mathbb{Z}_q$, the GDH problem is to find g^{ab} accessing an oracle that solves the DDH problem.

2) Security of the Protocol P1:

If H is a random oracle function and the gap Diffie--Hellman problem is hard in the group *G*, then protocol P1 is secure in the eCK model.

III. Result and Discussion

A. Implementation of Protocol P1

The implementation of the Diffie-Hellman protocol is found under the directory crypto in the OpenSSL cryptographic library. While implementing the protocol P1, we have kept the original Diffie-Hellman key exchange protocol implementation intact. Our new implementation has been done as a separate key exchange protocol for the OpenSSL cryptographic library. We have provided our implementations in check.c, gen.c, key.c, and p1.h files.

```
struct p1_st {
     /* This first argument used to pick up
         errors when a DH is passed instead
         of a EVP_PKEY
    */
    int pad;
    int version;
    BIGNUM *p;
    BIGNUM *g;
    long length;         /* optional */
    BIGNUM *pub_key;     /* A/B parameter*/
    BIGNUM *priv_key;    /* a/b parameter*/
    BIGNUM *pub_key1;    /* X/Y parameter */
    BIGNUM *priv_key1;   /* x/y parameter */
    BIGNUM *z1;          /* z1 parameter */
    BIGNUM *z2;          /* z2 parameter */
    BIGNUM *z3;          /* z3 parameter */
    BIGNUM *z4;          /* z4 parameter */
    int flags;
    BN_MONT_CTX *method_mont_p;
    /* Place holders if we want to do X9.42
        DH
    */
    BIGNUM *q;
    BIGNUM *j;
    unsigned char *seed;
    int seedlen;
    BIGNUM *counter;
    int references;
    CRYPTO_EX_DATA ex_data;
    const DH_METHOD *meth;
    ENGINE *engine;
};
```

Fig. 1 P1.h: Structure used in the execution of protocol P1.

The file p1.h defines the structure which is used to store all the inputs and outputs of computation during the protocol execution (Fig. 1)

```
int P_check(const P *p, int *ret)
{
        /*code here*/
}

int P_check_pub_key(const P *p, const BIGNUM
     *pub_key, int *ret)
{
        /*code here*/
}
```

Fig. 2 check.c.

check.c (Fig. 2) contains two functions that check all the parameters calculated during the protocol execution including p, q and confirm that they are likely enough to be valid. If any of the parameters are invalid, a flag is set indicating the reason for being invalid. The error code values are updated if any problem is found.

```
int DH_generate_parameters_ex (DH *ret, int
    prime_len, int generator, BN_GENCB *cb)
{
    /*code here*/
}
```

Fig. 3 gen.c.

gen.c (Fig. 3) contains a function that generates parameters *p* and *q*. The function **DH_generate_parameters_ex** generates Diffie-Hellman parameters, and stores them in the structure defined in p1.h. The pseudo-random number generator must be seeded prior to calling the function **DH_generate_parameters_ex**.

```
static int compute_key(unsigned char *key,
    const BIGNUM *pub_key, const BIGNUM *
    pub_key1, DH *dh)
{
   /*code here*/
}
```

Fig. 4 key.c.

key.c (Fig. 4) computes the shared secret keys (Z_1, Z_2, Z_3, Z_4), from the long-term or ephemeral private key (*a*/*b* or *x*/*y* respectively) in ***dh** and the other party's long-term or ephemeral public value (*A*/*B* or *X*/*Y* respectively) in ***pub_key**, then stores it in ***key**. Finally, SHA-256 is used to calculate the session key using the above-computed Z_1, Z_2, Z_3, Z_4 values.

