JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 18, 125-139 (2002)

Short Paper

Two-Pass Hybrid Key Distribution Protocol Based on ECC

SUNG-MIN LEE AND TAI-YUN KIM
Department of Computer Science and Engineering
Korea University
Seongbuk-gu, Seoul, Korea
E-mail:{smlee, tykim}@netlab.korea.ac.kr

In this paper we propose a two-pass hybrid key distribution and authentication protocol. The proposed protocol minimizes the number of message exchanges and the key management problem as it eliminates KDC, by using both symmetric-key and asymmetric-key schemes. In addition, it guarantees explicit entity and key authentication via a signature scheme based on elliptic curve cryptosystems (ECC) whose efficiency is superior to existing signature schemes with only two-message exchanges. As each entity has the same number of exponential operations, it also guarantees load balance among each entity's processing. We present proofs of security of our protocol using the formal methods Casper and FDR. The proposed protocol can be efficiently applied to various communication systems in distributed computing environments.

Keywords: key distribution, authentication, formal methods, ECC, security analysis

1. INTRODUCTION

In today's distributed systems, secure communication is very important. It is also important to use a secure and efficient key distribution in distributed computing environments. For this reason, much effort has been invested in providing security services in a variety of network and operating system environments [6]. There are many protocols based on public key and symmetric key schemes. In some of these protocols, limitations and flaws have been discovered [4, 10]. As the amount of mobile communication systems increases rapidly, it is difficult and inefficient for a central key distribution center (KDC) to keep the keys of all mobile systems for secure communication. In conventional key distribution protocols, a session key is generated by the KDC and each entity's secret keys that should be shared with the KDC for secure session key exchange, are also managed. So the load is especially biased on the KDC.

In this paper we propose a hybrid key distribution and authentication protocol which needs only two messages for complete key establishment and entity authentication, and spreads the load equally between the entities. The proposed protocol minimizes the number of message exchanges and the key management problem by supporting end-to-end

Received December 27, 1999; revised April 23 & August 1, 2000; accepted October 5, 2000.
Communicated by Chi Sung Laih.

end-to-end computation for key establishment using both symmetric and asymmetric schemes without KDC. It also provides explicit on-line entity and message authentication via an ECC-based signature scheme that is more efficient than other signature schemes. When the server authenticates entities in the protocol, the server needs each entity's authentic signature public key. In our protocol it can be obtained from a certificate transferred and checked for validity using the certifier's signature. So, the server in our protocol needs to keep only certifier's public key for authentication. Our protocol relieves entities of the problem of key management.

Security of key distribution protocol is very important. We analyze the proposed protocol by using a formal methods Casper [18] and FDR. We first specify the proposed protocol using Casper and generate a CSP description, and then check its security using its model checker FDR.

The remainder of this paper is organized as follows. Section 2 discusses previous protocols and their problems. Section 3 describes the proposed key distribution protocol. Section 4 describes the authentication mechanism of the proposed protocol. In section 5 security of the proposed protocol is analyzed using formal methods. Section 6 compares our results with existing protocols. Finally, section 7 provides our conclusions.

2. PREVIOUS KEY DISTRIBUTION PROTOCOLS

In this section we review previous protocols, such as the symmetric key based protocol, the public key based protocol and the TMN protocol along with their problems.

2.1 Classical Key Distribution Protocol

In classical key distribution protocol, a trusted server generates and distributes secret data using a symmetric key method to users. Even though this method is fast in encryption and decryption, it needs a symmetric key pre-distribution mechanism [19, 20]. If a classical key distribution method is employed for the key encryption, then the KDC should manage each entity's secret key. Load is especially biased on the KDC since it generates a session key and manages secret keys. There are many of classical key distribution protocols, such as Needham-Schroeder shared key protocol [1], Otway-Rees [2], Kerberos [3], etc.

