A Pattern Language for Key Management

Sami Lehtonen
VTT Information Technology
E−mail: Sami.Lehtonen@vtt.fi

Juha Pärssinen
VTT Information Technology
E−mail: Juha.Parssinen@vtt.fi

Abstract
Many services in a distributed public network − like the Internet − require secure communications. Security in communications consists of integrity, authenticity, confidentiality and non−repudiability. These aims can be achieved with cryptography. Key management plays a fundamental role in secure communications, as it is the basis of all cryptographic functions.

This paper describes a pattern language for key management. Eleven patterns are described: *Alice & Friends*, *There is somebody eavesdropping*, *The Real Thing*, *Signed Envelope*, *Face−to−Face*, *Address Book*, *Sealed Envelope*, *Sealed and Signed Envelope*, *Seal Ring Engraver*, *Key in the Pocket* and *The Forged Seal Ring*. These patterns are designed to answer basic key management requirements in respect of secure communications.

1. Introduction

The aim of this paper is to identify the basic problem domain of a key management system. This information should make it easier to understand existing implementations. Each single process or communication phase has been written as a pattern. These patterns cannot be used alone as a basis for an implementation. However, most cryptographic algorithms are available off−the−shelf and this document can serve as a checklist when choosing algorithms. The reader is assumed to understand the basics of asymmetric cryptography (public key encryption) and the idea of symmetric cryptography. This information can be obtained from [Overview].

Designing basic key management system started with identifying typical components of a key management system. Each component could be found in almost any key management system. Major design issues in these components were written as patterns. Typical components of a key management system are described in the next section.

2. Typical Components of Key Management systems

A keying relationship is the state wherein communicating entities share common data (keying material) to facilitate cryptographic techniques. This data may include public or private keys, initialization values, and additional non−secret parameters.

This fact leads to a definition of key management. Key management is the set of techniques and procedures supporting the establishment and maintenance of keying relationships between authorized parties. [Menezes96]

A basic key management system should be able to generate, store and exchange keys as well as to verify their authenticity. Additionally, it should be able to identify communicating parties. Usually, secure communications use symmetric keys for pure encryption (confidentiality and integrity) and asymmetric keys for symmetric key exchange and digital signatures.

The first component of a typical key management system is a key database, which answers the need of storing the keys. There exist public keys and correspondent private keys as well as session keys. Different types of keys require different kinds of storing.

A reliable session key source must be implemented to ensure effective encryption in data transfers. In fact, the whole problem domain culminates in the creation of a session key. If good session keys can't be created there is no use even for a perfect key exchange method. Key generator was identified as the second component of a key management system.

A secure connection requires that both participants have the same session key. This session key must be created by one participant and delivered to the other securely. Key exchange is the third component.

The last but not least component is key authentication component. The participants need to know whether they got the key from a right person. This eliminates the possibility of a man−in−the−middle attack.

3. The Pattern Language

Relationships between patterns in this pattern language are described in the figure 1.

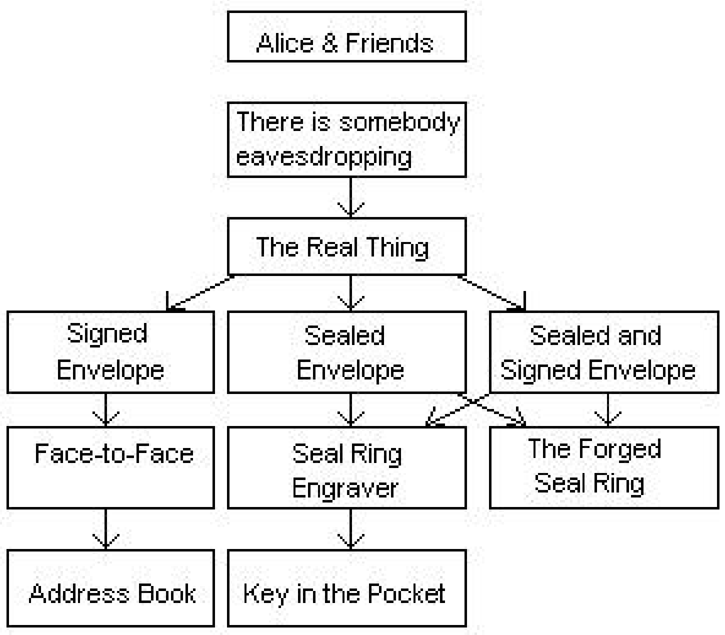

Figure 1. Relationships between patterns.

