



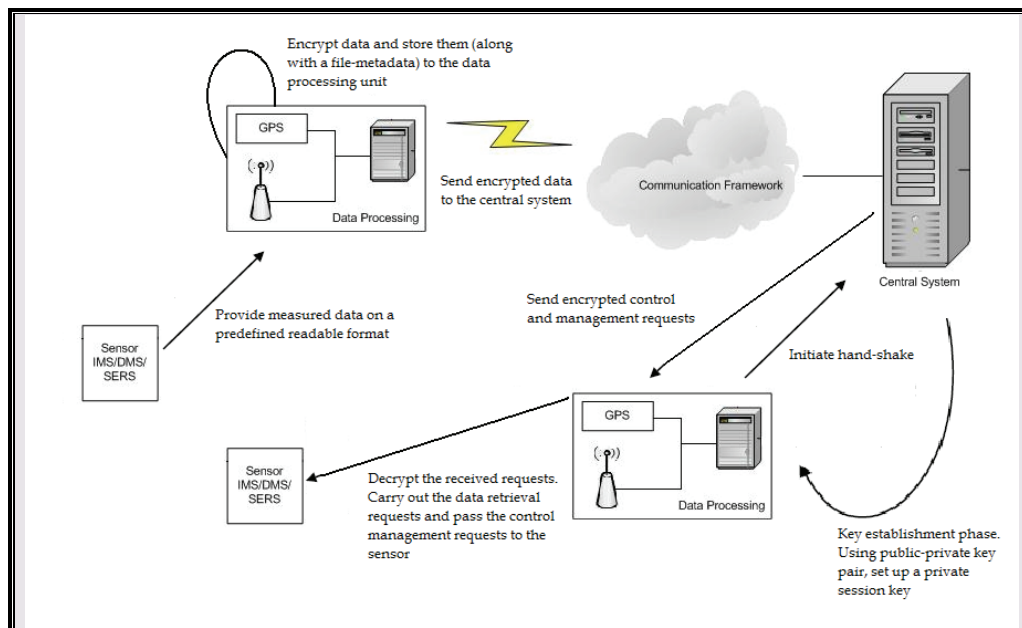
LOTUS

D200.2

Evaluation of currently known techniques and open problems in establishing a security infrastructure

TASSOS DIMITRIOU

ATHENS INFORMATION TECHNOLOGY



Funding has been received from the European Commission's
Seventh Framework Programme (2007-2013)
Dissemination Level:
Public

Project No. 217925
LOTUS
Localisation of Threat Substances in Urban Society

D200.2

Evaluation of currently known techniques and open problems in establishing a security infrastructure

Due date of deliverable:	September 30, 2009
Actual submission date:	October 8, 2009
Updated:	January 22, 2010
Version:	2
LOTUS report no:	LOTUS TR-09-008
FOI reg. no.:	FOI-2009-486
Author(s):	Tassis Dimitriou (AIT)
Number of pages:	71
Start date of project:	January 1, 2009
Duration:	3 years

Content

- Content..... 3
- 1 Introduction..... 4
 - 1.1 Purpose and scope 4
 - 1.2 Document Background..... 5
- 2 Setting up a secure communication channel..... 6
 - 2.1 General..... 7
 - 2.2 Symmetric and Asymmetric cryptography..... 10
 - 2.3 Symmetric key cryptographic techniques..... 11
 - 2.3.1 Encryption/Decryption..... 11
 - 2.3.2 Hash Functions 12
 - 2.3.3 Message Authentication Codes..... 13
 - 2.4 Public key cryptographic techniques 15
 - 2.4.1 Encryption/Decryption..... 15
 - 2.4.2 Digital Signatures..... 15
 - 2.5 Key establishment..... 17
 - 2.5.1 Key establishment using preconfigured public/private keys..... 18
 - Key agreement schemes 19
 - Key transport schemes..... 26
 - 2.5.2 Key establishment using preconfigured symmetric keys..... 30
 - 2.5.3 Password based key establishment 33
 - 2.5.4 Abstracting away the key establishment process..... 36
 - 2.6 LOTUS requirements for establishing a secure channel 40
 - 2.6.1 Threat modelling 41
 - 2.6.2 Issues to be addressed 48
- 3 Secure Storage..... 52
 - 3.1 Introduction..... 52
 - 3.2 Challenges and guidelines 52
 - 3.3 Existing solutions 54
 - 3.4 LOTUS requirements for secure storage..... 57
 - 3.4.1 Threat Modelling..... 58
 - 3.4.2 Issues to be addressed 62
- 4 LOTUS Secure Communication Scheme (LSCS) 63
 - 4.1 LSCS System Architecture..... 63
 - 4.2 High-Level System Design 64
 - 4.2.1 Components/Modules 64
 - 4.2.2 LSCS Framework Components 65
 - 4.2.3 LSCS Protocol Components 66
- 5 Conclusions..... 68
- 6 References..... 69

1 Introduction

1.1 Purpose and scope

This document is to provide a general description of the techniques to be used in order to establish a secure communication channel between the Central System and a mobile router in LOTUS as well as managing and storing encrypted data locally in a mobile router. This set of techniques will be used in the LOTUS Secure Communication Scheme (LSCS) report to guide the development process of a secure communication framework for the LOTUS project. The LSCS report will be continuously updated and will be used as the steering document for requirements specification, system design and development of the framework.

The document begins by offering a guided tour on cryptographic primitives and protocols (Sections 2.1 to 2.4) that will be helpful in understanding the concepts to be developed in subsequent chapters. Section 2.5 is the heart of this report describing how key establishment can be achieved in a way that is more appropriate to the LOTUS communication scenario. Key establishment is the process by which two entities (in our case a mobile router and the central system) can agree on a secret cryptographic key so that they start communicating securely with each other. Section 2.5.4 is an attempt to abstract the details of this process in a way that is more applicable to the LOTUS case. Two generic protocols are presented, KA-R and KA-IR, which differ only on the assumption of public keys available for key establishment. These protocols will be instantiated during the development of LSCS. Chapter 2 concludes with the LOTUS requirements for establishing a secure channel, a discussion of the threat model assumed and some more issues to be addressed during development.

As data in LOTUS must always be kept encrypted in a mobile router, Chapter 3 discusses the options and challenges of doing so. It considers existing solutions (Section 3.3), the LOTUS requirements for secure storage (Section 3.4), and concludes with a threat model and some issues to be addressed.

Finally, Chapter 4 offers a snapshot of the LOTUS Secure Communication Scheme (LSCS). It presents the overall architecture of the system and a high level discussion of the components that make up LSCS. The LSCS framework will be described in more detail in a subsequent *confidential* document.

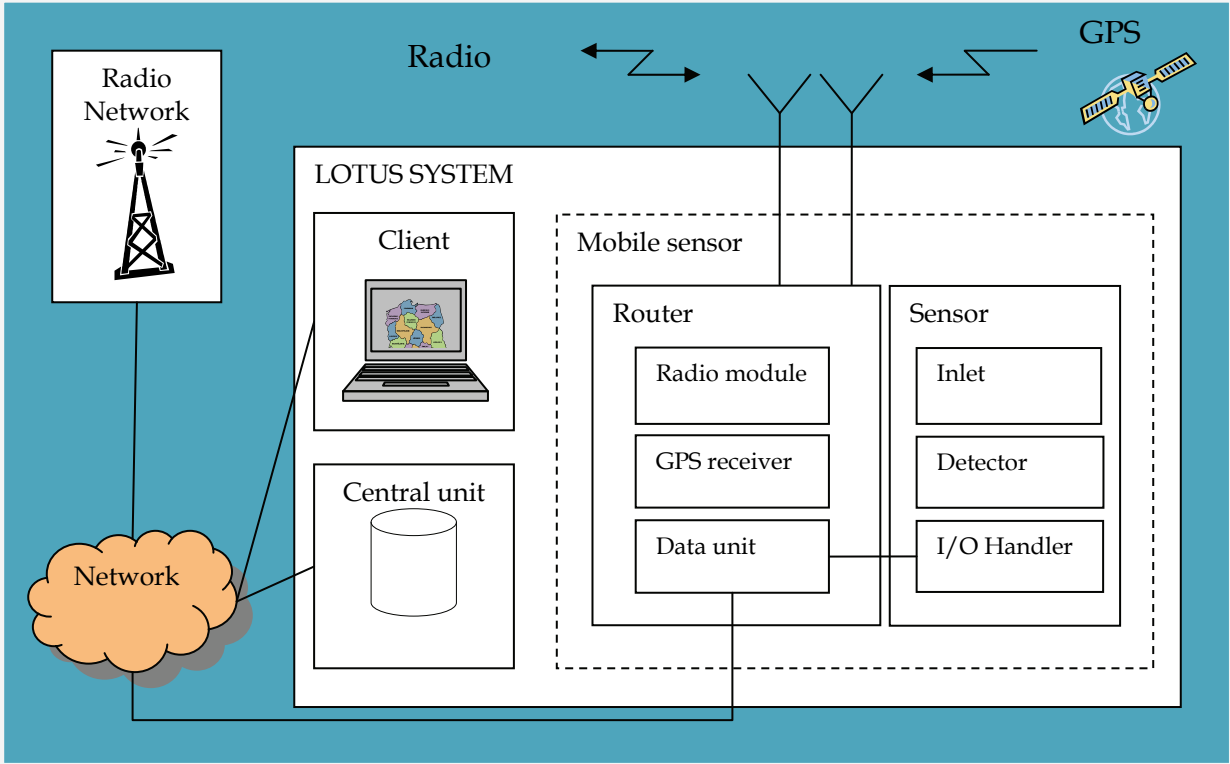
1.2 Document Background

This document is based on information from

- Project LOTUS description: European Commission, 7th Framework Program, Theme 10, Security: LOTUS Localisation of Threat Substances in Urban Society. FP7-SEC-2007-1 LOTUS n° 217925, Description of Work, November 2008.
- Fredrik Robertsson: *System requirement specification - D200.1 Deliverable*. LOTUS Technical Report LOTUS-TR-09-005, (Confidential UE/Restraint UE), April 2009.

2 Setting up a secure communication channel

In what follows, we assume that two communicating entities wish to communicate with each other securely. In our context “secure” means to provide, among other things, guarantees for confidentiality, integrity and authenticity. The two communicating entities may come under different disguises; however, in the context of the LOTUS project, communication will occur between a mobile router and the central system as illustrated in the following figure.



The mobile router will be responsible for interfacing the sensors with the central system, securely transmit data from the sensors, accept commands from the central system and manage data locally for further processing. In order to establish such security infrastructure, cryptographic keys will have to be established between these two different entities. In particular, keys are needed in order to provide for confidentiality and integrity protection to the communications within the LOTUS networking environment.

In the remaining of this section we will give a brief introduction to the foundations of cryptography and the mechanisms that can be used to achieve the required properties. While this can be no substitute for a more detailed study of the subject, the main terms will be introduced with the goal of enabling the remainder of this section to be understood.

2.1 General

In order to identify the set of requirements for communications to be secure, it is important to define the scope and boundaries of the security solution. The assumptions that are made for this purpose are:

1. The security solution should cover the communications between the mobile router and the central system. *It is outside the scope of this discussion to cover the security requirements of other networks and/or of the platforms and databases that the data will be finally handled by and stored in.*
2. The solution will cover a set of technological security requirements. *It is assumed that data handling practices and privacy policies of legitimate deployments, if enforced, are supervised by some regulatory framework.* For example, in order for the users to take full advantage of the services that can be offered by a legitimate deployment, it is necessary to build some level of trust, *which should not be accomplished using solely technical means*, i. e. the users have to trust that the organizations responsible for the deployments are complying with regulations (e.g. for data handling practices) in order to use the services offered.

The general security requirements that must be covered by this security framework through security protocols and mechanisms are:

- ***Confidentiality and secrecy of collected and communicated data.***
Confidentiality is about ensuring that information is accessible only to those authorized to have access to it, and providing guarantees that the information processed by a system is protected against unauthorized access.

Confidentiality is achieved through encryption of the data so that only those with the correct decryption key can recover it. In cryptographic protocols, confidentiality is essential to ensure that keys and other data are available only as indented.
- ***Integrity of the data that is processed or communicated.*** Data integrity provides protection against unauthorized data modification, insertion, substitution or deletion. It also includes providing the means for assuring *data freshness*. In a communication context where the attacker can be *active*, data integrity refers to the ability of the receiver to detect whether changes have occurred.

Integrity can be achieved through the use of hash functions in combination with encryption, or by the use of a message authentication code to create a separate check value. Data integrity is essential in most cryptographic protocols to protect the value of exchanged data.
- ***Authentication of the originator and recipient of messages.*** Authentication provides the means for verification of network identities, and is a prerequisite for covering the other security requirements. Data Origin Authentication provides evidence of the origin of the received data. It is a fundamental step in

achieving *entity* authentication in protocols as well as establishing keys between two parties.

Several cryptographic mechanisms may be used to provide authentication services. Most commonly, authentication is provided by digital signatures or message authentication codes.

- ***Authorization for accessing data or network resources.*** Authorization entails providing the means to control access and approve actions by authenticated entities. Depending on the scenario and role of each node in the network, the information it communicates may have different security requirements. A non-cryptographic analog of the interaction between authentication and authorization is the examination of an individual's credentials to establish their identity (authentication); upon proving identity, the individual is then provided with the key or password that will allow access to some resource, such as a locked room (authorization).

It is important to note that in many applications, a *combination* of cryptographic services (confidentiality, data integrity, authentication, etc.) is usually desired. Not all mechanisms, however, are cryptographic in nature. For example, physical security may be used to protect the confidentiality of certain types of data, and identification badges or biometric identification devices may be used for entity authentication. However, cryptographic mechanisms consisting of algorithms, keys, and other keying material often provide the most cost-effective means of protecting the security of information. This is particularly true in applications where the information would otherwise be exposed to unauthorized entities.

When properly implemented, some cryptographic algorithms provide *multiple* services. For example:

1. A message authentication code can provide data integrity as well as authentication since the symmetric key is shared between a pair of entities.
2. A digital signature algorithm can provide authentication and data integrity as well as non-repudiation¹.
3. Certain modes of encryption can provide confidentiality, data integrity, and authentication when properly implemented.

However, it is often the case that different algorithms must be employed in order to provide all the desired services as illustrated in the following example of secure information exchange between a pair of Internet entities:

¹ Non-repudiation is the concept of binding an entity to its actions, e.g. a sender of a message cannot later deny having sent the message. This is typically provided using a digital signature scheme. Protocols for authentication and key establishment do not usually make use of this concept, but non-repudiation provides the important properties of integrity and data origin authentication.

Consider the case where some of the exchanged information requires just integrity protection, while other information requires both integrity and confidentiality protection. It is also a requirement that each entity that participates in an information exchange knows the identity of the other entity.

The designers of this example system decide that a Public Key Infrastructure (PKI) needs to be established and that each entity wishing to communicate securely is required to physically authenticate his or her identity at some Registration Authority.

