THE WEB-BASED APPLICATION OF KEY EXCHANGE PROTOCOLS FOR DIGITAL AND MOBILE SIGNATURES

A Thesis Submitted to
The Graduate School of Engineering and Sciences of
Izmir Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

In Computer Software

By

Evren AKALP

September 2008

İZMİR

We approve the thesis of **Evren AKALP**

.....

Assoc. Prof. Dr. Ahmet KOLTUKSUZ
Supervisor

.....

Asst. Prof. Dr. Gökhan DALKILIÇ
Committee Member

.....

Asst. Prof. Dr. Tuğkan TUĞLULAR
Committee Member

XX September 2008

.....	……..
Prof. Dr. Sıtkı AYTAÇ	**Prof. Dr. Hasan BÖKE**
Head of the Computer Engineering Department	Dean of the Graduate School of Engineering and Science

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor, Assoc. Prof. Dr. Ahmet Koltuksuz, for his patience, guidance and encouragement. He was the one who shows me the right way when I have problems. Thanks for his support and effort thesis work.

Furthermore, it was a pleasure for me to work Dr. Serap Atay who helped me in many parts of the thesis.

I would also like to thank Sevgi Uslu who cooperated with me in many studies.

Finally, I want to thank to my parents who always supported me throughout my education as well as in my graduate study.

ABSTRACT

THE WEB-BASED APPLICATION OF
KEY EXCHANGE PROTOCOLS FOR DIGITAL AND MOBILE
SIGNATURES

Many people want to communicate and send their data securely, for this reason they use encryption methods. Symmetric cryptosystems are commonly used for securing data. They are fast and reliable. But there is a problem of exchanging keys in symmetric cryptosystems. Sender and receiver must share keys to get the plain text. In this thesis, different scenarios for key exchange protocols are developed. Solutions for securing communication to block man in the middle attack are also defined in this thesis.

ÖZET

SAYISAL VE MOBİL İMZA İÇİN ANAHTAR DEĞİŞİMİ PROTOKOLLERİNİN WEB TABANLI UYGULAMASI

Birçok kişi iletişimde ve veri transferinde güvende olmayı ister, bu yüzden şifreleme metotlarına başvururlar. Simetrik kriptosistemler veri güvenliğinde sık kullanılan bir yöntemdir. Hızlı ve güvenilirdirler. Fakat simetrik kriptosistemlerde anahtar değişimi problemi baş gösterir. Gönderen ve alıcıda aynı anahtarın bulunması zorunludur. Bu tez ile anahtar değişim protokolleri için değişik senaryolar üretilmiştir. Ayrıca transfer sırasında gelebilecek saldırılara karşı nasıl önlemler alındığı da açıklanmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

In this chapter some background information is provided. The key exchange protocols which utilized in all of the scenarios of this thesis are mentioned in this chapter. All these protocols have been implemented in Crympix cryptographic library by earlier researchers (Hışıl 2005 , Uslu 2007). The software created for this thesis connects to Crympix in order to to generate the necessary parameters for each scenario. The digital signature algorithms are implemented and used for authorization of users

1.1. Key Exchange

People want to secure their communications and data transfers. For this reason, they want to encrypt their data through a cryptographic system. One of these cryptographic systems is called as Symmetrical Cryptosystem. Symmetrical cryptosystems utilize same key for the purposes of both encryption and decryption process and in doing so it is required that the both sides of the communicating parties must have the same key obtained prior to the communication event take place.

Having same the key obtained thus explained above for both sides is a problem since the key must either be shared in advance and/or be distributed securely thru the secure channels which might be expensive to get and even more expensive to maintain. Therefore, the communicating parties need a protocol for exchanging the keys. These protocols are commonly known as the Key Exchange Protocols.

1.2. Key Exchange Protocols

As aforementioned, the whole idea about the key exchange protocol is get the key distributed to the only communicating parties of the communication. Now, this key is usually called the session key which means that key is active during the communication and another key will be generated for a new communication since the session key will be destroyed upon the termination of the communication. Key exchange protocols utilize mathematical approaches in order to obtain the session keys for both sides without transferring it. So that

nobody can get the session key by listening into the data transfer channel in between two sides other than the legitimate parties.

Even if we are using a symmetrical cryptosystem we still have to utilize the public and private key pairs in key exchange protocols for they are not used for cryptographical purposes but used for calculating the session key.

Key pairs are generated from specific domain parameters in both sides. These domain parameters and public keys are not secret. Only the private keys of both sides are secret. However, regenerating private key from domain parameters and public keys must be so difficult due to the utilization of mathematical protocols.

Three key exchange protocols explained briefly in this chapter, both protocols have domain parameters for generating public-private key pairs. Although all protocols look the same in the structure, their mathematical background is very different.

1.2.1. Diffie-Hellman Key Exchange

Difficulty of calculating discrete logarithms in finite fields is the main starting point of Diffie-Hellman Key Exchange Algorithm.

Discrete logarithm can be explained in formal definition as:

"Let G is a finite cyclic group of order n. Let α be a generator of G, and let $\beta \in$ G. The discrete logarithm of β to the base α, which is denoted as $\log_\alpha \beta$, is the unique integer x, $0 \leq x \leq n-1$, such that $\beta = \alpha^x$ (Menezes et al. 1996)."

Also discrete logarithm problem can be explained as:

"Given a prime p, a generator α of $\mathbf{Z}_p^*$ and an element $\beta \in \mathbf{Z}_p^*$, find the integer x, $0 \leq x \leq p-2$, such that $\alpha^x \equiv \beta \pmod p$ (Menezes et al. 1996)."

Discrete logarithm function is not soluble in polynomial time. This means that this function is an NP complete problem. Functions which are NP complete are suitable for use in cryptographic protocols.

Diffie Hellman Key Exchange Algorithm has three domain parameters (ANSI X9.42 1998):

- **p:** A prime defining the Galois Field GF(p), which is used as a modulus in the operations of GF(p), where $2^{(L-1)} < p < 2^{L}$, for $L \geq 1024$, and L is a multiple of 256.
- **q:** A prime factor of p-1 such that p = jq+1 and $q > 2^{m-1}$. GF(p)* has a cyclic subgroup of order q.
- **g:** A generator of the q-order cyclic subgroup of GF(p)*, that is, an element of order q in the multiplicative group of GF(p).

The key pair includes:

- x_A: *A* private key which is selected as $1 \leq x \leq (q-1)$.
- y_A: *A* public key which is calculated as $y_A = g^{x_A} \pmod{p}$.

The shared key is calculated as:

$$K = (y_B)^{x_A} \pmod{p} = (g^{x_B})^{x_A} \pmod{p} = g^{x_A \cdot x_B} \pmod{p} \quad (1.1)$$

$$K = (y_A)^{x_B} \pmod{p} = (g^{x_A})^{x_B} \pmod{p} = g^{x_A \cdot x_B} \pmod{p} \quad (1.2)$$

1.2.2. Elliptic Curve Diffie-Hellman Key Exchange

Elliptic curve cryptographical algorithms and protocols are used in Elliptic Curve Diffie-Hellman Key Exchange Algorithm.

The characteristics of elliptic curves, such as being defined on a finite cyclic group, and having operations as addition and doubling for the points on the elliptic curve makes these curves suitable for cryptographic protocols. Because when a scalar multiplication, which is actually a point addition, of an integer number (n) and a point (P) on curve is performed, even if the start and stop points and all other parameters about curve are known, it is hard to find the integer n (Uslu 2007).

The domain parameters used in Elliptic Curve Diffie Hellman Key Exchange are listed below (ANSI X9.63 1999):

- *p:* *A* prime defining order of the finite field F_q, where q = p and p > 3
- *a, b:* Two field elements in F_p which define the equation of the elliptic curve

$$E : y^2 = x^3 + ax + b (\text{mod } p)$$

- x_G, y_G**:** Two field elements in F_p which define a point $G(x_G, y_G)$ of prime order on E (note that E $\neq \infty$)
- ***n:*** The order of the point G
- ***h:*** The cofactor defined as $h = \#E(Fp) / n$

The key pair includes:

- x_A*:* A private key which is selected in the interval [1, n-1].
- Q_A*:* A public key which is a point $Q(x_Q, y_Q)$ on elliptic curve and calculated using point multiplication $Q = x_A.G$

The shared key is calculated as:

$$K = x_A.Q_B = x_A.x_B.G \text{ by User A} \quad (1.3)$$

$$K = x_B.Q_A = x_A.x_B.G \text{ by User B} \quad (1.4)$$

1.2.3. Elliptic Curve MQV Key Exchange

Actually, the man in the middle attack is the main problem for key exchange algorithms and, MQV has been developed against these attacks (Law, et al. 1998).

The fundamental distinction between MQV and other key exchange protocols is that MQV uses two different key pairs. One of these key pairs is called static or long-term key pair which is bound to the entity for a certain period of time, typically through the use of certificates. The other key pair is called ephemeral or short-term key pair which is generated for each run of the protocol (Law, et al. 1998).

The domain parameters are (ANSI X9.63 1999):

- *p: A* prime defining the order of the finite field F_p, where p > 3
- *a, b:* Two field elements in F_p which define the equation constants of the elliptic curve

$$E : y^2 = x^3 + ax + b (\text{mod } p)$$

- x_G, y_G**:** Two field elements in F_p which define a point $G(x_G, y_G)$ of prime order on E
- *n:* The order of the points over E in F_p ; $\#E(F_p) = n$
- *h:* The cofactor defined as $h = \#E(Fp)/n$

These parameters are specified by assuming that the elliptic curve E is defined on a finite field that has odd characteristic (F_p).

- The static (or long-term) key pair includes:
- w_A : A static private key.
- W_A : A static public key.