Alice & Friends is a separate pattern that describes a naming scheme used in this paper. *The Real Thing* solves the major design issue in key generator component. Different key exchange methods are described in *Signed Envelope*, *Sealed Envelope* and *Sealed and Signed Envelope*. These patterns concern both key exchange and authentication components.

Both *Face−to−Face* and *Seal Ring Engraver* make it possible to create and deliver asymmetric key pairs and authenticate them, thus solving design issues with the authentication component. *The Forged Seal Ring* is an essential addition for ensuring security when using *Sealed Envelope* or *Sealed and Signed Envelope*.

Design issues concerning key database component are solved in *Address Book* and *Key in the Pocket* patterns.

4. Patterns

4.1 Common Key Management Patterns

4.1.1 Alice & Friends

Context
Two persons are establishing a secure connection. Some other entities may also participate.

Problem
How to identify roles and name them in a standard way?

Forces
- It is easier to understand and remember communication between humans than some other entities.
- When referring to participants several times their names should be short and easy to remember.

Solution
To ease up terminology we use the following dramatis personae.

Alice	First participant. Client A. Also known as initiator.
Bob	Second participant. Client B, Also known as responder.
Eve	Eavesdropper.
Mallory	Malicious active attacker.
Trent	Trusted arbitrator. Cast in a role of a certification authority (CA).

Table 1. Dramatis Personae

Schneier introduces this dramatis personae in [Schneier96]. Also other characters exist but only these are needed in this paper.

Resulting Context
It is easy to refer to Alice as "she" and to Bob as "he" without a risk of misunderstanding. Capital letters indicate the role in communications. This naming system is widely used in literature and technical papers concerning cryptography.

Related Patterns
This pattern is a background for all other patterns. Characters introduced here are used throughout the rest of this paper.

4.1.2 There is somebody eavesdropping.

Context
Alice wants to communicate with Bob securely. Information should not be revealed to Eve.

Problem
How to prevent data from being eavesdropped?

Forces
- Encrypted data is readable only to those who know the right keying material.
- Encrypting algorithms usually don’t remain secret.
- Symmetric cryptography is more effective than asymmetric cryptography in the means of key length.
- Symmetric algorithms use the same key for encrypting and decrypting.

Solution
Alice and Bob use a symmetric algorithm for encrypting data. Symmetric cryptography does not require as much processing power as asymmetric methods. Some keying material is used to encrypt all data before transmission. The data is decrypted at the other end after receival.

Resulting Context
Data is secured during the transfer. Eve can't directly read the data.

Running Example
When browsing the net, Alice can be quite sure there is somebody eavesdropping somewhere between her and the server. Banks, for instance, use TLS (Transport Layer Security), to secure their web applications. TLS is formerly known as SSL (Secure Sockets Layer).

Other Examples
There are several secure symmetric algorithms (3DES, blowfish and IDEA just to name few).

Related Patterns
Alice and Bob decided to use symmetric encryption to secure data during transfer. This introduces a new problem; how to create a good key. This new problem is solved in *The Real Thing*.

4.1.3 The Real Thing

Context
Secure communications between Alice and Bob are possible with encryption. Alice and Bob have decided to use symmetric encryption.

Problem
How to generate good symmetric encryption keys securely?