This authentication requires the presentation of proper credentials such as a driver's license, passport, or birth certificate. The authenticated individuals then generate a public key which is incorporated with the user identifier and other information into a digitally signed message for transmission to a Certification Authority (CA). The CA then composes the user's public key certificate by signing the public key of the user and the user's identifier along with other information such as date of creation, expiration date, name of CA, etc. This certificate is returned to the public key owner so that it may be used in conjunction with the private key for entity authentication and key agreement purposes.

In this example, any two entities wishing to communicate must exchange public key certificates containing public keys that are checked by verifying the CA signature on the certificate (using the CA public key which must be known by both entities).

The public key of each entity and each entity's own private key is then used in a key agreement scheme to produce a secret value shared between the two entities. The shared secret may then be used to derive one or more shared symmetric keys. For example, one of the shared keys can be used to encrypt for confidentiality, while another key can be used for integrity and authentication. The receiver of the data protected by the key(s) has assurance that the data came from the other entity indicated by the public key certificate, that the data remains confidential, and that the integrity of the data is preserved.

Alternatively, if confidentiality is not required, integrity protection, entity authentication, and non-repudiation can be attained by establishing a signature key pair and corresponding certificate for each entity. The private signature key of the sender is used to sign the data, and the sender's public signature verification key is used by the receiver to verify the signature. In this case a single algorithm provides all three services.

2.2 Symmetric and Asymmetric cryptography

Cryptographic techniques can be divided into two main categories, known as secret key and public key techniques (or symmetric and asymmetric ones) depending on the nature of the keys used.

In the first category, all cryptographic algorithms use *shared* secret keys. Both the sender and the receiver share the same key, which is used by the sender to encrypt (or integrity protect) a message, and by the receiver to decrypt (or verify the integrity of) a message.

However, in the 1970s, Diffie and Hellman [DH76] came up with the concept of public-key cryptography, in which every entity is associated with a pair of keys: One key is called the *public key*, and can be made widely known, while the other is the *private key*, which must be kept secret and is normally known only to the key owner. The use of these two keys depends on the type of public key algorithm and the service we want to achieve.

- For *public key encryption algorithms*, the public key is used to encrypt the data and the corresponding private key is used to decrypt the data and recover the original message. This means that anyone who knows the user's *authentic* public key can encrypt data so that only the user (using his/her private key) can decrypt.
- In *digital signature algorithms*, the private key is used by the sender of a message to generate a *digital signature* on the message. Anyone with knowledge of the public key can then verify the signature, and thereby the origin and the integrity of the message.

In what follows we analyze in more detail these two basic cryptographic techniques.

2.3 Symmetric key cryptographic techniques

The three main cryptographic techniques that are based on symmetric cryptography are Encryption, Message Authentication Codes and Hash functions.

2.3.1 Encryption/Decryption

Encryption is used to provide confidentiality for data. The data to be protected is called plaintext when in its original form. Encryption transforms the data into ciphertext. Ciphertext can be transformed back into plaintext using decryption.

Two important algorithms for encryption/decryption are AES and Triple DES. Each of these algorithms operates on blocks of data (for example 64 or 128 bits) during an encryption or decryption operation. For this reason they are also called *block ciphers* as opposed to *stream ciphers* that operate at the bit level. A block cipher will take as input a block of plaintext and a secret key and will produce a block of ciphertext. To decrypt the ciphertext, the inverse operation is followed.

A block cipher is a *deterministic* algorithm which means that the same plaintext block will always encrypt to the same ciphertext block whenever the same key is used. If the multiple blocks in a typical message are encrypted separately, an adversary can easily substitute individual blocks, possibly without detection (this is the case for the ECB mode of operation). Furthermore, certain kinds of data patterns in the plaintext, such as repeated blocks, are apparent in the ciphertext.

Cryptographic modes of operation have been defined to alleviate this problem by combining the basic cryptographic algorithm with variable initialization vectors and some sort of feedback of the information derived from the cryptographic operation.² The NIST Recommendation for Block Cipher Modes of Operation [NIST800-38] defines modes of operation for the encryption and decryption of data using block cipher algorithms such as AES and Triple DES. The most commonly used modes of operation are Cipher Block Chaining (CBC) and COUNTER (CTR). These modes of operation, however, must be combined with integrity guarantees to provide security against the more powerful chosen ciphertext attacks.

There are a number of widely known and widely used block ciphers, of which the best known is almost certainly the Data Encryption Standard (DES), which has been a de facto standard for certain industries for over 20 years. However, the DES secret

² *Semantic security* is the notion of security [GM84] that tries to address these issues by making encryption *probabilistic*. Semantic security is equivalent to the notion of *ciphertext indistinguishability*. This notion of security works well against passive attackers but it does not consider the case of (adaptive) chosen ciphertext attacks (CCA), where an attacker is able to request the decryption of chosen ciphertexts. Consequently, semantic security is now considered insufficient for securing a general-purpose encryption scheme.

key only contains 56 bits, which means that with modern technology, it is possible to go over all possible 2^{56} keys until the correct one has been found.³ As a result DES is no longer considered appropriate for securing communications, although a more secure version exists – Triple DES [NIST800-67] – which uses two or three different keys, increasing the security level to 112 or 168 bits, respectively.

To overcome the limitations of DES, the Advanced Encryption Standard (AES) was developed in the late 1990s [FIPS197] that uses keys of length at least 128 bits. AES encrypts and decrypts data in 128-bit blocks, using 128, 192 or 256 bit keys. The usual notation for AES for the different key sizes is AES-x, where x is the key size.

2.3.2 Hash Functions

Many algorithms and schemes that provide a security service use a hash function as an intermediate component. A hash function takes as input an arbitrary string of bits and gives as output a *short, fixed-length*, value that is a function of the entire input. The output of the hash function is commonly known as hash-value, message digest, or fingerprint of the message.

Hash functions must be *one-way* in that they are simple and efficient to compute but hard to invert, i.e. given a particular hash value, it is computationally infeasible to find an input that produces the chosen hash value. Such property can be used for a variety of security services. For example, if it is necessary to protect the integrity of a large set of data (e.g. a database, a file, etc.), the entire set of data can be hashed and the output stored in some safe place. At some later time, the same data can be input to the hash function and the result compared to the stored value. If they match, the owner of the data can be confident that the data have not changed. As another example, a hash function is used in most practical digital signature schemes. When we want to sign a document, what we usually do is sign the hash value of the document. This way, signatures can be easy to generate since a hash value always has a fixed length, irrespective of the size of the document, and more secure since it makes it harder to forge a signature. Finally, hash functions can be used to generate Message Authentication Codes that provide both integrity and authenticity.

³ Lenstra and Verheul [LV01] provide guidelines for the determination of key sizes for symmetric cryptosystems, RSA, and discrete logarithm based cryptosystems. Nowadays it is believed that performing up to 2^{56} symmetric encryption/decryptions is “easy” while 2^{80} operations is “hard”. Since most symmetric algorithms have key sizes that are multiples of 64, keys of size 128 bits provide a very good level of security. For public key algorithms an equivalent level of security is 2048 bits.

The number of bits output by the hash function is determined by the design of the hash function. With a well-designed cryptographic hash function, it is not feasible to find a message that will produce a given hash value (pre-image resistance), nor is it feasible to find two messages that produce the same hash value (collision resistance). These properties are important when one uses hash functions with digital signature schemes, among other things.

Five typical hash functions [SHS] are SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512. The hash size produced by SHA-1 is 160 bits, 224 bits for SHA-224, 256 bits for SHA-256, 384 bits for SHA-384, and 512 bits for SHA-512. Algorithms standards must specify either the appropriate size hash function or the hash function size selection criteria if the algorithm can be configured to use different size hash functions.

2.3.3 Message Authentication Codes

Message Authentication Codes (MACs) provide data authentication and integrity. A MAC acts as a cryptographic checksum that is used to provide guarantee that the data has not been changed and that the MAC has been computed by the expected entity. Although message integrity is often provided using non-cryptographic techniques known as error detection codes, these codes can be altered by an adversary to effect an action to the adversary's benefit. The use of a MAC can alleviate this problem.

A MAC algorithm is a cryptographic function that takes as input a message and a secret key and outputs a short, fixed length output. This output is then sent or stored with the message and can be used to protect its integrity and guarantee its origin. If the receiver of a MAC has the correct secret key, the key can be used to verify the received MAC value. If this agrees with the value sent then the recipient knows that the message has not been changed and that it must have been sent by someone who knows the secret key. To be effective, it must be infeasible to forge a MAC, i.e. compute a MAC for a message without knowing the secret key, and it must also be infeasible to recover the secret key from valid MAC values.

Two types of algorithms for computing a MAC generally exist: MAC algorithms that are based on block ciphers, and MAC algorithms that are based on hash functions. Perhaps the most widely known method for generating a MAC is the "CBC-MAC" that is obtained using a block cipher such as AES or TripleDES operating in CBC mode. The key and block size used to compute the MAC depend on the algorithm used. If, however, the same block cipher is used for both

encryption and MAC computation in two different cryptographic operations, the same key *must not* be used for both the MAC and the encryption operation.⁴

The other way to generate a MAC is to use a hash function which gives rise to a construct called HMAC whose security has been shown in [BCK96]. The algorithm requires a single pass through the entire data. A variety of key sizes are allowed for HMAC; the choice of key size depends on the amount of security to be provided to the data and the hash function used.

In summary, MACs can be used to provide both data integrity and data origin authentication. However, MACs cannot provide *non-repudiation*. This is because the process of verifying a MAC is essentially the same as the process of generating the MAC. Hence the sender of MACed message can potentially “deny” it and claim that the recipient “forged” the MAC using his/her knowledge of the secret key. Thus a MAC cannot be used as long-term evidence that a message has been sent. This is a service provided by digital signatures.

⁴ For an illustration of an attack, see for example, <http://en.wikipedia.org/wiki/CBC-MAC>

2.4 Public key cryptographic techniques

Asymmetric key algorithms, also known as public key algorithms, use two related keys (a key pair) to perform various operations: a *public* key that may be known by everyone and a *private* key that must be kept secret. The two keys are linked in a mathematical way; however, knowledge of the public key does not reveal any information about the private key. Asymmetric algorithms are used to encrypt messages, to sign messages and to establish cryptographic keying material.

2.4.1 Encryption/Decryption

Public key encryption algorithms transform a block of plaintext into a block of ciphertext. Unlike symmetric algorithms, however, *different* keys are used for encryption and decryption. More precisely the public key of the intended recipient is used for encryption and the private key of the recipient is used for decryption.

Perhaps the most widely known algorithm today is the RSA algorithm, named by its inventors Rivest, Shamir and Adleman who first publicly described it [RSA78]. It is the first algorithm known to be suitable for signing as well as encryption, and one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys (longer than 2048 bits) and the use of up-to-date implementations.

Public key encryption is no more or less secure than symmetric encryption (security depends on the key size), however its use of large keys makes it computationally expensive, and hence slower to compute than symmetric algorithms such as DES and AES.

2.4.2 Digital Signatures

Digital signatures are used to provide authentication, integrity and *non-repudiation*. A digital signature is computed using the signer's private key and can be verified efficiently by anyone that has knowledge of this public key. The most common use of a signature gives a value, which like a MAC, can be attached to a message that needs to be protected.

Digital signatures can be used to authenticate the sender of a message since the signature could not have been created without knowledge of the signing key. Similar to a MAC, integrity is also provided since any change of the message after a signature has been created would invalidate the signature. Unlike a MAC however, signatures also provide *non-repudiation*. This is because an entity that verifies the validity of a signature cannot in principle forge a signature. Thus a signature must have been generated by the entity that possesses the private key.

Example of signature functions include those based on RSA, and those based on the security of the discrete logarithm problem, like the Digital Signature Algorithm [DSA].

2.5 Key establishment

Key establishment schemes are used when two entities wish to agree on a secret key so that they communicate securely with each other. In such schemes two parties can exchange messages in such a way so that they agree on a shared key based on the messages exchanged and secret information known only to them.

This section will by no means be a comprehensive treatment of protocols for authentication and key establishment. There have been previous surveys that can be of value to the reader who wants to find more about this fascinating topic. The extensive survey of Clark and Jacob [CJ97] includes a library of about 40 protocols as well as introductory material and an annotated bibliography. The two chapters in the *Handbook of Applied Cryptography* [HAC96] also provide a source of valuable information. Finally, the book by Boyd and Mathuria [BM03] gives a comprehensive treatment on protocols for authentication and key establishment.

In this section we will look at some typical ways that key establishment can be achieved that will be more applicable to the LOTUS communication scenario along with references to the most popular protocols. Our goal will be to abstract many of the details of the actions to be performed by the two parties (Section 2.5.4) in a way that will ease the development of the LOTUS secure communication scheme.

In general, two types of key establishment are defined: key transport and key agreement. *Key transport* is the distribution of a key (and other keying material) from one entity to another. The keying material is usually encrypted by the sending entity and decrypted by the receiving entity. If a symmetric algorithm (e.g., AES) is used to encrypt the keying material to be distributed, the sending and receiving entities need to know the symmetric key wrapping key (i.e., key encrypting key). If a public key algorithm is used to distribute the keying material, a key pair is used as the key encrypting key; in this case, the sending entity encrypts the keying material using the receiving entity's public key; the receiving entity decrypts the received keying material using the associated private key.

Key agreement is the participation by both entities in the creation of shared keying material. Among the class of public key protocols for key establishment, key agreement appears to be more popular than key transport since it can result in higher quality keys (since both parties contribute to the generation of the shared key). The traditional way of meeting these requirements would be to use public key cryptography; however, the use of symmetric keys may also be possible. Proposed methods are discussed in the following subsections along with a typical key establishment exchange of messages.

2.5.1 Key establishment using preconfigured public/private keys

The traditional way of solving the key establishment problem would be for each entity to be set up with a unique private key before deployment. The associated public key would be signed by a recognized authority (e.g. the owner of the deployed network), and the obtained certificate, together with the authority's trusted public key, would also be preconfigured into the node.⁵ After deployment, nodes wishing to establish keys between each other would simply swap their certificates, verify the validity of the certificates with the authority's trusted public key, and then exchange symmetric keys with each other using one of the many available protocols (e.g. [DH76]).

The main problem with this approach is the amount of processing power required for the algorithms used, which generally makes it infeasible if some of the entities involved have limited computational or storage capabilities (such as a person's mobile phone). However, as noted previously, it can be possible to do this in some cases. In addition, various optimizations have been worked on which make use of lightweight schemes. For example, protocols have been designed that take advantage of advancements in elliptic curve cryptography⁶.

Another issue to be considered is whether the protocol requires more private key operations (signatures or decryptions) or more public key operations (signature verifications or encryptions). For example, RSA is more efficient for public key operations, while for most algorithms based on discrete logarithms, the opposite is true. Furthermore, a protocol that requires mainly public key operations may be much more efficiently implemented using RSA with a small public exponent.

In general, depending on the application scenario, the use of public key cryptography may be possible. However, it is also true that if its use can possibly be avoided, then it should be.