The ephemeral (or short-term) key pair includes:

- r_A : An ephemeral private key which is selected as a statistically unique and unpredictable (random) integer in the interval [1, n-1].
- R_A : An ephemeral public key which is a point $R(x_R, y_R)$ on elliptic curve and calculated using point multiplication $R_A = r_A.G$

The implicit signature is symbolized as s_A (or s_B) and calculated as (Uslu 2007):

$$s_A = (r_A + \overline{R}_A.w_A) \bmod n \quad \text{by User A}$$

$$s_B = (r_B + \overline{R}_B.w_B) \bmod n \quad \text{by User B} \tag{1.5}$$

Then the shared key is calculated as (Uslu 2007):

$$\begin{aligned}
K &= h.s_A.(R_B + \overline{R}_B.W_B) \\
&= h.((r_A + \overline{R}_A.w_A) \bmod n).(R_B + \overline{R}_B.W_B) \\
&= h.(((r_A + \overline{R}_A.w_A) \bmod n).R_B + ((r_A + \overline{R}_A.w_A) \bmod n).\overline{R}_B.W_B) \\
&= h.(r_A.R_B + \overline{R}_A.w_A.R_B + r_A.\overline{R}_B.W_B + \overline{R}_A.w_A.\overline{R}_B.W_B) \\
&= h.(r_A.(r_B.G) + \overline{R}_A.w_A.(r_B.G) + r_A.\overline{R}_B.(w_B.G) + \overline{R}_A.w_A.\overline{R}_B.(w_B.G)) \\
&= h.G.(r_A.r_B + r_B.w_A.\overline{R}_A + r_A.w_B.\overline{R}_B + w_A.w_B.\overline{R}_A.\overline{R}_B) \qquad \text{by User A}
\end{aligned} \tag{1.6}$$

$$
\begin{aligned}
K &= h.s_B.(R_A + \overline{R}_A.W_A) \\
&= h.((r_B + \overline{R}_B.w_B) \bmod n).(R_A + \overline{R}_A.W_A) \\
&= h.(((r_B + \overline{R}_B.w_B) \bmod n).R_A + ((r_B + \overline{R}_B.w_B) \bmod n).\overline{R}_A.W_A) \\
&= h.(r_B.R_A + \overline{R}_B.w_B.R_A + r_B.\overline{R}_A.W_A + \overline{R}_B.w_B.\overline{R}_A.W_A) \\
&= h.(r_B.(r_A.G) + \overline{R}_B.w_B.(r_A.G) + r_B.\overline{R}_A.(w_A.G) + \overline{R}_B.w_B.\overline{R}_A.(w_A.G)) \\
&= h.G.(r_A.r_B + r_A.w_B.\overline{R}_B + r_B.w_A.\overline{R}_A + w_A.w_B.\overline{R}_A.\overline{R}_B) \qquad \text{by User B} \qquad (1.7)
\end{aligned}
$$

1.3. Digital and Mobile Signatures

For an agreement upon a document, a handwritten signature has been utilized as proof of authorship for a long time. Since a handwritten signature is authentic, unforgeable, not reusable, unalterable, and supposed to be nonrepudiated it is therefore quite convenient theoretically. However, many methods are generated to break these supposedly unique characteristics. For instance, in today's computer world it would be very easy to cut and paste a valid graphical signature from one document just to copy to another one.

Trents are used for verifying signatures of users. Every one must communicate with a Trent to verify the signature of the other one they are talking to. Every user has two keys one shared and one secret. Shared keys are stored in a Trent called Certification Authority. If one user wants to send data to another, he/she then signs data with secret key and only his/her shared key can open that signatures so that the receiver gets sender's shared key from certification authority and verifies the signatures (Schneier 1996).

1.4. Crympix

CRYMPIX is a multiprecision cryptographic library which was designed and developed by Hüseyin Hışıl as a master degree thesis "A Distributed Multiprecision Cryptographic Library Design" (Hışıl 2005).

As a multiprecision library CRYMPIX uses large numbers for cryptographic applications by splitting them to small pieces called word. Words have a length of n for an n-bit processor.

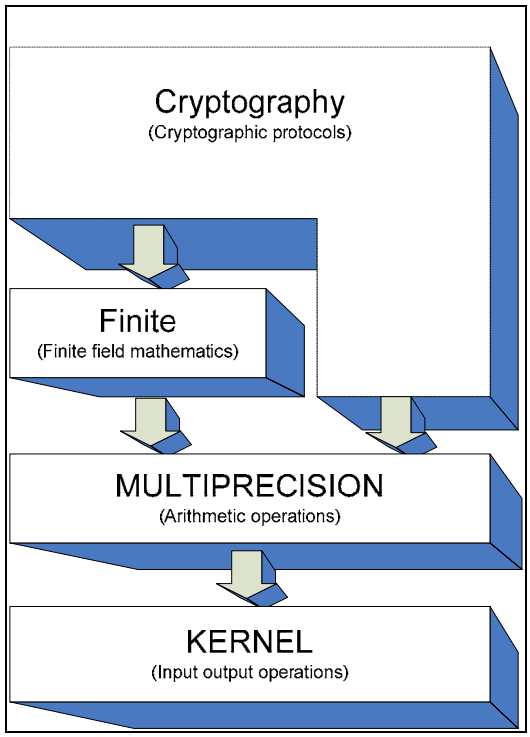

Figure 1.1. Architecture of CRYMPIX

Portability is very important for a cryptographic library so CRYMPIX is developed in ANSI C platform which is suitable for different platforms. Also, word lengths can be changed for different processor types.

CRYMPIX has many structures for multiprecision integers, finite fields, prime numbers and elliptic curves. So, many mathematical forms can be implemented in CRYMPIX easily.

CRYMPIX divided into several layers shown in Figure 1.1. Basic arithmetic operations, such as addition, subtraction, multiplication, division, exponentiation, modulus and modular exponentiation, are implemented in the *multiprecision layer*. Input-output operations and some base functions are placed in the *kernel layer*. *Finite layer* implements the finite field mathematical forms and operations. The cryptographic protocols such as encryption-decryption algorithms are placed in *cryptography layer* (Uslu 2007)

CHAPTER 2

SCENARIOS

Two scenarios are explained in this chapter. The first one is for communication of two users directly without any other computerized media between. In addition, the second one is for a communication through a server.

2.1. Key Exchange Without Server

In this scenario two people are communicating with each other directly, so that they must generate parameters and they must calculate a session key by themselves. For this reason, the hardware is important for the run-time performance of the application.

First of all the person who wants to communicate with the other has to determine which Key Exchange Protocol to use. Also he/she must determine which user will generate domain parameters. All communication for getting a session key is shown in Figure 2.1, and the communication steps are as follows

Step 1: User A prepare communication request.

Step 2: User A sends communication request message to User B.

Step 3: User B accepts communication request.

Step 4: User A generates domain parameters and Public, Private Key pairs.

Step 5: User A sends required domain parameters and Public Key to User B.

Step 6: User B takes domain parameters and generates own public, private key pair.

Step 7: User B calculates session key.

Step 8: User B sends own public key to User A.

Step 9: User A calculates session key.

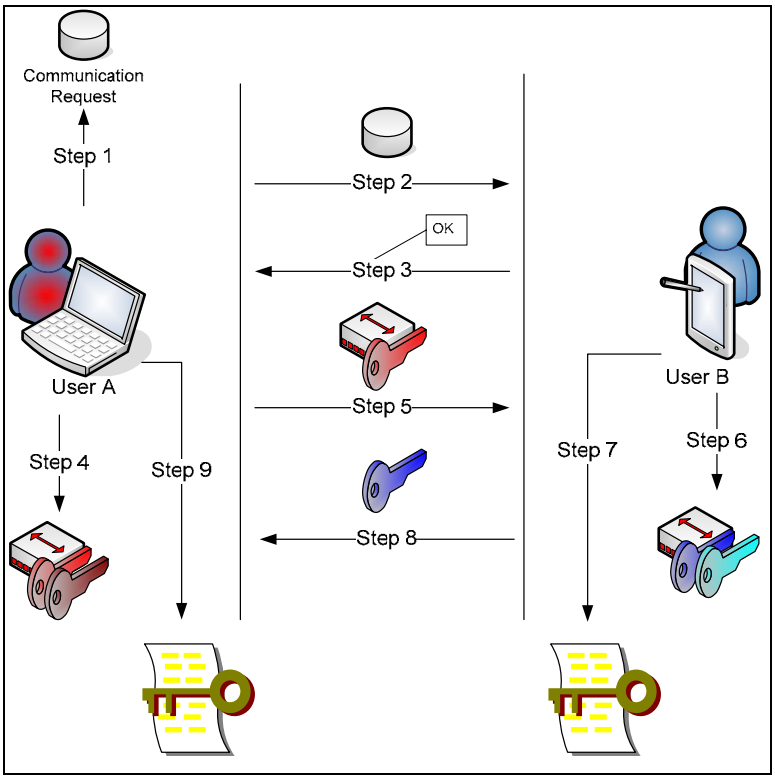

Figure 2.1. Key exchange without server

This scenario provides the users with an opportunity to generate domain parameters in their device or in remote user's device. This is useful when you have a device with limited configuration like pocket PCs and/or PDAs. For instance, a pocket pc wants to communicate with a laptop having a dual core processor; and since the laptop is more powerful than pocket pc, the pocket pc may select the option, as remote machine will generate domain parameters.

In Figure 2.2, one can see the options for generation of the parameters and choices of the protocols.