Forces
- A weak key generation algorithm is easier to cryptanalyze than strong encryption algorithms.
- A strong key generation algorithm requires truly random number generator or at least a cryptographically secure pseudo−random number generator.
- The seeding material for a pseudo−random number generator should be as long as (or longer than) the session key needed.
- A pseudo−random number generator algorithm generates always the same output with the same seeding material.
- Algorithms don't remain secret.
- The seed generator (or even a part of it) should not be available to others.
- If Eve knows the key generation algorithm and can limit the possible random seeds to a reasonably small group (furthermore, the possible session keys) she can decrypt the data being transferred.
- Pseudo−random number generator can be secured with use of a one−way hash function.
- Basically any combination of 1's and 0's could be used as an encryption key. All combinations should be equally likely.
- Some encryption algorithms have weak keys.
- A session key should be long enough. A longer key requires more processing power to be cracked with brute force.
- If a session key is too long it requires too much processing power to be used.

Solution
Alice gathers enough seeding material from a reliable source. She then generates a 128−bit key with a one−way hash function from that seeding material. Alice compares the generated key against a list of known weak keys of the encryption algorithm to be used. If the key is known to be weak, she generates another one.

Running Example
TLS specification contains an example that uses keystroke timing values. These values taken from a PC compatible 18,2 Hz timer provide 1 or 2 secure bits each, even though the total size of the counter is 16 bits or more. To seed a 128−bit pseudo−random number generator, one would thus require approximately 100 such timer values. [TLS99]

Other Examples
Time between falls of a toddler learning to walk is truly random. Of course, this is not very easy to measure. An excellent random source is a piece of plutonium. The time between radiating gamma particles is random, allthough it slowly gets longer while your lifetime may get shorter.

Westphal Electronic [Westphal] offers a computer attachable thermal noise based true random bit generator, which is capable of generating 800 000 random bits per second.

Resulting Context
Alice and Bob are able to create strong session keys for symmetric encryption. After all, the session key is the real thing of cryptography.

Related Patterns
This pattern solves the problem introduced in *There is somebody eavesdropping.* Now Alice and Bob have a problem with exchanging the session key that Alice just created. This problem can be solved in different ways. The first solution is presented in *Signed Envelope*, second in *Sealed Envelope* and third in *Sealed and Signed Envelope*.

4.2 Key management without Trent's help

4.2.1 Signed Envelope

Context
Alice and Bob are going to encrypt data to be transferred between them. Alice has created a session key. They don't have certificates because certificates are expensive.

Problem
How to deliver the session key?

Forces
- The symmetric session key could not be just sent over the Internet as plaintext; anyone eavesdropping the connection could use the key to decrypt the data.
- Asymmetric encryption algorithms are used for transferring session keys.
- An asymmetric key pair consists of a private key and a public key. Everything encrypted with the private key can only be decrypted with the public key and vice versa.
- Many suitable public key encryption based key exchange algorithms exist.

Solution
Alice encrypts the session key with Bob's public key, adds a digital signature with her private key and sends it to Bob. Bob can verify the digital signature with Alice's public key and decrypt the session key with his private key.

Resulting Context
Alice can be sure that she is communicating with Bob because only Bob can decrypt the session key. Bob can also be sure he got the session key from Alice. Both of them have a digital verification of each other's identities.

Related Patterns
This pattern solves the problem of key delivery, which was introduced in *The Real Thing*. Other solutions are presented in *Sealed Envelope* and *Sealed and Signed Envelope*. However, using this pattern leads to another problem. Alice and Bob need to exchange their public keys. This problem is solved in *Face−to−Face*.

4.2.2 Face−to−Face

Context
Alice and Bob are going to encrypt data to be transferred between them. They have decided to use public key encryption for session key transfer.

Problem
How to exchange public keys?