⁵ One of the problems associated with public key cryptography is the management of public keys. Although public keys do not have to be secret, their integrity and authenticity must be maintained through the use of *certificates* signed by trusted third parties. In our discussion, we will assume that public keys are authentic. Users of protocols, however, must be aware that this is an important simplification that needs to be addressed.

⁶ http://en.wikipedia.org/wiki/Elliptic_curve_cryptography

Key agreement schemes

Figure 2-1 depicts the steps implemented by an entity when establishing secret keying material with another entity using one of the key agreement schemes to be described in this section. Depending on the key agreement scheme, the entity could be either the key agreement initiator or responder. *Key confirmation* is included to provide assurance that the owner possesses the correct shared key.

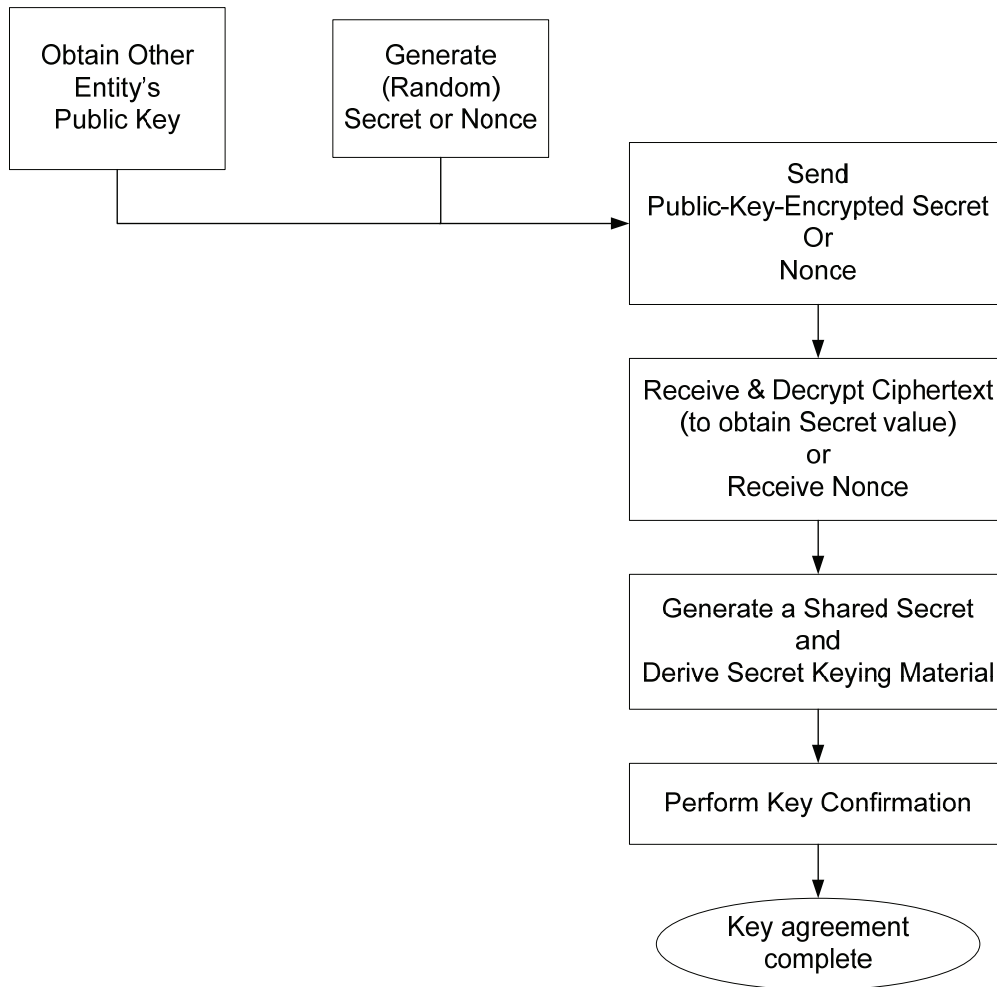


Figure 2-1: Key Agreement Process

Each participant obtains the identifier associated with the other entity, and verifies that the identifier of the other entity corresponds to the entity with whom the participant wishes to establish secret keying material.

Each entity generates either a (random) secret value (which becomes a shared secret when transmitted to the other entity) or a nonce, as required by the particular key agreement scheme. Each participant in the key agreement process uses the appropriate public and/or private keys to establish a shared secret and then derives secret keying material from the shared secret (and other available information) with

the help of a key derivation function. The key derivation function usually depends on the underlying protocol used. Typically it will be a one-way hash function of the shared secret and other exchanged data. A generic construction, known as KDF1, is specified in the IEEE P1363-2000 standard [P1363-2000]. Key confirmation may be used to provide assurance that the key pair owner possesses the (correct) private key.

Examples of Key Agreement protocols based on Diffie-Hellman

The Diffie-Hellman protocol dates back to 1976 [DH76]. This protocol has introduced public key cryptography to the academic world⁷ and has been the basis for numerous new protocols. It suffers, however, from lack of authentication which has been the goal of subsequent protocols. This is the first practical scheme that is used to exchange a secret key between two communicating parties with no need of a secure confidential channel. The protocol is shown in Table 2-1.

Table 2-1: Diffie-Hellman key agreement

Alice		Bob
Alice picks a random number $a \in [1, p-1]$		
$t_A = g^a \text{ mod } p$	t_A →	Bob picks a random number $b \in [1, p-1]$
	t_B ←	$t_B = g^b \text{ mod } p$
$Z_{AB} = (t_B)^a \text{ mod } p$		$Z_{AB} = (t_A)^b \text{ mod } p$

In this protocol, Alice and Bob are assumed to have agreed on an element g that generates a multiplicative group G (for example the set $[1, p-1]$ or a multiplicative subgroup of it, where p is a prime number). Then they select random numbers a and b and compute the public values t_A and t_B which they send to each other. Finally, they agree on the common secret Z_{AB} which is equal to $g^{ab} \text{ mod } p$.

Diffie-Hellman is secure against passive attackers since it is considered infeasible to derive the shared secret from the public values t_A and t_B (this is called the *Diffie-Hellman assumption*). It falls prey, however, to more active attackers since it does not

⁷ It is now known that Clifford Cocks, a British cryptographer, invented the first public-key cryptosystem in 1973 [Sin99]. Cocks’s encryption algorithm, named “non-secret encryption” is based on the difficulty of integer factorization and was essentially the same as RSA.

support the authenticity of the key agreed. An active adversary, in the middle of communications between Alice and Bob, can manipulate the messages exchanged to succeed in the so called *person-in-the-middle attack*. In particular the attacker (Eve) relays her own public value $t_E = g^e \bmod p$, succeeding in establishing separate keys g^{ae} and g^{eb} with Alice and Bob, respectively.

The shared secret derived by the protocol shown in Table 2-1 is called *ephemeral* because it depends on randomly chosen values that last only until the session key has been computed. If Alice and Bob already have public keys then they can use their public keys together with a random value to form a new shared key as the following protocol illustrates.

Table 2-2: Using a static Diffie-Hellman key

Alice		Bob
Alice picks a random number $a \in [1, p-1)$ and obtains Bob's long term public key y_B		Bob's private key is x_B Bob's public key is $y_B = g^{x_B}$
$t_A = g^a \bmod p$	t_A \longrightarrow	
$Z_{AB} = (y_B)^a \bmod p$		$Z_{AB} = (t_A)^{x_B} \bmod p$

In this protocol, Alice forms a shared secret using her random value a and Bob's public key y_B by calculating $Z_{AB} = y_B^a$. Similarly, Bob can reconstruct this secret by raising the value t_A received by Alice to its secret key x_B . Evidently, Bob cannot authenticate A since anybody could have generated t_A and cannot ensure the freshness of the key unless since it does not provide any fresh input. Alice, however, can obtain implicit key authentication.

One way to improve upon the previous situation is to use a protocol defined in [IEEE P1363] where the shared secret is the concatenation of both the ephemeral Diffie-Hellman key and the static one $(g^{x_A x_B} \bmod p)$ derived from participants' public keys. This protocol shown in Table 2-3 provides *forward secrecy*⁸ since an attacker must know one of the values a or b to find Z_{AB} . However, the protocol does not

⁸ Forward secrecy is the security property where compromise of the long-term keys of participants does not compromise the session keys established in previous protocol runs of these participants.

prevent *key compromise impersonation*⁹ since knowledge of the long term public keys is sufficient to calculate Z_{AB} . One way to defend against this last attack it to use the MQV protocol [MQV95] which differs from the previous one only in the calculation of the shared secret Z_{AB} .

Table 2-3: Combining an ephemeral Diffie-Hellman key with a static one

Alice		Bob
Alice picks a random number $a \in [1, p-1)$		
$t_A = g^a \text{ mod } p$	t_A \longrightarrow	Bob picks a random number $b \in [1, p-1)$
	t_B \longleftarrow	$t_B = g^b \text{ mod } p$
$Z_{AB} = (t_B)^a \parallel (y_B)^{x_A}$		$Z_{AB} = (t_A)^b \parallel (y_A)^{x_B}$

All these protocols and more can be augmented with bilateral key confirmation as shown in Table 2-4. Initially both parties agree on shared secret Z_{AB} using the methods described previously. From the shared secret a MAC key is derived with the help of a key derivation function KDF. “OtherInfo” includes information that has been exchanged as well as the identities of the participants, types of algorithms used, number of bits to be output by the KDF, etc.

After the MACs have been exchanged and verified, each party obtains assurance of the validity of the key. The “OtherData” included in the MAC computation must be chosen based on values exchanged and the IDs of participants. This value must be different in each MAC computation to ensure that no MAC can be replayed to the other party simply by reflecting the response back to the originator.

⁹ This is an attack where an adversary upon compromising the long term key of a user U, can masquerade to U as a *different* user V.

Table 2-4: Adding key confirmation

Initiator (Alice)		Respondent (Bob)
Derivation of shared secret Z_{AB} using a key agreement protocol. Derived Keying Material = KDF (Z_{AB}, OtherInfo)		
Verify received MAC	← T_B	$T_B = \text{MAC}(\text{MAC}_{\text{Key}}, \text{OtherData})$
$T_A = \text{MAC}(\text{MAC}_{\text{Key}}, \text{OtherData}')$	T_A →	Verify received MAC

Closing this discussion on key agreement protocols, we should note that extra information can be added in the protocol steps in order to *authenticate* the messages exchanged between the participants. A typical case is to *sign* the ephemeral public keys by the participant that generates them as illustrated in Table 2-5. This protocol is a modified (and more secure) version of the basic Station-to-Station protocol developed in [DOW92]. The basic version (containing no identifiers) was attacked by Lowe [Low96] showing that mutual authentication could not be achieved. For a more complete treatment of key agreement protocols, the user is referred to [BM03].

Table 2-5: Modified STS protocol

Alice		Bob
Alice picks a random number $a \in [1, p-1)$		
$t_A = g^a \text{ mod } p$	t_A →	Bob picks a random number $b \in [1, p-1)$
Verify received signature	$t_B, \text{Sig}_B(t_A, t_B, A)$ ←	$t_B = g^b \text{ mod } p$
$Z_{AB} = (t_B)^a \text{ mod } p$	$\text{Sig}_A(t_A, t_B, B)$ →	Verify received signature $Z_{AB} = (t_A)^b \text{ mod } p$

We conclude this section by introducing Internet Key Exchange (IKE), a “real world” protocol that can be used to set up a security association (SA) in the IPsec

protocol suite. IPsec operates at the internet layer of the networking protocol stack, providing *transparent* security for all layers above it, protecting all transport and application layer protocol messages.

A *Security Association (SA)* is an agreement about how two hosts (or two IPsec gateways) will provide for security. The SA specifies the algorithms that will be used to provide for confidentiality, authentication, message integrity and replay protection. IKE is the protocol that handles all the steps needed to establish a security association. These include agreement on the algorithms to be used, authentication and exchange of symmetric session keys. The protocol is shown in Table 2-6.

Table 2-6: Snapshot of Internet Key Exchange

Alice		Bob
Alice picks a random number $a \in [1, p-1)$ and nonce N_A		
$t_A = g^a \text{ mod } p$	t_A, N_A \longrightarrow	Bob picks a random number $b \in [1, p-1)$ and nonce N_B
	t_B, N_B \longleftarrow	$t_B = g^b \text{ mod } p$
Compute $Z_{AB} = g^{ab} \text{ mod } p$ Derived MAC Key and Session key K_{AB} using KDF (Z_{AB}, N_A, N_B)		
$M_A = \text{MAC}(A, \text{MAC Key}, t_A, t_B, \text{OtherData})$	$\{A, \text{Sig}_A(M_A)\}_{K_{AB}}$ \longrightarrow	Decrypt and verify signature/MAC value
	$\{B, \text{Sig}_B(M_B)\}_{K_{AB}}$ \longleftarrow	$M_B = \text{MAC}(B, \text{MAC Key}, t_A, t_B, \text{OtherData})$
Decrypt and verify signature/MAC value		

The first two steps of the protocol (initial negotiation of parameters has been omitted) constitute an ordinary Diffie-Hellman exchange in which the two parties agree on a shared key Z_{AB} . Once this phase is complete, the two parties use a key derivation function to derive encryption and MAC keys from Z_{AB} . Finally, the

signatures are used to provide for authentication between the two hosts as well as integrity of parameters that have been agreed in the previous protocol steps. The authentication can be performed using either pre-shared key (shared secret) or signatures or public key encryption. Once this phase is complete, the IKE hosts use the secure channel established to negotiate Security Associations on behalf of other services like IPsec. The negotiation results in a minimum of two unidirectional security associations, one in each direction.

Currently, IKE is at version two (IKEv2) as was updated in December 2005 by RFC 4306 [IKE].

Key transport schemes

Figure 2-2 illustrates the steps executed by protocol participants during key transport where one party selects the secret keying material. Such schemes may be using either public key techniques or a combination of public key and symmetric key techniques. The party that sends the secret keying material is called the sender, and the other party is called the receiver.