2.2 Public-key-based Key Distribution Protocol

A key distribution protocol based on public-key encryption scheme which is invented by Diffie and Hellman [14] involves one party choosing a symmetric key and transferring it to a second, using that party's encryption public key. As the order of the finite field should be very large, the scheme is secure [22]. If a public key distribution method is employed for the key encryption, then the management problem is reduced. However, its encryption and decryption speed is slower than the symmetric key scheme [19, 20]. There are several public key-based key distribution and authentication protocols such as Needham-Schroeder PK, X.509 [19], and so on.

2.3 TMN Protocol

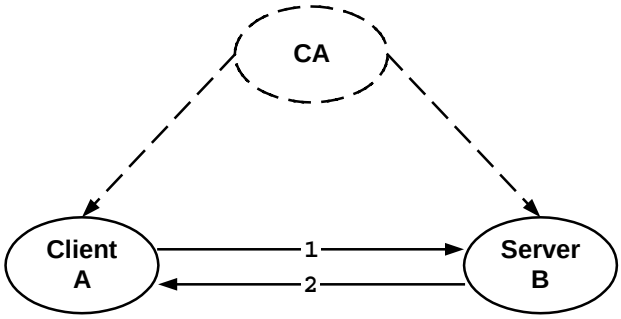
The TMN protocol is suitable for a mobile communication systems [9]. The protocol concerns three players: an initiator, a responder, and a key distribution center (KDC) who mediates between them. It employs two sorts of encryption. A public key encryption is employed for uplink channels (from a user terminal to a network center) so that the KDC is free from key management problems. A secret key encryption is employed for downlink channels (from a network center to user terminals) so that it enables high speed performance. But it has several problems. Lowe and Roscoe found some holes in security of the protocol using the process algebra CSP and its model checker FDR [10]. And for authentication the protocol needs an additional pre-processing procedure that the KDC issues each entity's secret and manages it and all users' information.

3. PROPOSED KEY DISTRIBUTION PROTOCOL

The objectives of the proposed protocol are to minimize the number of message exchanges required between the parties for complete key establishment, to distribute a balanced load (i.e. the messages sent, messages received, and exponential computations) to each entity in the protocol, and to reduce the key management problem using a hybrid scheme. In addition it provides an efficient entity and message authentication scheme since it uses an elliptic curve based signature whose key size is considerably smaller than existing digital signatures.

Three types of principles are involved in our protocol: clients, servers, and certificate authority (CA). The CA is a trusted third-party which issues certificates for each entity. We assume that all public keys of the principles involved in our protocol had been registered in the CA, and client *A* and server *B* keep certifier CA's public key to verify the authenticity of each entity's certificate that is received. We also assume that client *A* and server *B* receive the needed certificates from CA, and keep them initially via an off-line method. This procedure is required only once as a pre-processing procedure. In our protocol certificate consists of an ECDSA [7] signature verification key and a RSA [15] encryption key. Client *A* uses the RSA encryption key obtained from $cert_B$ for the session key encryption key. It also uses the signature verification key obtained from $cert_B$ to authenticate the reality of server *B*. Neither entity has to access the CA after the initialization process, since the certificate for signature verification is sent from its owner. So we consider the process of issuing and sending a certificate as a pre-processing procedure in the proposed protocol for environment setup.

Fig. 1 shows key distribution steps of the proposed protocol. In message (1), the protocol uses the RSA encryption algorithm for session key encryption (i.e. $P_B(I_B, r_A)$). RSA is a public key scheme based on security due to the difficulty of factoring large numbers. In message (2), it uses the Vernam cipher which is considered as a classical key encryption scheme. The Vernam encryption of two keys r_A and r_B, which we write as $V(r_A , r_B) \equiv r_A \oplus r_B$, is their bit-wise addition modulo 2. Note that $V(r_A , V(r_A , r_B)) = r_B$, so if entity *A* knows r_A ,then it can decrypt $V(r_A , r_B)$ to obtain r_B.



(1) A -> B : $P_B(I_B, r_A), N_A, S_A(P_B(I_B, r_A), N_A), cert_A$

(2) B -> A : $I_A, V(r_A, r_B), S_B(I_A, V(r_A, r_B)), Er_B(N_A, L), cert_B$

Fig. 1. Key exchange steps.