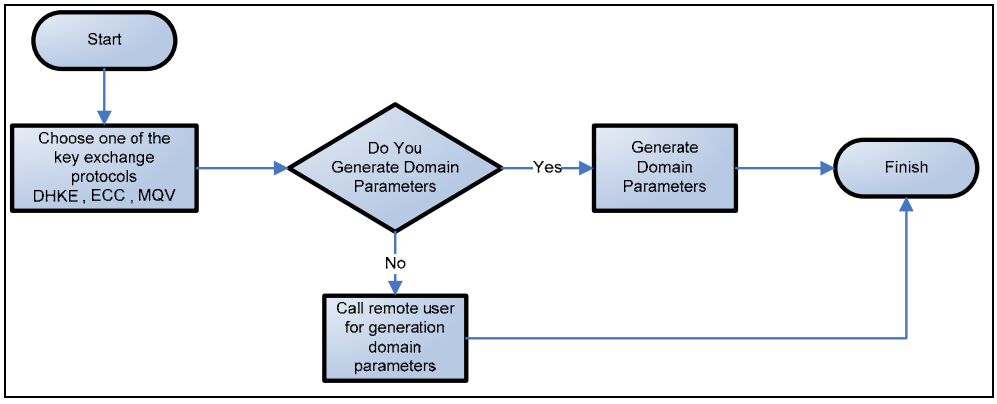

Figure 2.2. Choosing protocols and generating parameters

Although users have a chance for generating domain parameters in remote user, all users must generate private and public key pairs. So each device must at least generate key pairs. But this option gives a chance to decrease generation time for the devices which have limited hardware.

2.2. Key Exchange With Server

Generating domain parameters is a time consuming operation. Moreover, this time gets to be changed due to the hardware configurations of the client machines. In all key exchange protocols domain parameters are public so that they can be generated by a powerful server.

However, clients because of secrecy must do generating private and public keys. This means that in order to make private key secretly the clients must generate it themselves. Like a formerly mentioned scenario, one client has to determine which Key Exchange Protocol will be used in this scenario too and, the necessary domain parameters are taken from server as well.

All communication for getting a session key is shown in Figure 2.3; and the communication steps are as follows:

Step 1: User A prepare communication request.

Step 2: User A sends communication request message to User B.

Step 3: User B accepts communication request.

Step 4: User A requests necessary domain parameters from server

Step 5: User A generates Public, Private Key pairs.

Step 6: User A sends required domain parameters and Public Key to User B.

Step 7: User B takes domain parameters and generates own public, private key pair.

Step 8: User B calculates session key.

Step 9: User B sends own public key to User A.

Step 10: User A calculates session key.

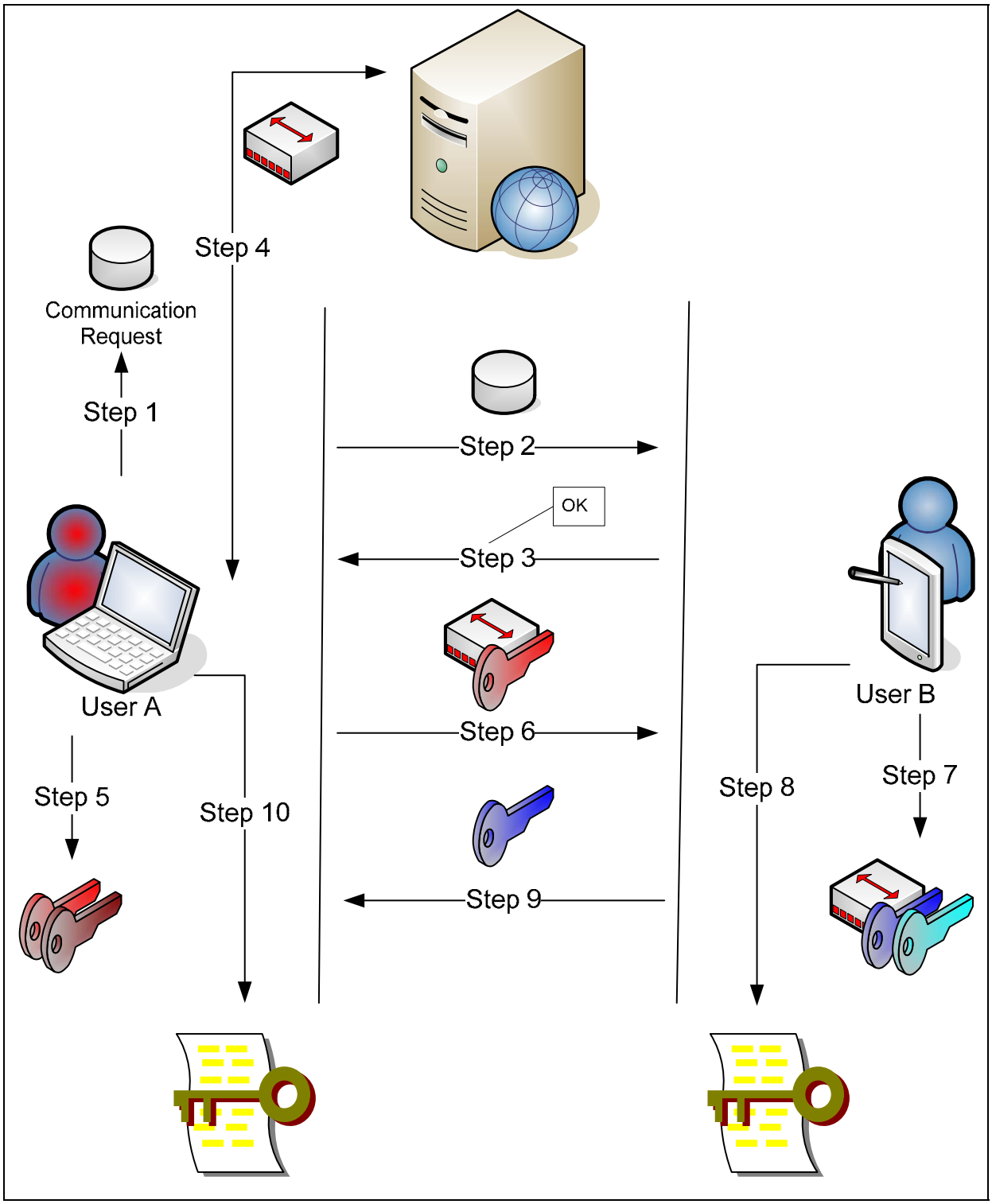

Figure 2.3. Key exchange with server

This scenario aims to decrease the load of the client devices. Mobile client devices have limited capacity so that every operation done by server is important for them. We have a chance to generate public and private keys and generation of session key in server. Therefore, the client device only transfers data to server. Nevertheless, in that scenario, every operation is done in server and if there are many clients, there may be a denial of service due to a very heavy load. In addition, another important point is that if somebody listens the transmission as a men_in_the_middle_attack, he/she can get domain parameters, private and public keys and session key.

Another disadvantage is that, all parameters are taken to the server and if somebody can reach server he/she can obtain all data about the session key.

Hence we only generate domain parameters, which are not secret in key exchange protocol, by server. Therefore, clients must generate public and private key pairs as well as session keys.

CHAPTER 3

AUTHENTICATION

Authentication is very important in key exchange scenarios, because one has to be sure that sender of the data is really the person who is to be communicated with. In these scenarios, the man in the middle attack is the most dangerous attack.

Figure 3.1 demonstrates the man in the middle attack for key exchange scenarios. In this attack scenario, a person who wants to obtain the session key is located in the middle of the transfer. This person (the Man in the Middle) is able to obtain all the messages from both User A and User B, and sends her own fabricated messages instead of the original messages. To do this, the attacker generates one session key for User A and another for User B, and intercepts all the messages from both users while being in the middle of the transmission. The communication steps in Figure 3.1 are outlined as follows.

Step 1: User A prepares the communication request.

Step 2: User A sends the communication request message to User B.

Step 3: Man in the Middle intercepts the message

Step 4: Man in the Middle sends new communication request to User B.

User B thinks that the request comes from User A.

Step 5: Man in the Middle accepts the communication request of User A.

User A thinks that the acceptance comes from user B.

Step 6: User B accepts the communication request, and Man in the Middle catches it.

Step 7: User A generates the domain parameters and the public-private key pairs.

Step 8: User A sends the required domain parameters and the public key to User B.

Step 9: Man in the Middle intercepts the domain parameters and generates the public-private key pair.

Step 10: Man in the Middle computes the session key for the communication with User A.

Step 11: Man in the Middle sends the generated public key to User A.

User A thinks that the key belongs to User B

Step 12: User A calculates the session key.

Step 13: Man in the Middle generates new domain parameters for User B.