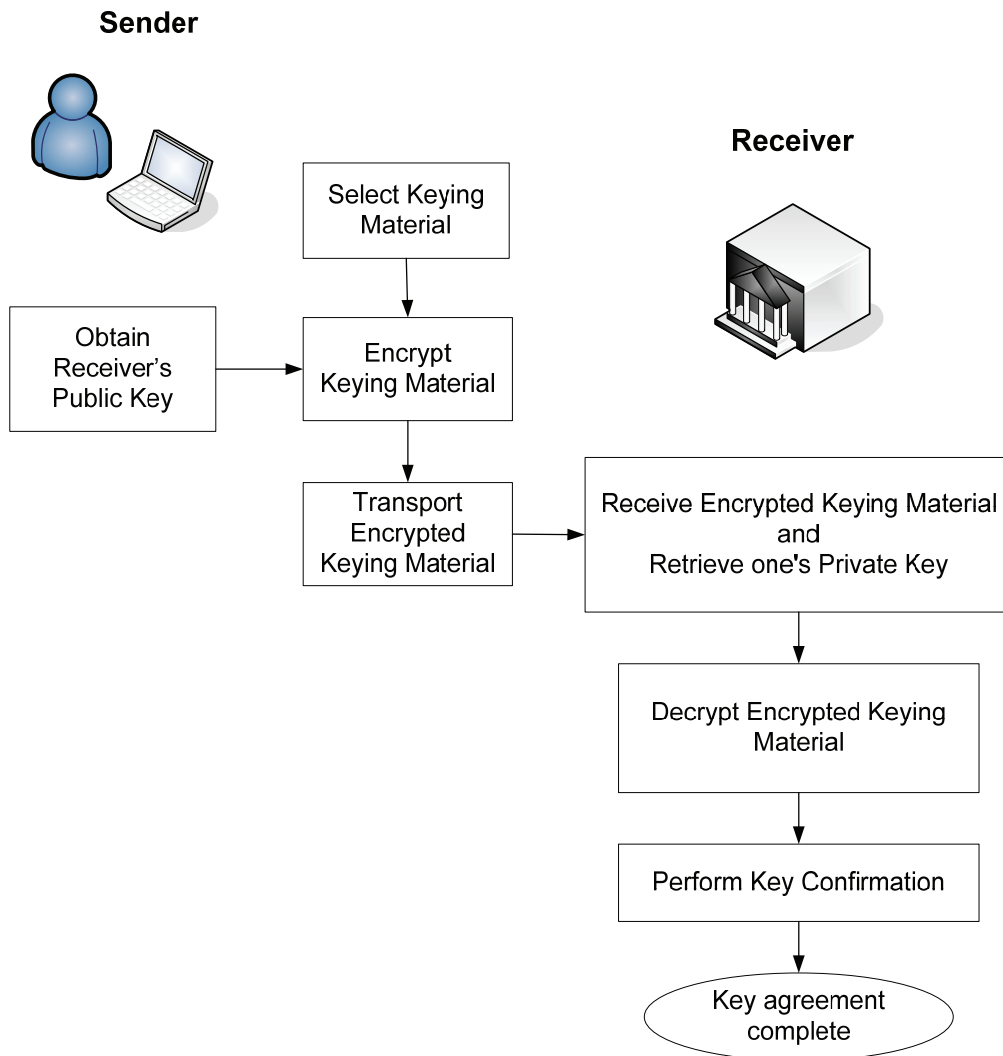


Figure 2-2: Key Transport Process

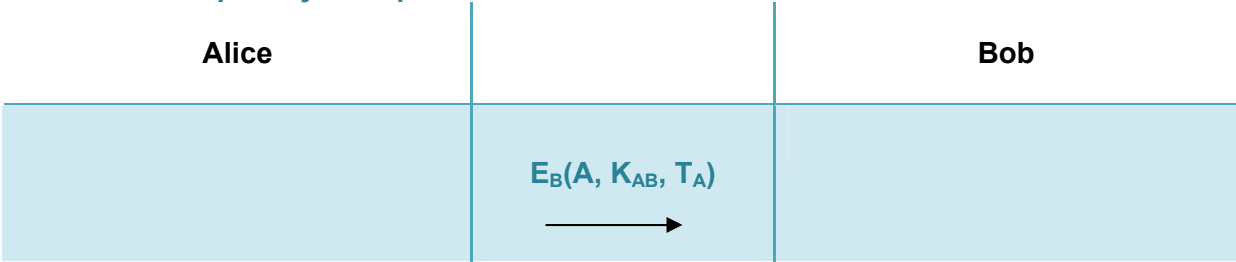
The sender obtains the identifier associated with the receiver and retrieves the receiver's public key. The sender selects the secret keying material (and, perhaps, other data) to be transported to the other entity and either encrypts that material directly, or, employs a combination of secret value encapsulation and key-wrapping. The resulting ciphertext is transported to the receiver. Using its private key, the receiver decrypts the ciphertext and obtains the plaintext keying material.

Key confirmation can be used to obtain assurance that the receiver has correctly recovered the keying material from the ciphertext. The sender can also obtain assurance that the receiver was in possession of the correct private key.

Examples of Key Transport protocols

In this section we will abstract away details of the public key algorithms used by using generic encryption and signature operations. Perhaps the simplest protocol is the one shown in Table 2-7 that can be found in the ISO/IEC 17770-3 standard.

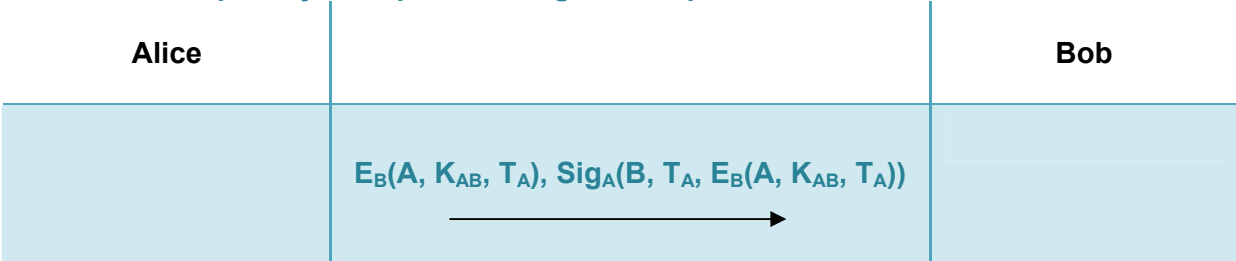
Table 2-7: Simple key transport



In this protocol, the session key K_{AB} is chosen by Alice and is sent to Bob encrypted with Bob’s public key. T_A is a *timestamp* to ensure freshness of the key. The problem, however, with this protocol is that Bob can have no assurance that this message indeed originates from Alice as he cannot authenticate her. Alice can be sure that the key is fresh (since she chooses it) and can only be shared by her and Bob. Again, however, no key confirmation is provided.

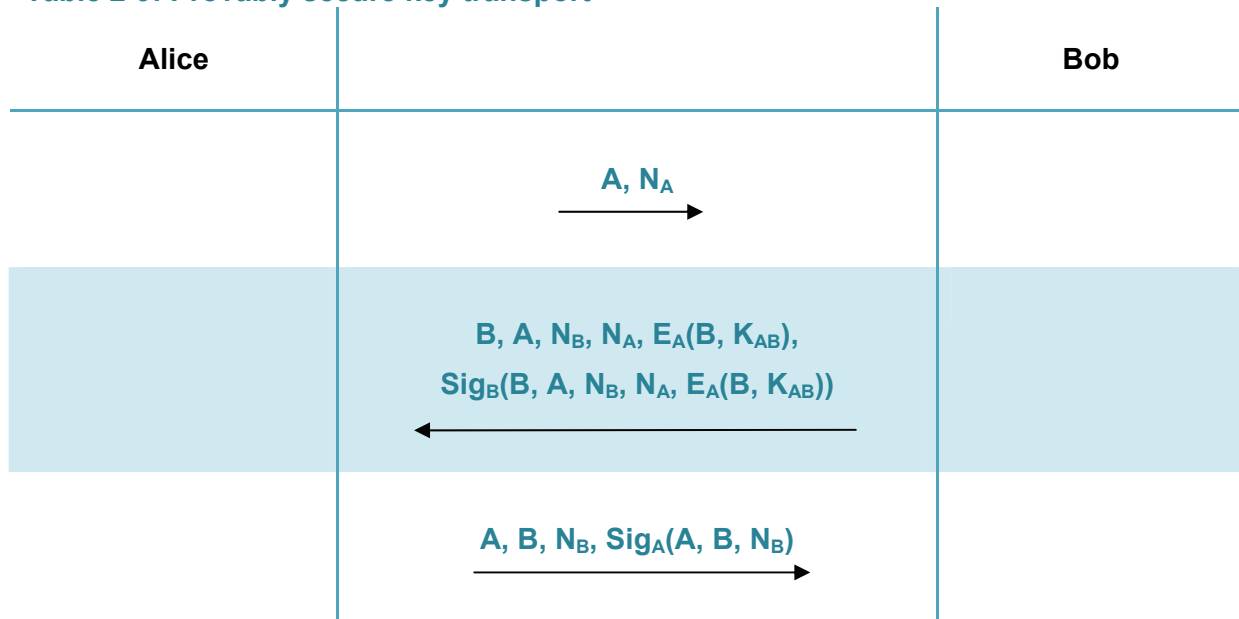
Some of the problems mentioned above can be solved if Alice signs the message. Then Bob can be sure that the message originates from Alice, while achieving key confirmation for him. As before, the protocol provides a good key for Alice but no key confirmation from Bob. This protocol is shown in Table 2-8.

Table 2-8: Simple key transport with signature operation



Adding more rounds to the protocol can provide mutual authentication and key confirmation as demonstrated in Table 2-9.

Table 2-9: Provably secure key transport



This protocol, from [BWM98], starts by Alice sending a nonce N_A to Bob. Then Bob picks a symmetric key K_{AB} and sends it encrypted using Alice's public key. At the same time it signs the message with his private key so that Alice authenticates its response. Finally, in the third message, Alice provides her signature to prove her presence to Bob. Notice that Bob does not obtain key confirmation but this can be solved by adding a MAC on the last message where the MAC key has been derived from the session key.

We conclude this section by introducing one "real world" protocol that allows client/server applications to communicate across a network in a way designed to prevent eavesdropping, tampering, and message forgery. This is the Transport Layer Security (TLS) protocol which has been defined in RFC 2246 in January 1999 and has been standardized by the Internet Engineering Task Force (IETF).

In typical TLS use case, authentication is *unilateral*: only the server is authenticated (the client knows the server's identity and its public key certificate), but not vice versa due to the lack of a universal public key infrastructure that can handle certificates for clients (i.e. end users). *Mutual authentication* requires that the TLS client also hold a certificate (which is not usually the case in the typical end-user/browser scenario). To provide for mutual authentication in the absence of client certificates, shared secrets or password based authentication can be used.

TLS involves three basic phases:¹⁰

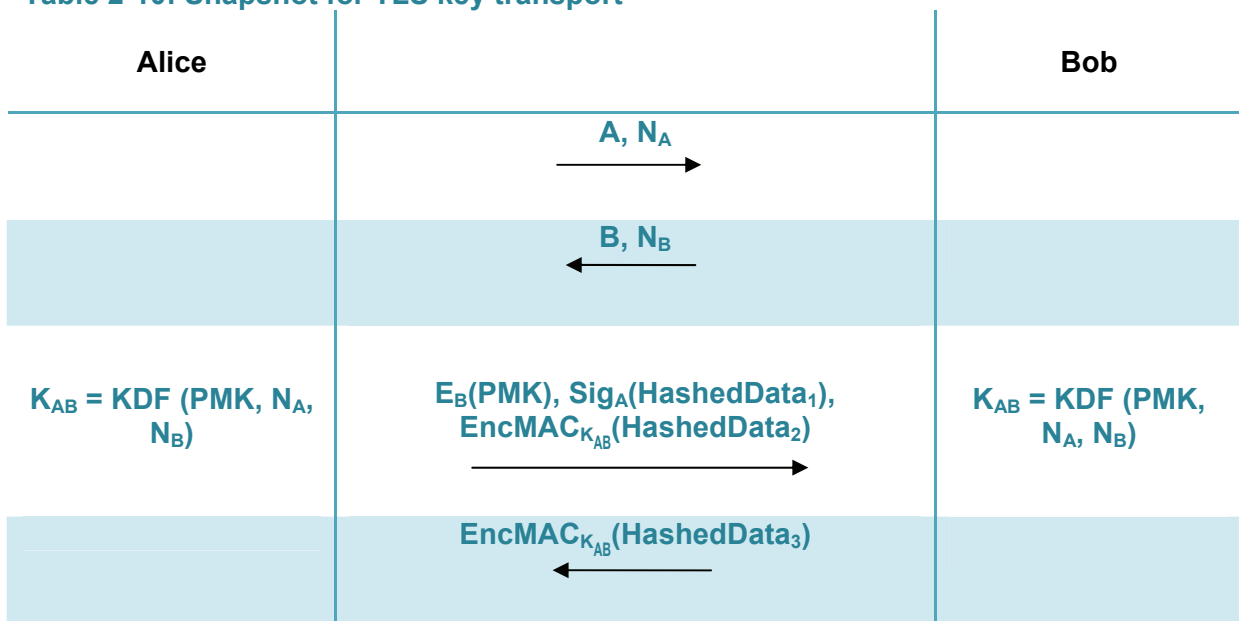
1. Peer negotiation for algorithm support
2. Key exchange and authentication

¹⁰ See for example http://en.wikipedia.org/wiki/Transport_Layer_Security

3. Symmetric cipher encryption and message authentication

During the first phase, the client and server negotiate cipher suites, which determine the ciphers to be used, the key exchange and authentication algorithms, as well as the message authentication codes (MACs). The key exchange and authentication algorithms are typically public key algorithms such as RSA, although other algorithms could be added to the available protocols. The message authentication codes are made up from cryptographic hash functions using the HMAC construction. Table 2-10 illustrates a simplified version of the protocol where a pre-master secret PMK is sent from Alice (the client) to Bob (the server).

Table 2-10: Snapshot for TLS key transport



In this protocol, “HashedData” correspond to hash values of messages that have been exchanged so far. The shared key is derived from the pre-master secret and the nonces supplied by the two parties (hence also B can affect the value of the shared key) using an appropriate key derivation function. Key confirmation for the use of key is provided by the EncMAC operation which alleges to the use of an operation that provides for both confidentiality and integrity.

2.5.2 Key establishment using preconfigured symmetric keys

If two nodes already share a common symmetric key, then the use of computationally expensive public key cryptography can be avoided. However, this is complicated by the fact that it would be highly unlikely that *any* two nodes in the network will share a secret key among them. An alternative solution would be to use trusted third parties that act as intermediaries in order for keys to be established.

The exact details of such schemes depend on the trust assumptions that can be made about a particular deployment scenario and do not seem to be very helpful in the LOTUS envisioned scenario. Thus we will not examine them in detail; we will give instead a few common representative examples from each category.

Table 2-11: ISO/IEC 11770-2 Mechanism 1

Alice		Bob
$K'_{AB} = \text{KDF}(K_{AB}, T_A)$	$\text{EncMAC}_{K_{AB}}(T_A)$ \longrightarrow	$K'_{AB} = \text{KDF}(K_{AB}, T_A)$

Table 2-11 is from the ISO/IEC 11770 Part 2 International Standard [ISO11770-2] which describes a number of protocols that may rely or not on a trusted server. The specific example uses one message from Alice to Bob that provides implicit key authentication. Using the existing shared key K_{AB} , a timestamp is transmitted in a way that guarantees both confidentiality and integrity. The new session key is derived from the application of a key derivation function to the timestamp T_A .

