International Journal of Computer Applications (0975 – 8887)
Volume 7– No.5, September 2010

A Comparison Study on Key Exchange-Authentication protocol

Razieh Mokhtarnameh* Nithiapidary Muthuvelu* Sin Ban Ho* Ian Chai*

*Faculty of Information Technology, Multimedia University,
Jalan Multimedia, 63100 Cyberjaya, Selangor, Malaysia

ABSTRACT

A key exchange protocol enables two parties to share a common key for encrypting a large amount of data. Authentication is an essential requirement prior to the key exchange process in order to prevent man-in-the-middle attack. It is important to understand the capabilities and performance of the existing key exchange protocols before employing the protocols in our applications. In this paper, we compare Secure Socket Layer, Secure Shell, and Identity-based key exchange protocols by quantifying the performance, complexity, and level of security of each protocol. Detailed experiments and observations are conducted to examine the protocols in terms of disk usage, computation time, and data transmission time. The analysis shows that the identity-based key exchange maintains similar security level as the other protocols, while conveying better performance.

General Terms

Security, protocol, cryptography.

Keywords

Key exchange protocol, performance, security, complexity.

1. INTRODUCTION

The Internet provides great access to valuable data, not only to legal users, but also to the hackers, data thieves, and network sniffers. As a result, data confidentially is an essential requirement for secure data communication over the Internet; protecting data from being disclosed to unintended parties while being communicated between the authorized entities.

It is common to utilize symmetric encryption to encrypt large amount of data being transferred between the authorized entities. In this symmetric encryption, a shared or common key is used by the entities to encrypt and decrypt the data. This common key is shared between the entities using a key exchange protocol which involves exchanging messages over an open channel. Thus, there is a need to prevent the sniffers from obtaining a copy of the key.

Key exchange together with entity authentication will avoid the man-in-the-middle attacks [17]. These attacks take place when a trusted server is impersonated by a malicious server. Therefore, the key exchange protocols are associated with authentication protocols.

The performance of authentication and key exchange protocols affects the overall data communication process in terms of time and CPU cycles, especially when a particular application involves massive, frequent communication of sensitive data. Peer-to-peer systems, ad hoc network environments, and distributed systems are examples of environments which involve frequent key exchange. It is usually desirable to limit the amount of data compromised if an attacker learns the key.

Our focus is to study and examine the key exchange protocols in order to quantify their capabilities in terms of security level, computation time, and data transmission time. These performance analyses will help us to decide on the appropriate or most suitable protocol when one has to apply it to an application. As for this purpose, three key exchange protocols (which are associated with authentication protocols) are chosen, namely, Identity-based Key Exchange (ID- KEX) [25], Secure Socket layer (SSL) [14] (that uses Public Key Infrastructure (PKI) [24] for mutual authentication and key exchange) and Secure SHell (SSH) [4].

The rest of the paper is organized as follows: the related work on key exchange protocols is presented in section 2. Section 3 describes the experimental setup for evaluating the three key exchange protocols (ID-KEX, SSL, SSH). Section 4 provides a study on their security and complexity levels. Complexity consists of the amount of disk space being used by the protocols and the performance trade-offs. The performance trade-offs in terms of computational and communication costs are identified and discussed further. The results and discussions are included in subsections of section 4. Section 5 concludes the paper with a discussion on the future work.

2. RELATED WORK

2.1 Diffie-Hellman Session Key Agreement

Diffe-Hellman session key agreement is the first key exchange protocol, proposed by Diffie and Hellman [11] as shown in Figure1. Diffie-Hellman key exchange by itself achieves perfect forward secrecy because no long-term keying material exists at the end of the session to be disclosed. However, it does not provide authentication of the communicating parties; hence it is vulnerable to a man-in-the-middle attack.