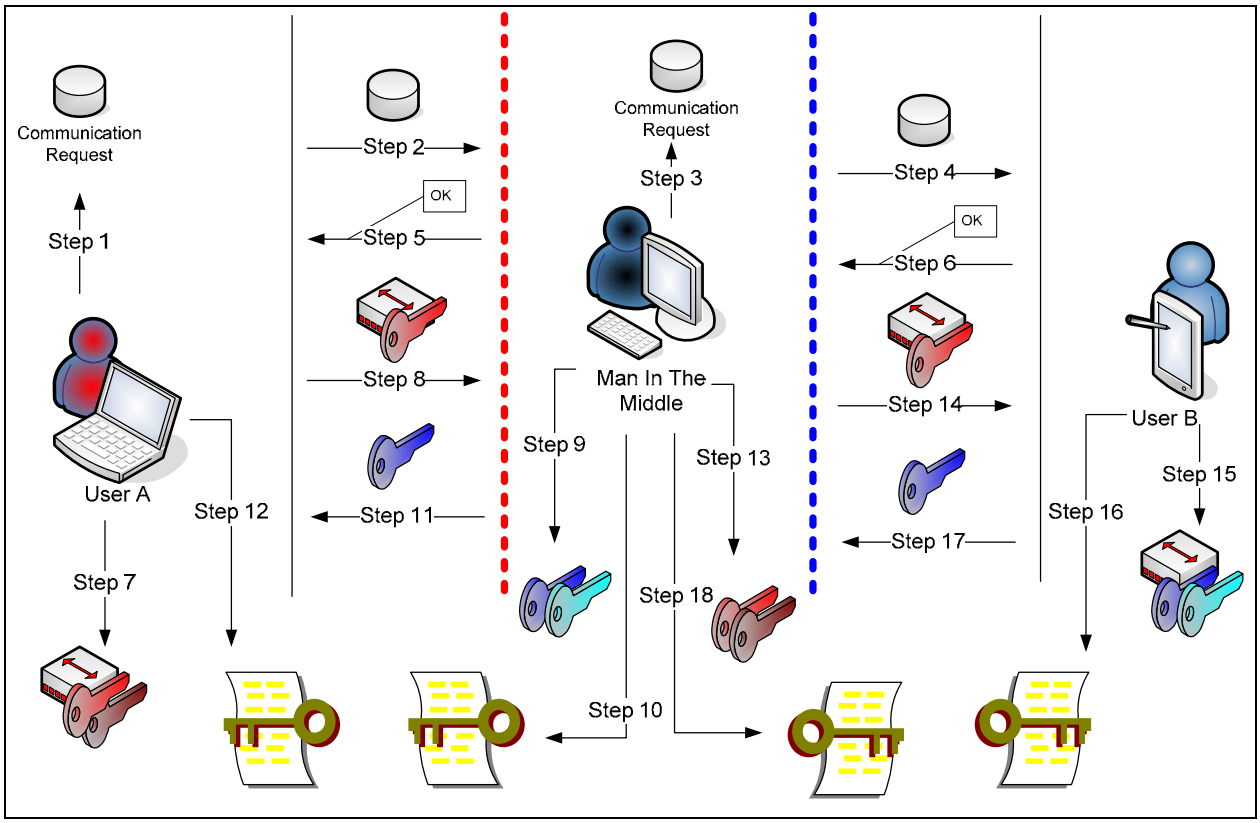

Figure 3.1. Man in the middle attack on key exchange

Step 14: Man in The Middle sends the domain parameters to User B.

User B thinks this that the data come from User A.

Step 15: User B receives the domain parameters and generates public-private key pair.

Step 16: User B computes the session key.

Step 17: User B sends the public key to User A.

Step 18: Man in the Middle intercepts the message and generates the session key for the communication with User B.

As mentioned above, the man in the middle is a very dangerous attack, because User A and User B have different session keys and they never really communicate with each other throughout the whole scheme. In addition, the man in the middle can recover all the messages encrypted with the session keys. Also, users can not decrypt each other's messages without the man in the middle due to the fact that their session keys are different.

In this attack, the man in the middle recovers each message from each user by decrypting with this user's session key. After the decryption, the man in the middle can either

modify the message or only reads it. Later, the message is sent to the other user after being encrypted with the proper session key.

It is very critical to avoid from or determine the occurrence of this type of attacks. For this reason, the certification method is used to prevent the man in the middle attacks.

While using the certification, each user has her own certificate and knows that the message is coming from the user who claims to be the sender, because only the owner of the certificate is able to sign the message.

There are two different ways of using certification. One of them is to use Internal Business Server and the other is the presence of External Certification Authority (CA).

3.1. With Internal Business Server

This method is mainly used in business solutions. If a company chooses this option, the company has to store the user public keys. This means that company has to make a secure service to respond to the certification confirmation requests, and provide the required physical, software and hardware based security. In other words, this company should achieve the standards of a Certification Authority.

The Key Exchange with Server communication scenario is subject to change when Authentication is existent. In Figure 3.2, the Key Exchange with Server scenario with Authentication on internal server is provided. The communication steps are as follows.

Step 1: User A prepares the communication request including its own certificate.

Step 2: User A sends the communication request message to User B.

Step 3: User B checks User A's certificate from the CA of the internal server.

Step 4: User B accepts the communication request including its own certificate.

Step 5: User A checks User B's certificate from the CA of the internal server.

Step 6: User A requests the necessary domain parameters from server including its own certificate.

Step 7: Server checks User A's certificate from internal CA.

Step 8: Server sends the necessary parameters to User A including its own certificate.

Step 9: User A checks Servers certificate from internal server CA.

Step 10: User A generates Public-Private Key pairs.

Step 11: User A sends the required domain parameters and the Public Key to User B including its own certificate.

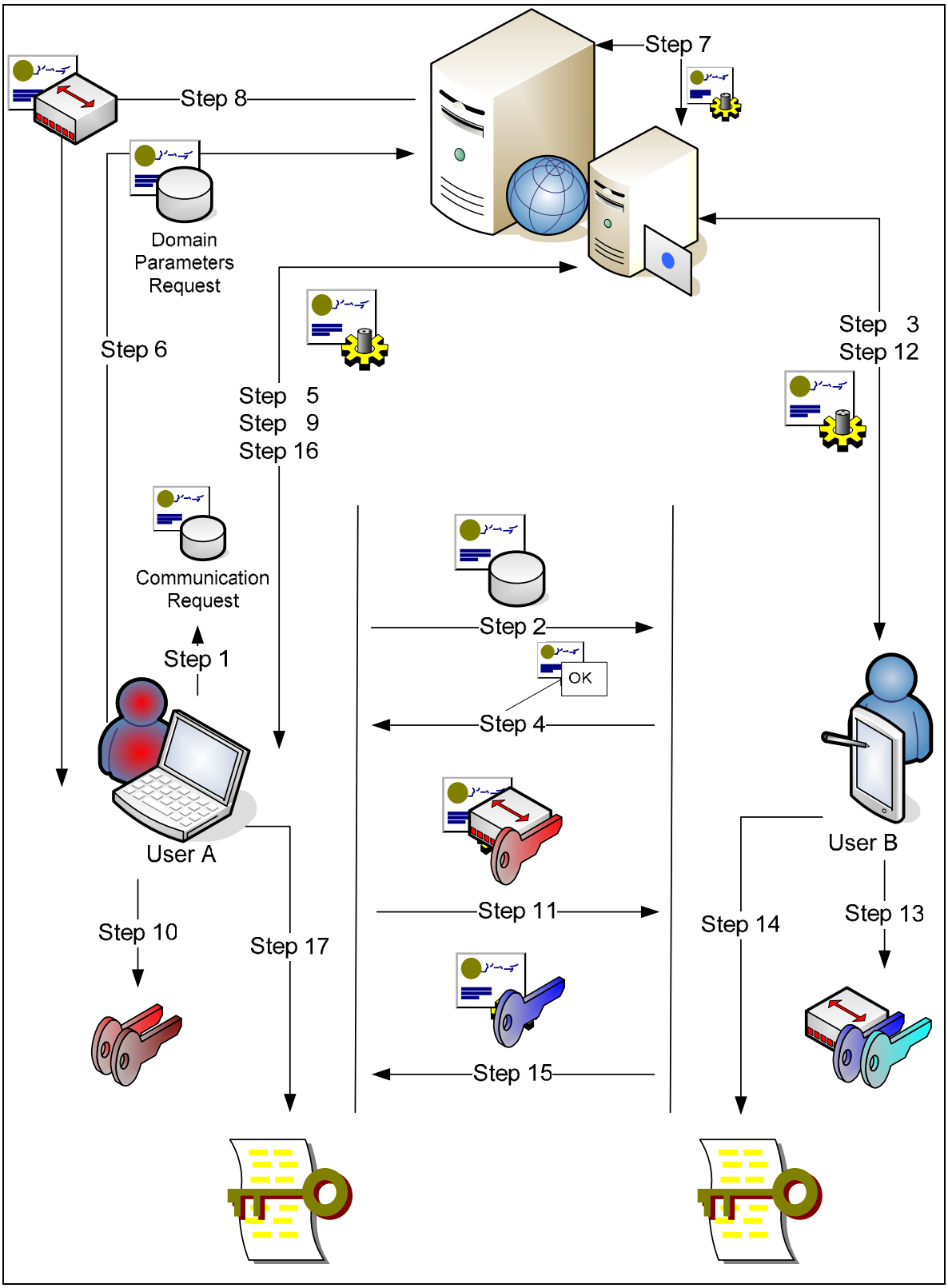

Figure 3.2. Authentication with internal server

Step 12: User B checks User A's certificate from the CA of the internal server.

Step 13: User B takes the domain parameters and generates own public-private key pair.

Step 14: User B computes the session key.

Step 15: User B sends its own public key to User A including its own certificate.

Step 16: User A checks User B's certificate from the CA of the internal server.

Step 17: User A computes the session key.

The above authenticated scenario provides the company with no need of any external CA. However, it is expensive and difficult to build and run a key management system inside the company. Therefore, many companies prefer external certification authorities.

Using the authentication mechanism, man in the middle attack is blocked because only the owner of the certificate is able to sign the message. The important points to consider are the security of the internal CA server and the user devices. If the attacker breaks in the server and modifies the certificates, the whole scheme fails. Furthermore, if the attacker manages to breach the client device and obtain the certificate or utilize the device without permission, the whole scheme fails again.

In order to decrease the transmission time, an option is to store the user certificates during the transmission, so that users do not have to validate the certificates with the internal CA every time.

When session keys are generated, all the stored certificates are removed. Using this mechanism, four steps can be omitted for this scenario and devices do not have to wait for the server response, so they can perform faster. In addition, server does not answer many requests because only two certification check requests arrives for any key exchange protocol, and this causes server load to decrease.

