Analysis of Key Agreement Protocols

Brita Vesterås

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2006

Institutt for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

This master's thesis study the recently proposed key agreement protocols. Since they are so new, there has not been any detailed analysis of them; hence one cannot be sure of how secure they really are. The security of these protocols is essential; since they are responsible for the integrity and confidentiality of all transactions from the moment the protocol is employed. If the key agreement procedure contains security flaws, unintended parties can get hold of the key and use it to decrypt all the transmitted messages that are encrypted with this key.

In order to secure the systems, one is always interested in finding the key agreement protocol that achieve the highest possible level of security. Because of this, new protocols are suggested all the time. Some of these does not have any security proof, and even those protocols that have been through an analysis may still contain flaws.

This master's thesis analyse a few of these protocols and try to detect their weaknesses and find what kind of attacks that can be a threat to each of the protocols. We also try to enhance the protocols in which we find weaknesses.

Sammendrag - Abstract in norwegian

Masteroppgaven tar for seg noen av de siste foreslåtte protokollene for nøkkel-utveksling. Ettersom de er så nye, har de ikke gjennomgått en detaljert analyse, og kan derfor inneholde sikkerhetshull. Sikkerhet i protokollene er viktig, ettersom de er ansvarlig for integriteten og konfidensialiteten til all kommunikasjon fra det øyeblikket de tas i bruk.

For å unngå problemer er det viktig å finne den protokollen som har best sikkerhet. Nye protokoller blir foreslått for å prøve å tilfredsstille kravene, men det er ikke noe bevis for at de har klart det.

Denne masteroppgaven vil analysere noen av disse nye protokollene og prøve å avdekke deres svakheter. Vi prøver også å forbedre de protokollene der vi finner svakheter.

Preface

This master thesis is a part of the fulfillment of my Master's degree in Information Security at Gjøvik University College. I have been a student at Gjøvik University College from autumn 2001 to spring 2006. I spent the first three years studying for a Bachelor degree in Computer Science, and the last two to earn my Master's degree.

The essence of my study is computer security. Since more and more computer systems are used in our daily life, keeping these systems safe and secure is an important issue and a great challenge. An effective means to achieve this is to maintain a high level of integrity and confidentiality by encrypting information. Key agreement protocols play an important role in this context. Courses like Cryptology and Network Security have provided me with a good knowledge and inspiration to explore this topic further.

Working with this project has given me more knowledge to what kind of challenges we are facing regarding secure key agreement, and also information about some key elements to reduce the possibility of abuse.

Acknowledgement

I would like to thank my supervisor, Professor Chik How Tan, for his inspiration and support throughout the entire project period. He has been of great help, and given me constructive comments and ideas.

Thank you, Hossein, for the support and collaboration during this master study.

I would also like to thank my family, and Solveig, Desirée and Inger Lise for giving me the time and space I needed to finish this project.

Basic definitions

adversary - someone whose purposes are in conflict with one's own purposes.

availability - information and resources being available at the time of request.

collision-free hash function - a hash function where the chance of producing the same digest from two different messages is small.

confidentiality - keeping information secret from other than the intended recipient(s).

denial-of-service attack - an attack on a computer system or network which overwhelms the system. This results in users not getting the service they want.

digest - the result of a hash function.

discrete logarithm - similar to ordinary logarithms, but in a group context.

discrete logarithm problem - given g, h in a group G, finding k such that $h = g^k$.

Elliptic Curve Cryptography (ECC) - a cryptographic method that is based on elliptic curves.

Elliptic Curve Discrete Logarithm Problem (ECDLP) - the problem of finding k given the points kP and P.

field - an algebraic structure where addition, subtraction, multiplication and division (but not division by zero) may be performed.

finite field - a field that contains a finite number of elements.

hash function - a function that converts a string of any size into a fixed size string containing random-looking elements. A kind of digital fingerprint.

integrity - assurance that data has not been changed.

key control - the ability to choose the key, or the parameters used in key computation.

key establishment - mechanisms that enable two parties to establish some kind of secret.

prime - a number with no divisors (cannot be split up into smaller factors). A positive integer with no other divisors except 1 and itself.

revoke key - remove a key from use.

factorisation - finding prime factors of an integer.

safe prime - a prime number of the form $2p \pm 1$, where p is also prime.

Contents

List of Figures

1 Introduction

Key establishment, which is a fundamental building block in cryptography, is in [4] defined to be any process whereby a shared secret key becomes available to two or more parties, for subsequent cryptographic use.

There are two kinds of key establishment protocols; key transport protocols and key agreement protocols. *Key transport* protocols is created by one principal and securely transmitted to the second principal. In *key agreement* protocols, both principals contribute information which is combined in order to create the secret key.

Key establishment protocols have traditionally been among the hardest protocols to design.

There are several challenges concerning key exchange. These are:

- ensuring that the keys are exchanged so that sender and receiver can perform encryption and decryption
- preventing an eavesdropper from getting to know the key
- give the receiver some proof that a message was encrypted by the party who claims to have sent the message

This master thesis analyses a few recent key agreement protocols, and tries to find their weaknesses. It also tries to enhance protocols with weaknesses.

1.1 Keywords

Security, key agreement, key agreement protocols, cryptology.

1.2 Topic of this thesis

This thesis provides an overview of the different attacks against key agreement protocols, and the basics that these protocols are based on. With this knowledge in mind, a few protocols are analysed to see if they have any vulnerabilities that can be used to attack the protocols.

1.3 Problem description

Key agreement protocols are considered one of the hardest protocols to design, and are one of the most important parts of a system when it comes to integrity and confidentiality of data.

There are continuously new proposals for key agreement protocols. The problem concerning the security of these protocols are that they have no security proof. There has been no thorough analysis of the protocols, and there is not enough knowledge regarding attack methods in key agreement protocols.

The work of this thesis can be divided into three parts:

- provide a summary of different ways to attack key agreement protocols
- describe design criterias of the protocols
- analyse various key agreement protocols

1.4 Justification and motivation

This is an area that is rapidly changing. Considering the fact that information technology continuously becomes a greater part of our lives, security is a more and more important factor.

Many key agreement protocols have been proposed, but many of them are without security proof. Even with the security proof, the protocol may contain weaknesses that may be exploited with a new kind of attack. Because of this, we must continuously analyse protocols to make sure that they are sound.

After completion of this master thesis, one would have a more detailed understanding of the attack methods of key agreement protocols, some information on design of key agreement protocols and an analysis of how secure the protocols that have been analysed are.

1.5 Research questions

The research questions that will be explored in this thesis are:

- What are the attack methods of key agreement protocols?
- What are the design methods of key agreement protocols?
- What are the weaknesses of the protocols this thesis analyses?

1.6 Choice of methods

The methods which will be used to find answers to the research questions, is a combination of litterature study and a more experimental approach.

The answers to the first two research questions can be found in the litterature. The task will be to find the relevant information, summarise it and present it in a structured way.

We will be able to answer the third research question after an analysis of the five protocols. There are many different ways to do this. They span from an "ad-hoc" method where you put yourself in the attacker's place and try to attack the protocol, to formalised methods with strict rules. In this thesis, we have chosen to use a few protocol features as the basis of our analysis. In this way, we will get results in the time frame available, and a comparison of the results from the different protocols will be possible.

1.7 Structure of the thesis

The rest of the thesis is organised as follows: chapter 2 provides the theory and background for the thesis. Here we present the problems key agreement protocols are based on, like hash functions, the discrete logarithm problem, the factorisation problem and the elliptic curve discrete logarithm problem. Then we review some of the basic design methods, like Diffie-Hellman, elliptic curve cryptosystem and the MQV[1] protocol. We also present the common ways to attack key agreement protocols. Finally, we look at what features we will focus on in the analysis of the protocols of this thesis.

Chapter 3-7 analyse various key agreement protocols. Chapter 3 considers Wen-Lin-Hwang's protocol for clients with low computing power. In chapter 4 we present Popescu's protocol, which focuses on being efficient and providing authentication. Harn-Hsin-Mehta's protocol, based on a single cryptographic assumption, is described in chapter 5. Chapter 6 contains an analysis of Tseng's protocol, which focuses on preventing denial-of-service attacks. In chapter 7 we present Yoon-Yoo's protocol, where prevention of off-line password guessing and modification attacks are the priority.

The reason why exactly these protocols has been chosen, is that they are quite different from one another. They are also proposed recently and without security proof.

The findings of chapter 3-7 are summarised in chapter 8. In chapter 9 we make some conclusions. Chapter 10 considers future work and possible ways to improve protocols.

[1]Menezes, Qu and Vanstone

2 Preliminaries

There are some key features that is being used to create cryptographic protocols. These features include one-way functions to provide the security. From these basic means we get simple protocols that recent protocols are based on. Considering the important part computer systems have in our daily life, we need to make sure that our systems is as secure as possible. The way to do this is to analyse the cryptographic protocols thoroughly, so that they are as sound as possible. Then we may be assured that the protocol will resist attacks.

This chapter gives a brief summary of related work. It provides an overview of the one-way functions or "problems" key agreement protocols take advantage of, and presents some of the basic protocols that the recent protocols are based on. We also describe the most common methods to attack key agreement protocols. The last section provides the features we will focus on during our protocol analysis.

2.1 Related work

Key agreement protocols are the common way for two principals to achieve secure communication by establishing a session key to encrypt the data that is being sent between them [38]. These kind of protocols have a long history, the first known protocol was Diffie-Hellman in 1976. Since then, many key agreement protocols have been proposed.

These kind of protocols are one of the hardest to develop. The reasons for this are that key agreement protocols are interactive protocols between two or more parties and that there are many different ways to attack the protocols [17]. Designers are still trying to improve the security of the protocols, but despite of the designers' best effort and intentions some protocols still contain flaws.

The development of key agreement protocols is based on try-and-fail [17]. A new protocol is developed which provides good security. Then a weakness is found in the protocol, or a new kind of attack is discovered, and the protocol is not longer safe. The security attributes is updated based on the new knowledge, and the whole process starts again. There is so far no formalised way to develop protocols.

Going into details about this in this thesis is impossible, but more information can be found in [1] and professional journals. This thesis only focuses on recent key agreement protocols and new attack methods.

2.2 One-way functions

A one-way function [29] is described as a mathematical function which is easy to compute in one direction (the forward direction), but difficult to compute in the opposite direction. We also call this a non-reversible function. A trapdoor one-way function is a one-way function where the inverse computation is easy as long as you have some extra information (the trapdoor). If you do not have this extra information, the computation is hard.

Cryptosystems with private/public key pairs are based on trapdoor one-way functions. The private key is the extra information you need in order to compute the function in both directions. If you do not possess the private key, you may only compute the function in the forward direction. The forward direction is used for encryption and signature verification, and the inverse direction is used for decryption and signature generation.

No function has so far been proven to be a one-way function [29]. If there is some proof that a function believed to be one-way can in fact be computed both ways, it will make all cryptosystems based on that function insecure.

Here we describe some examples of one-way functions.

2.2.1 Hash Functions

A hash function is a kind of signature for a stream of data that represents the content. It converts a string of any length into a string of a fixed size. A hash function should behave as much as possible as a random function, but needs to be deterministic and efficient.

Even a slight change in the input value will result in a major change in the output value. This is called the Avalanche effect [41].

A hash function should provide the following properties [43]:

- given an output h it should be hard to find the corresponding input m such that $h = H(m)$.
- given an input m_1 it should be hard to find another input m_2 such that $H(m_1) = H(m_2)$.
- it should be hard to find two different messages m_1 and m_2 that result in the same hash value ($H(m_1) = H(m_2)$).

Hash functions may be used to:

- verify file integrity - make sure that the file has not been modified.
- store passwords - passwords should not be stored in clear text. Instead, the resulting hash value is stored.
- signing a document - the document is first hashed, and the result is encrypted with the private key in order to assure the recipient who sent the document.

The two most used hash functions are MD5 and SHA-1.