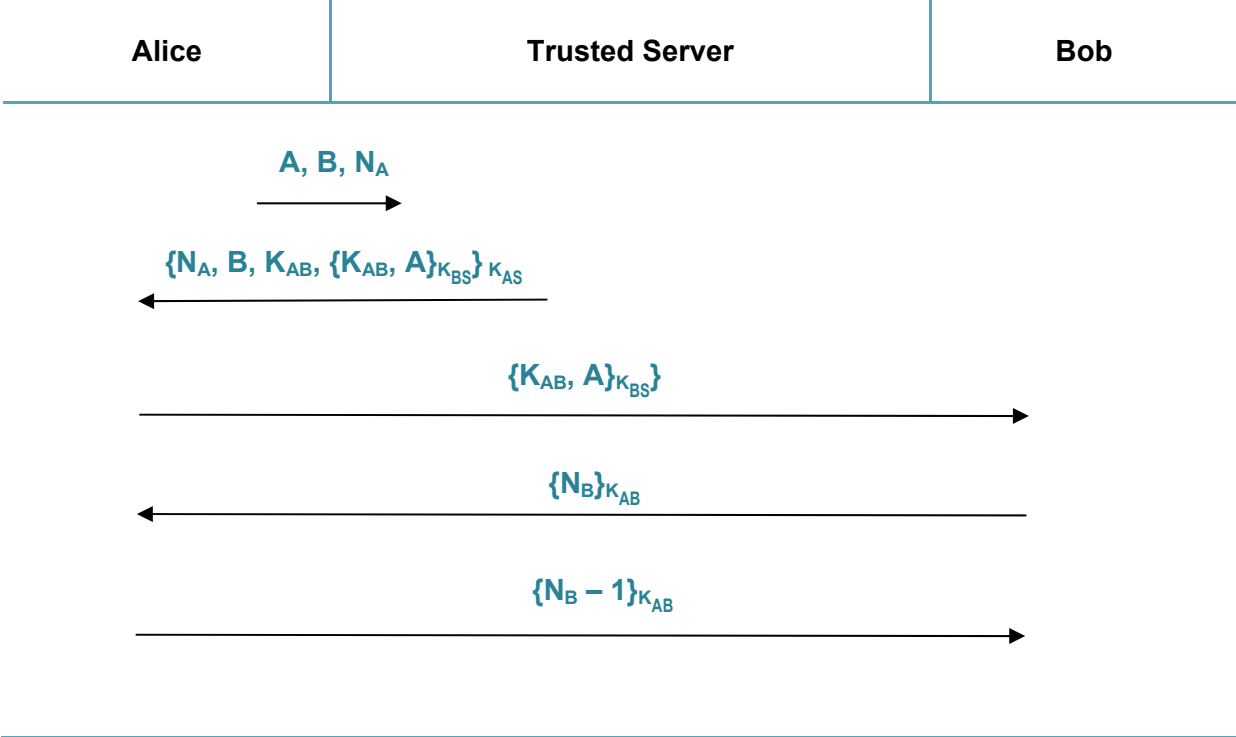
A simple extension that provides for *mutual authentication* is shown in Table 2-12. Both parties affect the outcome of the key derivation function by using supplied keying material K_A and K_B . If synchronized clocks are hard to maintain, timestamps can be replaced by nonces supplied by both parties. These nonces must be included in the encrypted messages to bind them together and avoid certain types of attacks.

Table 2-12: ISO/IEC 11770-2 Mechanism 5

Alice		Bob
	$\text{EncMAC}_{K_{AB}}(T_A, B, K_A)$ \longrightarrow	
$K'_{AB} = \text{KDF}(K_{AB}, K_A, K_B)$	$\text{EncMAC}_{K_{AB}}(T_B, A, K_B)$ \longleftarrow	$K'_{AB} = \text{KDF}(K_{AB}, K_A, K_B)$

Turning now to key establishment with the help of a trusted server S, the most known example is the protocol developed by Needham and Schroeder [NS78], shown in Table 2-13.

Table 2-13: Needham-Schroeder protocol



In this protocol, N_A and N_B are nonces issued by Alice and Bob, respectively. K_{AS}/K_{BS} are the keys shared between the server and Alice/Bob. K_{AB} is the shared key to be established with the help of the trusted server.

Initially, Alice contacts the server and asks for a fresh key to communicate with Bob (Message 1). The server responds with such a key encrypted with the key K_{AS} it shares with Alice (Message 2). Alice can be sure that the message (and hence the key) is fresh since she can recover the nonce she issued in the first step. Unfortunately, the protocol breaks down in the next step as pointed by Denning and Sacco [DS81] since Bob has no guarantee that this message is not a reply of an older message in which the attacker was able to recover the key K_{AB} . In such an attack, a malicious party can use the compromised key to masquerade as Alice to Bob. To overcome the attack, Denning and Sacco suggested the use of timestamps in the second and third message to allow Bob verify the freshness of the key.

It is interesting to note that this protocol (with timestamps) is used as a building block in Kerberos, an authentication protocol, which allows nodes communicating over an insecure channel to prove their identity to one another in a secure manner. The designers of Kerberos aimed primarily at a client-server model where a client (Alice), with the help of an authentication server (AS), manages to get access to a

service (Bob). Kerberos messages are protected against eavesdropping and replay attacks and the protocol provides for mutual authentication – both the user and the server can verify each other's identity (for a review on Kerberos, see [NT94]).

Finally, we should note that several protocols for key establishment with the help of a trusted server can be found in the ISO/IEC 11770-Part2 standard. In some of these, the shared key is chosen by the trusted server while in others the session key is chosen by either A or B. The server in the first case acts as a key distribution center while in the second as a key translation center, making the key available to the other party. The interested reader is referred to the standard for more details.

2.5.3 Password based key establishment

In the previous sections we saw key establishment protocols that depend on the existence of a secret key that was used to bootstrap the security infrastructure. In the asymmetric setting, this key (or keys) can be the private key of a party (or both parties) that can be used to decrypt information received from the other party or sign messages for authentication purposes. In the symmetric setting, this key can be a key shared between the two parties.

In many cases, however, the secret is a simple password, consisting of a number of ASCII characters, which needs to be relatively easy to remember. The question then becomes: Is it possible to base security on such a weak key (usually no more than 8 characters) that can be easily brute-forced by an attacker (i.e. the attacker can search for all possible passwords in a short time)?

Password-based protocols for authentication and key establishment are designed so that someone who eavesdrops on the communication line between client and server, or someone who impersonates either end will not obtain enough information to recover the secret password, even after observing a large number of legitimate message exchanges. In general, the basic properties we expect from such protocols can be seen below:

- Off-line verification of password guesses should not be possible. Someone who collects enough exchanged messages during a valid session should not be able to recover the password or eliminate a large number of them using a basic dictionary attack.
- On-line password verification should be limited to one per session. Someone impersonating either entity will be able to do a single on-line password guess. A false guess should result in an authentication failure, which should generate alarms and further actions if they occur in large numbers.

In this section we will review two such protocols (EKE and SRP) and highlight some of their basic properties since LOTUS cannot benefit from any password based authentication scheme.

Bellovin and Merritt [BM92] created the first strong password protocol that called EKE standing for *encrypted key exchange*. The idea is that Alice and Bob share a weak secret W (usually the hash of a password) that can be used to encrypt the ephemeral Diffie-Hellman numbers exchanged between them. Alice knows W because she can derive it from the password, while Bob (the server) has stored W in an appropriate database or password file. This means that compromise of the server does not automatically compromise the password. The protocol is shown in Table 2-14.

Table 2-14: Basic EKE protocol

Alice		Bob
Alice picks a random number $a \in [1, p-1)$		Bob knows $W = f(\text{password})$
$t_A = g^a \text{ mod } p$	$A, E_W(t_A)$ →	Bob picks a random number $b \in [1, p-1)$
	$E_W(t_B), E_{K_{AB}}(N_B)$ ←	$Z_{AB} = (t_A)^b \text{ mod } p$ $K_{AB} = \text{KDF}(Z_{AB}, \text{OtherData})$ $t_B = g^b \text{ mod } p$
$Z_{AB} = (t_B)^a \text{ mod } p$ $K_{AB} = \text{KDF}(Z_{AB}, \text{OtherData})$	$E_{K_{AB}}(N_A, N_B)$ →	Verify N_B
Verify N_A	$E_{K_{AB}}(N_B)$ ←	

As in basic Diffie-Hellman (Table 2-1), the shared secret Z_{AB} is equal to g^{ab} although the key derivation function to derive the shared key from Z_{AB} is not specified. The last two messages of the protocol aim at providing key confirmation of the new shared secret K_{AB} .

The intuition why the protocol is secure is that a Diffie-Hellman transmitted value looks like a random number. An eavesdropper doing a trial decryption on $E_W(t_A)$ or $E_W(t_B)$ cannot verify a password guess because decrypting with any password will still look like a random number.¹¹ And someone impersonating one side or the other can verify on a single password but this is an on-line, auditable guess. Elimination of one of the encryptions with W can lead to attacks as noted in [Pat97] and [STM95] depending on the precise format of the messages used for authentication.

¹¹ The original EKE paper did not specify the encryption algorithm used. Some later protocols used algorithms that map the shared password to an element of the group where the Diffie-Hellman exchange takes place. This is needed in order to avoid *partition attacks* as introduced by Bellare and Merritt. The idea is that an adversary guessing the password can attempt to decrypt $E_W(t_A)$ or $E_W(t_B)$ and examine whether the resulting plaintext is a valid Diffie-Hellman value. If not, the guessed password is incorrect and can be discarded.

The EKE protocol suffers from the obvious vulnerability that if someone finds W (say by stealing the server database), they could impersonate the user to the server. *Augmented* versions of password protocols have the additionally property of preventing the attack mentioned before. The idea is for a server to store a quantity derived from W (through the application of a one-way function) that can be used to verify the password, but the client is required to *know* the password and not just the derived quantity stored in the server. All protocols have augmented versions but we will conclude this section by describing SRP (for Secure Remote Password) that comes only in augmented form.

Table 2-15: Secure Remote Password (SRP) protocol

Alice		Bob
<p>Alice picks a random number $a \in [1, p-1]$</p>		<p>Bob has stored $g^W \bmod p$, where $W = f(\text{password})$</p>
<p>$t_A = g^a \bmod p$</p>	<p>A, t_A →</p>	<p>Bob picks a random number $b \in [1, p-1]$ and u</p>
	<p>S, u ←</p>	<p>$Z_{AB} = (t_A g^{Wu})^b \bmod p$ $K_{AB} = \text{KDF}(Z_{AB}, \text{OtherData})$ $t_B = g^b \bmod p$ $S = t_B + g^W$</p>
<p>$Z_{AB} = (S - g^W)^{a+uW} \bmod p$ $K_{AB} = \text{KDF}(Z_{AB}, \text{OtherData})$ $T_A = \text{MAC}(\text{MAC}_{\text{Key}}, \text{OtherData})$</p>	<p>T_A →</p>	<p>Verify T_A</p>
<p>Verify T_B</p>	<p>T_B ←</p>	<p>$T_B = \text{MAC}(\text{MAC}_{\text{Key}}, \text{OtherData})$</p>

SRP (Table 2-15) was invented by Wu [Wu98] and is a popular choice by the IETF for strong password protocols (documented in RFC 2945). The server (Bob) stores $g^W \bmod p$, where W is derived from Alice's password through the application of a cryptographic hash function. The shared secret is $Z_{AB} = g^{ab} \cdot g^{wub}$ which can be computed by A and B by information available only to them.

2.5.4 Abstracting away the key establishment process

Examples of both key agreement and key transport schemes have been presented in the previous sections. Here we will attempt to abstract many of their details and present a high level overview of the messages exchanged and the operations required to enable the establishment of a shared session key. This will be helpful in the development of the LOTUS secure communication framework, although many of the details will be instantiated in the LCSC report.

Typical key agreement message exchange

In a key agreement scheme, two parties, the initiator and the responder, establish keying material over which both have contributed input. We will consider two basic cases: if only the respondent's keying material is used, this scheme will be denoted by KA-R, otherwise if both parties' keys are used the scheme will be denoted by KA-IR. Key confirmation may be included to provide assurance that the participants share the same keying material. In what follows a general flow diagram will be provided for each key agreement scheme. Authentic public keys may be distributed by the parties themselves or by a third party, such as a Certification Authority (CA).

KA-R scheme

In this scheme (Table 2-16), the respondent *does not contribute* to the formation of the shared secret; instead, a nonce is used as a responder-selected contribution to the Key Derivation Function (KDF), ensuring that both parties influence the derived keying material.

In what follows, $(\text{PubKey}_B, \text{PrivKey}_B)$ denotes the respondent's (Bob) public-private key pair. SVE_{GEN} is a Secret Value Encapsulation function that is used to encrypt a secret value Z generated by A with the help of a pseudo-random generator. This value Z is encrypted with the respondent's public key PubKey_B , producing a ciphertext C . Note that the secret value Z will become a shared secret when recovered by B.

The other party (Bob) uses the SVE_{REC} operation to recover the shared secret Z from the ciphertext C using its private key PrivKey_B . Then it generates a nonce N_B and sends it to the initiator. The nonce is a time-varying value that has a negligible chance of repeating and can be used to indicate freshness of messages.

Once the nonce is sent, both parties can derive keying material (session keys, etc.) from the shared secret Z and "other information" such as IDs of both parties,

previous messages transmitted, etc., using a key derivation function KDF. The length of the keying material can be of any size and is limited only by the length that can be output by the KDF.

The final step is the MAC computation by party B and verification by party A that can be used to provide *unilateral* key confirmation from B to A. In this case the derived keying material is parsed to produce a MAC_{Key} which is used to produce a tag T . This tag is sent to A. If this tag matches the one computed by A then the successful establishment of keying material is confirmed to A.

Table 2-16: KA-R scheme with unilateral key confirmation

Initiator (Alice)		Respondent (Bob)
Retrieve respondent's public key establishment key $PubKey_B$		
Generate Z $C = SVE_{GEN}(Z, PubKey_B)$	C →	$Z = SVE_{REC}(PrivKey_B, C)$
	N_B ←	Generate nonce N_B
Derived Keying Material = $KDF(Z, OtherInfo)$		Derived Keying Material = $KDF(Z, OtherInfo)$
	T ←	$T = MAC(MAC_{Key}, OtherData)$
$T \stackrel{?}{=} MAC(MAC_{Key}, OtherData)$		

In this scheme, only the identifier of B is bound to a public key-establishment key. A (the initiator) has assurance that no third party can recover Z from C (without the compromise of B's private information). The respondent, however, has no such assurance. *In particular, B has no assurance as to the accuracy of the identifier claimed by the initiator and, therefore, has no assurance as to the true source of the ciphertext C .*

With respect to the freshness of the keying material, the initiator has assurance that the derived key is fresh due to its selection of the random Z value by itself. B has similar assurance since it contributes the nonce N_B to the KDF input.

Through the inclusion of MAC output T and by successfully comparing the received value T with its own computation, the A obtains assurance that

1. B has correctly recovered Z from C ;
2. that both parties agree on the values of “other data”;
3. that the derived keying material has been correctly computed by B;
4. B is in possession of the correct private key that corresponds to the public key used in the transaction, and
5. B has actively participated in the transaction.

Consequently, responder authentication is implicitly provided by the binding of party B’s identifier to the public key-establishment key.