Moreover, at the time of installation, it is possible to store server certificate inside the user devices so that the users do not have to check server certificate for every communication.

In Figure 3.3, the Authenticated Key Exchange with Server scenario for only one time certificate control on internal server is outlined. The related communication steps are given below.

Step 1: User A prepares the communication request including its own certificate.

Step 2: User A sends the communication request message to User B.

Step 3: User B checks User A's certificate from the CA of the internal server and saves the certificate.

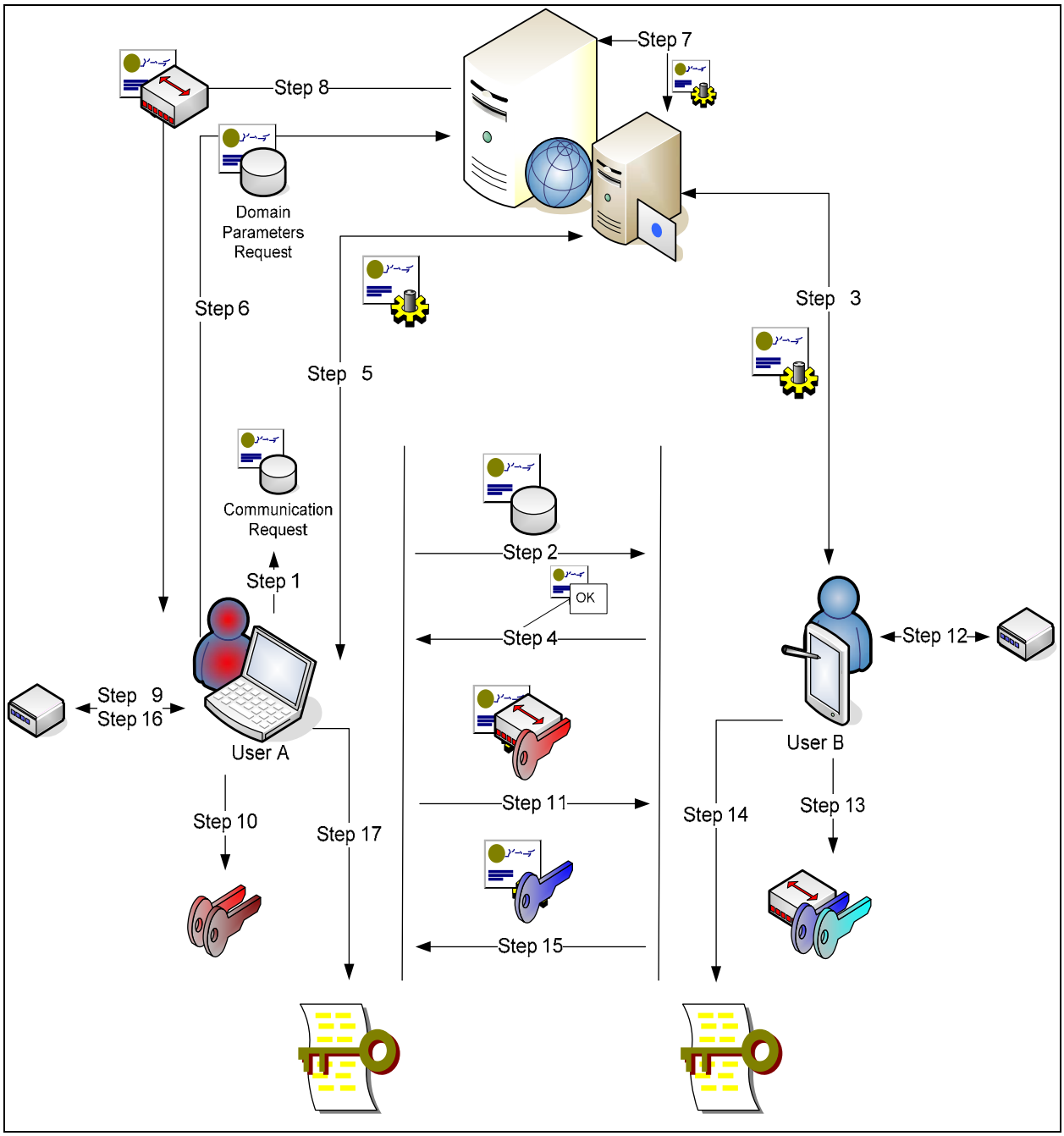

Figure 3.3. Authentication with internal server by stored certificates

Step 4: User B accepts the communication request including its own certificate.

Step 5: User A checks User B's certificate from the CA of the internal server and saves the certificate.

Step 6: User A requests the necessary domain parameters from server including its own certificate

Step 7: Server checks User A's certificate from internal CA.

Step 8: Server sends the necessary parameters to User A including its own certificate.

Step 9: User A checks the server certificate from the stored certificate.

Step 10: User A generates the Public-Private Key pairs.

Step 11: User A sends the required domain parameters and the Public Key to User B including its own certificate.

Step 12: User B checks User A's certificate from the stored certificate.

Step 13: User B takes domain parameters and generates its own public-private key pair.

Step 14: User B computes the session key.

Step 15: User B sends the public key to User A including its own certificate.

Step 16: User A checks User B's certificate from the stored certificate.

Step 17: User A computes the session key.

3.2. With External Certification Authority

3.2.1. Key Exchange Without Server Scenario

This scenario is designed primarily for two parties who want to communicate in secret or who have limited server support but strive to overcome the man in the middle attack. In this scheme everything is performed on the client side meaning that all generation, calculation and signing operations are executed in client devices. For this reason, the process might be slower if the client devices are not powerful enough. The related communication steps are described as follows and depicted in Figure 3.4.

Step 1: User A prepares the communication request.

Step 2: User A sends the communication request message to User B including its own certificate

Step 3: User B checks User A's certificate from the external CA.

Step 4: User B accepts the communication request including its own certificate.

Step 5: User A checks User B's certificate from the external CA.

Step 6: User A generates the domain parameters and Public-Private Key pairs.

Step 7: User A sends the required domain parameters and the Public Key to User B including its own certificate.

Step 8: User B checks User A's certificate from the external CA.

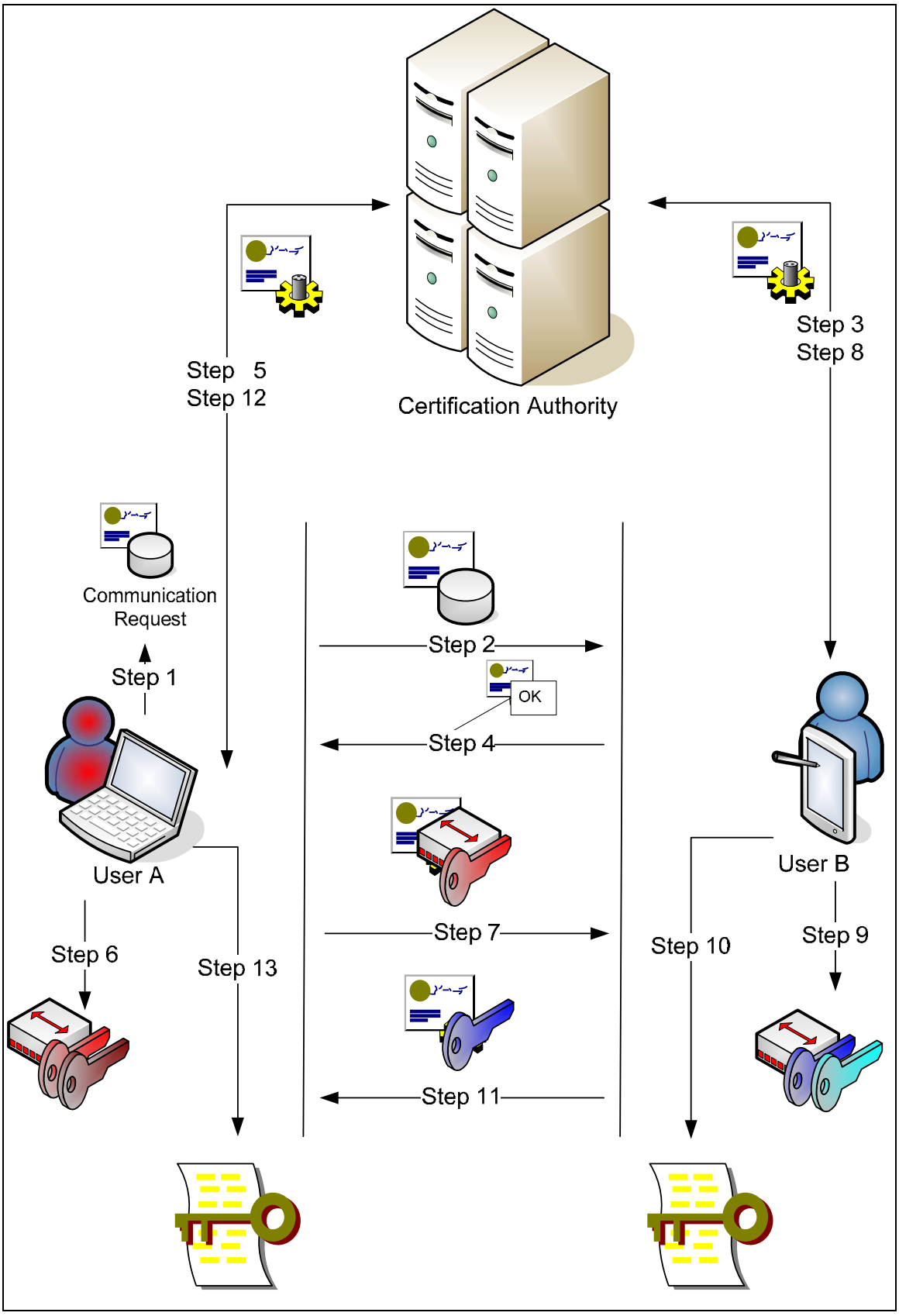

Figure 3.4. Authentication from CA without server

Step 9: User B takes the domain parameters and generates the public-private key pair.