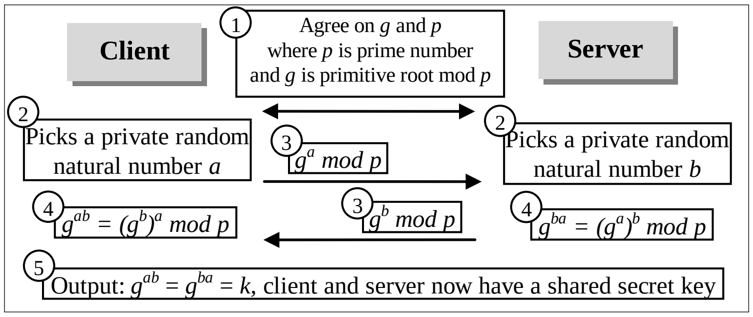

Client
Server
1 Agree on g and p where p is prime number and g is primitive root mod p
2 Picks a private random natural number a
2 Picks a private random natural number b
3 $g^a \bmod p$
3 $g^b \bmod p$
4 $g^{ab} = (g^b)^a \bmod p$
4 $g^{ba} = (g^a)^b \bmod p$
5 Output: $g^{ab} = g^{ba} = k$, client and server now have a shared secret key

Figure 1. Diffie-Hellman key exchange (DH-KEX)

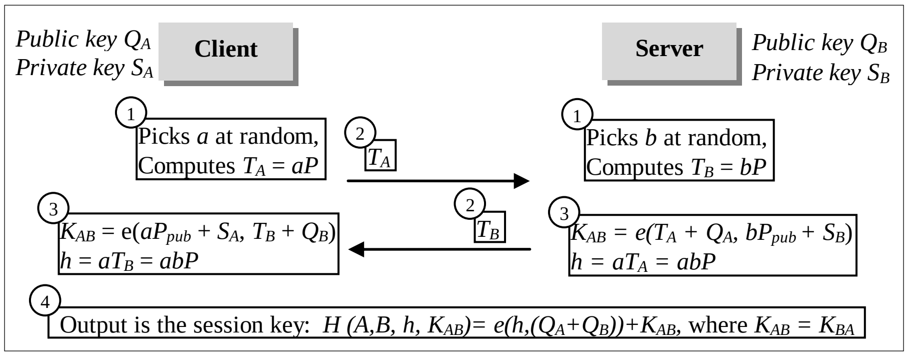

Public key Q_A Private key S_A
Client
Server
Public key Q_B Private key S_B
1 Picks a at random, Computes $T_A = aP$
2 T_A
1 Picks b at random, Computes $T_B = bP$
3 $K_{AB} = e(aP_{pub} + S_A, T_B + Q_B)$ $h = aT_B = abP$
2 T_B
3 $K_{AB} = e(T_A + Q_A, bP_{pub} + S_B)$ $h = aT_A = abP$
4 Output is the session key: $H(A,B, h, K_{AB}) = e(h,(Q_A+Q_B))+K_{AB}$, where $K_{AB} = K_{BA}$

Figure 2. ID-based key exchange (ID-KEX)

2.2 Station-To-Station (STS) Protocol

In order to fix the security flaw in the Diffie-Hellman protocol, the Station-To-Station (STS) protocol was proposed in [12]. To add authentication, the STS protocol requires both the parties to have a pair of public keys for signature generation and verification, and to know a publicly released symmetric key encryption. In contrast, note that the Diffie-Hellman protocol does not have these assumptions. These assumptions can be included into the protocol by sending public key certificates if the keys are not known in advance. In the STS protocol, STS protocol uses signatures to authenticate the communicating parties. It encrypts the signatures with the session key subsequently to show the knowledge of this session key. However, signatures and certificates cause the messages to increase considerably in size.

2.3 Secure Socket Layer (SSL)

SSL [14] involves negotiating and establishing secure connections, and securing the data transmission. SSL handshake uses certificates and PKI [24] for mutual authentication and key exchange. PKI binds public keys with particular user identities by means of a certificate authority (CA). The CA is the trusted entity that signs and issues digital certificates [13] to other parties. A digital certificate contains a public key and the identity of the owner and the validity period of the certificate. Therefore, authentication is performed through sending and verifying certificates which involve a great overhead. SSL key exchange can use an RSA algorithm, an asymmetric technique for session key exchange which encrypts the session key from the client to the server. A Diffie-Hellman key exchange can also be used which is more secure since both parties agree on the session key without having to send the key across the wire.

2.4 ID-based Authenticated Key Agreement

Many protocols were proposed for ID-KEX [20] [19] [9] [10]. Paterson and Price [18] noted that the aim in designing a good ID-KEX protocol is to achieve all the properties of the best usual key agreement protocols while trying to maximize efficiency. The public key can be chosen by any client in the system as it is generated from public information (email address or network address). Each party, then contacts the trusted authority (TA) once to authenticate and get the required private key. Therefore, there is no earlier distribution of keys between individual participants. Yuan and Li [25] proposed an efficient ID-KEX Protocol as shown in Figure 2. A key agreement protocol is said to be authenticated if it offers the guarantee that only the participating parties of the protocol can compute the agreed key. Therefore, this ID-KEX protocol is authenticated because it uses public and private keys to generate a shared secret.