2.2.2 Discrete logarithm problem

Discrete logarithms [29, 45] are similar to regular logarithms, except that they are used in a group context. In cryptography, this means in the group Z_p*. This means the set of integers $1, 2, ..., p - 1$ under multiplication modulo the prime p. This may look something like:

$$4^5 \bmod 11 \tag{2.1}$$

Finding the result of this logarithm is easy, we compute $4^5 = 1024$ and then find the remainder after dividing with 11. The result is

$$4^5 \bmod 11 = 1 \tag{2.2}$$

But, if the equation looks like this:

$$4^k \bmod 11 = 9 \tag{2.3}$$

and we want to find the value of k, we are facing a challenge. Because of the modular context, there may be many possible answers. By raising 4 to the power of higher and higher values of k, we see that one value which gives the correct result is $k = 3$,

$$4^3 \bmod 11 = 9 \tag{2.4}$$

Equation 2.3 is an example of the discrete logarithm problem. There exists no efficient algorithm in order to solve this kind of problem. When the values are bigger than this, it can be very time-consuming. Also, the fact that there may be several values of k which gives the same result, complicates it even more.

2.2.3 Factorisation problem

Factorisation can be described as splitting an integer into a set of smaller integers [29]. These integers are called factors, and when you multiply them together, the result is the original integer. Prime factorisation means splitting an integer into factors that are prime numbers. Any positive integer can be represented as the product of prime numbers in one way only. Multiplying two prime integers is easy, but the factorisation of the result is hard. These two last characteristics are exactly what several cryptosystems take advantage of.

An example of a cryptosystem who use this idea is RSA [29]. RSA was developed in 1978 by Ronald Rivest, Adi Shamir and Leonard Adleman. The main idea is to choose two large primes p and q. These two values is the private key. Then we compute a value $n = p * q$, which is the public key. The assumption is that, if the values are sufficiently large, it will not be possible to retract p and q knowing n.

2.2.4 Elliptic curve discrete logarithm problem

The basic in elliptic curve cryptography is using points on the curve to perform computations [29, 46]. There is in advance defined a base point P and a modulo prime value p. Each user selects a value k which is his private key, and computes $Q = k * P \bmod p$ as his public key.

Since k is his private key, he does not want anyone to find this value. And this is where we take advantage of the elliptic curve discrete logarithm problem. Performing the reverse operation and find k when you know the value of Q is considered to be difficult, or perhaps impossible, to derive. And to complicate the matter even more, because of the presence of $\bmod p$, there may be several values for k that will give Q as a result, but you have no information about which value is the correct one.

2.3 Design methods

Most of the new key agreement protocols are based on some standard protocols. Here we describe the most important of these standards. We also present the assumed "problems" that is used in cryptography in order to keep information available to the intended parties, and unavailable to others.

2.3.1 Diffie-Hellman key agreement protocol

Diffie-Hellman key agreement protocol [30, 44, 2], invented in 1976 by Whitfield Diffie and Martin Hellman, is the earliest example of an asymmetric key establishment technique. It is still found in many secure connectivity protocols on the Internet.

Diffie-Hellman is a cryptographic protocol for secure exchange of a shared secret between two parties over an untrusted network. The two parties may not have ever communicated previously, but with their new shared secret key they can encrypt their communications over the insecure channel. The perhaps most important part of the protocol is that the key is not sent over the connection, so that it can be detected by an eavesdropper.

In [2] we get an explanation of the protocol. Alice and Bob agree on two numbers *p* and *g*. *p* is a large prime number and *g* is called the base generator. Alice picks a secret number *a* and computes her public number

$$x = g^a \bmod p \tag{2.5}$$

Bob picks a secret number *b* and computes his public number

$$y = g^b \bmod p \tag{2.6}$$

Alice and Bob then exchange their public numbers *x* and *y*. Alice computes

$$K_A = y^a \bmod p = (g^b \bmod p)^a \bmod p = (g^b)^a \bmod p = g^{ab} \bmod p \tag{2.7}$$

Bob computes

$$K_B = x^b \bmod p = (g^a \bmod p)^b \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p \tag{2.8}$$

The resulting keys are

$$K_A = K_B = K \tag{2.9}$$

and they have a secret key that they can use to encrypt messages.

Figure 1 shows an execution of the protocol.

Alice	*g, p*	***Bob***
a		
		b
$x = g^a \bmod p$		
	$\xrightarrow{x}$	
		$y = g^b \bmod p$
	$\xleftarrow{y}$	
$\begin{aligned} K_A &= y^a \bmod p \\ &= (g^b \bmod p)^a \bmod p \\ &= (g^b)^a \bmod p \\ &= g^{ab} \bmod p \\ &= K \end{aligned}$		
		$\begin{aligned} K_B &= x^b \bmod p \\ &= (g^a \bmod p)^b \bmod p \\ &= (g^a)^b \bmod p \\ &= g^{ab} \bmod p \\ &= K \end{aligned}$

Figure 1: Diffie-Hellman key agreement protocol

The special part about this protocol is that they have agreed on a key without actually passing this key between them and risking that it has been intercepted by anyone else. Even if any third party should get hold of x and y, and also knows the parameters g and p, he will still not be able to compute the key. In order to compute the correct key, he needs to know either a or b, but these values are kept secret.

The strength in this protocol, which makes it impossible to calculate a or b given x or y, lies in the modular arithmetic. This makes it a one-way function, because there are potentially a huge number of values of Alice's number a that would give the same x value (see section 2.2.2).

This key exchange mechanism does not specify any prior agreement or subsequent authentication between the participants. The protocols without that requirement are called anonymous key agreement protocols.

A drawback with the basic Diffie-Hellman protocol is that it has no authentication of the parties. This means that an adversary that controls the channel can perform a man-in-the middle attack to make the two communicating parties compute separate keys with the adversary. The two parties believe that they are sharing a key, and are unaware of what has really happened.

2.3.2 Elliptic Curve Cryptosystems

Elliptic curves as algebraic entities have been studied for over 100 years [24, 46]. The first time it was introduced in cryptography was in 1985.

An elliptic curve is defined by the equation

$$y^2 = x^3 + a * x + b \tag{2.10}$$

where x and y are variables, and a and b are constants. All the values come from a *field*. A field is a group of values that are familiar to one another. Some examples are real numbers, complex numbers and rational numbers. Integers modulo a prime number are an example of a *finite field*. This is the field that is used for cryptographic purposes. Here the equation has an extra element at the end:

$$y^2 = x^3 + a * x + b \bmod p \tag{2.11}$$

Elliptic curve cryptography depends on the difficulty of solving the discrete logarithm for the group of an elliptic curve over some finite field. This problem is called Elliptic Curve Discrete Logarithm Problem, ECDLP.

The most used finite field are the Galois field $GF(p)$, which contains the integers modulo a prime number p. $GF(2^n)$ are also often used.

When we have the elliptic curve E, and a field $GF(q)$, we look at the group of points $E(q)$ of the form (x, y), where both x and y are in $GF(q)$. There is also defined a group operation $" + "$. Then we define a second operation $" * "$. The two operations work as follows: If P is some point in $E(q)$, then we define

$$2 * P = P + P \tag{2.12}$$

$$3 * P = P + P + P \tag{2.13}$$

and so on.

When the elliptic curves are used in cryptography, one decides a base point P and a modulo prime value p, which are published. Each participant selects its own private key k and then publishes the value $Q = k * P \bmod p$ as the public key. If Alice and Bob have private keys k_A and k_B, and the public keys Q_A and Q_B, then Alice can calculate

$$k_A * Q_B = k_A * (k_B * P) \tag{2.14}$$

and Bob can compute the same value as

$$k_B * Q_A = k_B * (k_A * P) \tag{2.15}$$

Then they have agreed upon a secret value that is easy for them to compute, but difficult for any third party to derive.

Figure 2 shows an execution of the protocol.

Alice	*P, p*	***Bob***
k_A		
		k_B
$Q_A = k_A * P \bmod p$		
	$\xrightarrow{Q_A}$	
		$Q_B = k_B * P \bmod p$
	$\xleftarrow{Q_B}$	
$K = k_A * Q_B \bmod p$		
$= k_A * (k_B * P) \bmod p$		
		$K = k_B * Q_A \bmod p$
		$= k_B * (k_A * P) \bmod p$

Figure 2: Elliptic curve key agreement

One of the advantages of elliptic curve cryptography is that one can use a smaller key size than other methods, and still achieve the same level of security.

The additive group described above can be compared to the multiplicative group of powers of an integer g modulo a prime p: $g^0, g, g^2, \ldots$. The problem of finding k when you know kP and P is in this group similar to solving the discrete logarithm problem. It is therefore called the elliptic curve discrete logarithm problem (ECDLP).

ECC is regarded as the strongest asymmetric algorithm at a given key length. It may therefore be useful over links that have very tight bandwidth requirements.

2.3.3 MQV protocol

Menezes, Qu and Vanstone proposed a protocol called MQV [6, 15]. This protocol is used to establish a shared secret between two parties. Both parties generate dynamic private/public key pairs and exchange their public keys. Then each party calculates an implicit signature by using his own private key and the other party's public key. This signature is used to generate the shared secret. The secret generated by each party will be the same only if they are based on the corresponding public keys. This means that a man-in-the-middle attack will not be possible.

In [6, 15] we get the details of the protocol. The protocol is based on elliptic curves, which is described in section 2.3.2. P is a point on the elliptic curve $E(q)$. $\overline{X}$ and $\overline{Y}$ represent the first L-bits of the x-coordinate of the points X and Y, respectively. $L = \left[\frac{(\log_2 q)+1}{2}\right]$. The factor h is a co-factor, $h = \frac{|G|}{q}$.

A has the private/public key pair (a, A). Similarly, B has the private/public key pair (b, B). A selects a random value x and calculates

$$X = xP \tag{2.16}$$

B selects a random value y and calculates

$$Y = yP \tag{2.17}$$

The values X and Y are exchanged.

A calculates

$$s_A = (x + \overline{\chi}a) \bmod q \tag{2.18}$$

and

$$S_B = (Y + \overline{\gamma}B) \bmod q \tag{2.19}$$

B calculates

$$s_B = (y + \overline{\gamma}b) \bmod q \tag{2.20}$$

and

$$S_A = (X + \overline{\chi}A) \bmod q \tag{2.21}$$

Both A and B calculate the shared secret

$$K = h\, s_A\, S_B = h\, s_B\, S_A \tag{2.22}$$

Figure 3 shows an execution of the protocol.

MQV makes sure that no third party can intercept the messages.

Alice	*P, q*	***Bob***
x		
$X = xP$		
	$\xrightarrow{X}$	
		y
		$Y = yP$
	$\xleftarrow{Y}$	
$s_A = (x + \overline{\chi}a) \bmod q$		
$S_B = (Y + \overline{\gamma}B) \bmod q$		
		$s_B = (y + \overline{\gamma}b) \bmod q$
		$S_A = (X + \overline{\chi}A) \bmod q$
$K = h\, s_A\, S_B$		
		$K = h\, s_B\, S_A$

Figure 3: MQV protocol

2.4 Attack methods of key agreement protocols

There are several different ways to perform an attack on a key agreement protocol. In this subsection, we will briefly describe how attacks may be characterised, the difference between active and passive attacks and present some common attack methods.

2.4.1 Characterising attacks

The practical importance of an attack is dependent on the answers to these questions [42]:

- What knowledge and capabilities are needed as a prerequisite?
- How much additional secret information are deduced?
- How much effort is required?

How likely the different attacks are to be used in real life, varies a lot. A common assumption for cryptographers is to prepare for a worst case scenario, and think that if a scheme can resist unrealistic attacks, it will also resist attacks that it could face in reality.

2.4.2 Active and passive attacks

There are several different ways to attack key agreement protocols. We often divide them into passive and active attacks.

A *passive attack* is an attack which only require some network monitoring equipment. This is the easiest kind of attack, but also the kind that is easiest to protect against. All you need is to encrypt the transmitted messages, and the attacker will get no information out of the monitoring.

An *active attack* involves alteration or modification of a message. It requires more work, both from the attacker's side and the defender's side.