B. Deploying the Protocol P1 in a client/ server environment

We establish an OpenSSL client connection with the server running on ***localhost*** port ***44330***, using the ECK cipher suite, which is created by us. The ECK cipher suite uses the protocol P1 implementation for key exchange. Fig. 5 shows the server certificate (the cipher suite ECK is highlighted).

```
Last login: Fri Jun 30 02:08:22 on ttys001
Serans-MBA:~ seralahthan$ openssl s_client -connect localhost:44330 -cipher ECK
CONNECTED(00000005)
depth=0 C = SL, ST = Western, L = Colombo, O = UOP, OU = Faculty of Engineering, CN = Seran
verify error:num=18:self signed certificate
verify return:1
depth=0 C = SL, ST = Western, L = Colombo, O = UOP, OU = Faculty of Engineering, CN = Seran
verify return:1
---
Certificate chain
 0 s:/C=SL/ST=Western/L=Colombo/O=UOP/OU=Faculty of Engineering/CN=Seran
   i:/C=SL/ST=Western/L=Colombo/O=UOP/OU=Faculty of Engineering/CN=Seran
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEOTCCAyGgAwIBAgIJAOOcxEkGkj2rMA0GCSqGSIb3DQEBCwUAMHAxCzAJBgNV
BAYTAlNMMRAwDgYDVQQIEwdXZXN0ZXJuMRAwDgYDVQQHEwdDb2xvbWJvMQwwCgYD
VQQKEwNVT1AxHzAdBgNVBAsTFkZhY3VsdHkgb2YgRW5naW5lZXJpbmcxDjAMBgNV
BAMTBVNlcmFuMB4XDTE3MDYyOTIwNTE1MFoXDTE4MDYyOTIwNTE1MFowcDELMAkG
A1UEBhMCU0wxEDAOBgNVBAgTB1dlc3Rlcm4xEDAOBgNVBAcTB0NvbG9tYm8xDDAK
BgNVBAoTA1VPUDEfMB0GA1UECxMWRmFjdWx0eSBvZiBFbmdpbmVlcmluZzE0MAwG
A1UEAxMFU2VyYW4wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDJoG2Q
Xt9DtMc/mGKAFMLq0N4HGucCyFgqanV4uOMmHFCKmvBiTA/ct0tFcxTnguFI7JDK
MHe6BURtZ5F0hn/XAKyjLxws0hQYx5atVhIGhRZfJHmM/VoyOiZabKSAqvEzLpwk
xQujVUMZFgzph1wGKYucoOySwAYYAD624r/9i/GUaBnnxTdH4KZFJJXudsZY93tT
ClwGVffmAbGZZu5MuprJLpELS8mpIq8iKV2ojOkMS4lbdKE0V8+TXw2erLmTMFlf
C7o1wUj6HU891XfeC8AE63KLi3EnO4Zu+/7ClZTR8i6Ymy61flEQ4LHkPd80UAAr
NFsiaU2cU4DFmPRLAgMBAAGjgdUwgdIwHQYDVR0OBBYEFPycb5dfuh1gcgsFaWv1
EGuANDQVMIGiBgNVHSMEgZowgZeAFPycb5dfuh1gcgsFaWv1EGuANDQVoXSkcjBw
MQswCQYDVQQGEwJTTDEQMA4GA1UECBMHV2VzdGVybjEQMA4GA1UEBxMHQ29sb21i
bzEMMAoGA1UEChMDVU9QMR8wHQYDVQQLExZGYWN1bHR5IG9mIEVuZ2luZWVyaW5n
MQ4wDAYDVQQDEwVTZXJhboIJAOOcxEkGkj2rMAwGA1UdEwQFMAMBAf8wDQYJKoZI
hvcNAQELBQADggEBAFy4MyeJ3j65QIzEnHY5NvZPcRPOpu2aEbE1b60xnLItg6C8
+PpJ/DmIjcJLey8bsso4JHjxVGj4aPOyjB1adWwiU4EAejVjoPckRvEqCUROA7iK
ydMk2uZHIn927byKt7wbbMGmtDdWoECBp6SIqITBlrxFOMylnYZck3a+UU3VQcPq
TxltrtFFs3CM9u2yU9Dg7EZcFl9f+NEfrd21MO2LAG4M40DYnFMAEbn/6AlyB9ZG
2btiNmWYedxBBDMeMrnyP6nTQJdcI0+QKiWtP96cE+Rv5QJB04tcpY6gAgTzkO8q
8fVNxxjSRAxknBsT3t4jjD6/wjqRYMR6bfsK/e4=
-----END CERTIFICATE-----
subject=/C=SL/ST=Western/L=Colombo/O=UOP/OU=Faculty of Engineering/CN=Seran
issuer=/C=SL/ST=Western/L=Colombo/O=UOP/OU=Faculty of Engineering/CN=Seran
---
```

Fig. 5 Server certificate (the cipher suite ECK is highlighted)