2.5 SSH (Secure SHell)

SSH is a secure network protocol used by the user to log into a remote computer running an SSH server [4]. It was designed to replace telnet which is an earlier protocol that passes username and password in plain text. However, SSH provides a secure transmission by encrypting the authentication strings and all the other data exchanged between the hosts. Ylonen and Lonvick explored three layers of the SSH protocol; the Transport Layer Protocol [23] provides host authentication, confidentiality (encryption), and integrity; the User Authentication Protocol [21] authenticates the client-side user to the server and provides a number of authentication methods; and the Connection Protocol [22] multiplexes the encrypted tunnel into several logical channels.

SSH supports both password authentication and public key authentication. Although passwords are convenient and they require no additional configuration or setup from the users, they can be guessed, and the hacker can get into the system. Public key authentication provides better security as every machine creates a public/private key by itself. SSH clients and servers maintain and check a database containing identifications for all the hosts that have been involved in the interactions. Therefore, the first time when a user connects to a remote entity, the user has to know or trust that the key fingerprint for that entity is correct as SSH does not practice a central authority to assure access for each entity.

The Diffie-Hellman key-exchange protocol has been the subject of many works. Canetti and Krawczyk [6] analyzed key-exchange protocols (Diffie-Hellman and key-transport) authenticated via symmetric or asymmetric techniques to obtain the proof of security. In [7] they presented a security analysis of the Diffie-Hellman key exchange protocol authenticated with digital signatures used by the Internet Key Exchange (IKE) standard. In addition, many ID-based authenticated key agreement protocols were proposed and compared with others [20] [19] [9] [10]. Lee, Malkin, and Nahum [16] focused on the different parts of SSL such as the strength of SSL/TLS servers; Castelluccia, Mykletun, and Tsudik [8] analyzed the performance of SSL/TLS Handshakes and suggested an improvement. Moreover, the performance of pre-shared and Public Key Exchange Mechanisms for TLS protocol has been reported by Kuo, Tschofenig, Meyer, and Fu [15].

3. Methodology

Our experimental environment contains one client machine and one server machine. The client machine is equipped with an Intel ® Pentium® 4 CPU 2.66 GHz with 512 MB of RAM and 40 G HDD. The server machine is prepared with an Intel® Pentium ®4 CPU 2.80 GHz with 512 MB of RAM and 50 G HDD. The length of the RSA keys are 2048 bits.

In our experiments, OpenSSL [2] is used for implementing SSL since it is the best-known open library for secure communication at the time of this writing. As shown in Figure 3, a context object is used to create a new connection object for each new SSL connection. These connection objects are used to do SSL handshakes, reads, and writes.

The PBC (Pairing-Based Cryptography) library [3] is used for implementing ID-KEX [25]. It is a C library that performs the mathematical operations underlying pairing-based cryptosystems. Figure 4 shows the algorithm for Yuan and Li ID-KEX. Pairings involve two groups of prime order *q*. The PBC library refers to them as G_1 and G_2, where G_1 is an additive group and G_1 denotes a related multiplicative group. The pairing is a bilinear map that takes two elements as input from G_1 and outputs an element of G_2. The parameters can be found in Figure 2.

```
➢ Generate CA certificate for all the machines
  with OpenSSL commands
➢ Generate key and certificate for each machine
  with OpenSSL commands
1.    initialize_ctx(): context
  initialization.
      a) SSL_library_init():Initialize the
         library which primarily loads up the
         algorithms that OpenSSL will be using
      b) SSL_CTX_new(): create the context
      c) SSL_CTX_use_certificate_chain_file():
         load the certificate chain
      d) SSL_CTX_use_Private-Key_file():load
         the private key
2.    tcp_connect(): create a TCP connection
  between client and server
3.    SSL_connect(): perform the SSL
  handshake to authenticate server and client,
  and establish the shared key
4.    check_cert(): check the host's
  certificate
```