2.4.3 Different kind of attacks

There are many different ways an attacker can exploit a protocol This subsection contains a brief description of some of the most common attack methods [1].

Eavesdropping: Eavesdropping [1] means that an adversary captures information that is being sent in the protocol. Eavesdropping has existed throughout time, where someone overhears things they were not supposed to and when the communicating parties are not aware of him. In the information technology sense it means unintended parties getting hold of messages without the intended parties knowing it.

There is no way of preventing eavesdropping, but it is possible to protect the content of the messages. This can be done by using encryption. It helps assure the confidentiality of the message, since only those who know the key can decrypt the message. The eavesdropper can still get hold of the message, but has no way of knowing the content. This requires that the key has been kept secret, so that only the two communicating parties have information about the key.

This is one of the most basic kinds of attack, and more complex attacks might include eavesdropping as part of the attack. Eavesdropping is a kind of passive attack.

Modification: In a modification attack [1], the adversary alters the information that is sent in the protocol. This is a kind of active attack, since the attacker has a stronger role in this situation than a passive attack, where he just listens to the communication.

A way to prevent this kind of attack is to use cryptographic integrity measures.

Replay: A replay attack [1, 48] is an attack where a valid transmission is being recorded, and then later repeated, to the same or a different principal, for attacking purposes. This is done either by the originator or by an adversary who intercepts the data and retransmits it. This is a fundamental kind of attack, which is often used as a part of more complex attacks.

One way to avoid replay attacks is using session tokens [48]. These are used only one time, and will therefore reveal any attempts of replay. Another way of preventing this kind of attack is by using timestamps.

Reflection: A reflection attack [1] is a way of attacking a challenge-response authentication system that uses the same protocol in both directions. The idea is to trick the target into providing the answer to its own challenge. This attack is only possible if the protocol allow parallel runs.

The way to prevent this attack is to require the initiating party to first respond to challenges before the target party responds to its challenges. Another solution is to require the key or protocol to be different for each direction of communication.

Denial of service attack: A denial of service attack [1] is when attackers send many invalid requests to a server without establishing a server connection in order to overwhelm a server and stop legitimate users from getting a connection with the server. The attack may be divided in two groups: attacks trying to use up the computational resources (like CPU resources or storage resources) of the server and those trying to exhaust the number of allowed connections.

There are no complete solution to this kind of attack. The only defense is to reduce the impact each connection have on the server, by minimising computational calculations and the number of values that the server needs to store for each connection.

Typing attack: A typing attack [1] means that the adversary replaces a message field of one type with a message field of another type. This will make the recipient misinterpret a message, and accept a protocol element as another one (of a different type). For example could a principal identifier be falsely accepted as a key.

Cryptanalysis: Cryptanalysis [1, 42] is the study of methods for obtaining the meaning of encrypted information, without access to the secret information that is normally required [42]. In most cases, this kind of attack focuses on finding the secret key.

Frequency analysis is the basic tool for breaking classical ciphers and reveal the secret key. In normal conversation, some letters appear more frequently than others (i.e. "E" is the most frequent letter in the English language). This information can be used to try to decrypt substitution ciphers (encryption algorithms that replaces each letter with the same value each time); and from this reveal the secret key, which can then be used to decrypt following messages.

Certificate manipulation: Certificate manipulation is when the adversary modifies certificate information to perform an attack on a protocol. The certificate of a principal acts as an assurance from a trusted third party that the principal's public key really does belong to that principal [1]. The trusted authority does not necessarily receive some evidence that the principal really possesses the private key that corresponds with the value he claims to be his public key. This may cause problems when an adversary gains a certificate containing a public key value without having the corresponding private key.

Protocol interaction: Protocol interaction means that the adversary chooses a new protocol to interact with a known protocol. Most of the long-term keys are meant to be used for a single protocol only [1, 7]. Despite of this, some keys are used in more than one protocol. For example could a protocol which use decryption to prove possession of an authentication key be used by an adversary to decrypt messages from another protocol, provided that the same key is being used [1].

The ways to prevent this kind of attack is to use different keys for each protocol, and to include protocol details (like identifier and version number) in an authenticated part of the messages.

2.5 Protocol Analysis

As we said in section 1.6, there are many different ways to analyse protocols, but in this thesis we have chosen to focus on whether or not the protocols meet some specified goals. Here we present these goals and give a short description of them.

A secure protocol is capable of resisting both passive and active attacks. In addition to this, some other features are highly desired [3]:

1. *Known key security*: a protocol run should result in a unique secret session key. If this key is compromised, it should have no impact on other session keys.
2. *Forward secrecy*: The fact that long-term private keys are compromised, should have no impact on the secrecy of previously established session keys.
3. *Key-compromise impersonation resilience*: If entity A's long-term private key is compromised, an adversary is able to impersonate A. But this should not enable him to impersonate other entities to A.
4. *Unknown key-share resilience*: If entity A wants to create a secret key with B, it should not be possible that A is tricked into sharing a key with entity C.
5. *Key control*: Neither of the entities should be able to force the session key to a value of his choice.

We would also like to have some kind of authentication in our protocols, so that the principals can verify each other's identity. Authenticated key agreement protocols may be divided into two categories: *public-key-based key agreement protocols* and *password-based key agreement protocols*. The first category adopts signature schemes in order to provide mutual authentication. The second category allows both principals to share a secret password in advance to provide the authentication property. An authenticated key agreement protocol must achieve *one* of these two security goals [7] (from A to B):

- *implicit (weak) key confirmation:* B is assured that A is the only other entity that could know the correct key.
- *explicit (strong) key confirmation:* B is assured that A is the only other entity that knows the correct key (B has verified that A has received the correct key).

In an Authenticated Key Exchange (AKE) protocol, two requirements must be met [40]:

- *mutual authentication*: (two party authentication) both parties authenticate themselves in such a way that both parties are assured of the other party's identity.
- *secure communication*: the two communicating entities establishes a session key in order to secure later transmissions.

These features are greatly used for analysis of key agreement protocols. It is also these factors we will consider when analysing the protocols of this thesis.

Assumptions regarding the attacker's resources is not important in this kind of analysis, when we see if the protocols fulfill these specific goals. We illustrate this with the definition of forward secrecy: The way the long-term key is compromised, whether it has been revealed by a brute force attack or it was not properly protected by its owner, is not the focus here. What we are interested in, is the consequenses if this happens.

3 Wen-Lin-Hwang's protocol

In 2005, Wen, Lin and Hwang [40] proposed a protocol for key establishment for low computing power devices, like cellular phones, PDAs[1] and smart cards.

The article focuses on designing an AKE-LPC (Authenticated Key Exchange - for Low Power Computing clients) protocol, based on hybrid key architecture. A hybrid key architecture means that one entity (often a server) stores a pair of matching public/private keys while the other entity shares a secret with the server.

The authors propose two protocols, the last with a slight modification to achieve explicit mutual authentication. We will only consider the last one.

3.1 Notations

Here we describe the notations used in this protocol.

- U_C, U_S - Client (U_C) and server (U_S)
- s_C - A k-bit master key that U_C stores
- (pk_S, sk_S) - A public/private key pair held by U_S
- E_{pk_S}(x) - Encryption of x using the server's public key pk_s
- D_{sk_S}(y) - Decryption of y using the server's private key sk_s
- r_C, r_S - two random numbers (of k-bit length) of client (r_C) and server (r_S)
- f(), H() - one-way hash functions

3.2 Protocol

From the beginning, the client U_C stores it's master key s_C. The server U_S holds the private and public key pair (sk_S, pk_S), and maintains a public table which contains all identities (like U_C) and their corresponding verifiers (like $f(U_C, s_C)$). The table record for client U_C will be $U_C, f(U_C, s_C)$.

The protocol consists of two phases, the *precomputation* phase and the *protocol execution* phase. In the precomputation phase, U_C selects a random number r_C. U_C then computes the cipher text y from $E_{pk_S}(U_C, s_C, r_C)$. U_C then stores r_C and y, see figure 4.

In the *protocol execution* phase, U_C sends y to U_S. U_S decrypts y to obtain (U_C, s_C, r_C). U_S then checks if $f(U_C, s_C)$ matches with the value in the table. If it is a match, U_S then

[1]Personal Digital Assistant

(s_C) ***Client***	***Server*** (sk_S, pk_S)
$r_C \leftarrow \{0,1\}^k$	
$y = E_{pk_S}(U_C, s_C, r_C)$	

Figure 4: Wen-Lin-Hwang's protocol - precomputation phase

selects a random number r_S. If not, U_S terminates the protocol. U_S creates an authentication value $Auth_S = H(H[y, r_C, r_S], 2)$ and sends $r_S, Auth_S$ to U_C, see figure 5.

U_C verifies $Auth_S$ and creates an authentication value $Auth_C = H(H[y, r_C, r_S], 1)$ and sends to U_S. U_C computes the session key $SK = H(H[y, r_C, r_S], 0)$.
U_S verifies $Auth_C$. If it is okay, U_S computes the session key $SK = H(H[y, r_C, r_S], 0)$.

(s_C, r_C, y) ***Client***		***Server*** (sk_S, pk_S)
	$\xrightarrow{y}$	
		$(U_C, s_C, r_C) = D_{sk_S}(y)$
		$match\ with\ table?$
		$r_S \leftarrow \{0,1\}^k$
		$sk' = H(y, r_C, r_S)$
		$Auth_S = H(sk', 2)$
	$\xleftarrow{r_S, Auth_S}$	
$sk' = H(y, r_C, r_S)$		
$H(sk', 2)? = Auth_S$		
$Auth_C = H(sk', 1)$		
	$\xrightarrow{Auth_C}$	
		$H(sk', 1)? = Auth_C$
$SK = H(sk', 0)$		$SK = H(sk', 0)$

('?=' means checking if the two sides of the equation are equal)

Figure 5: Wen-Lin-Hwang's protocol - execution phase

3.3 Analysis of protocol

This protocol does not meet the forward secrecy goal. The long-term secret keys in this protocol are s_C and sk_S. s_C is used in $y = E_{pk_S}(U_C, s_C, r_C)$, and sk_S is used to decrypt y to get back the three values U_C, s_C, r_C. We assume that an attacker has eavesdropped on a previous key agreement session, and has the transmitted messages y, $(r_S, Auth_S)$ and $Auth_C$. If he manage to get hold of the key sk_S, he can decrypt the message y. Then he has y, r_C and r_S, which is all he needs to create the previously established session key $SK = H(H[y, r_C, r_S], 0)$. He can then use this key to decrypt all previously sent messages that was encrypted with this key. So this protocol fails to meet the forward secrecy goal.

We assume that an attacker has been eavesdropping on a previous run of the protocol between client and server, see figure 6.

$$\begin{array}{lcr}
\textbf{(}s_C\textbf{) \textit{Client}} & & \textbf{\textit{Server}}\ (sk_S, pk_S) \\
\hline
r_{C_1} \leftarrow \{0,1\}^k & & \\
y_1 = E_{pk_S}(U_C, s_C, r_{C_1}) & & \\
\hline\hline
 & \xrightarrow{y_1} & \\
 & & (U_C, s_C, r_{C_1}) = D_{sk_S}(y) \\
 & & \text{match with table?} \\
 & & r_{S_1} \leftarrow \{0,1\}^k \\
 & & sk_1' = H(y_1, r_{C_1}, r_{S_1}) \\
 & & Auth_{S_1} = H(sk_1', 2) \\
 & \xleftarrow{r_{S_1}, Auth_{S_1}} & \\
sk_1' = H(y_1, r_{C_1}, r_{S_1}) & & \\
H(sk_1', 2) ?= Auth_{S_1} & & \\
Auth_{C_1} = H(sk_1', 1) & & \\
 & \xrightarrow{Auth_{C_1}} & \\
 & & H(sk_1', 1) ?= Auth_{C_1} \\
SK_1 = H(sk_1', 0) & & SK_1 = H(sk_1', 0) \\
\hline
\end{array}$$

('?=' means checking if the two sides of the equation are equal)

Figure 6: Attack sequence on Wen-Lin-Hwang's protocol A - previous protocol run

They then use the session key to exchange some encrypted messages, see figure 7.