KA-IR scheme

In this scheme (Table 2-17), *both* parties contribute to the formation of the shared keying material. This is the same as the basic KA-R scheme, however, the shared secret Z is the concatenation of sub-secrets Z_A and Z_B . The scheme can be augmented with key-confirmation which can be *bilateral* as well.

Table 2-17: KA-IR scheme with unilateral key confirmation

Initiator (Alice)		Respondent (Bob)
Retrieve respondent’s public key establishment key $PubKey_B$		Retrieve initiator’s public key establishment key $PubKey_A$
Generate Z_A $C_A = SVE_{GEN}(Z_A, PubKey_B)$	C_A →	$Z_A = SVE_{REC}(PrivKey_B, C_A)$
$Z_B = SVE_{REC}(PrivKey_A, C_B)$	C_B ←	Generate Z_B $C_B = SVE_{GEN}(Z_B, PubKey_A)$
$Z = Z_A Z_B$		$Z = Z_A Z_B$
Derived Keying Material = KDF (Z , OtherInfo)		Derived Keying Material = KDF (Z , OtherInfo)
Derived Keying Material = KDF (Z , OtherInfo)	T ←	$T = MAC(MAC_{Key}, OtherData)$

$T \stackrel{?}{=} \text{MAC}(\text{MAC}_{\text{Key}}, \text{OtherData})$		
---	--	--

In this scheme, *each* entity has an identifier that is bound to a public key-establishment key. Therefore, both parties have assurance that an attacker cannot recover sub-secret Z_X from ciphertext C_X , where X can be either A or B. Consequently, A and B both have assurance that they are the only two parties capable of deriving the keying material corresponding to $Z = Z_A || Z_B$.

With respect to the freshness of the keying material, the initiator has assurance that the derived key is fresh due to its selection of the random Z_A value. B has similar assurance since it contributes the secret Z_B to the KDF input.

Through the inclusion of MAC output T and by successfully comparing the received value T with its own computation, A obtains assurance that

1. B has correctly recovered Z_A from C_A (or in the case of bilateral confirmation that A has correctly recovered Z_B from C_B);
2. that both parties agree on the values of “other data”;
3. that the derived keying material has been correctly computed by B (or A in the case of bilateral confirmation);
4. B (or A) is in possession of the correct private key that corresponds to the public key used in the transaction, and
5. B (or A) has actively participated in the transaction.

2.6 LOTUS requirements for establishing a secure channel

LOTUS is an early warning system for chemical substances in which sensor data is to be exchanged between mobile sensors and a central system. Figure 2-5 illustrates a high level view of LOTUS Secure Communication Scheme (LSCS). Mobile routers are equipped with a box holding a GPS, a data processing unit, a radio module unit and different sensors (IMS, DMA and SERS sensors). As highlighted, the basic functionalities of the LSCS system are to encrypt measured data, store them to the processing data unit (along with a file-metadata) and send them to the central system. Data are provided by the sensors to the data unit where some processing is done before transmitting the encrypted raw data readings using a communication framework.

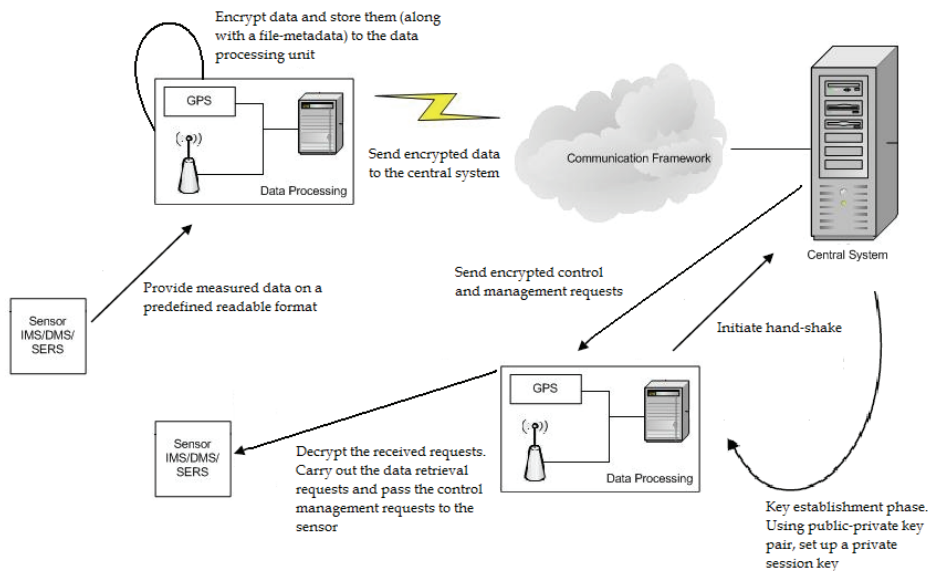


Figure 2-3: High level view of the LOTUS Secure Communication System

More specifically, all mobile routers initiate the key establishment phase for agreeing on the symmetric key to be used throughout the communication session. This session key is brought into play for encryption/decryption of processing data and decryption of received control and management requests. Data retrieval requests are carried out by the LSCS system whereas control requests are forwarded to the mobile sensor. Once data is ready for transmission, LSCS uses the underlying commercial GSM radio network as its primary communication framework.

After successfully establishing a secure communication channel with the central system, the mobile router will start transmitting measured data provided by the sensors. The first step is to encrypt this data using the shared session key. However, according to the LOTUS project requirements, at least 24 hours of measurements

must be stored in the data unit as well. In order to do this, file – metadata is created for each data collection that is to be stored containing information like time period of measurement, type of sensor, etc. All encrypted data collections along with their file metadata are stored locally to the data unit of the mobile router. File metadata is used for distinguishing stored information upon request from the central system. Since it is also encrypted, the only way for finding or recovering blocks of data is via the symmetric session key.

The exact details of secure storage envisioned by LOTUS will appear in an accompanying document, the LSCS System Requirements and Design Specification report, which will be updated regularly as implementation proceeds. Here we highlight the general characteristics of the system that will guide us in the developmental process. In the next section, we proceed to capture the *threats that apply in the context of communication between a LOTUS mobile router and the central system*.

2.6.1 Threat modelling

Threat modeling allows one to systematically identify and rate the threats that are most likely to affect a system. By identifying and rating threats, appropriate countermeasures can be developed, starting with the threats that present the greatest risk.

In this section we will analyze threats by breaking them into three broad categories: Network, host and application threats. Then we perform a practical risk analysis, focusing only on the networking part. This analysis is not intended to be perfect or comprehensive; however it will give us an idea of the various issues involved in designing a secure communication channel.

Application threats target the application itself, however vulnerabilities found in an application may allow a malicious user to produce an exploit at the network or host level. Typical application threats include buffer overflows, cross-site scripting and SQL injection attacks; Brute force and dictionary attacks; Elevation of privileges, disclosure of confidential data and data tampering; Unauthorized access to administration configuration data, lack of individual accountability, etc. To defend against these types of threats a proper framework is needed that considers the categories where security mistakes are most frequently made and can be used to provide guidance to application developers. This framework may include guidelines for input validation, authorization and configuration management, parameter manipulation, auditing and logging, and so on. Using this framework, one can focus consistently on the key design and implementation choices that mostly affect the application's security and avoid common pitfalls that arise during the development process.

Host threats are directed at the system software upon which applications are developed. Top host level threats may include: Infection by viruses, Trojan horses, and Worms; Profiling for system and OS details; Password guessing and cracking; Arbitrary code execution and unauthorized access; Denial of service attacks, etc. Typical defenses against each threat category may include such things as i) Stay tuned with the latest operating system service packs and software patches. ii) Block all unnecessary ports and disable unused functionality including protocols and services. iii) Harden weak, default configuration settings. iv) Use TCP/IP and IPSec filters for defense in depth. v) Configure applications, services, and operating system with denial of service in mind, and so on.

Finally, the development of secure web applications relies heavily on the existence of a secure networking infrastructure. The role of a secure network is not only to protect itself from TCP/IP-based attacks, but also to ensure the integrity of the traffic that is been forwarded. Top *networking threats* are based primarily on the following common pitfalls:

- *Using security mechanisms that rely solely on the IP address of the sender.* This may open the way for impersonation attacks since it is relatively easy to send IP packets with a fake source IP address (IP spoofing).
- *Passing session identifiers or other identifying information over unencrypted network channels.* This can lead to IP session hijacking attempts.
- *Passing clear text authentication credentials or other sensitive data over unencrypted communication channels.* This could allow attackers to get access control credentials or obtain and possibly tamper with other sensitive data items.

In what follows, we examine in more detail the following network level threats: Scanning, Sniffing, Spoofing, Session hijacking and Denial of service (DoS) and some of their known variants along with typical countermeasures.

- *Scanning:* Network devices can be discovered and profiled in much the same way as other types of systems. This is done by sending probing packets that scan the network for IP addresses and services running on specific hosts. Attackers usually start with host scanning attacks that help identify IP addresses with active hosts using them. Once the hosts have been enumerated, the next logical step is to identify services running on potential victims. This is achieved by port scanning techniques. Armed with this information, an attacker may target known vulnerabilities that may have not been addressed with security patches.
 - Typical countermeasures include configuring routers and firewalls to prevent responses to scanning requests, and operating systems to disable unused protocols and unnecessary services (ports).

- *Spoofing or Impersonation* is a means of hiding one's true identity on the network. There are two main reasons for doing this. One is to hide the identity of the attacker and hence the original source of the attack by crafting packets with fake source addresses. Another is to exploit the trust between hosts; an attacker may convince a potential victim to accept a malicious packet if it appears to be coming from a trusted source.
 - Countermeasures include the use of proper filtering at the firewall level to reject outgoing packets that do not contain a local IP address, or incoming packets with a local IP address.

- *Sniffing or Eavesdropping*: This is the most basic attack that can be applied to any protocol in which the attacker captures information sent in the protocol messages. Eavesdropping is considered a *passive* attack in which the attacker simply listens to communications, as opposed to active attacks where the attacker can modify, replay, and inject messages. Using a simple packet sniffer, an attacker can easily monitor traffic on the network for data such as plaintext passwords or configuration information.
 - The basic countermeasure in this case is to encrypt communication fully, including authentication credentials and requests. This is the approach followed by the LOTUS Secure Communication Scheme (LSCS). After an initial handshake, a secure communication channel is created where all data is transferred in a secure and authenticated way. This prevents sniffed packets from being usable to an attacker.

- *Intruder-in-the-middle / Impersonation*: In this attack, an adversary pretends to be one of the entities that wish to communicate. The adversary is considered an intruder since he/she modifies, deletes and usually generates totally new messages by sitting in between the communicating entities.
 - Countermeasures to help prevent Intruder-in-the-middle and impersonation attacks include the use of encrypted session negotiation and the use of encrypted and authenticated communication channels. In the case of the LOTUS system, this will guarantee that mobile routers will be the only ones communicating with the central system and vice versa.

- *Session Hijacking*: Session hijacking occurs when the attacker captures an authentication token and takes control of another user's session.
 - Countermeasures include transmission of authentication tokens over secure channels.

- *Replay*: In this attack, an adversary uses the information captured by a single protocol execution to the same or different protocol participant. The adversary repeats this execution to gather information about the long-term key, encryption algorithms, random number generators, etc.
 - Countermeasures include the use of freshness identifiers (random number and timestamps) along with authentication data that ensure the freshness of exchanged messages.

- *Denial of Service*: In a denial of service (DoS) attack, the attacker prevents legitimate users to have access to a service by either using up all the computational resources of the server or by exhausting the number of allowable connections to the server. The SYN flood attack is a common example of a network level denial of service attack. The attack exploits a potential vulnerability of TCP allowing an attacker to send more requests to a server than it can actually handle.
 - Countermeasures to prevent denial of service include i) hardening the TCP/IP stack by applying appropriate mechanisms to ensure that the connection queue is never exhausted, ii) use appropriate forms of authentication to quickly differentiate between legitimate connections and DoS attempts.

At this point we will *rate* the threats based on the risks they pose. This will allow us to identify the threats that present the greatest risk. In particular, we will derive a risk rating (by assigning High (3), Medium (2), and Low (1) values) for a given threat by asking the following questions:

- *Discoverability*: How easy is it to find the vulnerability?
- *Exploitability*: How easy is it to launch an attack in terms of skills and resources needed?
- *Stealthiness*: How difficult would it be to detect the attack?
- *Reproducibility*: How easy is it to reproduce the attack?
- *Damage potential*: How great is the damage if the attack succeeds?

The result of this analysis along with possible countermeasures is shown below:

#	Description of Threat	Risk Score	Potential Damage	Discoverability	Exploitability	Stealthiness	Reproducibility
1	<i>Scan</i> : Detect network topology, use port scans to detect open ports	8	2	2	2	1	1
2	<i>DoS</i> : SYN flooding, Smurf, other low-level attacks.	10	2	3	3	1	1
3	<i>DoS</i> : upload GBs of data to take up all free space.	8	2	2	2	1	1
4	<i>Auth</i> : guess username and password.	10	2	2	2	1	3
5	<i>Auth</i> : impersonate other communicating entity	10	2	2	2	2	2
6	<i>Auth</i> : trick VPN into using a less secure auth protocol.	10	2	2	2	2	2
7	<i>Auth</i> : spoof hacker's source IP/MAC address to bypass firewall.	12	3	3	2	3	1
8	<i>Auth</i> : bypass requirement to authenticate at all.	10	2	2	2	2	2
9	<i>Auth</i> : use malware on users' computers to steal passwords.	10	2	2	3	2	1
10	<i>Auth</i> : Unauthorized access due to lack of policies to permit host connectivity	12	3	3	2	3	1
11	<i>Hijack</i> : hijack live web connections	7	2	2	1	1	1
12	<i>Sniff</i> : sniff credentials in transit over network.	11	3	2	1	3	2
13	<i>Disclosure</i> : monitor confidential data transmitted over network.	12	3	2	3	2	2
14	<i>Disclosure</i> : crack SSL encryption on sniffed HTTPS packets.	7	2	1	1	2	1
15	<i>Disclosure</i> : crack encrypted credential data like passwords.	7	2	1	1	2	1
16	<i>Disclosure</i> : extract keys from VPN servers.	8	2	2	1	2	1
17	<i>Replay</i> : capture and replay packets for a transaction.	10	2	2	2	2	2
18	<i>Tampering</i> : parameter manipulation	9	3	1	2	1	2

Countermeasures

- 1 Use firewalls to mask services that should not be exposed
- 2 Filtering, patching and updating of system software
- 3 Ensure authenticity of access requests and downloads
- 4 Enforce the use strong passwords, check for weak passwords
- 5 Use of strong authentication techniques. SSL certificates
- 6 Restrict options. Use strong algorithms. Make good security the default
- 7 Firewall dependent. Use of proper filtering rules
- 8 Always ask for authentication
- 9 Use of antivirus/antispyware software
- 10 TCP/IP filtering of IPSEC policies to restrict host connectivity
- 11 Ensure liveness of participants. Establishment of secure channels using unpredictable nonces
- 12 Establish a secure communications channel before sending data
- 13 Secure the communications channel. Use of SSL/IPSec or similar
- 14 Use of strong encryption
- 15 Use of strong encryption
- 16 Restrict access to server. Use of strong passwords and keystores
- 17 Ensure freshness of messages transmitted. Disregard replayed ones
- 18 Enforce integrity. Use of Digital signatures and Message Authentication Codes

Communication Channel Considerations

Sensitive application data and authentication credentials that are sent to and from the central system should be encrypted to provide privacy and integrity. This mitigates the risk associated with eavesdropping and tampering. This threat is not considered to be negligible in the LOTUS environment for two reasons: i) there may be a very increased incentive for making the LOTUS data unintelligible or useless, and ii) the application is not located in a closed and physically secured network.