Notation

$E_x(y)$ denotes the result of applying a symmetric encryption function using a key x.
$P_x(y)$ denotes the result of applying x's RSA public key function to y.
$S_x(y)$ denotes the result of applying x's private key function (i.e. ECDSA signature generation function) to y.
I_x denotes an identifying string for entity x.
N_x denotes a nonce generated by x.
P_x denotes the x's public key.
$Cert_x$ denotes a public key certificate: $cert_x = (I_x, P_x, S_{CA}(I_x, P_x))$. The certificate contains x's identity and signature public key, plus trusted server CA's signature S_{CA} over these.
L denotes the lifetime of a session key issued by server B.

The proposed protocol begins with client *A* initiating the session by generating random number r_A as a key-encryption key and then sending message (1) to server *B* containing $P_B(I_B,r_A)$, *A*'s nonce N_A, *A*'s signature over $P_B(I_B,r_A)$ and N_A, together with its certificate $cert_A$ which contains *A*'s identity I_A, *A*'s ECDSA signature public key P_A, plus trusted certificate server CA's signature S_{CA} over these: $S_{CA}(I_A,P_A)$. *B* receives the message and authenticates *A*'s identity, integrity of the message via CA's signature and *A*'s signature since *B* keeps CA's signature public key then decrypts $P_B(I_B, r_A)$ using its RSA private key and obtains r_A. *B* need not keep the client's public key to verify *A*'s signature, since the client sends the message containing its public key (i.e. $cert_A$). *B* then generates random number r_B, which will be used as a session key between *A* and *B*, its lifetime *L*. *B* issues the Vernam encryption of two keys r_A and r_B and then sends message (2) which consists of $cert_B$, *A*'s identity, *A*'s nonce and lifetime of r_B which are encrypted under the session key r_B, signature over I_A and $V(r_A,r_B)$ to client *A*. Finally, *A* can verify *B*'s identity using *B*'s signature. *A* then decrypts $V(r_A,r_B)$ by its key-encryption key r_A and gets $V(r_A, V(r_A,r_B)) = r_B$ as a session key with *B*. Using r_B, *A* gets the lifetime of r_B and authenticates the session key r_B explicitly as it gets its own nonce. It also verifies that the nonce

N_A is the same as that included in message (1). Matching A assures freshness of the session key r_B. Therefore, A and B share mutual secret r_B and its lifetime.

4. AUTHENTICATION BASED ON ELLIPTIC CURVE CRYPTOSYSTEMS

For authentication, conventional key distribution protocols based on symmetric encryption scheme use a nonce [5]. Our proposed hybrid protocol uses a digital signature. While a nonce-based protocol guarantees only entity authentication, the proposed protocol provides both integrity of messages and entity authentication. Using a digital signature scheme, we authenticate mutual entity and integrity of the message and session key with only two message exchanges. The drawback of a signature-based authentication scheme is that it is slower than a nonce-based authentication scheme. But using an ECC-based signature, we can improve the efficiency when each entity authenticates one another and the integrity of the messages. In this section we give an overview of the elliptic curve cryptosystem and authentication scheme via ECDSA signature.

4.1 ECC Overview

Elliptic Curve Cryptosystems (ECC) were first suggested by Miller and Koblitz [11]. A main feature that makes ECC attractive is the relatively short operand length relative to a public-key cryptosystem such as RSA and systems based on the discrete logarithm in finite fields. ECC can provide various security services such as key exchange, privacy through encryption, and sender authentication and message integrity through digital signatures [12]. The benefit of elliptic curves comes from their ability to take any two points on a specific curve, add them together, and get another point on the same curve. More importantly for cryptography is the difficulty of figuring out which two points were added together to get there. Although ECC can be based on finite fields of any characteristic, practical systems have only been implemented over prime fields or Galois fields of characteristic two [13].