Forces
- Alice and Bob must create their key pair beforehand.
- Alice must be sure she got the public key from Bob and vice versa.
- Public keys are usually transferred out−of−band.
- If public keys are sent over a network, Mallory could send her public key to Bob and pretend to be Alice. Bob could not know if it is Alice or not. Other possible threat is a man−in−the−middle attack. Mallory listens to a communication between Alice and Bob, and when they are exchanging their public keys Mallory hijacks those packets and replaces them with another ones. After this she simply decrypts the session key that Alice sends and encrypts it with another key and forwards it to Bob.
- Public keys could be in any format when traded, but it is convenient to hand them over in a digital format.

Solution
Both Alice and Bob write their public keys on a digital media (a floppy disk, for example). They meet each other face−to−face and hand over their public keys.

Resulting Context
Alice and Bob can identify each other when they are face−to−face. In other words, they can be 100% sure they got the public key from the real owner of that particular key.

Example
This is the most secure way of exchanging public keys. For example, people who use PGP (Pretty Good Privacy) arrange public key exchange meetings.

Related Patterns
This pattern solves the problem of exchanging public keys that *Sealed Envelope* rises. Both Alice and Bob need a place to store other people’s public keys. This problem is solved in *Address Book*.

4.2.3 Address Book

Context
Alice and Bob have created asymmetric key pairs and swapped their public keys. They may have several other public keys from other people.

Problem
How to assure integrity and authenticity of public keys without loosing accessibility?

Forces
- The private key must be available only to Alice.
- Encrypting all keys will affect accessibility.
- Integrity of all keys must be ensured.
- Session keys should not be kept outside the encrypting device or software.

Solution
Alice saves her private key on a digital media (a floppy disk, perhaps) and puts it in a place that is not accessible to others. Her public key and other people's public keys don't need encryption but their integrity must be assured. Alice adds a digital signature, which is a calculation of her private key and the public keys in her database, and stores a backup copy of the key database in a safe place. If someone modifies Alice's key database the digital signature won't match anymore revealing the attempt to attack. If this happens, Alice can always fetch the backup copy.

Resulting Context
Alice doesn't need an additional password for her key database. Alice can check the integrity of stored public keys by verifying the digital signature. If the database is modified and/or corrupted she can restore the backup copy.

At this time, Alice and Bob are ready to start communicating with encrypted connections.

Related Patterns
This pattern solves the public key storing problem that arose with the use of *Face−to−Face*.

4.3 Key management with Trent

4.3.1 Sealed Envelope

Context
Alice and Bob are going to encrypt data to be transferred between them. Alice has created a session key. Alice wants to remain anonymous. Bob doesn't care who Alice is.

Problem
How to deliver the session key?

Forces
- There exists many trustworthy third parties.
- Certificate is a digital document signed by a third trusted party that contains a public key and other information about the owner.
- Because of the digital signature public keys can be transferred over the network in a plaintext format without the risk of being modified.

Solution
Bob sends his certificate (public key is certified) to Alice. Alice encrypts the session key with Bob's public key and sends it to Bob. Bob decrypts the session key with his private key.

Resulting Context
Bob gets the session key from Alice. Alice can be sure that she is communicating with Bob because only he can decrypt the session key. Alice never reveals her identity to Bob.

Running Example
Transport Layer Security (TLS) protocol is usually used in this way, which is the most common way to secure HTTP connections. The TLS negotiation with server certificate is described in Figure 2.