In anticipation of the capabilities of the attackers, we will abide by the model of Dolev and Yao [DY81] that is accepted as the standard threat model for the development of all cryptographic protocols. In that model, the attacker (Eve) can do the following:

- *She is able to eavesdrop on all messages sent in the cryptographic protocol. In particular, she can obtain any message passing through a network.*
- *She is able to modify all messages sent using any information available. In addition she can re-route any message to any other protocol participant. This includes the ability to generate and insert completely new messages and thus send messages to any protocol participant by impersonating any other entity.*
- *She can be a legitimate user of the network, and thus she can initiate a protocol with any other user. Additionally, she will have the opportunity to become a receiver in any protocol interaction.*
- *She can start any number of parallel protocol runs between protocol participants, including different runs with the same protocol participants.*

Thus in the Dolev-Yao threat model, any message sent to the network is considered to be available to Eve. Additionally, any message received from the network must be treated as if it have been sent by Eve (perhaps in an attempt to fool protocol participants). However, there are some things that Eve cannot do. In particular,

- *Eve cannot guess a random number (nonce) when drawn from an appropriate large space.*
- *Without the correct decryption key, Eve cannot recover a plaintext from a given ciphertext or create a valid ciphertext from previous ones.*
- *Eve cannot recover the private key matching a given public key.*
- *Eve cannot control private areas of the computing environment (e.g. the memory of device engaging in a cryptographic operation).*

2.6.2 Issues to be addressed

Our proposed security framework will make use of lightweight security mechanisms to establish a secure channel between a mobile router and the central system using the techniques developed in Section 2.5.4. Additionally, an effort will be made to enable selection of the “lightest” set of algorithms and parameters for the current context, in order to avoid too strong security and the consequent wasted resources when not needed. We also propose the use of the “lightest” algorithms and implementations for equivalent levels of security. This is explained in more detail below.

The ‘security level’ will determine the mechanisms and protocols that are used to provide authentication, encryption, message freshness and integrity for each request. The security level may be ‘High’, ‘Medium’ or ‘Low’.

- *Low Level* – provides non-privileged services and allows exchange of non-sensitive data
- *Medium Level* – provides for a good level of protection, even if the data exchanged is not necessarily sensitive
- *High Level* – provides privileged access to service and/or exchange of highly sensitive data.

In terms of lightweight security algorithms and implementations, what is needed is algorithms and implementations specifically tailored to the constraints of the particular environment, as well as flexibility within these to allow different levels of security to be applied (and resources to be saved).

The networking environment typically requires four main types of security algorithms: encryption; integrity; message freshness; and authentication. As a general principle, we propose to make use of symmetric key cryptographic algorithms where possible. From several experimental evaluations of security protocols, it is known that symmetric cryptographic operations are between two to three orders of magnitude less costly than asymmetric cryptographic operations.

Where asymmetric cryptography is required, its use will be kept to the minimum possible, perhaps for an initial key establishment phase. Alternatively one could make use of *elliptic curve* cryptography. Similar evaluations to those for symmetric cryptographic algorithms show that for asymmetric operations, elliptic curve cryptography is considerably less costly than traditional public key cryptography [WRZ06]. This is both in terms of processing power required and the size of messages.

Encryption

The approach adopted in this framework for introducing flexibility in the encryption process is to use combinations of different parameters of AES. The assumption is that the shorter the key lengths are and the smaller the number of rounds is, the weaker the algorithm is, but fewer resources are required to perform the encryption. The three security levels are introduced as a combination of block length, different number of rounds and different key length. Another option is simply to either use encryption or to not use it at all in order to provide a difference between the security levels.

Integrity

We propose that symmetric key based Message Authentication Codes (MACs) are used to provide efficient integrity protection to data. A CBC-MAC based on AES could be used, and in this case a similar technique of varying the number of rounds and key length could be used to vary the level of strength in the integrity algorithm (and resources used) because MAC size depends on the length of the blocks that the data is split to when applying the algorithm.

Hash based MACs, such as one based on SHA-1, could be used instead. In this case, there is no scope to vary the strength of the algorithm used. However, to provide a lower level of security, the output could be truncated from its usual 160 bits (for SHA-1) to something shorter (e.g. 64 bits or 128 bits), which would reduce the size of messages.

Finally, an option is simply to either use integrity protection or not as a way of varying the security level.

Message freshness

This is usually provided through either sequence numbers or timestamps. Sequence numbers generally provide stronger security than timestamps, particularly in an environment where clocks may not be accurate. However, they require maintenance of state on the nodes. Whether or not use of sequence numbers or timestamps is more resource efficient is not obvious, and is likely to be highly specific to a particular scenario.

However, not using either will be more resource efficient and therefore the security level could best be varied in this way rather than changing between timestamps and sequence numbers.

Authentication

In any networking system, there are two types of authentication that may be needed. Pairs of nodes may need to authenticate each other for point-to-point communications (pairwise authentication), or for group communications the source

of the data may need to authenticate itself (source authentication, not applicable in this context).

For *pairwise authentication*, this may either be achieved through the use of a PKI (i.e. asymmetric cryptography with certificates) or through the use of shared secret keys that have been distributed in some secure way (e.g. manually, a key pre-distribution scheme, use of a trusted third party etc).

In terms of resources used on the node, it will generally be the case that the use of a PKI will be more resource intensive than shared secret keys. However, this may depend on the scheme used to distribute secret keys. For example, if a trusted third party is used, then the increased communication required may outweigh the processing power saved from not needing asymmetric cryptography. Possible schemes which could be used to distribute secret keys will likely depend on the particular scenario and how practical the different schemes are in it.

In terms of security, this will again depend on the particular shared secret key distribution scheme used. There need be no practical difference in the security provided by the PKI approach compared to the use of shared secret keys. Again, this may depend on the precise key distribution scheme used, but this will be highly scenario dependent. Having said all that, it may be that varying the security level and saving resources is possible. For example, the only practical choice for shared secret distribution could be a common network specific key that is manually provided to all nodes, which they may then use to authenticate each other.

This uses minimal resources and for low security situations may be sufficient. However, the nodes could switch to using a PKI if the context changes to require high security. Note that if the PKI approach is to be used, then *elliptic curve cryptography* should be considered, such as Elliptic Curve Diffie-Hellman key agreement, due to its relative efficiency as discussed above. Also, note that authentication cannot be switched on or off independently. If encryption or integrity protection is required, then so is authentication, and if authentication is required, then generally speaking so is encryption or integrity protection.

These ideas are summarized in table 2-18.

Table 2-18: Providing different security levels

Level of Security	Low	Medium	High
Encryption	AES, 128-bit block length, 128-bit keys 1024 bits for public keys	AES, 128-bit block length, 128-bit keys 2048 bits for public keys	AES, 128-bit block length, 256-bit keys 4096 bits for public keys
Integrity protection	128-bit MACs based on block cipher in CBC mode or hash function (MD5)	128-bit MACs based on block cipher in CBC mode or hash function (SHA256)	256-bit MACs based on block cipher in CBC mode or hash function (SHA512)
Freshness	Relative freshness through message sequence numbers	Strong freshness through timestamps	Strong freshness through timestamps
Authentication services	Symmetric network and group keys	Key establishment with pre-configured public keys and key confirmation. Alternatively, symmetric link keys for pairwise authentication	Key establishment with pre-configured public keys and key confirmation

3 Secure Storage

The important of secure storage in LOTUS stems from the fact that mobile routers must be able to store measurements locally for 24 hours. This chapter will provide an overview of storage solutions and will discuss in some detail the requirements for secure storage set forth in LOTUS.

3.1 Introduction

One of most common causes of unauthorized release of personal information appears to be lost backup tapes and stolen laptops. Sensitive information contained on tapes that travel to off-site storage is at risk and many organizations fail to protect sensitive information stored in these tapes. Additionally, laptops contain confidential information (documents, presentations, emails, network access credentials, etc.) and the value of this information may outweigh the value of the laptop hardware. Thus, with access to the right hardware and software, retrieving information (credit card numbers, health data, passwords, account IDs, banking information, social security numbers, etc.) becomes a trivial task. This is magnified even further by the fact that most organizations focus on securing the network perimeter while largely ignoring internal network protection. Thus once an attacker penetrates an organization's network, she can usually take her pick of databases, files, or email message stores she would like to browse. Additionally, sensitive data fall prey to internal attacks caused by disgruntled employees or individuals that make honest mistakes that yet cost an organization money, customers, or investors. *Using encryption to hide information not required for day to day activities is usually a good way to minimize exposure to employee activities.*

Penalties for not preventing accidental or malicious exposure of sensitive information go beyond fines associated with regulatory constraints. An organization could also experience negative publicity, public loss of confidence, eventually leading to business failure. Facing these issues, what should be done to protect data at risk, for both security and liability reasons?

3.2 Challenges and guidelines

Encrypting data at rest seems to be the answer. Encrypting stored data, however, can be expensive and often encryption may not be welcomed by relevant parties. Some of the reasons that have deterred the use of encryption for everyday use are outlined below [SPICE]:

- *Performance degradation:* Adding encryption when saving files and decryption time when accessing files incurs an extra overhead. However, on current

processors, the impact of encryption/decryption is negligible and need to be considered only in CPU constrained devices such as mobile phones, PDAs, sensor nodes, etc.

- *Data loss*: Data becomes irretrievable if the key or pass phrase, known only to the user, is forgotten or lost. In principle the use of portable computing devices as storage for Restricted or Critical information should be avoided. However, the risk of data destruction on a *portable computer* is an acceptable risk, and is not a defensible argument for failing to deploy encryption on such devices.
- *User inconvenience*: Use of encryption adds another password prompt between the user and their data. This is still a trade-off for improving security of data.
- *Encryption key management*: A key management procedure is needed to associate encrypted files with keys for accessing those files. This also includes Management of encryption systems, certificates, passwords, etc.
- *Complexity*: Configuring encryption may still be too challenging for average end users since this may entail a combination of different products, and the same products may not work across all the platforms a user uses (PDA, business laptop, home computer, etc.)

In general encrypting all data seems like overkill, since one must be looking at the value of doing so or at how the encryption process fits into an overall data protection strategy. Some useful guidelines for deciding what to encrypt and when to encrypt it are outlined below [Mog05]:

- *Encrypt data that moves (physically or virtually)*: If data is subject to physical or virtual loss, it should be encrypted. Encryption protects data that is subject to loss even with strong access controls. This includes
 - Laptops,
 - Tapes,
 - E-mail with sensitive information – The store-and-forward nature of e-mail makes it is highly subject to inappropriate disclosure throughout its lifetime. Sensitive e-mails containing private health information, nonpublic financial information or trade secrets should be encrypted despite the difficulty, or distributed through a more secure method.
 - Enterprise data exchange – Transmission of sensitive information to business partners or customers is another example of when data should be encrypted. In this case, creating a secure communication channel is usually more appropriate than encrypting the source data files if the endpoints are secure
 - Other systems subject to physical loss – If servers or storage devices are in a physically insecure location and subject to physical loss, encryption will protect that data in the event of theft.

- Portable storage devices – Flash drives, portable hard drives, or even CD-ROMs and DVDs with sensitive information are easy to lose and may need to be encrypted.

- *Encrypt for separation of duties:* In many cases access controls are not granular enough to protect sensitive data from certain kinds of access. For example, if a user has access to a customer or employee record, she can view all information about that person and not just specific fields. Encrypting information in these fields is one way to prevent access only to authorized people. Encryption is also effective to restrict access in shared repositories without having to restructure the access controls for the entire repository.

3.3 Existing solutions

In this section we highlight some of the solutions proposed to date starting from simpler to more advanced ones that deal with distributed storage and access of data [SSS07].

Full Disk Encryption

Full disk encryption (FDE) can be used to encrypt the entire drive, including the operating system, applications and data files. Access to data is permitted only after successful authentication to the FDE solution. FDE works by redirecting a computer's master boot record (MBR) to a special pre-boot environment (PBE) that controls access to the computer. The MBR determines what software (e.g. OS) will be executed when the computer boots from the media. However, with FDE this action is deferred until the PBE first authenticates the user. Once this is done, encryption and decryption is transparent to the user.

The FDE functionality can also be embedded into hardware (e.g. disk controller) in a way that encryption code and authentication values are stored securely in the hard drive. This way, both functions can be performed by the hard drive with no OS participation, thus leading to very little performance overhead. Another difference between software and hardware-based FDE solutions is central management of keys and recovery for software-based solutions versus local management for hardware-based ones. On the other hand, hardware solutions provide better protection of cryptographic keys since cryptographic processing does not rely on the OS and computer memory thus protecting the keys from exposure.

Virtual Disk Encryption

With virtual disk encryption (VDE) a file is created (virtual disk) that can hold ordinary files and folders. Access to this virtual disk is allowed only after authentication is performed. Once this is done, the software will automatically encrypt or decrypt files for the user. A benefit of using VDE as opposed to FDE is that backup of sensitive data is trivial (virtual disk is simply copied to a backup facility) and mobility is supported (the virtual disk can be moved to another computer provided authentication and encryption utilities exist in the form of executables)

File or Folder Encryption

File or folder encryption is the process by which individual files or folders are encrypted, permitting access to them only after authentication is performed. This is the simpler method of the three. When a user attempts to open an encrypted file/folder, the software will attempt to authenticate the user. Once this is performed, the chosen file will be decrypted. File/folder encryption does not provide any protection for data outside the specified files including swap files that may contain data while the file was held in memory. Also this method does not protect the confidentiality of filenames and other metadata which could leak some information to the attacker.

Networked Storage Solutions

The first generation of storage consisted of data that was stored directly on servers and came under the name Direct Access Storage (DAS). DAS solutions suffered from several limitations on issues such as provisioning, maintenance and scalability. Furthermore, as organizations moved towards distributed access, centralized data storage became a key challenge. This created the need to distribute storage across a network and led to cheaper and more reliable storage systems in the form of Network Attached Storage (NAS) and Storage Area Network (SAN) solutions.

The Storage Networking Industry Association (SNIA) defines NAS as a set of storage elements that connects to a network and provide file access services to computer systems. A NAS storage element consists of an engine, which implements the file services, and one or more devices, on which data is stored. On the other hand, a SAN is a network whose primary purpose is the transfer of data between computer systems and storage elements and among storage elements. A SAN consists of a communication infrastructure, which provides physical connections and a management layer which organizes the connections, storage elements and computer systems so that data transfer is secure and robust [SNIA].