4.2 Explicit Entity and Message Authentication via ECDSA

ECC can be applied to digital signature schemes. We can authenticate the identity of certain users and integrity of messages using a digital signature based on ECC. The Elliptic Curve Digital Signature Algorithm (ECDSA) is a digital signature scheme whose security relies on the elliptic curve discrete logarithm problem (ECDLP) [7]. ECDSA is the elliptic curve analogue of the DSA [16]. But, it is touted as being at least as secure and considerably faster than either RSA [15] or DSA, with similar key and signature lengths. For example, an elliptic curve $E(Z_P)$ with a point $P \in E(Z_P)$ whose order is a 160-bit prime offers approximately the same level of security as DSA with a 1024-bit modulus p and RSA with a 1024-bit modulus n[7]. ECDSA is being proposed as an ANSI X9.62 standard. The key generation procedures for ECDSA are shown in Fig. 2. Each entity of our protocol generates a public and private key pair, then publishes the public key (E, P, n, Q) (i.e., it registers its public key in CA). We assume that all public keys of the principles involved in our protocol have been registered in CA.

1.Select an elliptic curve E defined over Z_P. The number of points in $E(Z_P)$ should be divisible by a large prime n.
2.Select a point $P \in E(Z_P)$ of order n.
3.Select a statistically unique and unpredictable integer d in the interval $[1,n-1]$.
4.Compute $Q = dP$.
5.A's public key is (E, P, n, Q); A's private key is d.

Fig. 2. ECDSA key generation of client *A*.

By using ECDSA signature, each entity of the proposed protocol can authenticate each other. In the protocol, a pair of entities (i.e. *A* and *B*) needs authentication using ECDSA. Fig. 3 shows an explicit authentication procedure between client *A* and server *B* via ECDSA signature. The procedure is as follows. First, client *A* generates message *m* which is concatenated with $P_B(I_B,r_A)$ and N_A, and selects a statistically unique and unpredictable integer *k* in the interval $[1,k-1]$. *A* then computes $kP = (x_1,y_1)$ and $r=x_1 \bmod n$. If $r=0$, then select *k* again. *A* computes $k^{-1} \bmod n$ and $s = k^{-1}\{h(m)+dr\} \bmod n$, where *h* is the secure hash algorithm (SHA-1). If $s=0$, select *k* again. Finally, *A* computes its signature. The signature over message *m* is the pair of integers (r,s). To verify *A*'s identity and the integrity of message *m*, *B* first extracts *A*'s authentic public key (E,P,n,Q) from $cert_A$, that is, a certificate binding party *A* to a public key suitable signature verification as it verifies the authenticity of $cert_A$, since *B* keeps certifier CA's public key. *B* then verifies that *r* and *s* are integers in the interval $[1,n-1]$ and computes $w=s^{-1} \bmod n$ and $h(m)$. *B* computes u_1 and u_2, and then computes $u_1P + u_2Q = (x_0,y_0)$ and $v=x_0 \bmod n$. Finally, *B* accepts the signature if and only if $v=r$, otherwise rejects it.

Client (A)	***Server (B)***
$m=P_B(I_B,r_A)$, N_A $kP=(x_1,y_1)$ and $r= x_1 \bmod n$, where $k \in [1,n-1]$ $s=k^{-1}\{h(m)+dr\} \bmod n$	
$m,(r,s)$, $cert_A$ ⟶	
	$w=s^{-1} \bmod n$ $u_1=h(m)w \bmod n$, $u_2=rw \bmod n$ $u_1P + u_2Q =(x_0,y_0)$, $v=x_0 \bmod n$ $Decision \in \{Accept, Reject\}$

Fig. 3. Explicit authentication procedure between *A* and *B* via ECDSA.

5. SECURITY ANALYSIS

In this section, we briefly describe methods for analyzing security protocols. Security of the proposed protocol is described with respect to both passive and active attacks. Then the proposed protocol is analyzed using formal methods such as Casper and FDR.

5.1 Methods for Analyzing Security Protocols

In recent years, a method for analyzing security protocols using the process algebra CSP and its model checker FDR has been developed. Protocol descriptions are interpreted in Hoare's language of Communicating Sequential Processes (CSP) [21], as are specifications such as confidentiality. Formal Systems' FDR is used as the model checking method. This technique has proved remarkably successful, and has been used to discover a number of attacks upon protocols [17]. The procedures for analyzing security protocols are as follows:

- Each agent taking part in the protocol is modelled as a CSP process.
- The most general intruder who can interact with the protocol is also modelled as a CSP process.
- The resulting system is tested against specifications representing desired security properties. FDR searches the state space to investigate whether any insecure traces can occur.