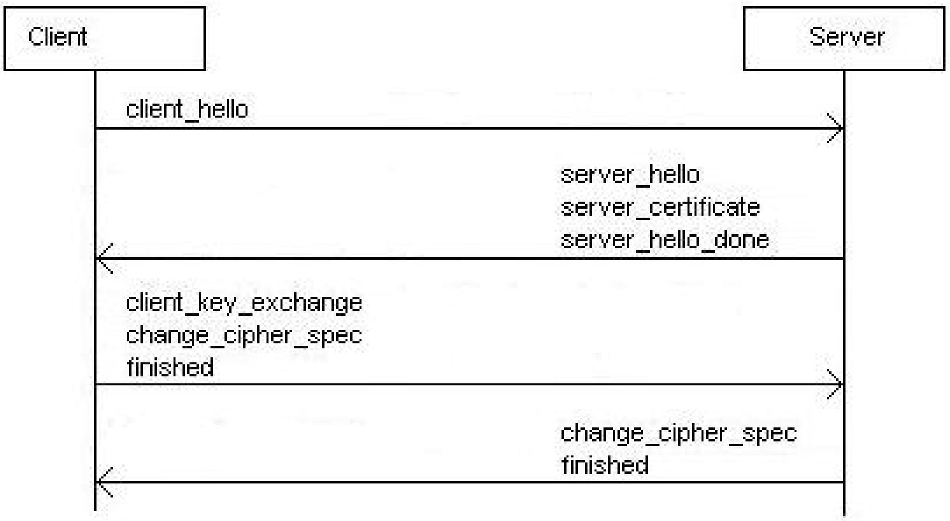

Figure 2. TLS negotiation with server certificate

The client proposes possible cipher suites in the client_hello message. Server chooses the cipher suite and tells it to the client in the server_hello message. Server sends it’s certificate in the server_certificate message and indicates the end of hello phase with the server_hello_done message.

The client generates the session key and encrypts it with the server’s public key enclosed in server_certificate. The client then sends change_cipher_suite, which indicates that the client is ready to start communication with data encryption, and finished messages. The server responds with the same.

Related Patterns
This pattern solves the problem of key delivery, which was introduced in *The Real Thing*. Other solutions are presented in *Signed Envelope* and *Sealed and Signed Envelope*. However, using this pattern leads to another problem. Bob needs a certificate from Trent. This problem is solved in *Seal Ring Engraver*. Alice needs to know whether Bob’s certificate is valid or not. This is solved in *The Forged Seal Ring*.

4.3.2 Sealed and Signed Envelope

Context
Alice and Bob are going to encrypt data to be transferred between them. Alice has created a session key. Both of them require authentication.

Problem
How to deliver the session key?

Forces
- Certificate is a digital document signed by a third trusted party that contains a public key and other information about the owner.
- Because of the digital signature certificates can be transferred over the network in a plaintext format without the risk of being modified.

Solution
Alice and Bob exchange their certificates (public keys are certified). Alice encrypts the session key with Bob’s public key, adds a digital signature with her private key and sends it to Bob. Bob can verify the digital signature with Alice’s public key and decrypt the session key with his private key.

Resulting Context
Bob gets the session key from Alice. Bob can also be sure he got the session key from Alice by verifying the digital signature with her public key. Alice can be sure that she is communicating with Bob because only Bob can decrypt the session key. Both of them have a digital verification of each other's identities.

Running Example
TLS (Transport Layer Security) protocol can optionally use client certificate for user authentication (web banking, for example). The TLS negotiation with both server and client certificate is described in Figure 3.