Step 10: User B computes the session key.

Step 11: User B sends its public key to User A including its own certificate.

Step 12: User A checks User B's certificate from the external CA.

Step 13: User A computes session key.

Since certification is used throughout the whole transmission, the man in the middle attack is prevented. Users deny all packets with invalid certifications and close the communication. As a result, they can be sure that the packets come from the user with whom the communication is intended to be established.

Another important point is that the users check the certificates from external CA during the transmission and this is a time consuming process. If user devices are mobile and the users are charged for each data packet, the operation becomes quite expensive.

Like Authentication with Internal Business Server scenario, users can store certificate of the other user at the time of the first certificate check of the CA. Then, during every transmission the certificate is checked from the stored one. Therefore, for the whole communication the certificate is checked only once. It is important that the certificate is deleted from the stored address when the communication ends.

Users only check the certificate from the CA when stored certificate is different than the signed data which is sent by the remote user. This means that the remote user has changed her certificate during the communication. If the signed data is still different, there is either a man in the middle attack or some corruption occurs during the transmission of the message.

Figure 3.5 portrays the Authenticated Key Exchange without Server scenario for only one time certificate control on internal server, whose communication steps are given below.

Step 1: User A prepares the communication request.

Step 2: User A sends the communication request message to User B including its own certificate

Step 3: User B checks User A's certificate from the external CA and saves the certificate.

Step 4: User B accepts the communication request including its own certificate.

Step 5: User A checks User B's certificate from the external CA and saves the certificate.

Step 6: User A generates the domain parameters and the Public-Private Key pairs.

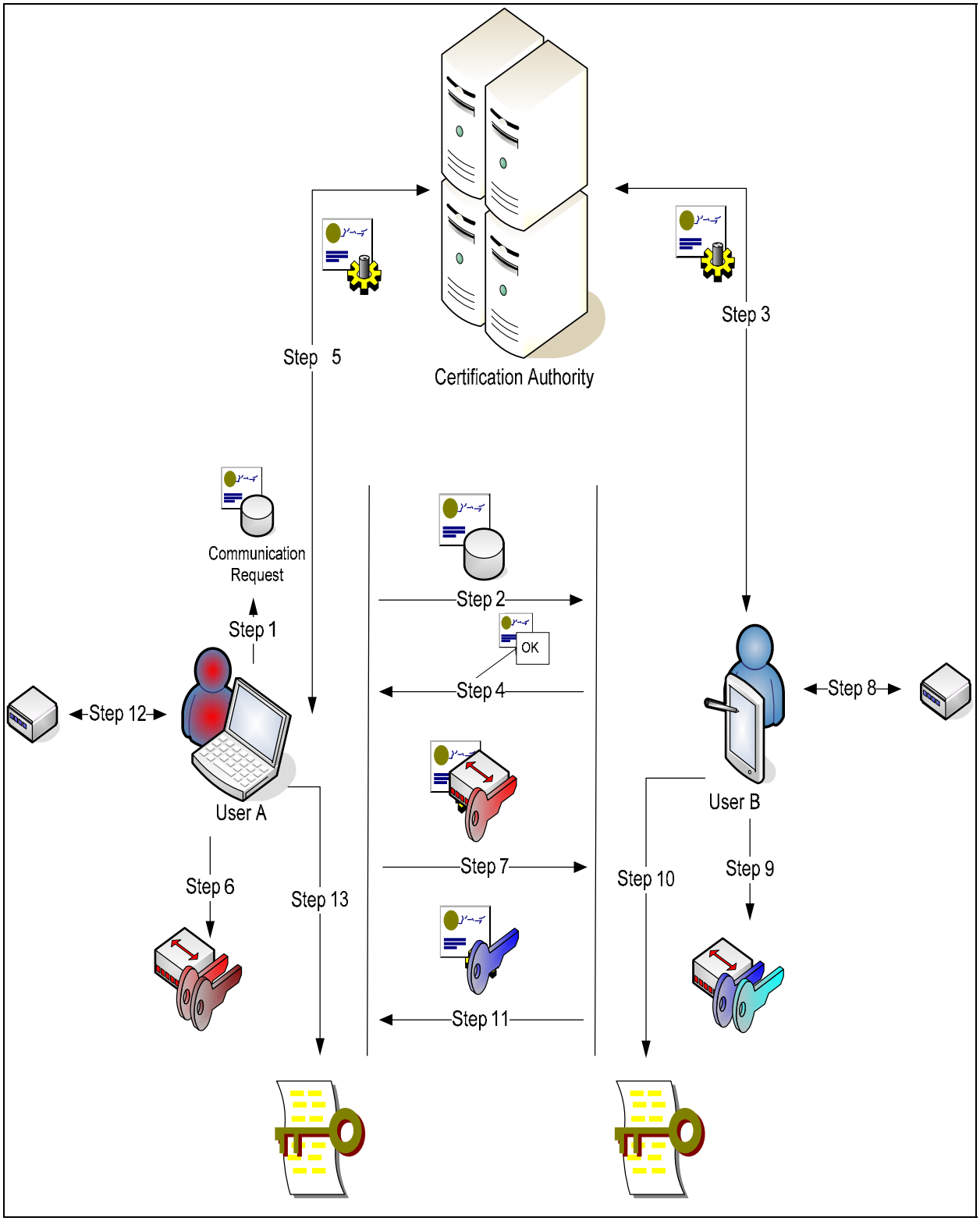

Figure 3.5. Authentication from CA without server by stored certificates

Step 7: User A sends required the domain parameters and the Public Key to User B including its own certificate.

Step 8: User B checks User A's certificate from the stored certificate.

Step 9: User B takes the domain parameters and generates the public-private key pair.

Step 10: User B computes session key.

Step 11: User B sends its public key to User A including its own certificate.

Step 12: User A checks User B's certificate from the stored certificate.

Step 13: User A computes the session key.

3.2.2. Key Exchange With Server Scenario

Building a certification authority mechanism is quite expensive and a time consuming operation. For this reason, companies mainly prefer to buy this service from an external CA. However, they also have many servers working inside the company. Therefore, they want to perform some of the hard and time consuming calculations on these servers.

At this point, the Key Exchange with Server scenario is extended by the usage of certification mechanism from External CA. In every operation, certificates are checked from the External CA. Although, this is a little bit time consuming operation, in order to make communication more secure, it should be performed. The communication steps are listed below and, the related scenario is depicted as Figure 3.6.

Step 1: User A prepares the communication request including its own certificate.

Step 2: User A sends the communication request message to User B.

Step 3: User B checks User A's certificate from the External CA.

Step 4: User B accepts the communication request including its own certificate.

Step 5: User A checks User B's certificate from the External CA.

Step 6: User A requests the necessary domain parameters from server including its own certificate

Step 7: Server checks User A's certificate from External CA.

Step 8: Server sends the necessary parameters to User A including its own certificate.

Step 9: User A checks Server certificate from External CA.

Step 10: User A generates the Public-Private Key pairs.