Figure 3. Algorithm for SSL

```
Initialization steps:
   1. Setup program to generate system
      parameters (P, P_pub)
   2. Extract program to calculate the
      private key for each machine (S_a)
   3. Client and Server programs to
      calculate the shared key (K)
Client program:
➢ Initializing pairing and elements
ex:  element_init_G1(P,  pairing);//initialize
  element P from G_1
➢ Client/server connection
➢ Calculate the shared key (Q_a and Q_b are
  public keys and T_b is received from server
  program)
element_random(a);//assign random element to a
element_mul_zn(Ta, P, a);//T_a = P * a
element_mul_zn(h, Tb, a);//h = T_b * a
element_mul_zn(temp1, Ppub, a);
element_add(temp1, temp1, Sa);//temp1=temp1+S_a
element_add(temp2, Tb, Qb);
pairing_apply(Kab, temp1, temp2, pairing);
//applying pairing as K_ab = e(temp1,temp2)
element_add(temp3, Qa, Qb);
pairing_apply(temp4, h, temp3, pairing);
element_add(K, temp4, Kab);

Note: The Server program is similar to the
client program
```

Figure 4. Algorithm for ID-based authentication key exchange (ID-KEX)

```
➢ Generate key-pairs by openssh
➢ Setting the options
     1.      ssh_options_new()
     2.      ssh_options_set_username()
     3.      ssh_options_set_host()
     4.      ssh_options_set_ssh_dir()
     5.      ssh_options_set_identity()
➢ ssh_new(): create the session
➢ ssh_set_options(): give options to the
  session
➢ ssh_connect(): connecting the ssh server
➢ ssh_get_pubkey_hash(): create the hash of
  the server public key
➢ ssh_is_server_known():checks the user's
  known host file for server authentication
➢ ssh_userauth_autopubkey(): client
  authentication
➢ channel open session(): opening the channel
```

Figure 5. Algorithm for SSH

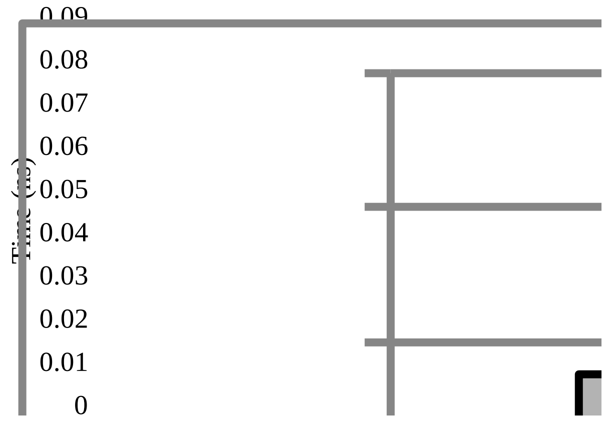


Figure 6. Connection and Computation Time in KEX Protocols (kextime).

Table 1. Cryptographic operations

	SSL (authentication + KEX)	ID-based authenticated KEX	SSH (authentication + DH-KEX)
Client	RSAverify + RSAencrypt + RSAsign	1 pairing + 3 point multiplication	RSAencrypt + 1modular exponentiation + 1 multiplication
Server	2* RSAverify + RSAdecrypt	1 pairing + 3 point multiplication	RSAdecrypt +1 modular exponentiation + 1 multiplication

Table 2. Processing Time (nanoseconds) for each KEX mechanisms

Methods	Connection time (ns)	Computation time (ns)	
		Client	Server
ID-KEX	0.000489	0.067591	0.053368
SSL	0.000795	0.078187	0.058589
SSH	0.008923	0.068206	0.055302

Next, libssh [1] is a C library used for SSH implementation (Figure 5). Client information (the host name of the server, the port, the binding address, the default username) has to be sent to the server before the client/server connection. The client information is given to an *ssh-connect* function as an option structure, then this information will be used repeatedly by the SSH implementation.

4. Results and Discussions

4.1 Performance Metrics

Performance for these key exchange mechanisms are analyzed and measured in terms of cryptographic computations and data transmission. Besides client and server processing time, a full handshake also involves delays due to message passing and network latency.

4.1.1 Computation Complexity

Table 1 depicts the cryptographic operations in the SSL, ID-based cryptographic scheme and SSH. An ID-KEX protocol is authenticated because it uses public and private keys to generate a shared secret. However, in an ID-KEX protocol, two parties transfer their parameters to each other and then calculate the shared key by using their public and private keys as in Yuan's algorithm [25]. Figure 2 and Table 1 show the cryptographic operations involved in the calculation of the shared key which cost one pairing and three point multiplications. The SSH computation includes a Diffie-Hellman key exchange, directory checking, and public key authentication for the client and server. SSH public key authentication involves one RSA encryption/decryption as shown in Table 1.