The attacker now has the information y_1, r_{S_1}, $Auth_{S_1}$, $Auth_{C_1}$, C_1 and C_2. He still does not have enough information to compute the session key, and hence he cannot decrypt the two messages.

$$\begin{array}{lcr}
\textbf{\textit{Client}} & & \textbf{\textit{Server}} \\
\hline
C_1 = E_{SK_1}(M_1) & & \\
 & \xrightarrow{C_1} & \\
 & & M_1 = D_{SK_1}(C_1) \\
 & & C_2 = E_{SK_1}(M_2) \\
 & \xleftarrow{C_2} & \\
M_2 = D_{SK_1}(C_2) & & \\
\hline
\end{array}$$

Figure 7: Attack sequence on Wen-Lin-Hwang's protocol B - transmission of encrypted messages

But if the attacker learns the value of the private key sk_S, he can decrypt y_1 and from that get the values U_C, s_C and r_{C_1}. He now has all he needs to compute the session key $SK_1 = H(y_1, r_{C_1}, r_{S_1})$. Then he may decrypt both C_1 and C_2 and retrieve the original messages. So the forward secrecy goal is not fulfilled.

The other goals are fulfilled in this protocol.

Known key security: The session key SK is computed from $H(H[y, r_C, r_S], 0)$. Besides from the value 0, all the values change each session. This means that even if the session key is compromised, it will have no effect on other session keys. So the protocol meets the known key security goal.

Key-compromise impersonation resilience: If the client's long-term private key s_C is compromised, an attacker can impersonate the client and create the message y. This is because the encryption algorithm, the servers public key pk_S and U_C is publicly known. All the attacker then needs in order to create the message $y = E_{pk_S}(U_C, s_C, r'_C)$ is a random value r'_C. But because the attacker does not know the server's secret key sk_s, he is not able to decrypt the value y and get the information he needs in order to compute the hash value $sk' = H(y, r_C, r'_S)$. He cannot get the correct value of r_C without decrypting y.

We now look at it in the other way, and assume that the server's private key sk_S is compromised. Then an attacker can decrypt the message y from a client. He can then complete the protocol and create a session key SK between the client and the attacker. But he still cannot impersonate another client to the server. He needs to know one of the client's private keys s_x in order to do this. So the protocol meets the key-compromise impersonation goal.

Unknown key-share resilience: Because of the $Auth$ messages the two parties exchange, they prove their identity to each other. As long as the server's private key sk_S is not compromised, only the server could decrypt y and get the r_S, which it needs to create an $Auth_S$ value that the client would accept. And still, as long as sk_S is not compromised, only the client who sent the first message will know the value of r_C and create an $Auth_C$ that the server would accept. So the protocol meets the unknown key-share resilience goal.

Key control: The session key $SK = H(H[y, r_C, r_S], 0)$ consists of two random values, one from each entity. Since the server picks his value last, and then already know both y and r_C, he can choose a r_S so that the session key results in the value that he wants. But, because of the hash functions, he needs to try different values of r_S until he finds a value that results in the wanted key. One of the assumptions regarding hash functions is that it is computationally infeasible to find a value x that will give the wanted result y (H(x) = y), as described in section 2.2.1. In this case it means that even though the server has three of the values needed to perform the session key (y, r_C and 0) and is in total control of the last value r_S, it would be practically impossible to find a value that, after the two hash functions, will result in the wanted key. So the protocol meets the key control goal.

Key confirmation: Because both principals verify each other's Auth values, they confirm that the other principal is computing the same session key. Hence the protocol provides strong key confirmation.

The protocol's specific goal: The goal for this protocol was to make the client perform as little computations as possible, and to achieve mutual authententication and secure communications, see section 2.5.

We see that the client only has to perform a few hash functions, which requires low resources.

The client is authenticated by sending his user ID U_C and secret s_C, encrypted with the server's public key pk_S in $y = E_{pk_S}(U_C, s_C, r_C)$. The server is authenticated by sending back f_C in $Auth_S = H(H[y, r_C, r_S], 2)$. If we assume that the server's private key sk_S is kept secret, which it should be, only the server could decrypt y and retrieve the value r_C.

Because of the two Auth messages, both parties know that they are using the same values y, r_C and r_S. They then know that they will compute the same session key.

So the protocol achieves mutual authentication and secure communication, and it has low computational requirements on the client side.

3.4 Improvement of the protocol

The original protocol did not meet the forward secrecy goal. If the long-term secret key s_C or sk_S were compromised, it would affect both the current and previous session keys.

If we should make a minor change to the protocol so that it meets the forward secrecy goal, it could look like the protocol in figures 8 and 9.

We introduce two new secret values, k_C and k_S. These values are used to create $n_C = y^{k_C}$ and $n_S = y^{k_S}$, which are used to compute the session key.

(s_C) Client	***Server*** (sk_S, pk_S)
$r_C \leftarrow \{0,1\}^k$ $y = E_{pk_S}(U_C, s_C, r_C)$ $k_C \leftarrow \{0,1\}^k$ $n_C = y^{k_C}$	

Figure 8: Wen-Lin-Hwang's protocol - precomputation phase - enhanced version

In our proposed protocol, the session key is computed by $SK = H(H[(n_C)^{k_S}, r_C, r_S], 0)$

(s_C, r_C, y, k_C, n_C) ***Client***		***Server*** (sk_S, pk_S)
	$\xrightarrow{y, n_C}$	
		$(U_C, s_C, r_C) = D_{sk_S}(y)$
		$\mathrm{match\ with\ table?}$
		$r_S \leftarrow \{0,1\}^k$
		$k_S \leftarrow \{0,1\}^k$
		$n_S = y^{k_S}$
		$sk' = H((n_C)^{k_S}, r_C, r_S)$
		$Auth_S = H(sk', 2)$
	$\xleftarrow{r_S, n_S, Auth_S}$	
$sk' = H((n_S)^{k_C}, r_C, r_S)$		
$H(sk', 2)? = Auth_S$		
$Auth_C = H(sk', 1)$		
	$\xrightarrow{Auth_C}$	
		$H(sk', 1)? = Auth_C$
$SK = H(sk', 0)$		$SK = H(sk', 0)$

('?=' means checking if the two sides of the equation are equal)

Figure 9: Wen-Lin-Hwang's protocol - execution phase - enhanced version

or $SK = H(H[(n_S)^{k_C}, r_C, r_S], 0)$. We use the idea from Diffie-Hellman, see section 2.3.1, when we make the first part of the inner hash value to be $y^{k_C k_S}$, where y^{k_C} and y^{k_S} is transmitted but not k_C or k_S alone. Even with the long-term private keys, an attacker can only get hold of the values n_C, n_S, r_C and r_S. But he does not know the secret random values k_C and k_S. So therefore he cannot compute the session key, and our protocol meets the forward secrecy goal.

4 Popescu's protocol

Popescu [28] has proposed an efficient and secure authenticated key agreement protocol that is based on the Diffie-Hellman key agreement. The protocol uses elliptic curve cryptography, which is described in section 2.3.2.

4.1 Notations

Here we describe the notations used in this protocol.

- P is a point on the elliptic curve $E(q)$
- H is a one-way hash function
- a, b are random integers from the interval [1, q-1]. These are the secret keys of user A and B, respectively
- $Y_A = -a * P$ and $Y_B = -b * P$ are two points on the elliptic curve. These are the public keys of user A and B, respectively
- $K_S = -b * Y_A = -a * Y_B = ab * P$ is the long term secret key of A and B

4.2 Protocol

The authenticated key agreement protocol is as follows: A generates a random integer k_A from the interval [1, q-1] and computes $V_A = -k_A * P$ and $e_A = H(x_{V_A}, x_{K_S})$, where x_{V_A} and x_{K_S} are the x-coordinate of the V_A and K_S points, respectively. A sends V_A and e_A to B.

Similarly, B randomly selects an integer k_B from the interval [1, q-1] and computes $V_B = -k_B * P$ and $e_B = H(x_{V_B}, x_{K_S})$. B sends V_B and e_B to A.

A computes $H(x_{V_B}, x_{K_S})$ and compares the result to e_B. If they are not identical, A terminates the execution. Otherwise A computes the point $K_A = -k_A * V_B$.

B computes $H(x_{V_A}, x_{K_S})$ and compares the result to e_A. If they are not identical, B terminates the execution. Otherwise B computes the point $K_B = -k_B * V_A$.

Figure 10 shows an execution of the protocol.

A and B then have computed the same key $K = -k_A * V_B = -k_B * V_A = k_A k_B * P$, and they have authenticated each other.

Alice	P, K_S	***Bob***
k_A		
$V_A = -k_A * P$		
$e_A = H(x_{V_A}, x_{K_S})$		
	$\xrightarrow{V_A, e_A}$	
		k_B
		$V_B = -k_B * P$
		$e_B = H(x_{V_B}, x_{K_S})$
	$\xleftarrow{V_B, e_B}$	
$H(x_{V_B}, x_{K_S})? = e_B$		
$K_A = -k_A * V_B$		
		$H(x_{V_A}, x_{K_S})? = e_A$
		$K_B = -k_B * V_A$

('?=' means checking if the two sides of the equation are equal)

Figure 10: Popescu's protocol

4.3 Analysis of protocol

This protocol does not meet the key-compromise impersonation resilience goal. If entity A's long-term private key a is compromised, an attacker can use this key together with B's public key Y_B and the publicly known value P to generate $K_S = -a * Y_B$. Then he can pretend to be A and execute the protocol with B.

We still assume that the attacker knows the private key a, only this time he wants to impersonate B to A. He computes the value K_S from $-a * Y_B$. He selects a random value $k_{B'}$ and computes $V_{B'}$ and $e_{B'}$. When he receives V_A from A, he can compute the session key between A and the attacker.

The attacker then can impersonate A to B, which is reasonable since it is A's key that have been compromised. But the attacker also has enough information to impersonate B to A. The attacker may either initiate a run of the protocol to A, while he pretends to be B (see figure 11), or he can intercept the transmissions A sends to B and respond back himself, posing as B (see figure 12). This is the part that makes the protocol fail to meet the wanted security feature.

The other goals are fulfilled in this protocol.

Known key security: The session key K is computed from $k_A k_B * P$. Both k_A and k_B are chosen randomly for each session. Even if this key is compromised, it will have no effect on other session keys. Therefore, the protocol meets the known key security goal.

Forward secrecy: The long-term private keys here are a and b. These values does not directly affect the session key. The session key K is computed by $k_A k_B * P$. So if the attacker wants to learn a previous session key, he must derive k_A from $V_A = -k_A * P$ and find the session key from $K_A = -k_A * V_B$ or derive k_B in the same manner. But deriving

C_B	P, K_S	***Alice***
k_C		
$V_C = -k_C * P$		
$e_C = H(x_{V_C}, x_{K_S})$		
	$\xrightarrow{V_C, e_C}$	
		k_A
		$V_A = -k_A * P$
		$e_A = H(x_{V_A}, x_{K_S})$
	$\xleftarrow{V_A, e_A}$	
$H(x_{V_A}, x_{K_S})? = e_A$		
$K_C = -k_C * V_A$		
		$H(x_{V_C}, x_{K_S})? = e_C$
		$K_A = -k_A * V_C$

('?=' means checking if the two sides of the equation are equal)

Figure 11: Attack sequence on Popescu's protocol - attacker C initiating a protocol run

these values from the V_x expressions are similar to solving the discrete logarithm problem, which is considered to be computationally infeasible, as described in section 2.2.2. So the protocol meets the forward secrecy goal.

Unknown key-share resilience: We assume an attacker C in the middle, trying to intercept all messages between A and B and replacing them with his own. He receives V_A, e_A from A, intended to B, and exchanges these values with V_C, e_C and sends these to B. B sends V_B, e_B to A, but C intercepts the message and replaces the values with V'_C, e'_C. A and C now share a key, and B and C share a key. But the part that makes this scenario impossible is the verification of the e_x messages. These verifications control the relationship between V_x and e_x. As long as no private keys are compromised, the attacker has no way of knowing the key K_S, and will therefore not be able to get the correct relationship between his values V_C and e_C, and V'_C and e'_C. When the verification fails, the protocol execution is terminated. Therefore the unknown key-share resilience goal are met in this protocol.