Following command is used to run the client:

```
$openssl s_client -connect localhost:44330 -cipher ECK
```

```
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: ECK, 2048 bits
---
SSL handshake has read 2440 bytes and written 472 bytes
Verification error: self signed certificate
---
New, TLSv1.2, Cipher is ECK-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECK-RSA-AES256-GCM-SHA384
    Session-ID: 0AE663FAB1FCF7D8ECC0FD413711EA878E3BE6C7A16F2F8D11540C8AD963DB09
    Session-ID-ctx:
    Master-Key: 0F57502827AFCF5674B8D58FD0E279E322C7F8170AD0B398AF09BB3A13417D7BFB45334E61C331BA7D6B419FD270067D
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 7200 (seconds)
    TLS session ticket:
    0000 - 28 cc 22 8f ac 95 16 b3-9f 1a 9d 97 d6 d4 fd a3   (.".............
    0010 - b5 8e a9 44 33 c9 31 a4-b6 3c 01 ad 0a db 0f 19   ...D3.1..<......
    0020 - 56 47 3a a5 63 98 49 e0-fe 1b 6d 83 6c da 25 c5   VG:.c.I...m.l.%.
    0030 - 96 31 be a2 bf 4a 49 81-18 89 10 21 32 e3 8f 91   .1...JI....!2...
    0040 - bf 4d f4 5d 9c 42 d5 60-79 5f 3c f8 fc e1 86 73   .M.].B.`y_<....s
    0050 - b5 1f b2 e7 6d 0c ac 76-13 7b 63 b4 08 39 a6 0b   ....m..v.{c..9..
    0060 - 4d 1d 90 47 48 1b 3b d6-2e db 2f e6 07 4e d6 c1   M..GH.;.../..N..
    0070 - 6c 96 2b 7d c2 b0 cb 5e-91 76 15 2c f0 81 0c b0   l.+}...^.v.,....
    0080 - 13 66 e5 8a b6 cc 86 01-e3 36 0d 1f 3d a4 51 ed   .f.......6..=.Q.
    0090 - 11 6f aa b8 2d 91 ce 1d-18 ae 30 82 ad 39 bf 57   .o..-.....0..9.W

    Start Time: 1498769790
    Timeout   : 7200 (sec)
    Verify return code: 18 (self signed certificate)
    Extended master secret: yes
---
```

Fig. 6 Client/ Server connection

Then the following command is used to run the server:

```
$openssl s_server -key key.pem -cert certificate.pem -accept 44330 -www
```

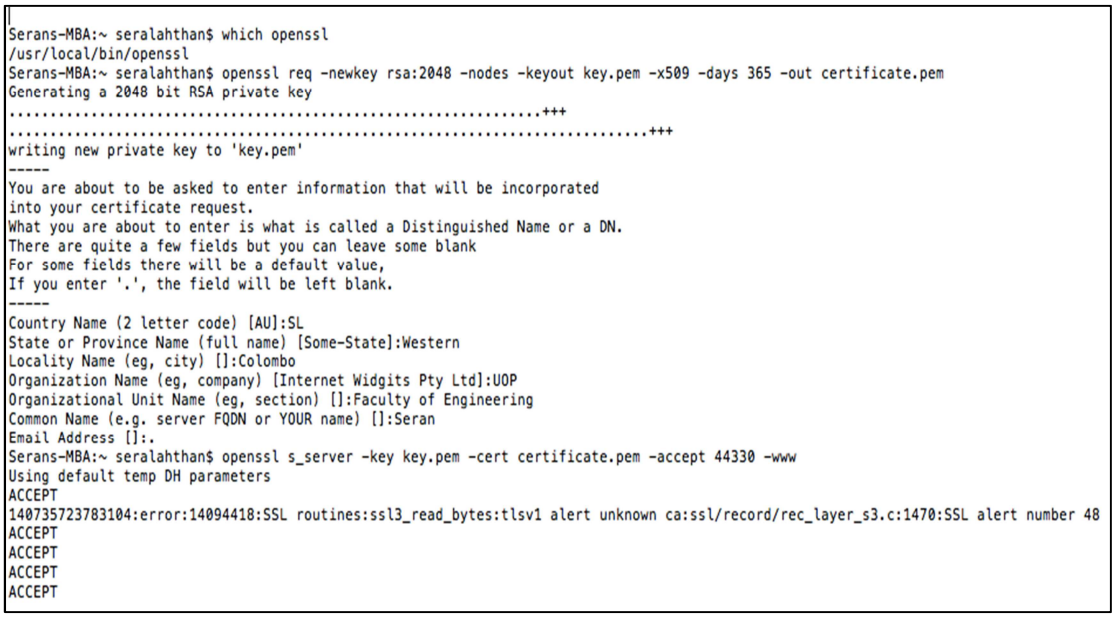

Fig. 7 shows generation of RSA private key certificate for connecting with the local server.