SSL computation involves verifying certificates, client and server authentication, and key exchanges that involves RSA encryption/decryption. The RSA key exchange is performed during the SSL handshake. RSA key exchange requires a message only from the client. The client generates a 256-bit random number, encrypts the number using the public keys of the server and sends it to the server. The server decrypts the random number as it possesses the private key. This random number is the key that used to perform symmetric encryption for data transmission after SSL handshake.

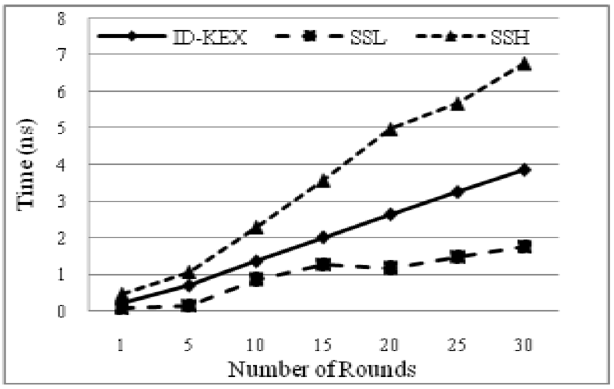
SSH encrypts all communications to and from the client and server while SSL attempts to make a connection with unencrypted channels. As shown in Table 2, SSH connection time is approximately 91% more than SSL and 94.52% more than ID-KEX. This is due to a significant cost related to the establishment of SSH sessions. Table 1 depicts the cryptographic operations in the SSL, ID-KEX and SSH.

The dominant cost for an ID-KEX is the evaluation of a pairing, whereas for SSL is the RSA encryption/decryption process. Pairing is approximately 47.79% slower than an RSA decryption with pre-computation which involves calculations of certain fixed parameters. However, the overall processing time for ID-KEX is less than SSL and SSH since they have overhead for their authentications. Figure 6 and Table 2 show connection time and client/server computation time in nanosecond (ns). In conclusion, SSL and SSH have more computation especially on client machine than ID-KEX.

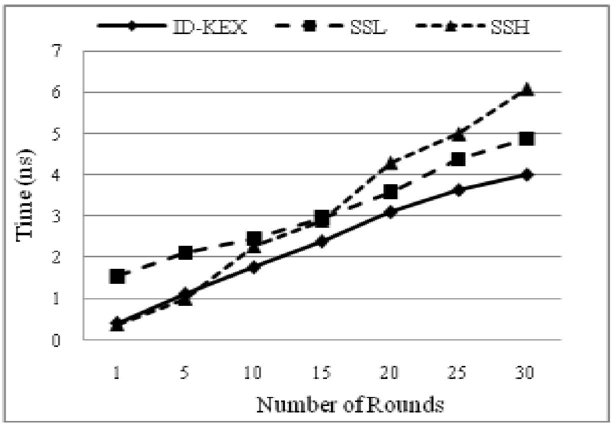
4.1.2 Data Transmission

The amount of data transmitted to complete the key exchange mechanisms are shown in Table 3. The ID-KEX protocol has the lowest transmitted amount of data as only two parameters are calculated (parameter by the client and parameter by the server) and exchanged; neither certificates nor public keys are transmitted.

Nevertheless, SSH needs more data to be exchanged since it has client and server authentication in which their public keys need to be transferred. As a result, ID-KEX is 1.5 times faster than SSH in terms of data transmission time. Furthermore, it is 9 times faster than SSL since SSL is using mutual authentication that exchanges certificates with the biggest data amount. Hence, SSL has the highest data transmission time.



Chart(a). Client



Chart(b). Server

Figure 7. Execution time for frequent authentication and KEX.

Table 3. Data Transmission Time for each KEX Mechanisms

Methods	Transmission time (ms)	Transmission data (bytes)
ID-KEX	0.092	2180
SSL	0.850	5544
SSH	0.139	5125

Table 4. Disk Space for each KEX Mechanism

Methods	Client (bytes)	Server (bytes)
ID-KEX	1310	1310
SSL	6800	6800
SSH	2134	2134

4.2 Frequent Key Exchange Experiments

Experiments were conducted to realize the effect of the protocols in an environment that involves frequent authentication and key exchange mechanisms. It is useful particularly for applications involve enormous, frequent transmissions of valuable data. Moreover, man-in-the-middle attack can be reduced through establishing a new shared key frequently. Figure 7 present the overall execution time for SSH, SSL, and ID-KEX protocols in the above mentioned environment.