Key control: A does not have any way of forcing the session key to a value of his choice. But since B receives V_A before he chooses a value k_B, he may choose a value which will make the session key be the wanted value. However, in order to do this, he needs to determine which value k_B will give the wanted K_B in the expression $K_B = -k_B * V_A$, and that is similar to solving the elliptic curve discrete logarithm problem (see section 2.2.2). So the protocol meets the key control goal.

Key confirmation: In the verification of the e values in this protocol, the principals only verify if the other principal has knowledge of the common secret K_S. But it does not give any information about whether or not the other principal has computed the same session key. So this protocol only provides weak key confirmation.

Alice		***C***	***Bob***
k_A			
$V_A = -k_A * P$			
$e_A = H(x_{V_A}, x_{K_S})$			
	$\xrightarrow{V_A, e_A}$	intercept	
		k_C	
		$V_C = -k_C * P$	
		$e_C = H(x_{V_C}, x_{K_S})$	
	$\xleftarrow{V_C, e_C}$		
$H(x_{V_C}, x_{K_S})? = e_C$			
$K_A = -k_A * V_C$			
		$H(x_{V_A}, x_{K_S})? = e_A$	
		$K_C = -k_C * V_A$	

('?=' means checking if the two sides of the equation are equal)

Figure 12: Attack sequence on Popescu's protocol - attacker C intercepting a conversation

The protocol's specific goal: The goal for this protocol is to be efficient. And as we can see, the protocol only use hash functions and elliptic curve computations, which does not require much resources. So this goal is met.

4.4 Improvement of the protocol

The original protocol did not meet the key-compromise impersonation resilience goal. If an attacker gets hold of A's private key a, he may use this knowledge to impersonate B to A. If we make a slight modification of the protocol, see figure 13, we may be able to fulfill this goal as well.

In order to authenticate the parties, we want each party to prove possession of its private key. We let B compute $V'_A = -b * V_A$, which he sends to A. A then checks the equation $-k_A * Y_B = V'_A$. If B really knows the private key that corresponds to his public key Y_B, then the two sides of the equation will be equal and B has authenticated himself. In the same way, A computes $V'_B = -a * V_B$, which he sends to B. B then checks the equation $-k_B * Y_A = V'_B$. If the two sides of the equation are equal, A has proven that he knows the value of his private key a.

If A's long-term private key a are compromised, an attacker may still be able to impersonate A to B. But because he cannot compute a correct value for V'_A, he can no longer use his information about A's private key to impersonate B to A. So the key-compromise impersonation resilience goal is fulfilled.

Alice	P, K_S	***Bob***
k_A		
$V_A = -k_A * P$		
$e_A = H(x_{V_A}, x_{K_S})$		
	$\xrightarrow{V_A, e_A}$	
		k_B
		$V_B = -k_B * P$
		$e_B = H(x_{V_B}, x_{K_S})$
		$V'_A = -b * V_A$
	$\xleftarrow{V_B, e_B, V'_A}$	
$H(x_{V_B}, x_{K_S})? = e_B$		
$-k_A * Y_B? = V'_A$		
$V'_B = -a * V_B$		
	$\xrightarrow{V'_B}$	
$K_A = -k_A * V_B$		
		$H(x_{V_A}, x_{K_S})? = e_A$
		$-k_B * Y_A? = V'_B$
		$K_B = -k_B * V_A$

('?=' means checking if the two sides of the equation are equal)

Figure 13: Popescu's protocol - enhanced version

5 Harn-Hsin-Mehta's protocol

In 2004, Harn, Hsin and Mehta [10] proposed a few key agreement protocols based on Diffie-Hellman, which only used one single cryptographic assumption.They argued that if a protocol could be constructed using one cryptographic assumption, it would be at least as secure as that with multiple assumptions.

They associate a protocol based on multiple assumptions with a house with several doors with different security mechanisms. The more doors a house has, the more ways a thief can break into the house. The weakest security mechanism of all is the easiest to overcome. And the more assumptions a protocol has, the more ways an attacker can try to attack the protocol. The security of the protocol is then reduced to the security of the weakest assumption. The designers propose three different protocols, each based on a single cryptographic assumption. We will look at the protocol based on RSA factorisation.

5.1 Notations

Here we describe the notations used in this protocol. The two communicating parties are i and j.

- p_i, q_i - two safe primes, $p_i = 2p'_i + 1$ and $q_i = 2q'_i + 1$
- g_i - a long-term generator with order $2p'_i q'_i$, publicly known
- d_i - a long-term private key, $d_i \in [0,\ 2p'_i q'_i - 1]$
- n_i - a long-term public value, $n_i = p_i q_i$
- e_i - a long-term public value, $e_i d_i \bmod (2p'_i q'_i) = 1$
- R_i - a long-term public value, $R_i = (g_i)^{d_i} \bmod n_i$
- $cert_i$ - a certificate containing the values n_i, R_i, e_i and g_i
- k_i - a short term key, $k_i \in [0,\ n_j - 1]$
- r_i - a short term key, $r_i = (g_j)^{k_i} \bmod n_j$
- s_i - the signature

5.2 Protocol

For the computations, each party needs some of the other party's public keys. A is given the values g_B, n_B, R_B and e_B from B's certificate, and B is given the values g_A, n_A, R_A and e_A from A's certificate.

A chooses a value k_A, computes $r_A = (g_B)^{k_A} \bmod n_B$ and sends r_A to B. B chooses a value k_B and computes $r_B = (g_A)^{k_B} \bmod n_A$. He also computes $k_{AB} = (r_A)^{d_B} \bmod n_B$ and $s_B = (r_B)^{d_B} k_{AB} \bmod n_B$, and sends the values (r_B, s_B) to A. A computes $k'_{AB} = (R_B)^{k_A} \bmod n_B$. A verifies s_B and k'_{AB} by checking the equation $(r_B)(k'_{AB})^{e_B} = (s_B)^{e_B} \bmod n_B$. If the two sides are equal, A computes $k_{BA} = (r_B)^{d_A} \bmod n_A$ and $s_A = (r_A)^{d_A} k_{BA} \bmod n_A$. A then sends s_A to B. B computes $k'_{BA} = (R_A)^{k_B} \bmod n_A$. B verifies s_A and k'_{BA} by checking the equation $(r_A)(k'_{BA})^{e_A} = (s_A)^{e_A} \bmod n_A$.

Figure 14 shows an execution of the protocol.

Alice		***Bob***
k_A $r_A = (g_B)^{k_A} \bmod n_B$		
	$\xrightarrow{r_A}$	
		k_B $r_B = (g_A)^{k_B} \bmod n_A$ $k_{AB} = (r_A)^{d_B} \bmod n_B$ $s_B = (r_B)^{d_B} k_{AB} \bmod n_B$
	$\xleftarrow{r_B, s_B}$	
$k'_{AB} = (R_B)^{k_A} \bmod n_B$ $(r_B)(k'_{AB})^{e_B} ?= (s_B)^{e_B} \bmod n_B$ $k_{BA} = (r_B)^{d_A} \bmod n_A$ $s_A = (r_A)^{d_A} k_{BA} \bmod n_A$		
	$\xrightarrow{s_A}$	
		$k'_{BA} = (R_A)^{k_B} \bmod n_A$ $(r_A)(k'_{BA})^{e_A} ?= (s_A)^{e_A} \bmod n_A$

('?=' means checking if the two sides of the equation are equal)

Figure 14: Harn-Hsin-Mehta's protocol

Here we get two session keys, k_{AB} for transmissions from A to B, and k_{BA} for transmissions from B to A.

5.3 Analysis of protocol

This protocol does not meet the forward secrecy goal. The long-term private keys are d_x. If we assume that d_B is compromised, an attacker can compute the key k_{AB} from $(r_A)^{d_B} \bmod n_B$ (n_B is publicly known, and r_A can be found by eavesdropping on the conversation between A and B). So clearly this will affect the current session key.

If the attacker has eavesdropped on previous conversations and archived the messages, both from the protocol execution and messages encrypted with previous session keys, he may use the knowledge of d_B to derive the previous session keys and read the encrypted messages.

First, we assume a previous run of the protocol between A and B, where the attacker C was eavesdropping on the conversation, see figure 15.

$$\begin{array}{lcr}
\textbf{\textit{Alice}} & & \textbf{\textit{Bob}} \\
\hline
k_{A_1} & & \\
r_{A_1} = (g_B)^{k_{A_1}} \bmod n_B & & \\
 & \xrightarrow{r_{A_1}} & \\
 & & k_{B_1} \\
 & & r_{B_1} = (g_A)^{k_{B_1}} \bmod n_A \\
 & & k_{AB_1} = (r_{A_1})^{d_B} \bmod n_B \\
 & & s_{B_1} = (r_{B_1})^{d_B} k_{AB_1} \bmod n_B \\
 & \xleftarrow{r_{B_1}, s_{B_1}} & \\
k'_{AB_1} = (R_B)^{k_{A_1}} \bmod n_B & & \\
(r_{B_1})(k'_{AB_1})^{e_B} ?= (s_{B_1})^{e_B} \bmod n_B & & \\
k_{BA_1} = (r_{B_1})^{d_A} \bmod n_A & & \\
s_{A_1} = (r_{A_1})^{d_A} k_{BA_1} \bmod n_A & & \\
 & \xrightarrow{s_{A_1}} & \\
 & & k'_{BA_1} = (R_A)^{k_{B_1}} \bmod n_A \\
 & & (r_{A_1})(k'_{BA_1})^{e_A} ?= (s_{A_1})^{e_A} \bmod n_A \\
\hline
\end{array}$$

('?=' means checking if the two sides of the equation are equal)

Figure 15: Attack sequence on Harn-Hsin-Mehta's protocol A - previous protocol run

$$\begin{array}{lcr}
\textbf{\textit{Alice}} & & \textbf{\textit{Bob}} \\
\hline
C_1 = E_{k_{AB_1}}(M_1) & & \\
 & \xrightarrow{C_1} & \\
 & & M_1 = D_{k_{AB_1}}(C_1) \\
 & & C_2 = E_{k_{BA_1}}(M_2) \\
 & \xleftarrow{C_2} & \\
M_2 = D_{k_{BA_1}}(C_2) & & \\
\hline
\end{array}$$

Figure 16: Attack sequence on Harn-Hsin-Mehta's protocol B - transmission of encrypted messages

The attacker C now has stored the information about r_{A_1}, r_{B_1}, s_{B_1}, s_{A_1}, C_1 and C_2. C can not decrypt the messages because he does not have enough information to compute the session key.

Then suppose that the attacker gets hold of the private key d_B. Then C can compute the session key $k_{AB_1} = (r_{A_1})^{d_B} \bmod n_B$. He can use this key to decrypt the ciphertext C_1 and retrieve the message M_1, see figure 16. But he can still not decrypt C_2. If the attacker continues to eavesdrop on the conversations B has, he can compute each session key for transmission from A to B until B changes his private key. So the forward secrecy goal is not fulfilled.

The other goals are fulfilled in this protocol.

Known key security: The session keys k_{AB} and k_{BA} are based on the random values k_A and k_B. Each k_x value are independent of both previous and future ones, so disclosure of the current secret session key will have no impact on other session keys. So the protocol meets the known key security goal.

Key-compromise impersonation resilience: If we assume that A's private key d_A is compromised, an attacker can execute the protocol and get a common key with B. (B believes he is sharing a key with A). So clearly the attacker can impersonate A. But if the attacker, still knowing d_A, tries to impersonate B to A, he will not be able to compute $k_{AB} = (r_A)^{d_B} \bmod n_B$, because he does not know d_B. A will detect this during the verification of s_B. So the protocol meets the key-compromise impersonation resilience goal.