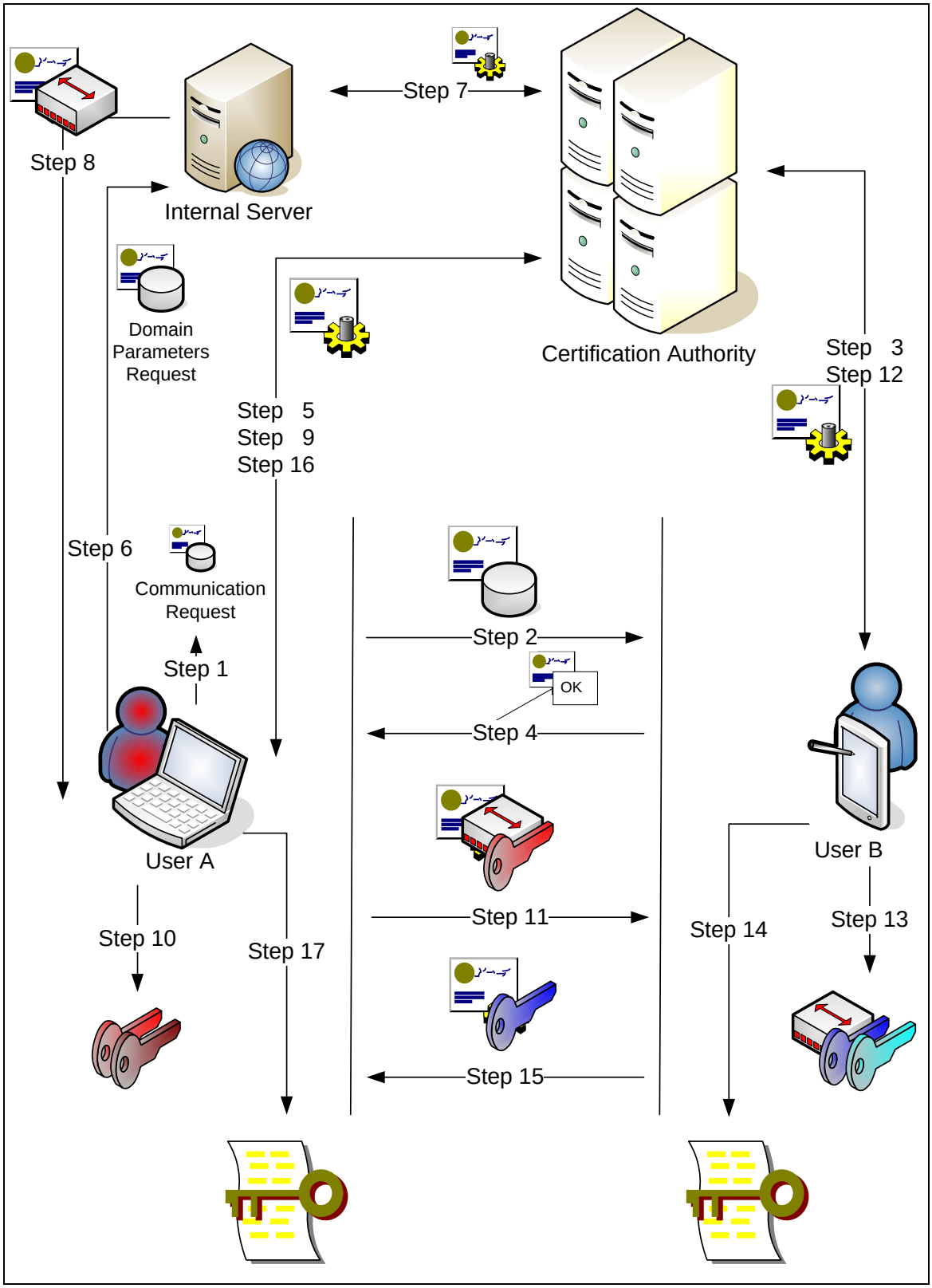

Figure 3.6. Authentication from CA with Internal Server

Step 11: User A sends the required domain parameters and Public Key to User B including its own certificate.

Step 12: User B checks User A's certificate from the External CA.

Step 13: User B takes the domain parameters and generates the public-private key pair.

Step 14: User B computes the session key.

Step 15: User B sends its public key to User A including its own certificate.

Step 16: User A checks User B's certificate from the External CA.

Step 17: User A computes the session key.

Again, the users might store the certificates of the others in the first check from the CA and check the certificate from the stored one for every transmission. After the communication ends, certificates must be deleted from the stored address.

Also, like internal server scenario, server certificate can be stored inside the devices so that client devices can perform the server checks faster.

Figure 3.7 shows the Authenticated Key Exchange without Server scenario for only one time certificate control on internal server. The communication steps are as follows.

Step 1: User A prepares the communication request including its own certificate.

Step 2: User A sends the communication request message to User B.

Step 3: User B checks User A's certificate from the External CA and saves the certificate.

Step 4: User B accepts the communication request including its own certificate.

Step 5: User A checks User B's certificate from the External CA and save the certificate.

Step 6: User A requests the necessary domain parameters from server including its own certificate

Step 7: Server checks User A's certificate from External CA.

Step 8: Server sends the necessary parameters to User A including its own certificate.

Step 9: User A checks Servers certificate from stored certificate.

Step 10: User A generates the Public-Private Key pairs.

Step 11: User A sends the required domain parameters and the Public Key to User B including its own certificate.

Step 12: User B checks User A's certificate from the stored certificate.

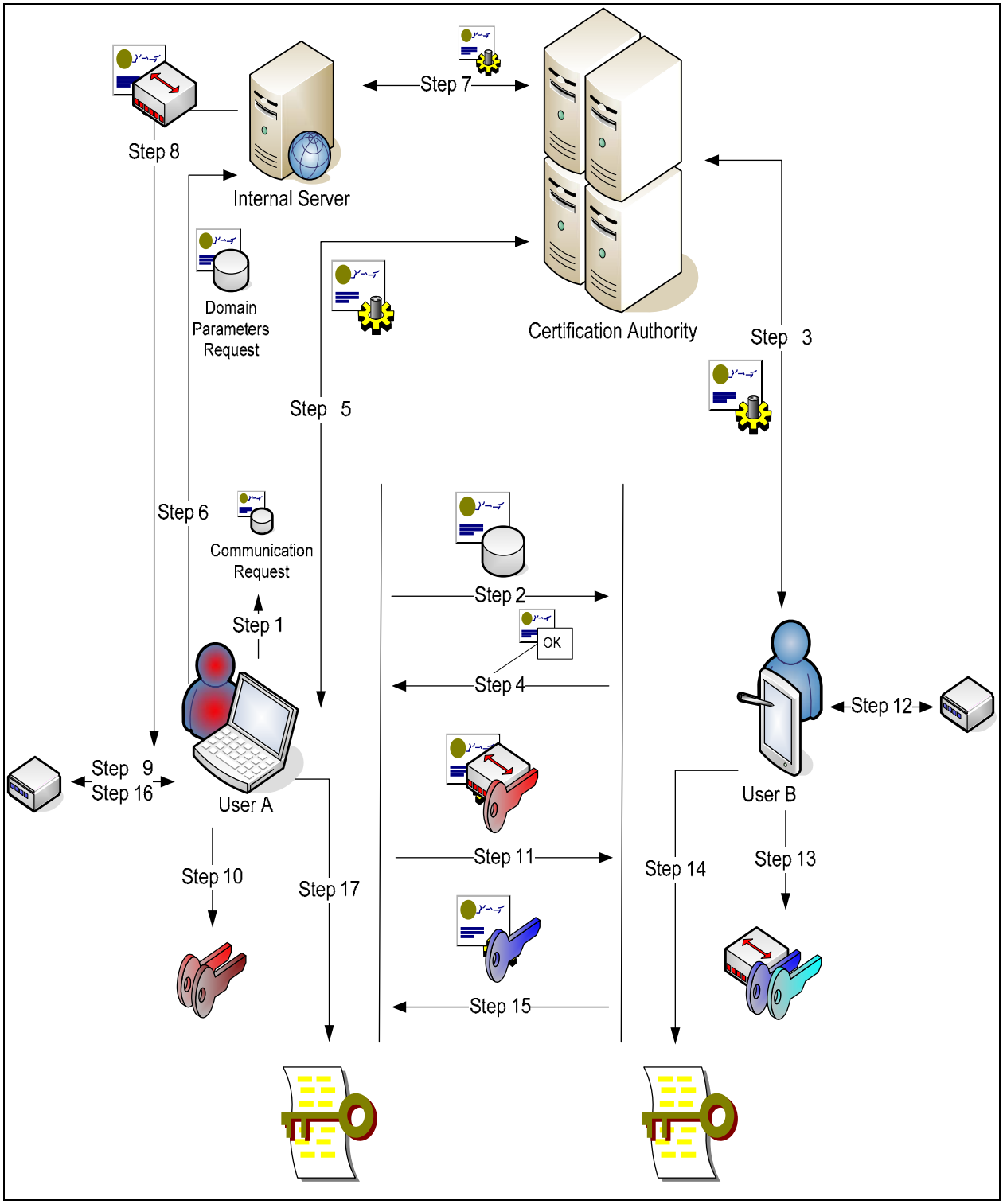

Figure 3.7. Authentication from CA with internal server by stored certificates

Step 13: User B takes the domain parameters and generates the public-private key pair.

Step 14: User B computes session key.

Step 15: User B sends its public key to User A including its own certificate.

Step 16: User A checks User B's certificate from the stored certificate.

Step 17: User A computes the session key.

If this scenario is employed in a company with unlimited internet access and sufficiently powerful hardware devices, the company should prefer to check the certificates in every transmission.

CHAPTER 4

SOFTWARE ARCHITECTURE

Software has been implemented for the scenarios told in the thesis. There are two components in this software. One of them is a windows application and the other is a web service application for generating necessary parameters. Crympix crypto library is used for generation operations. A c to c# wrapper has been implemented and included in this software to obtain necessary functions and structures in Crympix.

4.1. Protocol Implementation

Making key exchange software is a little bit complicated. There are two sides communicating with each other. Both can decide on the key exchange protocol to use and both can generate domain parameters. There are many different variations in the system shown in Figure 4.1.

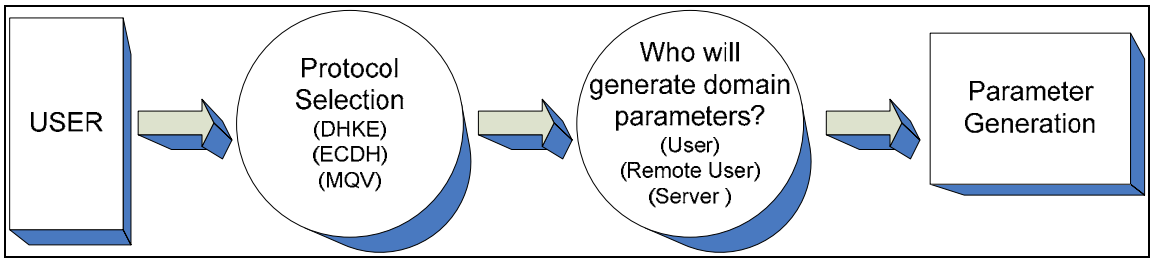

Figure 4.1. Variations of System

Each user can start the communication; the one who starts chooses the protocol for key exchange. Remote user is notified about the protocol and if he/she denies it, communication goes to the next step.

The next step is the generation of the domain parameters. Each user has an opportunity to generate domain parameters. Also users can send a job to the server and get the generated parameters.

All data transfers are in XML format in order to make this software adaptable with any other software. .Net platform has been chosen for this application but for mobile devices

especially, java applications can be written. So they can easily communicate with each other by XML structure.

Layered architecture is used in coding. So each operation has its own layer. This makes it easy to modify, update or fix the software. For example, all data are serialized in a specific layer and deserialization takes place in a remote machine in the same layer for the same data. In the case of XML structure change, only a change in the corresponding layer will be enough.