The SSH duration increases rapidly in comparison with others by increasing number of rounds on both client and server. It is about 42.29% more than ID- KEX and 72.81% more than SSL on client. Furthermore, it is about 37.13% more than ID-KEX and 41.68% more than SSL on server. Therefore, it is better to be used in applications that need infrequent communications. The duration of SSL increases about 52.89% less than ID-KEX on client. Although the duration of SSL increases about 7.24% less than ID-KEX on server, ID-KEX takes less time than SSL. As a result, ID-KEX indicates better efficiency on the server than the client. On the other hand, SSL has better efficiency on the client than the server especially on 20 rounds and more. Therefore, for applications which involve frequent key exchange and prefer more lightweight server, the ID-KEX can be a suitable choice. Whereas, SSL is more appropriate for applications that need frequent key exchange and lightweight client is desirable.

4.3 Disk Space Consumption

The length of the RSA keys used in SSL and SSH is 2048 bits. The length of the keys in ID-KEX is 128 bits. This is one of the features of ID-based cryptography that provides smaller key sizes. In ID-KEX protocol, the same amount of disk space is used on client and server machines. The public keys are created on the fly (using the hostname or IP address of the machine). Hence, the client and server just need to allocate limited space for its private key (of 317 bytes) and system parameters (of 993 bytes); the overall required space is 1310 bytes. The total size of SSL resource/server certificate and the relevant key is 5400 bytes. In addition, the certificate of the TA or CA costs 1400 bytes. Thus, the minimum disk space needed for key/certificate on resource/server machine is 6800 bytes. In SSH, on each client or server there are public key (of 391 bytes) and private key (of 1743 bytes). As shown in table 4, SSL uses more space than the others.

4.4 Security Considerations

In a man-in-the-middle attack [17], an adversary sits between the client and server, intercepting all traffic and altering or deleting messages at will. ID-KEX is immune to man-in-the-middle attack as each party must know the public key of other party to generate the shared key. Therefore, they know with whom they have the shared key. SSL by using PKI and SSH by using public key authentication are not susceptible to man-in-the-middle attacks. However, SSH can be vulnerable the first time you connect to a remote entity. You may trust the key fingerprint for adversary by mistake instead of the original server. According to [5] all the three protocols that are being compared in this paper provide Known-Key Security[1] and Perfect Forward Secrecy[2], as a fresh and unique session key is generated by client and server that depend on the ephemeral Diffie-Helman private keys for each run of protocols. The ephemeral Diffie-Helman private keys are temporary, which means after each handshake; both the client and server delete their temporary private keys.

ID-KEX or the protocols which uses Diffie-Helman algorithm for key exchange, provide No Key Control[3] security property. In Diffie-Helman no entity can decide the key separately. However, RSA requires a message only from the client. The client generates a 256-bit random number and send to server. As a result, client can control the key.

5. Conclusion and Future Work

We have presented a comparison study between SSL, SSH and ID-based key agreement protocols in their authentication and key exchange parts. Furthermore, we have discussed and analyzed their security and complexities aspects. The analysis shows that in order to provide a satisfying security level, SSL is slower than the others in terms of data transmission time and mutual authentication. While SSL is heavy weight, it is the standard behind many secure communications on the Internet. SSH has less complexity but can be vulnerable to man-in-the-middle attacks. We noticed that ID-KEX has better performance than SSH and SSL. Although the concept of ID-based public keys appears to be new at the time of this writing, it has some useful security features in terms of key size and management as compared to SSL using the PKI approach.

We have compared the protocols for frequent authentication and key exchange. The ID-KEX and SSL both can be suitable for frequent key exchange applications. However, various applications have different requirements or constraints. For example, ID-KEX can be more suitable for applications that require less overhead especially on the server side and less communication bandwidth such as wireless networks. The dynamic and ephemeral network topology demands frequent key exchanges, in ubiquitous and pervasive computing applications, such as sensor networks or RFID-systems, the devices in use as nodes of the network often require severe size limitations and power consumption constraints.