Unknown key-share resilience: If we assume an attacker C acting as a man-in-the middle intercepting and modifying messages as he pleases, in order to try to trick one of the parties into sharing a key with him. But because of the way both parties compute a trial value k'_{xx} based on some public information R_x of the party they want to establish a common key with, and as long as the k'_{xx} value is verified against the received values r_x and s_x, he will not succeed. So the protocol meets the unknown key-share resilience goal.

Key control: The only changing values used to generate the session keys are k_A and k_B. Both A and B chooses these values before they get to know the rest of the variables they will use in order to generate the keys. So neither party will have the possibility to force the value of the session keys. Therefore, the protocol meets the key control goal.

Key confirmation: Because both principal verifies the received value s_x and the computed value k'_{xx}, they can verify that the other principal has computed the same session key for that direction. So the protocol provides strong key confirmation.

The protocol's specific goal: The designers here want to make a protocol based on only one assumption, namely RSA factoring. We see that for an attacker to compute the session key from A to B, he will need to find k_A from $r_A = (g_B)^{k_A} \bmod n_B$, so that he may compute $k_{AB} = (R_B)^{k_A} \bmod n_B$. This means solving the discrete logarithm problem, which is described in 2.2.2. As an alternative he can try to find the private key d_B of B. In order to do this, he needs to extract d_B from $e_B d_B \bmod (2p'_B q'_B) = 1$. The attacker needs to find the values for p_B and q_B from n_B in order to find p'_B and q'_B. This is the same as solving the RSA factorisation problem, which is described in section 2.2.3.

The attacker only needs to solve one of these problems in order to find the session key for one direction. But we see that he could avoid the RSA factorisation and instead try to solve a discrete logarithm. Therefore, the protocol is not based solely on RSA factorisation.

5.4 Improvement of the protocol

The problem with the original protocol was that it did not meet the forward secrecy goal. If the long-term private key of either party was compromised, an attacker could compute both the current and previous session keys. If we should make a minor change to the protocol so that it meets the forward secrecy goal, it could look like the protocol in figure 17.

We introduce two new values, $m_A = (R_A)^{k_A} \bmod n_A$ and $m_B = (R_B)^{k_B} \bmod n_B$. These values are used to make the session keys dependent on more secret values than just the long-term secret keys.

$$
\begin{array}{lcr}
\textbf{\textit{Alice}} & & \textbf{\textit{Bob}} \\
\hline
k_A & & \\
r_A = (g_B)^{k_A} \bmod n_B & & \\
m_A = (R_A)^{k_A} \bmod n_A & & \\
& \xrightarrow{r_A,\, m_A} & \\
& & k_B \\
& & r_B = (g_A)^{k_B} \bmod n_A \\
& & m_B = (R_B)^{k_B} \bmod n_B \\
& & k_{AB} = (r_A)^{d_B k_B} \bmod n_B \\
& & s_B = (r_B)^{d_B} k_{AB} \bmod n_B \\
& \xleftarrow{r_B,\, m_B,\, s_B} & \\
k'_{AB} = (m_B)^{k_A} \bmod n_B & & \\
(r_B)(k'_{AB})^{e_B} ?= (s_B)^{e_B} \bmod n_B & & \\
k_{BA} = (r_B)^{d_A k_A} \bmod n_A & & \\
s_A = (r_A)^{d_A} k_{BA} \bmod n_A & & \\
& \xrightarrow{s_A} & \\
& & k'_{BA} = (m_A)^{k_B} \bmod n_A \\
& & (r_A)(k'_{BA})^{e_A} ?= (s_A)^{e_A} \bmod n_A \\
\hline
\end{array}
$$

('?=' means checking if the two sides of the equation are equal)

Figure 17: Harn-Hsin-Mehta's protocol - enhanced version

In our proposed protocol, the key for messages transmitted from A to B is computed by $k_{AB} = (r_A)^{d_B k_B} \bmod n_B$. So even if the attacker gets to know B's long-term private key d_B, he cannot compute the key because he does not know the secret random value k_B. If we look at A's computation of the same key, it constitutes of $k'_{AB} = (m_B)^{k_A} \bmod n_B$. m_B is transmitted in clear text, and n_B is publicly known. But an attacker will not know the value k_A. Extracting k_A from r_A or m_A, or k_B from r_B or m_B is the same as solving the discrete logarithm problem (see section 2.2.2), which is considered impossible .

The same way of thinking applies for the session key in the opposite direction. So knowing the long-term secret key d_A or d_B is not enough to be able to compute the current or previous session keys. You also need to know k_A or k_B. Therefore, our proposed protocol meets the forward secrecy goal.

6 Tseng's protocol

In 2005, Tseng [38] proposed a key agreement protocol with explicit key confirmation which require very little computational cost in order to reduce the impact of denial-of-service attacks. They focus on CPU-exhaustion and storage-exhaustion.

CPU-exhaustion means that the attacker tries to use up all the server's computational resources, and *storage-exhaustion* is when the attacker fills the server's memory with variables connected to the various requests to the server.

In this analysis, we will only look at the last protocol, namely the New Authenticated Key Agreement Protocol.

6.1 Notations

Here we describe the notations used in this protocol.

- p, q - two large primes, where q is a factor of $p-1$
- g - a generator in $GF(p)$ with order q
- H - a collision-free hash function
- I_i - identity information of user i
- x_i - secret key, $x_i \in [0,\ q-1]$
- y_i - public key, $y_i = g^{x_i} \bmod p$
- E_B, D_B - the server's encryption and decryption functions
- K_S - the server's master secret, used together with a timestamp to generate the secret key of E_i
- K - the final session key between A and B

The values p, q, g and H are published.

6.2 Protocol

The client A chooses a random number $a \in [0, q-1]$ and computes $r_A = g^a \bmod p$. The server B also chooses a random number $b \in [0, q-1]$ and computes $r_B = g^b \bmod p$, $s_B = b + x_B * H(r_B, y_B) \bmod q$ and $M_B = g^{s_B} \bmod p$.

A sends I_A to B. B computes $c_B = E_B(r_B, M_B, s_B)$. B sends r_B, c_B and I_B to A.

A computes $M'_B = r_B * (y_B)^{H(r_B, y_B)} \bmod p$, $V = H(M'_B, r_A, r_B, y_A)$ and $s_A = a +$

$x_A * V \bmod q$. A also computes $M_A = g^{s_A} \bmod p$, $e = H(M_A, r_A)$, $K'_A = (M'_B)^{s_A} \bmod p$ and $T_A = H(r_A, r_B, K'_A)$. A sends V, T_A, r_A, e, c_B and I_A to B.

B recovers r_B, M_B and s_B by decrypting c_B. B then checks to see if V is equal to $H(M_B, r_A, r_B, y_A)$. If it is not, the protocol is terminated. Otherwise, B computes $M'_A = r_A * (y_A)^V \bmod p$ and compares e to the result of $H(M'_A, r_A)$. If the values are not equal, the protocol is terminated. Otherwise, B computes $K'_B = (M'_A)^{s_B} \bmod p$ and checks if T_A is equal to $H(r_A, r_B, K'_B)$. Finally, B computes $T_B = H(r_B, r_A, K'_B)$. B sends T_B to A.

A checks if both sides of the equation $T_B = H(r_B, r_A, K'_A)$ are equal. If not, the protocol is terminated. If it is okay, A computes the session key $K = H(K'_A)$. B also computes the same session key from $K = H(K'_B)$.

Figure 18 shows an execution of the protocol.

6.3 Analysis of protocol

All the goals are fulfilled in this protocol.

Known key security: The session key $K = H(K'_x)$ is based on a hash value. This means that as long as there is a slight change in the input value K'_x, the output K will be totally different. Based on the assumption that a hash function is a one-way function, as presented in section 2.2.1, the disclosure of K will not reveal any information about other session keys.

But if we instead assume that the value K'_x is compromised, let us see what happens then. The value K'_x is based on the values s_A and s_B. These two values are computed using the random values a and b, respectively. These values change each time, with no distinct pattern. Therefore, it is not possible to get information about other session keys even if the value K'_x is compromised. So the protocol meets the known key security goal.

Forward secrecy: The long-term private keys in this protocol are x_A and x_B. These values are used to compute $s_A = a + x_A * V \bmod q$ and $s_B = b + x_B * H(r_B, y_B) \bmod q$. The values s_A and s_B are used to generate the values K'_A and K'_B, respectively. To compute K'_A we need M'_B and s_A, and to compute K'_B we need M'_A and s_B. We can compute $M'_B = r_B * (y_B)^{H(r_B, y_B)}$ (y_B is publicly known, and r_B we get from eavesdropping on the conversation), and M'_A we can find in the same fashion. But we cannot get the values s_A or s_B because we do not know the random values a and b. In order to find a or b, the attacker must derive these values from $r_A = g^a \bmod p$ or $r_B = g^b \bmod p$. Extracting these values would be similar to solving the discrete logarithm problem, as presented in section 2.2.2. Therefore, the protocol meets the forward secrecy goal.

Key-compromise impersonation resilience: If A's long-term private key x_A is compromised, an attacker can participate in a protocol run and set up a session key between him and B. B believes he is sharing a key with A.

Alice	p, q, g, H	***Bob***
a		
$r_A = g^a \bmod p$		
		b
		$r_B = g^b \bmod p$
		$H_B = H(r_B, y_B)$
		$s_B = b + x_B * H_B \bmod q$
		$M_B = g^{s_B} \bmod p$
	$\xrightarrow{I_A}$	
		$c_B = E_B(r_B, M_B, s_B)$
	$\xleftarrow{r_B, c_B, I_B}$	
$M'_B = r_B * (y_B)^{H(r_B, y_B)} \bmod p$		
$V = H(M'_B, r_A, r_B, y_A)$		
$s_A = a + x_A * V \bmod q$		
$M_A = g^{s_A} \bmod p$		
$e = H(M_A, r_A)$		
$K'_A = (M'_B)^{s_A} \bmod p$		
$T_A = H(r_A, r_B, K'_A)$		
	$\xrightarrow{V, T_A, r_A, e, c_B, I_A}$	
		$(r_B, M_B, s_B) = D_B(c_B)$
		$V \ ?= H(M_B, r_A, r_B, y_A)$
		$M'_A = r_A * (y_A)^V \bmod p$
		$e \ ?= H(M'_A, r_A)$
		$K'_B = (M'_A)^{s_B} \bmod p$
		$T_A \ ?= H(r_A, r_B, K'_B)$
		$T_B = H(r_B, r_A, K'_B)$
	$\xleftarrow{T_B}$	
$T_B \ ?= H(r_B, r_A, K'_A)$		
$K = H(K'_A)$		
		$K = H(K'_B)$

('?=' means checking if the two sides of the equation are equal)

Figure 18: Tseng's protocol

Let us assume that the attacker wants to use his knowledge of x_A to impersonate B to A. Because of the relationship between M_x and M'_x, where one is based on the private key x_x and the other is based on the corresponding public key y_x, the attacker will not be able to pull this off. He will generate s_B using the wrong private key. A computes M'_B and then K'_A based on B's public key y_B. They will then end up with different values of K'_x. A will discover this when he receives T_B, and terminate the protocol. So the protocol meets the key-compromise impersonation resilience goal.

Unknown key-share resilience: Because of the relationship between M_x and M'_x mentioned above, it is not possible to be tricked into sharing a key with another entity than what was intended. The T_A and T_B verifications will detect any attempts of this, and the protocol will be terminated. So the protocol meets the unknown key-share resilience goal.

Key control: The impact each party could have on the key is when they choose their random values a and b. B chooses his value early on, and will have no impact on the final key. A also does this early, but since he does not transmit his value, he can change it after he gets the value M'_B. But finding a value a which will give a s_A that results in the wanted session key, is very difficult. In order to do this, A needs to find a input value K'_A to the hash function, which will give the wanted session key as output. After that, A needs to get the s_A value from the expression $K = (M'_B)^{s_A} \bmod p$ in order to get a a value to use to compute r_A. But this scenario requires A to break the hash function and solve the discrete logarithm problem, which is presented in section 2.2.1 and 2.2.2. Both of these problems are considered computationally infeasible. So the protocol meets the key control goal.