IV. CONCLUSIONS

In this work we implement the protocol P1 (eCK-secure and NAXOS trick free authenticated key exchange protocol) to be used with the widely-used OpenSSL cryptographic library. OpenSSL implementations are widely used with the real-world security protocol suites, such as Security Socket Layer and Transport Layer Security. According to our understanding, this is the first OpenSSL implementation of an eCK-secure key exchange protocol. Thus, our work opens up the direction to use the recent advancements of cryptography for betterment of the real-world Internet communication.

As a future work, we aim to implement a leakage-resilient AKE protocols [9], [13]-[15] for OpenSSL, which is resilient to wide range of side-channel attacks, in addition to eCK security.

ACKNOWLEDGMENT

This research is supported by Faculty Computer Science, UTHM, Malaysia and Information Technology (FSKTM), Research Management Centre (RMC), H082 Tier 1/2018 and Gates IT Solution Sdn. Bhd. under its publication scheme.

REFERENCES

[1] W. Diffie and M. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory 20(6), pp 644-654, 1976.

[2] V. Boyko, P. MacKenzie, and S. Patel, *Provably Secure Password-authenticated Key Exchange using Diffie-Hellman*, EUROCRYPT 2000, pp 156 – 171, Springer, 2000.

[3] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, *Authentication and Authenticated Key Exchanges*, Des. Codes Cryptography 2(2), pp107 – 125, 1992.

[4] A. Fujioka, K. Suzuki, and B. Ustaoglu, *Utilizing Postponed Ephemeral and Pseudo-Static Keys in Tripartite and Identity-based Key Agreement Protocols*, IACR Cryptology ePrint Archive, 2009:423, 2009.

[5] D. P. Jablon, *Strong Password-only Authenticated Key Exchange*, SIGCOMM Computer Communication Revise 25(5), pp 5 – 26, 1996.

[6] H. Krawczyk, *HMQV: A High-performance Secure Diffie-Hellman Protocol*, CRYPTO 2005, pp 546 – 566, Springer, 2005.

[7] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, *An Efficient Protocol for Authenticated Key Agreement*, Des. Codes and Cryptography 28(2), pp 119—134, Springer, 1998.

[8] B. A. LaMacchia, K. E. Lauter, and A. Mityagin, *Stronger Security of Authenticated Key Exchange*, Provsec 2007, pp 1 – 16, Springer, 2007.

[9] J. Alawatugoda, D. Stebila, and C. Boyd, *Continuous After-the-fact Leakage-resilient eCK-secure Key Exchange*, IMA Cryptography and Coding 2015, pp. 277 – 294, Springer, 2015.

[10] M. Bellare and P. Rogaway, *Entity Authentication and Key Distribution*, CRYPTO 1993, pp 232 – 249, Springer, 1993.

[11] M. Bellare and P. Rogaway, *Provably Secure Session Key Distribution – The Three Party Case*, STOC 1995, pp 57 – 66, 1995.

[12] R. Canetti and H. Krawczyk, *Analysis of Key-exchange Protocols and Their Use for Building Secure Channels*, EUROCRYPT 2001, pp 453 – 474, Springer, 2001.

[13] J. Alawatugoda, *Generic construction of an eCK-secure key exchange protocol in the standard model*, International Journal of Information Security 16(5), pp 541 – 557, Springer, 2017

[14] J. Alawatugoda, *On the leakage-resilient key exchange*, Journal of Mathematical Cryptology 11(4), pp 541 – 557, Springer, 2017

[15] S. Chakraborty, J. Alawatugoda and C. Pandu Rangan, *Leakage-resilient non-interactive key exchange in the continuous-memory leakage setting*, Provsec 2017, pp167—187, Springer, 2017