4.2. Inheritance and Abstract Classes

Key Exchange protocols have similar methods, such as

- getParameters()
- getPublicKey()
- getPrivateKey()
- getKey()
- calculateDomainParameters()
- calculateKeyPair()

So in coding, an interface that is implemented by all key exchange protocols has been created. By this structure all protocols are the same objects for coders. If a private key is needed, there is no need to know the protocol that is selected. Only getPrivateKey() method is called and generated private key for the selected protocol is returned.

For Diffie-Hellman Protocol a sample code is depicted in Figure 4.2. From the figure it's apparent that a class can be implemented from the interface for connecting to another library. So any library can be used only by changing the protocol classes.

```
[DllImport("crympix.dll")]
static extern int calculateDHDP(string p, string q);
public void calculateDomainParameters()
{
    try
    {
        calculateDHDP(((DH)KeyCarrier.instance().cargo).modulN,
                ((DH)KeyCarrier.instance().cargo).generatorG);
    }
    catch (Exception ex)
    {   …       }
}
```

Figure 4.2. Sample Code of Protocol Methods

These type of structures make applications easy to understand and easy for coding. Also upgrade by other people will be much simpler.

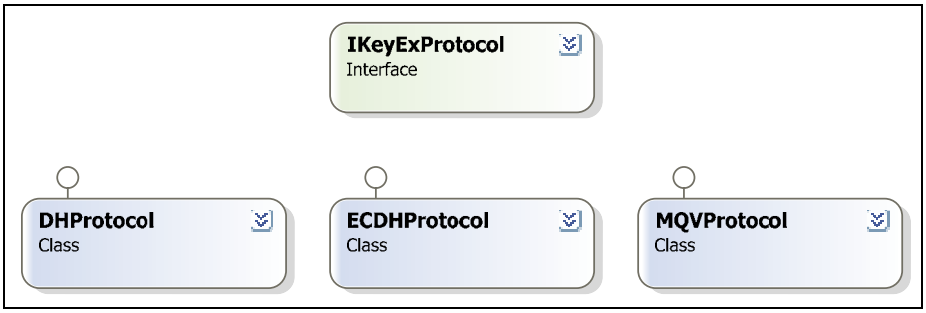

Figure 4.3. Class diagram of interface used in application

Abstract classes are also used in application for grouping same variables of keys such as key length, and for implementing similar methods like getProtocolType() or getKeyValues(). Every key object in applications is carrying its protocol type as well.

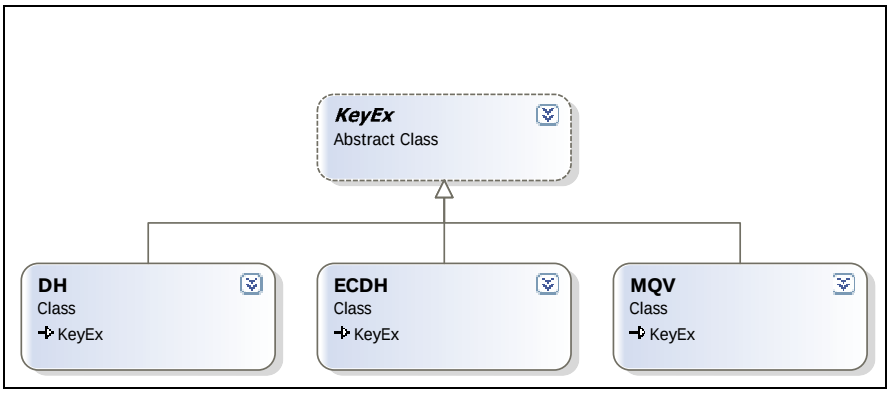

Figure 4.4. Class diagram of keys in application

A wrapper structure has been implemented like the structures in Crympix so that in .net applications one can get native dll structures of Crympix dll as .net objects. So upper levels of the software can easily get needed parameters from that object. Wrapper conversations can be seen in Figure 4.5 and in Figure 4.6

```
typedef struct{
      FP_t fp;
      FP_t q;
      MI_t g;
      MI_t x;
      MI_t X;
      MI_t Y;
      MI_t key;
}DH_FP_t[1], *DH_FP;
```

Figure 4.5 Crympix Diffie-Hellman Key Structure

```
public class DH:KeyEx
{
    private string m_generatorG;
    private string m_modulN;
    private string m_privateKey;
    private string m_key;
    private string m_publicKey;
    private string m_remotesPublicKey;

    public string remotesPublicKey {…}
    public string publicKey {…}
    public string generatorG {…}
    public string modulN {…}
    public string privateKey {…}
    public string key {…}

    public override KeyExchange.Business.Enums.ProtocolType getProtocolType()
    {
       return KeyExchange.Business.Enums.ProtocolType.DiffieHellman;
    }
    public override string getKeyValues()
    {
       …
    }
}
```

Figure 4.6 .Net Diffie Hellman Key Structure

4.3. XML and Web Services

Server side is a very important point in this application because most domain parameters will be generated in server side. Therefore, there must be an application in server side for getting the requests and returning the required domain parameters to the users.

XML Web services are the best solutions for this need. Because the future mobile devices with java applications can use this system for communication. XML will be the global language for these devices and all devices will use xml to send to or receive data from server.

Client applications are communicating with XML also, so that any application written in any programming language by using XML can communicate with this application with specific parameters.

Serialize and Deserialize methods are used for these operations. By serialize methods, application transforms objects to xml strings and makes it ready for transferring to other users. Deserialize methods get the xml strings and transform them into objects.

4.4. Adaptation to other Libraries

Application has been written in a layered architecture, the layers are shown in Figure 4.4. As told before, key exchange protocols are implementing IKeyExProtocol interface. So if another cryptographic library is chosen only new Key Exchange protocols must be written for calling those cryptographic libraries' methods which implement IKeyExProtocol.

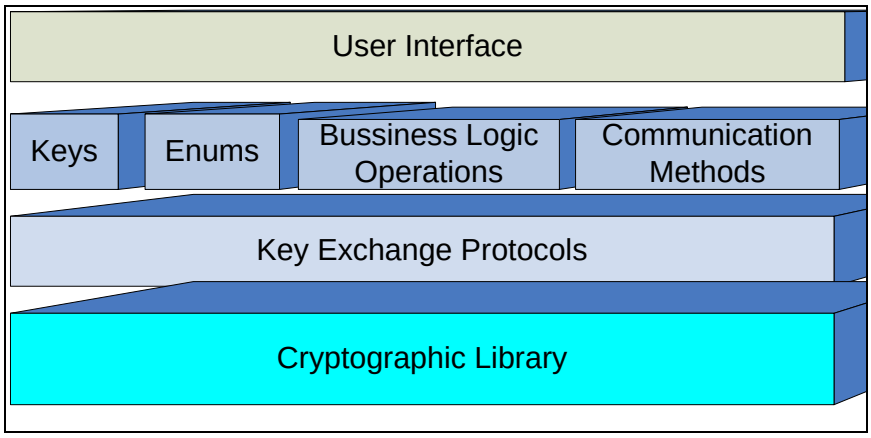

Figure 4.7. Layers of the application

For each cryptographic library only three classes for each protocol must be written. These are DHProtocol, ECDHProtocol and MQVProtocol.

CHAPTER 5

CONCLUSION AND THE FUTURE WORK

In the thesis many scenarios are explained. There are many possible algorithms and protocols for exchanging keys and to get a session key. However, as seen only implementing Key Exchange protocols is not enough. There can be some attacks. Therefore, in implementing a system, some security implementations and design must be done.

In this thesis, the importance of the attacks is shown. If attacks are not blocked, the attacker can get all the communication data without the knowledge of the sender or receiver.

In this thesis, a method is proposed to determine man in the middle attack, certification has the most important role in this operation. New generation mobile sim cards have users' certification inside, which means it is easy to use this application in mobile environments.

Application design is the other important point. By the layered design the application can be adapted to any cryptographic platform. This makes the application modular.

5.1. Future Work

Application has been implemented on .Net platform (www.microsoft.com 2008), for future work it can be developed in Java (java.sun.com 2008), Java Micro Edition (java.sun.com 2008) and .Net Compact Framework (msdn.microsoft.com 2008). By the improvements on mobile signatures and mobile hardware, mobile applications are getting more important so this application can be implemented in mobile frameworks for communicating mobile devices with others.

Also key exchange for more than two people scenarios can be developed and an application can be implemented. Those scenarios are important for companies for communicating with their staff.

REFERENCES

ANSI (American National Standards Institute) X9.42, 1998. Public Key Cryptography for the Financial Services Industry: *Agreement of Symmetric Keys Using Discrete Logarithm Cryptography.*

ANSI (American National Standards Institute) X9.63, 1999. Public Key Cryptography for the Financial Services Industry: *Key Agreement and Key Transport Using Elliptic Curve Cryptography.*

Developer Resources for Java Technology. http://java.sun.com (accessed June 05, 2008).

Hışıl, H. 2005. A distributed multiprecision cryptographic library design. *İ.Y.T.E Thesis of Ms.*

Law, L., Menezes, A., Qu, M., Solinas, S. and Vanstone, S., 1998. Efficient Protocol for Authenticated Key Agreement, Designs. *Codes and Cryptography* 28:119-134.

Menezes, A.J., Oorschot, P.C.V. and Vanstone, S.A., 1996. *Handbook of Applied Cryptography.* Boca Raton, FL: CRC Press..

Microsoft Dot Net Framework. http://www.microsoft.com/net (accessed June 05, 2008).

Microsoft Developer Network, http://msdn.microsoft.com (accessed June 05, 2008).

Schneier, B. 1996. *Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C.* New York: John Wiley & Sons.

Uslu, S. 2007. Basic Key Exchange Protocols for Secret Key Cryptosystems Under Crympix Library. *İ.Y.T.E Thesis of Ms.*