The task of producing the CSP description of the system is very time-consuming, and is doable only by people well practiced in CSP. Even experts often make mistakes that prove hard to spot. In order to address these problems, Lowe proposed Casper, which is written in the functional programming language Haskell [18], simplifying the specification process. The user specifies the protocol using a more abstract notation, and Casper compiles this into CSP code, which is suitable for checking using FDR. Casper script is split into up to nine sections, two of which are optional. Each section is headed by a line beginning with #. The following Casper script is shown in EBNF form.

Script ::= free-variables-section
processes-section
protocol-description-section
specification-section
[equivalences-section]
actual-variable-section
[functions-section]
system-section
intruder-section

5.2 Passive Attack vs. Active Attack

A passive attack involves an adversary who attempts to defeat a cryptographic technique by simply recording data and later analyzing it. The proposed protocol uses RSA, whose security relies on the difficulty of factoring integers, and ECDSA signature based on elliptic curve discrete logarithm problem (ECDLP): given an elliptic curve E defined over a finite field F_q and two points $P, Q \in E(F_q)$, find an integer l such that $lP = Q$ in E, provided that such an integer exists [7,11]. So, the proposed protocol is secure against passive attack.

An active attack involves an adversary who modifies or injects messages. The network environment is untrusted, communication links may be susceptible to wiretapping, interception and replay of messages. We describe the security of the protocol by employing an intruder. We assume the intruder can :

- Overhear messages in order to learn their contents, possibly intercepting these messages;
- Forge new messages using messages he knows;
- Use keys that he knows so as to take part in fake sessions or overhear sessions between two other entities.

Then we analyze the security of the proposed protocol using formal methods such as Casper and FDR.

5.3 Analyzing the Proposed Protocol Using Casper

In order to generate a CSP description of the proposed protocol, we first construct a Casper input file. The file must define not only the operation of the protocol, but also the system to be checked. The Casper input description hence, contains two distinct parts:

- A definition of the way in which the protocol operates, describing the messages passed between the agents, the tests performed by the agents, the types of the data items used, the initial knowledge of the agents, a specification of what the protocol is supposed to achieve, and a definition of any algebraic equivalences over the types used.
- A definition of the actual system to be checked, defining the agents taking part in the actual system and the roles they play, the actual datatypes to be used, and the intruder's abilities.

The type of variables and functions that are used in the protocol definition are defined under the heading "#Free variables". The definition of free variables of the proposed protocol takes the following form. We define the agents, symmetric keys, public keys, and nonce. The functions *PK* and *SK* return an agent's public key and secret key respectively.

#Free variables
a,b : Agent
ka, kb : SessionKey
PK : Agent -> PublicKey
SK : Agent -> SecretKey
InverseKeys = (PK, SK), (ka, ka), (kb, kb)
na : Nonce

Each agent running in the system will be represented by a CSP process. The names of the CSP processes representing the agents are defined below. The parameters and variables following keyword *"knows"* define the knowledge that the agent in question is

expected to have at the beginning of the protocol run. So client A of our protocol is expected to know his own identity *a*, the server's identity *b*, the session key *ka*, his nonce *na*, the public key function *PK*, and his secret key *SK(a)*.

#Processes
CLIENT(a,b,ka,na) knows PK, SK(a)
SERVER(b,a,kb) knows PK, SK(b)

The main part of the definition of the protocol is the definition of the sequence of messages in our protocol. The notation used is similar to the standard method of describing protocols, as in Figure 1. The notation *{m}{k}* means message *m* encrypted with key *k*. Casper assumes that the run is initiated by *A* receiving some message from a user, or the environment, including *B*'s identity. We represent this by an extra message 0 in the protocol description. *ka (+) kb* representing the Vernam encryption of *ka* and *kb*. We simplified the sequence of messages in Fig. 1 by eliminating needless messages. Thus, the complete protocol description takes the following form:

#Protocol description
0. -> a : b
[a!=b]
1. a -> b : {b, ka}{PK(b)}, na, {{b, ka}{PK(b)}, na}{SK(a)}
[a!=b]
2. b -> a : a, ka (+) kb, {a, ka (+) kb}{SK(b)}, {na}{kb}
[a!=b]

Two kinds of specifications are currently supported: secrets and authentication. Secret specifies that certain data items should be secret. The first secret specification above may be paraphrased as: *a* thinks that *ka* is a secret that can be known to only himself and *b*. The lines starting Agreement are authentication specifications. The first *Secret* function specifies that *a* is correctly authenticated to *b*, and the two agents agree on the data values *na*.

#Specification
Secret(a, ka, [b])
Secret(b, kb, [a])
Agreement(a,b,[na])

The types of variables to be used in the actual system to be checked are defined in a similar way to the types of free variables.

#Actual variables
A, B, M : Agent
rA, rB, rM : SessionKey
Na, Nm: Nonce
InverseKeys = (rA,rA), (rB,rB), (rM,rM)

Any functions used by the agents in our protocol description have to be defined under the #Function heading. Public key functions and secret key functions of agents are shown as follows:

#Functions
symbolic PK, SK

The system definition specifies which agents in the system need to be checked. We consider a system with a single client *A* and a single server *B*. They use *rA, rB, Na*.

#System
CLIENT(A,B, rA, Na)
SERVER(B, A, rB)

Finally, the operation of the intruder is specified by giving his identity, and the set of data values that he initially knows. The following description defines the intruder's identity to be *M*, and the intruder initially knows all the agents' identities, his session key *rM*, his nonce *Nm*, public key functions, and his secret key *SK(M)*.

#Intruder Information
Intruder = M
IntruderKnowledge = {A, B, M, PK, rM, Nm, SK(M)}

We have generated 513 lines CSP description of the proposed protocol from above complete script using Casper. Then we have tested the security of our protocol using the CSP description and FDR. FDR found that there is no attack upon the proposed protocol. Therefore, the proposed protocol is secure against active attack.

6. COMPARISON WITH EXISTING PROTOCOLS

Important properties of the protocol to establish keys are discussed in this section. Also, characteristics of the proposed protocol are analyzed and compared with existing key distribution protocols in terms of certain factors. In key establishment protocols, it is important to satisfy following characteristics:

- nature of the authentication: entity authentication and key authentication should be provided.
- reciprocity of authentication: each of the entity authentication and key authentication may be unilateral or mutual
- key freshness: a session key is fresh (from the viewpoint of one party) if it can be guaranteed to be new, as opposed to possibly an old session key being reused through actions of an adversary.
- efficiency: considerations include the number of message exchanges (passes) required between parties, bandwidth required by the message (total number of bits transmitted), processing complexities for each party.
- third party requirements.

The proposed protocol provides mutual entity authentication. A drawback of using a timestamp in place of a nonce is the need for loosely synchronized clocks [5]. So we use a nonce to assure the freshness of the received message. The protocol needs only two message exchanges for complete key establishment and load balancing (i.e. message sent, message received, and exponential operations) between the entities. It has not a key management problem since it eliminates KDC and uses both symmetric-key and public-key schemes. Also, it increases efficiency in authentication since it uses an ECC-based signature whose key size is considerably smaller than other signature schemes. The protocol is compared with existing protocols in Table 1. The items compared are encryption scheme, server type, use of timestamp, authentication and use of digital signature and key management problem.

Table 1. Characteristics of each protocol.