The comparison presented in this paper may well be tested in different applications. These may include, for example, peer to peer systems, ad hoc network environments, and distributed systems, as well as grid systems in which it is desirable to have lightweight and flexible security mechanisms.

6. REFERENCES

[1] The libssh project, http://www.libssh.org/

[2] The openssl project, http://www.openssl.org/

[3] The pbc library, http://crypto.stanford.edu/pbc/

[4] Barrett, D., Silverman, R. 2005. SSH: The Secure Shell (The Definitive Guide). O'Reilly, 2nd edition edn.

[5] Blake-Wilson, S., Johnson, D., Menezes, A. 1997. Key agreement protocols and their security analysis. In: 6th IMA International Conference on Cryptography and Coding. Lecture Notes in Computer Science, vol. 1355, pp. 30-45. Springer Berlin / Heidelberg.

[6] Canetti, R., Krawczyk, H. 2001. Analysis of key-exchange protocols and their use for building secure channels. In: EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques. pp. 453-474. Springer-Verlag, London, UK.

[7] Canetti, R., Krawczyk, H. 2002. Security analysis of ikes signature-based key-exchange protocol. In: In: Proc. CRYPTO02, Springer LNCS 2442. pp. 143-161. Springer-Verlag.

[8] Castelluccia, C., Mykletun, E., Tsudik, G. 2005. Improving secure server performance by re-balancing ssl/tls handshakes. In: in Proceedings of the 10th Annual USENIX Security Symposium. pp. 26-34.

[9] Chen, L., Kudla, C. 2002. Identity based authenticated key agreement protocols from pairings. In: In: Proc. 16th IEEE Security Foundations Workshop. pp. 219-233. IEEE Computer Society Press.

[10] Choie, Y. J., J.E., Lee, E. 2005. Efficient identity-based authenticated key agreement protocol from pairings. Applied Mathematics and Computation 162, 179-188.

[11] Diffie, W., Hellman, M.E. 1976. New directions in cryptography.

[12] Diffie, W., Van Oorschot, P.C., Wiener, M.J. 1992. Authentication and authenticated key exchanges. Des. Codes Cryptography 2(2), 107-125.

[13] Feghhi, J., Feghhi, J., Williams, P. 1999. Digital Certi_cates: Applied Internet Security. Addison Wesley Longman.

[14] Frier, A., K.P., Kocher, P. 1996. The secure socket layer. Technical report, Netscape Communications Corp.

---

[1] Known-Key Security: Client and server should generate a unique secret key in each round of key agreement protocol. Each key generated in one protocol round is independent and should not be exposed if other secret keys are compromised.

[2] Perfect Forward Secrecy: If secret key is compromised, the previously established session keys are not compromised.

[3] No Key Control: The key should be determined jointly by both entities. None of the entities can control the key alone.

[15] chun Kuo, F., Tschofenig, H., Meyer, F., Fu, X. 2006. Comparison studies between pre- shared and public key exchange mechanisms for transport layer security. In: 25th IEEE International Conference on Computer Communications. pp. 1-6.

[16] Lee, H.K., Malkin, T., Nahum, E. 2007. Cryptographic strength of ssl/tls servers: cur- rent and recent practices. In: IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement. pp. 83-92. ACM, New York, NY, USA.

[17] Ornaghi, A., Valleri, M. 2003 Man in the middle attacks. In: Black Hat Europe 2003. US.

[18] Paterson, K., Price, G. 2003. A comparison between traditional public key infrastructures and identity-based cryptography. Information Security 8(16), 57-72.

[19] Shim, K. 2003. Efficient id-based authenticated key agreement protocol based on the weil pairing. Electronics Letters 39(8), 653-654.

[20] Smart, N.P. 2002. An id-based authenticated key agreement protocol based on the weil pairing. Electronics Letters 38(13), 630-632.

[21] Ylonen, T., Lonvick, C.E. 2006. The secure shell (ssh) authentication protocol, rfc 4252.

[22] Ylonen, T., Lonvick, C.E. 2006. The secure shell (ssh) connection protocol, rfc 4254.

[23] Ylonen, T., Lonvick, C.E.2006. The secure shell (ssh) transport layer protocol, rfc 4253.

[24] 24. Younglove, R. 2001. Public key infrastructure. how it works. Computing & Control Engineering Journal 12, 99-102.

[25] Yuan, Q., Li, S. 2005. A new efficient id-based authenticated key agreement protocol. Cryptology ePrint Archive: Report 2005/309.