Key confirmation: Because both principals controls the received value T and make sure that the other principal has used the same value for K'_x as they have computed, they know that they will compute the same session key. So this protocol provides strong key confirmation.

The protocol's specific goal: This protocol tries to reduce the impact of denial-of-service attacks against the server (here B) by minimising the server's computations and storage needs.

On the computational side, we see that B needs to perform 5 hash functions, 1 encryption, 1 decryption, 2 modular multiplications and 4 modular exponentiations. This is quite a lot, but if we take one more look at the protocol, we see that about half of these computations comes after the verification of the client (here A). After responding to A, the protocol lets A perform some computation and send some values to B. B verifies this by performing the same computation and comparing it to the received value V. If the two values does not match, B terminates the protocol. If A is trying to perform a denial-of-service attack, he will probably not take the time to compute the values properly. B will then detect this fast, and stop the communication.

When it comes to the storage part, we see that B encrypts all values connected to this session and sends it back to A. This way he does not have to store anything before he has verified A and retrieved the values back in the third transmission of the protocol.

7 Yoon-Yoo's protocol

In 2005, Yoon and Yoo [50] proposed a key agreement protocol which they claim to be efficient and simple, and able to resist off-line password guessing and modification attacks. The protocol is based on elliptic curve cryptography.

7.1 Notations

Here we describe the notations used in this protocol.

- P - a common password between A and B
- $GF(q)$ - a finite field
- $E(q)$ - an elliptic curve over $GF(q)$
- Q - a point on the elliptic curve $E(q)$
- P' - a point on $E(q)$ computed from P with a password
- a, b - the secret random value of A and B, respectively
- H - a hash function

7.2 Protocol

A selects a random integer a and computes $X_1 = a * Q + P'$. A then sends X_1 to B.

B similarly chooses a random integer b and computes $Y_1 = b * Q$, $X = X_1 - P' = a * Q$ and $SK_B = b * X = a * b * Q$. B then sends Y_1 and $H(X, SK_B)$ to A.

A computes $SK_A = a * Y_1 = a * b * Q$. A then compares the expressions $H(a * Q, SK_A)$ and the received $H(X, SK_B)$. If they are not equal, the protocol terminates. Otherwise, A sends $H(Y_1, SK_A)$ to B.

B compares the expressions $H(b * Q, SK_B)$ and $H(Y_1, SK_A)$. If they are not equal, the protocol terminates. Otherwise, the protocol execution has finished, and A and B have agreed on the key $K = H(SK_A) = H(SK_B)$.

After the verifications of the hash values, both parties have authenticated each other.

Figure 19 shows the execution of the protocol.

Alice	P, Q	***Bob***
a		
$X_1 = a * Q + P'$		
	$\xrightarrow{X_1}$	
		b
		$Y_1 = b * Q$
		$X = X_1 - P'$
		$SK_B = b * X$
		$H_B = H(X,\ SK_B)$
	$\xleftarrow{Y_1,\ H_B}$	
$SK_A = a * Y_1$		
$H(a * Q, SK_A)\ ?= H_B$		
$H_A = H(Y_1,\ SK_A)$		
	$\xrightarrow{H_A}$	
		$H(b * Q, SK_B)\ ?= H_A$
		$K = H(SK_B)$
$K = H(SK_A)$		

('?=' means checking if the two sides of the equation are equal)

Figure 19: Yoon-Yoo's protocol

7.3 Analysis of protocol

All the goals are fulfilled in this protocol.

As long as there is no totally private keys in this protocol, only the shared secret password P, we find that the key-compromise impersonation resilience goal is not fit in this context, and choose to leave it out of the analysis. We may instead conclude that if this password is compromised, the principals may not be authenticated.

Known key security: The secret session key here is $K = H(SK_A) = a * Y_1 = a * b * Q$. The values a and b are random for each session key, so the compromise of the current session key will have no effect on other session keys. Therefore, the protocol meets the known key security goal.

Forward secrecy: The private key in this protocol are the password P, which is used to derive the point P'. If we assume that an attacker is eavesdropping on the conversation, he will have archived transactions of previous X_1's and Y_1's. If he later gets to know P and can derive P', he still need to get either a from X_1 or b from Y_1 in order to compute the previous session key. Finding these values means solving the elliptic curve discrete logarithm problem. So the protocol meets the forward secrecy goal.

Unknown key-share resilience: The information that authenticates the entities in this protocol is the knowledge of the point P'. The verification of $H(a*Q, SK_A) = H(X, SK_B)$, and the similar equation for B, assures both parties that the other party knows the password and was able to compute the correct point P'. As long as this point (and the password P which it is derived from) is not compromised, the attacker cannot trick entity A into sharing a key with another entity than B. So the protocol meets the unknown key-share resilience goal.

Key control: B has the possibility to choose a value b after receiving X_1 from A. The session key is computed by the hash value of b and the received value from A. Finding a b which gives the wanted output from the hash function is considered computationally infeasible, see section 2.2.1. Therefore, the protocol meets the key control goal.

Key confirmation: Because of the verification of the hash values H_A and H_B the two principals may be assured that the other principal has computed the same session key. Therefore, this protocol provides strong key confirmation.

The protocol's specific goal: The protocol's goal is to be efficient and simple, and to resist off-line password guessing and modification attacks.

From the diagram of the protocol, we see that it looks quite simple. It does not have many transmissions or computations. The computations in the protocol is based on elliptic curve computations and hash functions, which does not require much computational resources.

When it comes to the password-guessing, we see that they never use the real password. Instead they use an elliptic curve point that is derived from the password. If an attacker eavesdrops on the conversation, he can get the values X_1, Y_1, H_B and H_A. From the two first values, he may try to guess P' by testing some values P'' and find $X' = X_1 - P''$. But in order to verify his guess, he needs to compute $H' = H(X', SK'_B)$ and compare the result to H_B. If he could derive b from $Y_1 = b * Q$, he could compute $SK'_B = b * X'$ and verify his guess. But as we have seen in section 2.2.4, this is the same as solving the elliptic curve discrete logarithm problem. We may then conclude that the protocol resists password guessing attacks.

Because of the verification of the received H_x values, both parties know that the other party has received the correct value and that they are computing the same key. A modification attack is therefore not possible.

8 Summary

As we have seen in the previous chapters, the structure of the protocols varies.

Figure 20 shows a summary of the protocols' structure and what security features they provide. The protocols are listed in the same order as they were analysed:
P1 Wen-Lin-Hwang's protocol
P2 Popescu's protocol
P3 Harn-Hsin-Mehta's protocol
P4 Tseng's protocol
P5 Yoon-Yoo's protocol

Summary of the protocols

Feature	P1	P2	P3	P4	P5
# of transmissions	3	2	3	4	3
# of hash computations	8	4	0	12	6
# of encryptions	1	0	0	1	0
# of decryptions	1	0	0	1	0
# of modular multiplications	0	0	2	4	0
# of modular exponentiations	0	0	8	8	0
# of EC point computations	0	4	0	0	6
Known Key Security	OK	OK	OK	OK	OK
Forward Secrecy	NO	OK	NO	OK	OK
Key-Compromise Impersonation	OK	NO	OK	OK	-
Unknown Key-Share Resilience	OK	OK	OK	OK	OK
Key Control	OK	OK	OK	OK	OK
Key Confirmation	strong	weak	strong	strong	strong

Figure 20: Summary of analysed protocols

Different computations require different computational resources. Hash functions are easy to compute, and encryption and decryption is a bit more computationally expensive. Modular exponentiation requires most computational resources.

From the result of the analysis of the five protocols, we notice some security flaws. This does not come from lack of knowledge on the designer's part or a too hasty development, but it is simply because it is difficult, if not impossible, to focus on all aspects of security at the same time. Others think in a different way and with a slightly different perspective, and may therefore be able to detect problems the designers did not see. This is not necessarily a bad thing, it could instead be a good means for quality assurance.

The analysis we have seen does not prove the security of the protocols. But it is a helpful means for determining which protocols are sound. There is no assurance that the most advanced protocols are the best.

8.1 Key manangement

As we have seen from the analysis, the problems some of the protocols encountered may be avoided if the long-term secret keys are kept safe.

The life cycle of a key is [7]:

- creation
- distribution
- usage
- archiving
- destruction

The key generation is based on the specifications of the key (number of bits etc) and a source of random bits. The key needs to be as random as possible. The generation may be done by one of two parties: either by the user itself or by a TTP [1] on behalf of the user. If this is the case, it must be distributed to the user in a secure manner. For instance, the new key may be encrypted with a key shared between the user and the TTP during transmission.

All private keys need to be stored in a secure manner. This way, no unauthorised disclosure or modification of the keys may occur. A solution to this problem is to use tamper-resistant hardware device, where you need some sort of PIN code or password to get access to the content. Another possibility is to protect the key by encrypting it. We may also divide the key into several parts, and store them separately, in such a way that you need all parts of the key in order to make use of it. If the key's lifetime is fulfilled, and the key needs to be archived, it must still be stored in a secure manner.

When a key is to be destroyed, we need to remove all information about the key. Simply deleting it from the harddrive is not enough. We may overwrite the memory of the hard drive with random data in order to remove all traces of the key. Another solution may be to store the key on an external storage media, like a CD kept in a secure place. When the key is no longer in use, the media is physically destroyed.

When a long-term private key is disclosed, all messages transmitted with session keys based on that key may be affected. If we minimise the time a long-term key is in use, we minimise the impact it will have if the key is disclosed. Using each key for only one single purpose will also reduce the damage if the key is compromised.

If and when we detect that the long-term private key has been disclosed, we must immediately revoke that key and create a new one. The lifetime of a key must be less than the time it takes to find the key by exhaustive search.

As a summary, these are the things we need to remember when working with long-

[1]Trusted Third Party

term private keys in order to keep the key, and the whole system with it, secure [7]:

- limit the time a long-term key is in use
- use each key only for a single purpose
- store the key in a secure manner
- if you suspect that a secret key have been compromised, revoke it immediately and create a new key
- when a key's lifetime is fulfilled, the key must be destroyed in a secure way

By employing these simple measures, we have a good foundation for our system's security.

9 Conclusion

The purpose of this master thesis is to analyse a few key agreement protocols and try to detect their weaknesses. If they have any weaknesses, we want to see what kind of attacks may be launched against them. We also try to improve the protocols so that these attacks will no longer succeed.

We will here make some conclusions regarding the research questions in 1.5.

9.1 Attack methods of key agreement protocols

As we have seen in section 2.4.2, there exists two basic categories of attacks. These are passive attacks, where messages only is gathered but not altered, and active attacks, where there are some modification of the messages.

There are quite a lot of attack methods under these two categories, and we will just state them here:

- Eavesdropping - an adversary captures information that is sent in the protocol
- Modification - the information sent in the protocol is altered by an adversary
- Replay - a transmission is recorded, and then later retransmitted
- Reflection - sending the same message back to the sender in a new protocol run
- Denial of service attacks - the attacker overwhelms the server so that legitimate users will not get a connection with the server
- Typing attack - replacing a message field of one type with a message field of another type
- Cryptanalysis - the study of methods for obtaining the meaning of encrypted information
- Certificate manipulation - modification of the certificate information
- Protocol interaction - using a new protocol to interact with a known protocol

More information about these kind of attacks can be found in section 2.4.2.

9.2 Design methods of key agreement protocols

All key agreement protocols are based on some standard protocols. In 2.3 we defined Diffie-Hellman, MQV and the elliptic curve cryptography as the most used standards.

Cryptography are also based on some problems that are assumed to be hard to solve, unless you have some extra secret information. We have in 2.2 defined a few of these problems:

- Hash functions
- Discrete logarithm problem
- Factorisation problem
- Elliptic curve discrete logarithm problem

9.3 Weaknesses in the protocols

As we have seen in the analysis, three of the five analysed protocols fail to meet some of the security features described in 2.5. But mostly, the analysed protocols have good results.