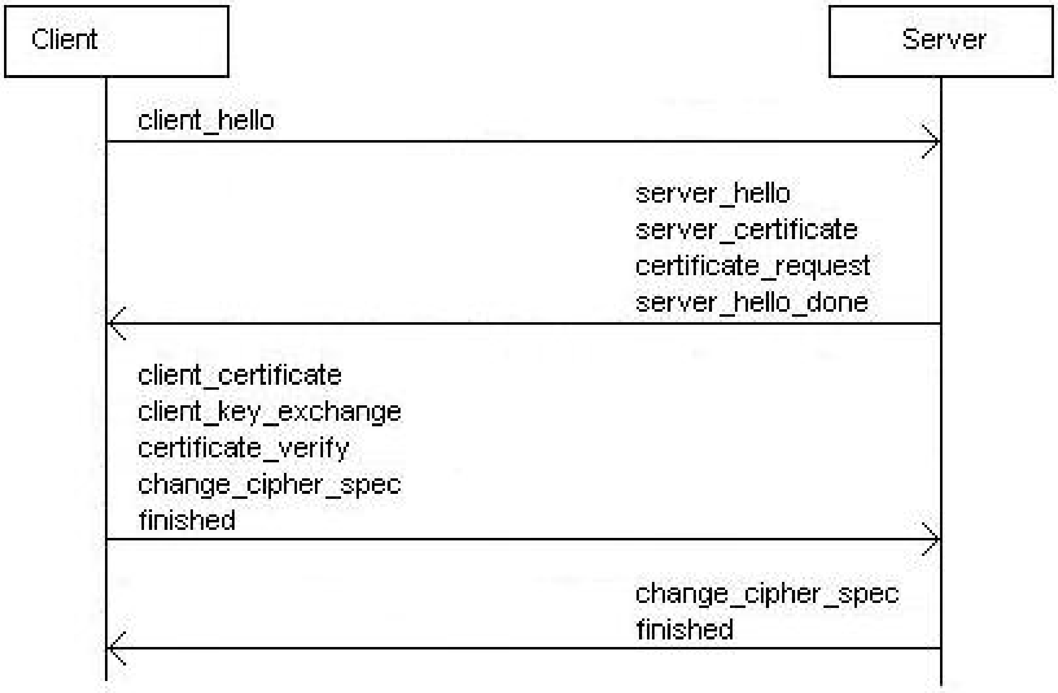

Figure 3. TLS negotiation with both client and server certificate

The client proposes possible cipher suites in the client_hello message. Server chooses the cipher suite and tells it to the client in the server_hello message. Server sends its certificate in the server_certificate message and requires the client to send its certificate with certificate_required message. Then the server indicates the end of hello phase with the server_hello_done message.

The client responds with its certificate. Then the client generates the session key and encrypts it with the server's public key enclosed in server_certificate. certificate_verify is a digital signature hashed from the previous messages. The client then sends change_cipher_suite, which indicates that the client is ready to start communication with data encryption, and finished messages. The server responds with the same.

Related Patterns
This pattern solves the problem of key delivery, which was introduced in *The Real Thing*. Other solutions are presented in *Sealed Envelope* and *Signed Envelope*. However, using this pattern leads to two new problems. Alice and Bob need to obtain certificates from Trent. This problem is solved in *Seal Ring Engraver*. Alice needs to know whether Bob's certificate is valid and vice versa. This problem is solved in *The Forged Seal Ring*.

4.3.3 Seal Ring Engraver

Context
Alice wants to exchange keys with certificates.

Problem
How to obtain a certified key pair?

Forces
- Sometimes Alice and Bob are not able to exchange their public keys Face−to−Face.
- Both Alice and Bob know a trusted person, Trent.
- Trent can calculate an asymmetric key pair, certify the public key and deliver it to Alice.
- A certificate signed by Trent can be used with anybody who knows Trent.

Solution
Trent calculates an asymmetric key pair and creates a certificate for the public key. He adds a digital signature to the certificate. Trent delivers the certificate and the corresponding private key to Alice.

Resulting Context
The certificate and the certified key pair has to be stored somewhere.

Running Example
There are companies that offer certificates for web servers. For example, [Verisign] can create key pairs and certify them.

Related Patterns
This pattern solves the problem of obtaining certificates that was introduced in both *Sealed Envelope* and *Sealed and Signed Envelope*. Having a certificate rises a problem of a secure storage for it. This problem will be solved in *Key in the Pocket* pattern.

4.3.4 Key in the Pocket

Context
Alice has her certified public key and a private key.

Problem
How to store a key pair and a certificate securely?

Forces
- The authenticity and integrity of stored keys must be ensured without loosing accessibility.
- The private key must be available only to Alice.

Solution
Alice’s key pair and certificate are stored on a smart card, which handles the use of these. Even Alice herself cannot read the private key.

Resulting Context
Alice has her key pair and certificate in a place she can easily access. A psychological benefit of smart card is that people understand physical keys, what they signify and how to protect them. Putting a cryptographic key in the same physical form makes storing and protecting that key more intuitive [Schneier96].