Properties / Protocol	Encryption scheme	Server type	Use of Timestamp	Authentication	Use of Digital signature	Key Management
Shamir's no-key protocol[19]	Symmetric key	None	No	No	No	No
Kerberos[3]	Symmetric key	KDC	Yes	Yes	No	Yes
Needham-Schroeder shared key[1]	Symmetric key	KDC	No	Yes	No	Yes
Otway-Rees [2]	Symmetric key	KDC	No	Yes	No	Yes
Needham-Schroeder PK[19]	Public key	None	No	Yes	No	Yes
TMN[9]	Hybrid	KDC	Yes	Optional	No	No
Beller-Yacobi [8]	Hybrid	None	No	Yes	Yes (RSA or Elgamal)	No
The Proposed Protocol	Hybrid	None	No (Nonce)	Yes	Yes (ECDSA)	No

Compared to the TMN protocol, the protocol needs only two message exchanges for complete key establishment and authentication while the TMN protocol needs four messages. For authentication TMN protocol KDC should generate each user i's secret s_i using a polyrandom function which only the KDC knows, then distribute s_i to user i by using off-line methods and should manage all users' information. Such processes keep

the KDC busy and make it inefficient. However, the proposed protocol provides on-line authentication with only two message exchanges using an ECDSA signature, which is more efficient than existing signatures.

Table 2 shows the number of messages for key establishment in each protocol. The number of messages is one important factor which affects efficiency of the protocol. The number of messages sent or received also affects each entity's processing complexity. The proposed protocol is a two-pass key distribution protocol, but it provides all the properties supported by other protocols with only two message exchanges. Also, the load balance of messages for each entity has been considered, since only one message is sent or received.

Table 2. The number of messages in each protocol.

Number of Messages / Protocol	Number of total messages	Number of messages sent			Number of messages received		
		A	B	KDC	A	B	KDC
Shamir's no-key protocol	3	2	1	None	1	2	None
Kerberos	4	2	1	1	2	1	1
Needham-Schroeder shared key	5	3	1	1	2	2	1
Otway-Rees	4	1	2	1	1	2	1
Needham-Schroeder PK	3	2	1	None	1	2	None
TMN	4	1	1	2	1	1	2
Beller-Yacobi	4	2	2	None	2	2	None
The Proposed Protocol	2	1	1	None	1	1	None

Our protocol may have two drawbacks. One is that the key generation procedure of the protocol may be unfair to *A* since the session key is decided by only *B*. But, our protocol is designed for a client/server model. So, the key is decided by server *B* like the TMN protocol. Although our protocol seems to be unfair to *A*, it can eliminate the KDC, as server *B* generates a session key. The other drawback is that computational cost is high, since it needs to execute eight exponential computations (i.e. four RSA-operations, four ECC-operations). However, the ECC-based authentication scheme of our protocol is superior to other signature schemes in key size and efficiency. Also, our protocol guarantees a balanced load, since the number of exponential operations computed by each entity is the same. Table 3 shows the computational cost of each protocol that uses a public key scheme.

Table 3. Computational cost.

Protocol \ Number	The Number of Exponential Computations			
	Total	A	B	KDC
Needham-Schroeder PK	5	2	3	None
TMN	4	1	1	2
Beller-Yacobi	6	3	3	None
The Proposed Protocol	8	4	4	None

7. CONCLUSIONS

In this paper we proposed a two-pass hybrid key distribution and authentication protocol. When designing an authentication protocol certain tradeoffs must be made. It is important to understand the issues involved in order to choose the best solution [5]. Issues of our protocol are security, the number of messages required, the key management problem, the load balance of each entity's computation, entity and key authentication, and key freshness.

The proposed protocol needs only two messages for complete key establishment. It minimizes the number of message exchanges and the key management problem as it eliminates KDC, and uses both symmetric-key and asymmetric-key schemes. Using a hybrid mechanism, it guarantees entity and key authentication with two-pass, and load balance of each entity's computation. The protocol authenticates using an ECDSA signature whose efficiency is superior to other digital signatures. And, it guarantees key freshness, as it uses nonce to prevent the session key from replaying. The proposed protocol has been specified as a CSP description using Casper. Its security has been analyzed using FDR. FDR has found there were no insecure traces in our protocol, proving that the proposed protocol is secure. Since it provides security and reliability the proposed protocol can be efficiently applied to various communication systems in distributed computing environments.