Two of the protocols fail to meet the forward secrecy goal, and one protocol fail to meet the key compromise impersonation resilience goal. All the problems are caused by disclosure of the long-term private key. If we take the necessary steps in order to keep this key secret, the encountered problems will not have any impact on the security. In section 8.1 we make some suggestions about how to keep the private key safe.

We have also proposed an improvement of each of these three protocols, where we try to enhance the parts where the protocols have problems. By performing slight modifications to the original protocol, we are able to fulfill the goals that the original protocol did not meet. It is important to notice that these changes are only suggestions, there may also be other ways to fulfill the goals.

Considering the fact that three of five protocols fail to meet some of the security features, we see that there is a need for better routines regarding key agreement protocols. Since the security of the entire system is based on the security of the cryptographic schemes that is employed, we need to get these schemes as secure as possible. Key agreement protocols is an important part of these schemes.

Today, everyone is free to propose and publish key agreement protocols. Others may read and verify them, and if they find weaknesses they perhaps publish this together with their improvement of the protocol. This is a good means for assuring the protocol's security, but which protocols that is reviewed is quite random. So the protocols that are not reviewed does not get the same security verification.

One way of improving these routines is by establishing a society of cryptographers that verifies all new protocol proposals before they are published. If they find some weaknesses in the protocol, the developers get to change the protocol and send it in again. This way, all protocols have been through an analysis and the security might be improved. The developers should not be intimidated by this society and not send in their work before it is "perfect"; this should only be a means for improving the security of the protocols, which benefits all parties.

10 Future work

The development over the last years have been increased use of communication technologies. We have no reason to think that this development will not continue. This results in an increasing demand for computer systems that have high availability, integrity and confidentiality.

The focus here are integrity and confidentiality, which we may achieve by using cryptographic measures. In order to keep these two at a high level, we need systems that are secure. A secure system is, as we have seen, dependent on the cryptography and therefore also on key agreement protocols.

The way new protocols are analysed now is a bit of a coincidence. The way things are done now, by having others to look at your work is a good start. However, getting some standards as to how to analyse protocols would help improve the security of the protocols. Also, the sooner the protocol can be analysed, the better. Then we manage to fix the problems before the protocols is taken into use.

As new attacks arise, protocols needs to be verified again and we may need to develop some new protocols who can resist the new attacks.

Bibliography

[1] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer Verlag, 2003.

[2] D. A. Carts. "A Review of the Diffie-Hellman Algorithm and its Use in Secure Internet Protocols". http://www.sans.org/rr/whitepapers/vpns/751.php, 2001.

[3] L. Chen and C. Kudla. "Identity Based Authenticated Key Agreement Protocols from Pairings". http://www.hpl.hp.com/techreports/2003/HPL-2003-25.pdf, 2003.

[4] R. Choo. Provably-Secure Mutual Authentication and Key Establishment Protocols Lounge. http://sky.fit.qut.edu.au/ choo/lounge.html.

[5] K.-K. R. Choo, C. Boyd and Y. Hitchcock. "The importance of proofs of security for key establishment protocols - Formal analysis of Jan-Chen, Yang-Shen-Shieh, Kim-Huh-Hwang-Lee, Lin-Sun-Hwang, and Yeh-Sun protocols". *Journal of Computer Communications, Special Issue of Internet Communication Security*, 2006.

[6] "Code & Cipher - Certicom's Bulletin of Security and Cryptography". http://www.certicom.com/download/aid-90/C&C_voll_iss2.pdf, 2003. Vol. 1, no. 2.

[7] A. W. Dent and C. J. Mitchell. "*User's Guide to Cryptography and Standards*", Artech House Computer Security Series, 2004, chapter 11-12, pp. 215-266.

[8] R. Dutta and R. Barua. "Overview of Key Agreement Protocols". http://citeseer.ist.psu.edu/cache/papers/cs2/433/http:zSzzSzeprint.iacr.orgzSz2005zSz289.pdf/overview-of-key-agreement.pdf.

[9] S. Friedl. "An Illustrated Guide to Cryptographic Hashes". http://www.unixwiz.net/techtips/iguide-crypto-hashes.html.

[10] L. Harn, W.-J. Hsin and M. Mehta. "Authenticated Diffie-Hellman key agreement protocol using a single cryptographic assumption". *IEE Proceedings on Communications*, vol. 152, issue 4, pp. 404-410, 2005.

[11] G. Horn, K. M. Martin and C. J. Mitchell. "Authentication protocols for mobile network environment value-addedservices". *Vehicular Technology, IEEE Transactions on*, vol. 51, 2002, pp. 383-392.

[12] R.-J. Hwang and F.-F. Su. "A new efficient authentication protocol for mobile networks". *Computer Standards & Interfaces*, vol. 28, 2005, pp. 241-252.

[13] K. Imamoto and K. Sakurai. "A design of Diffie-Hellman based key exchange using one-time ID in pre-shared key model". *Proceedings of the 18th International Conference on Advanced Information Networking and Applications (AINA)*, Fukuoka, Japan, pp. 327-332, March 2004.

[14] K. Imamoto and K. Sakurai. "Design and Analysis of Diffie-Hellman-Based Key Exchange Using One-time ID by SVO Logic". *Electronic Notes in Theoretical Computer Science*, vol. 135, 2005, pp. 79-94.

[15] B. Kaliski. "Unknown Key Share Attacks and the MQV Protocol". *ACM Transactions on Information and System Security*, vol. 4, no. 3, 2001, pp. 275-288.

[16] P. Keeratiwintakorn and P. Krishnamurthy. "An energy efficient security protocol for IEEE 802.11 WLANs". *Pervasive and Mobile Computing*, vol. 2, 2006, pp. 204-231.

[17] C. Kudla. *Modelling the security of two-party key agreement protocols*. http://www.cs.bris.ac.uk/Research/CryptographySecurity/Workshop/Slides/Kudla .ppt, November 2004.

[18] T. Kyntaja. "A Logic of Authentication by Burrows, Abadi and Needham". http://www.tml.tkk.fi/Opinnot/Tik-110.501/1995/ban.html.

[19] G. Horn, K. M. Martin and C. J. Mitchell. "Authentication protocols for mobile network environment value-added services". *Vehicular Technology, IEEE Transactions on*, vol. 51, 2002, pp. 383-392.

[20] L. Harn, M. Mehta and W.-J. Hsin. "Integrating Diffie-Hellman Key Exchange into the Digital Signature Algorithm (DSA)". *IEEE Communications Letters*, vol. 8, 2004, pp. 198-200.

[21] L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone. "An Efficient Protocol for Authenticated Key Agreement". *Design, Codes and Cryptography*, vol. 28, no. 2, 2003, pp. 119-134.

[22] J. Leiwo. "Cryptographic Protocols". http://www.tml.tkk.fi/Studies/T-110.498/2003summer/Slides/lecture10.pdf, June 2003.

[23] H. Li and M. Singhal. "A Key Establishment Protocol for Bluetooth Scatternets". *Proceedings of the Third International Workshop on Mobile Distributed Computing (ICDCSW'05)*, Washington, DC, USA, pp. 610-616, 2005.

[24] P. Malý. "Elliptic Curves Cryptography (ECC)". http://pepek.ic.cz/data/ECC.pdf.

[25] S.-L. Ng and C. Mitchell. "Comments on mutual authentication and key exchange protocols for low power wireless communications". *Communications Letters, IEEE*, vol. 8, 2004, pp. 262-263.

[26] R. C.-W. Phan. "Fixing the Integrated Diffie-Hellman-DSA Key Exchange Protocol". *IEEE Communications Letters*, vol. 9, 2005, pp. 570-572.

[27] R. C.-W. Phan. "Cryptanalysis of two password-based authentication schemes using smart cards". *Computers & Security*, vol. 25, 2006, pp. 52-54.

[28] C. Popescu. "A Secure Authenticated Key Agreement Protocol". *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference*, Dubrovnik, Croatia, pp. 783-786, May 2004.

[29] RSA Laboratories. RSA Laboratories' "Frequently Asked Questions About Today's Cryptography" Version 4.1. http://www.rsasecurity.com/rsalabs, 2000.

[30] RSA Security. What is Diffie-Hellman. http://www.rsasecurity.com/rsalabs/node.asp?id=2248.

[31] D. Salomon. "*Data Privacy and Security*". Springer Professional Computing, 2003.

[32] B. Song and K. Kim. "Two-Pass Authenticated Key Agreement Protocol with Key Confirmation". *Proceedings of Indocrypt 2000*, Calcutta, India, pp. 237-249, December 2000.

[33] M. A. Strangio. "Efficient Diffie-Hellmann two-party key agreement protocols based on elliptic curves". *2005 ACM Symposium on Applied Computing - Proceedings of the 2005 ACM symposium on Applied computing*, Santa Fe, New Mexico, pp. 324-331, March 2005.

[34] A.-F. Sui, L.C.K. Hui, S.M. Yiu, K.P. Chow, W.W. Tsang, C.F. Chong, K.H. Pun, H.W. Chan. "An improved authenticated key agreement protocol with perfect forward secrecy for wireless mobile communication". *IEEE Wireless Communications and Networking Conference*, New Orleans, Louisiana, pp. 2088-2093, March 2005.

[35] H.-M. Sun, B.-T. Hsieh and S.-M. Tseng. "Cryptanalysis of Aydos et al.'s ECC-based wireless Authentication Protocol", *Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04)*, Taipei, Taiwan, pp. 565-568, March 2004.

[36] C. H. Tan. "BAN logic" - lecture notes in IMT4101 Network Security. http://www.hig.no/imt/file.php?id=1836.

[37] T.-Y. Wu, T.-S. Tsai and H.-C. Chao. "The implication of the security key exchange during mobile IPv6 smooth handoff." *The 7th International Conference on Advanced Communication Technology*, Phoenix Park, Korea, pp. 965-970, February 2005.

[38] Y.-M. Tseng. Efficient authenticated key agreement protocols resistant to a denial-of-service attack. *International Journal of Network Management*, vol. 15, 2005, pp. 193-202.

[39] M. Volkmer and S. Wallner. "A Key Establishment IP-Core for Ubiquitous Computing". *Proceedings of the 16th International Workshop on Database and Expert Systems Applications*, Copenhagen, Denmark, pp. 241-245, June 2005.

[40] H.-A. Wen, C.-L. Lin and T. Hwang. "Provably secure authenticated key exchange protocols for low power computing clients". *Computers & Security*, vol. 25, 2006, pp. 106-113.

[41] Wikipedia - Avalanche effect. http://en.wikipedia.org/wiki/Avalanche_effect.

[42] Wikipedia - "Cryptanalysis". http://en.wikipedia.org/wiki/Cryptanalysis.

[43] Wikipedia - "Cryptographic hash functions".
http://en.wikipedia.org/wiki/Cryptographic_hash_function.

[44] Wikipedia - Diffie-Hellman.
http://en.wikipedia.org/wiki/Diffie-Hellman.

[45] Wikipedia - Discrete Logarithm Problem.
http://en.wikipedia.org/wiki/Discrete_logarithm_problem.

[46] Wikipedia - Elliptic Curve Cryptography.
http://en.wikipedia.org/wiki/Elliptic_curve_cryptography.

[47] Wikipedia - "Key-agreement protocol".
http://en.wikipedia.org/wiki/Key-agreement_protocol.

[48] Wikipedia - "Replay attack".
http://en.wikipedia.org/wiki/Replay_attack.

[49] A. Wool. "Lightweight Key Management for IEEE 802.11 Wireless LANs with Key Refresh and Host Revocation". *Wireless Networks*, vol. 11, 2005, pp. 677-686.

[50] E.-J. Yoon and K.-Y. Yoo. "New Efficient Simple Authenticated Key Agreement Protocol". *Proceedings of Lecture Notes in Computer Science*, vol. 3595, pp. 945-954, 2005.

[51] C. Zhuo, H. Fan, and H. Liang. "A New Authentication and Key Exchange Protocol in WLAN". *International Conference on Information Technology: Coding and Computing (ITCC'05)*, Los Alamitos, CA, USA, pp. 552-556, 2005.