Running Example
The Electronic ID Card by the Finnish Population Register Centre contains an asymmetric key pair and a certificate. This card can be used as authentication when using web services provided by the Finnish authorities.

Related Patterns
This pattern solves the problem of storing a certificate, which was introduced in *Seal Ring Engraver*.

4.3.5 The Forged Seal Ring

Context
It is possible that a certificate becomes insecure (the private key is compromised, for example). However, when exchanging the session key with certificate(s) Alice needs to know whether Bob's certificate is valid or not.

Problem
How Alice can be sure about the validity of Bob's certificate?

Forces
- Alice has Trent's public key. This key can be used to verify the digital signature in a certificate.
- The verification of the digital signature ensures that the certificate was − at one particular moment in the past − valid.

Solution
If Alice notices that her private key has been compromised, she informs Trent about it. Trent maintains a list of invalid certificates that have not been expired. This list is called a Certificate Revocation List (CRL). The CRL has it's own expiration time, which is shorter than certificate's.

Alice gets the CRL from Trent and stores it. When the CRL expires she gets a new CRL. When Alice gets a certificate from Bob she looks at the CRL if the certificate has been revoked. If Alice and Bob communicate with each other again and Alice has not obtained a new CRL there is no need to look again.

Resulting Context
Alice and Bob can be sure about the validity of each other's certificates.

Running Example
Certificate providers, such as [Verisign], have their own CRLs.

Related Patterns
This pattern solves the problem introduced in *Sealed and Signed Envelope* and *Sealed Envelope*.

Appendix A

First version of this story was told July 6th 2001 at EuroPloP2001 on the Focus Group for Pattern Sequences. This story presents one possible sequence of use of patterns in our language.

Let the story begin...

Alice wants to communicate with Bob but *there is somebody eavesdropping.* They want to keep their secrets and not reveal them to Eve the eavesdropper. Alice and Bob decide to use a symmetric algorithm to encrypt data they send to each other. One session key is used to encrypt all data before transmission and the data is decrypted using the same key at the other end after receival.

Next, Alice and Bob have to have a strong enough session key for symmetric encryption. After all, the session key is the *real thing* of cryptography.

Alice generates a key and checks whether it is weak or not. If the key is known to be weak, she simply generates another one. This session key is used only in one particular communication session.

There are at least three different ways to reliably exchange session keys between Alice and Bob in communication sessions. This time we will only tell you one. The other two are in our paper.

At this point in our story, Alice and Bob are exchanging a session key but they don't have any certificates because they are expensive. They decide to use asymmetric cryptography to create a *signed envelope, which* contains the session key. Alice sends this envelope to Bob through an insecure channel.

To make the *signed envelope*, Alice encrypts the session key with Bob's public key and adds a digital signature using her private key. Alice can be sure that she is communicating with Bob because only he can open the *signed envelope* and decrypt the session key. Furthermore, the digital signature tells Bob that it's from Alice.

To use *signed envelope*, Alice and Bob need to have each other's public keys. Both Alice and Bob write their public keys on a digital media (a floppy disk, for example). They meet each other *face−to−face*, identify each other and hand over their public keys.

Both Alice and Bob store other people's public keys in their *address book*s.

Public keys don't need encryption but their integrity must be ensured. Alice adds a digital signature, and stores a backup copy of the key database in a safe place. If someone modifies Alice's key database the digital signature won't match anymore revealing the attempt to attack. If this happens, Alice can always fetch the backup copy.

References

[Menezes96] A. Menezes, P. van Oorschot, S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996

[Overview] http://www.itsc.state.md.us/info/internetsecurity/Crypto/CryptoOverview.htm

[Schneier96] Bruce Schneier, "Applied Cryptography", Wiley & Sons, 1996

[TLS99] T. Dierks, C. Allen, "The TLS Protocol Version 1.0", RFC2246, 1999

[Verisign] http://www.verisign.com

[Westpexihal] http://www.westphal−electronic.de