REFERENCES

1. R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, Vol. 21, 1978, pp. 993-999.
2. D. Otway and O. Rees, "Efficient and timely mutual authentication," *Operating Systems Review*, Vol. 21, 1987, pp. 8-10.
3. J. T. Kohl, B. C. Neuman, and T. Y. T'so, "The evolution of the kerberos authentication system in distributed open systems," *IEEE Computer Society Press*, 1994, pp. 78-94.
4. S. M. Bellovin and M. Merritt, "Limitations of the kerberos authentication system," *ACM SIGCOMM Computer Communication Review*, 1990, pp. 119-132.
5. B. C. Neumana and S. G. Stubblebine, "A note on the use of timestamps as nonces," *Operating Systems Review*, Vol. 27, 1993, pp. 10-14.

6. R. Molva, G. Tsudik, E. van Herreweghen, and S. Zatti, "KryptoKnight authentication and key distribution system," in *Proceedings of European Symposium on Research in Computer Security*, 1992, pp. 154-174.
7. D. B. Johnson and A. J. Menezes, "Elliptic curve DSA (ECDSA): an enhanced DSA," Certicom Corp., URL: www.certicom.com.
8. M. J. Beller and Y. Yacobi, "Fully-fledged two-way public key authentication and key agreement for low-cost terminals," *Electronics Letters*, Vol. 29, 1993, pp. 999-1001.
9. M. Tatebayahi, N. Matsuzaki, and D. B. Newman Jr., "Key distribution protocol for digital mobile communication systems," *Advances in Cryptology: Proceedings of Crypto '89*, LNCS 435, Springer-Verlag, 1990, pp. 324-333.
10. G. Lowe and B. Roscoe, "Using CSP to detect errors in the TMN protocol," *IEEE Transactions on Software Engineering*, Vol. 23, 1997, pp. 659-669.
11. V. Miller, "Uses of elliptic curves in cryptography," *Advances in Cryptology CRYPTO'85,* LNCS 218, Springer-Verlag, 1986, pp. 417-426.
12. R. Schroeppel, H. Orman, and S. O'Malley, "Fast key exchange with elliptic curve systems," *Advnaces in Cryptology: Proceedings of Crypto '95*, Springer-Verlag, 1995, pp. 43-56.
13. J. A. Solinas, "An improved algorithm for arithmetic in a family of elliptic curves," *Advnaces in Cryptology: Proceedings of Crypto '97*, Springer-Verlag, 1997, pp. 357-371.
14. W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, Vol. IT-22, 1976, pp. 644-654.
15. R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communicaions of ACM*, Vol. 21, 1978, pp. 120-126.
16. D. Naccache, D. M'Raihi, D. Raphaeli, and S. Vaudenay, "Can D.S.A. be improved? complexity trade-offs with the digital signature standard," *Advances in Cryptology-EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 77-85.
17. A. W. Roscoe and M. H. Goldsmith, "The perfect "spy" for model checking cryptoprotocols," in *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.
18. G. Lowe, "Casper: A compiler for the analysis of security protocols," 1996. Available via URL http://www.mcs.le.ac.uk/~glowe/Security/Casper/index.html.
19. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
20. B. Schneier, *Applied Cryptography*, Second Edition, John Wiley & Sons, Inc., 1996.
21. C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.
22. R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, Revised Edition, Cambridge University Press, 1994.
23. Formal Systems Ltd., *Failures-Divergence Refinement-FDR 2 User Manual*, 1997. Available via URL http://www.formal.demon.co.uk/FDR2.html

Sung-Min Lee received B.S. in Computer Engineering from the Hallym University, 1997. He received M.S. in Computer Science and Engineering from the Korea University,

1999. He received Ph.D. in Computer Science and Engineering from the Korea University, 2001. At present, senior software engineer at TongYang Systems, Corp. Research Interests: cryptography, electronic commerce, network security, digital copyright protection, distributed object system, J2EE security.

Tai-Yun Kim received B.S. in Industrial Engineering Science from Korea University, 1981. He received M.S. in Computer Science from the Wayne State University, 1983. He received Ph.D. in Computer Science from the Auburn University, 1987. At present, professor at the Department of Computer Science and Engineering in the Korea University. Research Interests: computer networks, EDI systems, security, satellite communication and computer graphics.