These technologies offer advanced features for device and user authentication, however NAS and SAN implementations can be very complex and a full description of these technologies is beyond the scope of this report (for more information the reader is referred to [Poc03] and [CB03]).

3.4 LOTUS requirements for secure storage

LOTUS is an early warning system for chemical substances in which sensor data is to be exchanged between mobile sensors and a central system. Figure 3-1 illustrates a high level view of LOTUS Secure Communication Scheme (LSCS). Mobile routers are equipped with a box holding a GPS, a data processing unit, a radio module unit and different sensors (IMS, DMA and SERS sensors). As highlighted, the basic functionalities of the LSCS system are to encrypt measured data, store them to the processing data unit (along with a file-metadata) and send them to the central system. Data are provided by the sensors to the data unit where some processing is done before transmitting the encrypted raw data readings using a communication framework.

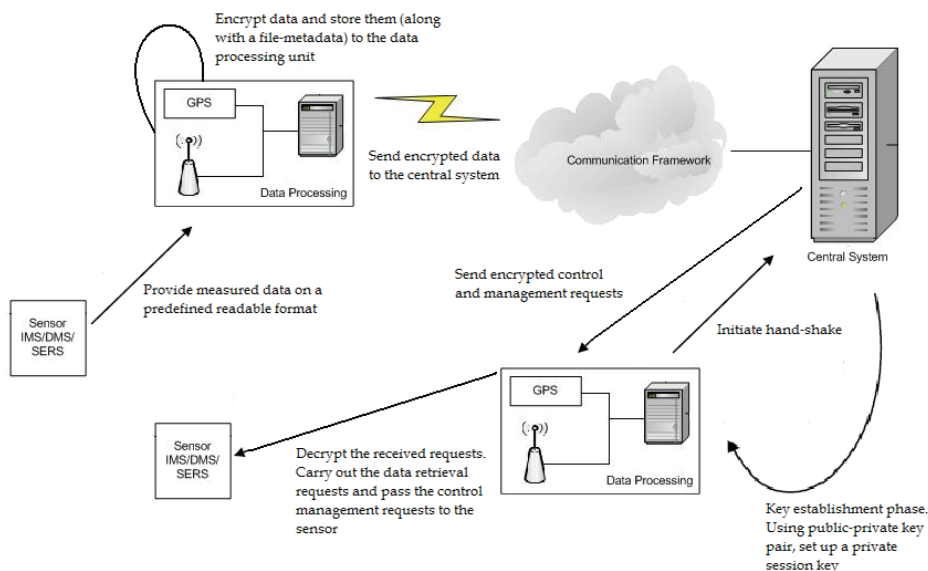


Figure 3-1: High level view of the LOTUS Secure Communication System

After successfully establishing a secure communication channel with the central system, the mobile router will start transmitting measured data provided by the sensors. The first step is to encrypt this data using the shared session key. However, according to the LOTUS project requirements, at least 24 hours of measurements must be stored in the data unit as well. In order to do this, file - metadata is created for each data collection that is to be stored containing information like time period of measurement, type of sensor, etc. All encrypted data collections along with their file metadata are stored locally to the data unit of the mobile router. File metadata is used for distinguishing stored information upon request from the central system. Since it is also encrypted, the only way for finding or recovering blocks of data is via the symmetric session key.

The exact details of secure storage envisioned by LOTUS will appear in an accompanying document, the LSCS System Requirements and Design Specification report, which will be updated regularly as implementation proceeds. Here we highlight the general characteristics of the system that will guide us in the developmental process.

3.4.1 Threat Modelling

Our focus in this section is to consider the threats related to securing data held locally at the *mobile router*. We will examine four main threats (Disclosure of sensitive data, Unauthorized access, Poor cryptography and Network eavesdropping) and propose relevant countermeasures.

- *Disclosure of Sensitive Data*: The LOTUS mobile router will store measurements of the sensors that may contain sensitive data, such as findings of illegal substances, etc. It is essential to protect the privacy and integrity of this type of data. Major vulnerabilities that can lead to the disclosure of sensitive information include:
 - Storing data with no encryption
 - Weak authorization
 - Weak encryption

Countermeasures: To prevent the disclosure of confidential data, data must be secured in persistent stores such as databases and configuration files, and during transit over the network. The use of strong encryption seems to be a must. Finally, only authenticated and authorized users should be able to access the data that is specific to them. Access to system level configuration data should also be restricted to administrators. The selection of strong passwords and the creation of a minimal set of accounts in the mobile router must be enforced by the administrator.

Description of Threat	Total Risk Score	Potential Damage	Probability of Threat			
			Discoverability	Exploitability	Stealthiness	Reproducibility
Disclosure of sensitive data	9	1	1	3	2	2

- *Unauthorized Access*: With inadequate authorization, malicious users may be able to access, modify or deleted restricted data. Vulnerabilities that can allow unauthorized access include:
 - Lack of authorization in data access code providing unrestricted access
 - Over-privileged accounts

Countermeasures: To prevent unauthorized access, one must use restricted Access Control Lists on the persistent data stores that contain sensitive data. As an extra protection measure, data must be encrypted. Finally, the use of identity and role-based authorization is considered a nice practice to ensure that only the user or users with the appropriate level of authority are allowed access to sensitive data. It also allows different views on data (e.g. separation of views between users who can see data and those who can modify data).

Description of Threat	Total Risk Score	Potential Damage	Probability of Threat			
			Discoverability	Exploitability	Stealthiness	Reproducibility
Unauthorized access	9	2	1	3	2	1

- *Poor cryptography*: Most applications use cryptography to protect data and to ensure it remains private and unaltered. Top threats surrounding the application's use of cryptography include poor key generation or key management, weak encryption and checksum spoofing.
 - *Poor key generation or key management*: Attackers can decrypt encrypted data if they have access to the encryption key or can easily derive the encryption key. Attackers can discover a key if keys are managed poorly or if they were generated in a non-random fashion.
 - *Weak encryption*: An encryption algorithm provides no security if it is vulnerable to brute force attacks. Home brew algorithms are particularly vulnerable if they have not been tested. Instead, one should use published, well-known encryption algorithms that have withstood years of rigorous attacks and scrutiny.
 - *Checksum spoofing*: Simple checksums do *not* provide for integrity. For example, one should not rely on plain hashes to provide data integrity for messages sent over networks since an attacker may change both the message and the hash value. Instead one should use cryptographic checksums (i.e. Message Authentication Codes) that provide a high level of integrity.

Countermeasures to address these threats include: Use of built-in encryption routines and secure key management. Use of strong random key generation functions that includes storage of keys in restricted locations. Regular expiration and update of keys. Use of proven cryptographic services provided by a platform. Use of HMAC for increased authentication services and integrity.

Description of Threat	Total Risk Score	Potential Damage	Probability of Threat			
			Discoverability	Exploitability	Stealthiness	Reproducibility
Poor cryptography	5	1	1	1	1	1

- *Network Eavesdropping*: The deployment architecture of most applications includes a physical separation of the data access code from the data server. As a result, sensitive data such as application-specific data or login credentials must be protected from network eavesdroppers. Top vulnerabilities include:
 - Clear text credentials passed over the network during authentication.
 - Unencrypted sensitive application data sent to and from the data server.

Countermeasures to limit vulnerabilities to network eavesdropping include: Use of authentication to avoid sending credentials over the network. Installation of server certificate on the data server that can be used in the automatic encryption of transmitted credentials over the network. Use of SSL connections or IPSec encrypted channels.

Description of Threat	Total Risk Score	Potential Damage	Probability of Threat			
			Discoverability	Exploitability	Stealthiness	Reproducibility
Network Eavesdropping for sniffed credentials and application data	10	2	2	2	2	2

Conclusions

The main threat that all storage technologies mitigate is *unauthorized access to information on a lost or stolen device*. Virtual disk encryption and file/folder encryption technologies can also mitigate some OS and application layer threats (malware, remote access, etc.) until the user authenticates to the software solution. However, once this authentication occurs, any process (including malware) can get decrypted data. In general, it makes sense to restrict this window of opportunity by encrypting only the absolutely necessary files. The more files are protected, the sooner the user is likely to authenticate which increases the window of exposure for decrypted files.

In the case of LOTUS, local storage at mobile routers serves mainly as secure backup in case the communicated data never reaches the central system. In this case the central system may request some of this data to be resent by the mobile router. It is for this reason that data must also be retained by the mobile router for 24 hours in encrypted form. *However, one should be aware that even if secure encryption technology is implemented there may still exist residual data on the device that remains unprotected. For example, when a file is encrypted and the original file is deleted, remnants of the original file may still exist on the local disk. These remnants may still be retrievable by an attacker who uses forensic tools even if no authentication has been performed. The same is true for the keys that reside in memory and are used to encrypt the files or the communicated data. For similar reasons, these are also easily recoverable. Our threat model does not include these cases as these are beyond the capabilities that must be provided by the LOTUS system.*

Essentially, anybody can have access to the mobile router since these are not tamper resistant devices. However, only the central system should be able to read the data collections that are kept in encrypted form. Hence the main threat that is covered is *unauthorized reading of communicated data that reside on a mobile router that has been lost or stolen*.

3.4.2 Issues to be addressed

The secure storage capabilities that will be provided by LOTUS include the following (details about the exact mechanisms will appear in the accompanying LSCS System Requirements and Design Specification report):

- Configuring storage encryption software, such as specifying encryption/decryption and integrity verification algorithms. Typical cryptographic primitives will include strong encryption algorithms such as AES and strong integrity algorithms such as SHA1 and SHA256.
- Ensure that security features don't result in performance degradation. An effort will be made to avoid double encryption by storing encrypted data the moment they are transmitted to the central system.
- Possibility of recovery of stored information from device failure. When data communications are encrypted and the key becomes corrupted, the original data can usually be re-sent. When data is at rest, however, the loss or corruption of keys can mean a loss of data. In general, however, no provision will be made for hardware-based key storage and key backup or escrow as this is beyond the scope of the current project.
- Authenticate access to storage device and from management or administration device to storage device will be handled by the central system and the underlying OS.
- Managing storage encryption authenticators and cryptographic keys. In particular, encryption and integrity protection algorithms must be selected, as well key strength for algorithms that support multiple key lengths. Also one or more cryptographic keys can be used to encrypt the stored data on the mobile router. Therefore, storage encryption will have to consider various aspects of key management such as
 - key generation
 - use
 - storage recovery, and
 - destruction

In addition to key management, there are several other aspects of cryptography that need to be considered. Creating a cryptography policy involves choosing appropriate encryption and integrity algorithms and key lengths. For example, AES should be used for encryption because of its strength and speed. Similarly, for integrity checking one can use HMAC-SHA, Cipher-based Message Authentication Code (CMAC) or Counter with Cipher Block Chaining-Message Authentication Code (CCM). An effort will be made to choose among the most efficient algorithms to date.

4 LOTUS Secure Communication Scheme (LSCS)

The LSCS report (a companion document to this one) will be used as the steering document for requirements specification, system design and the development of the secure communication framework.

LSCS is part of the LOTUS communication system which enables the use of networking mobile sensors for reporting measurements to the central system over a radio network. It is responsible for securing all radio transmissions. Since communication via radio shall be duplex, LSCS needs to be able to handle both encryption and decryption of data.

The areas of LOTUS system that are covered by LSCS include the following:

- Key establishment between mobile sensors and the central system. This will lead to the establishment of a secure communication channel amongst them.
- Encryption/Decryption of measured data.
- Secure data storage in the data unit of the mobile sensors router.

The scope of the LSCS report thus includes:

1. System features definitions to be implemented during development phase.
2. Component and Interface requirements that must be fulfilled.

4.1 LSCS System Architecture

Figure 4-1 describes the overall architecture of LSCS system. LSCS will be integrated in the mobile router. Comprising components will be responsible for:

- **Data handling.** Measured data provided by the sensors; data storage and retrieval.
- **Security handling.** Encryption/decryption of data packets to be transmitted and decryption of received requests.
- **Network Communication.** Transmission of encrypted data and handling of received message requests.

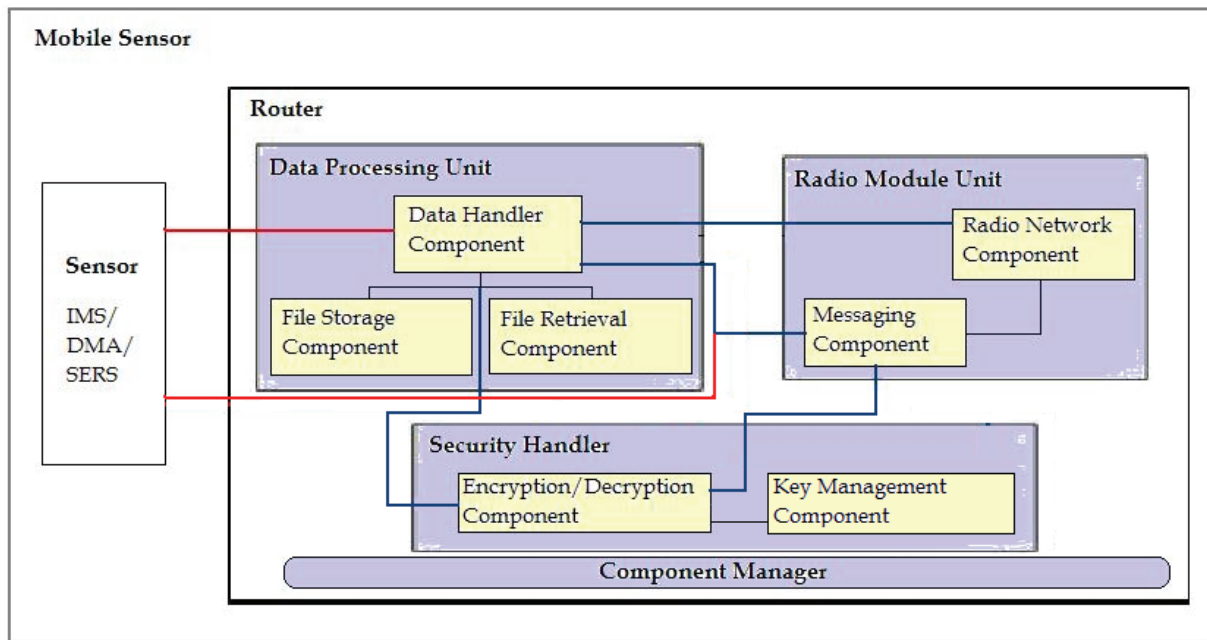


Figure 4-1. LSCS System Architecture

LSCS will be also a part of the central system for supporting:

- Reception and decryption of data packets.
- Encryption and transmission of message requests.

4.2 High-Level System Design

4.2.1 Components/Modules

In order to facilitate the quicker development of LSCS system prototype, we logically partitioned it into a set of components. As illustrated in Figure 4-2, the component design follows a layered approach.

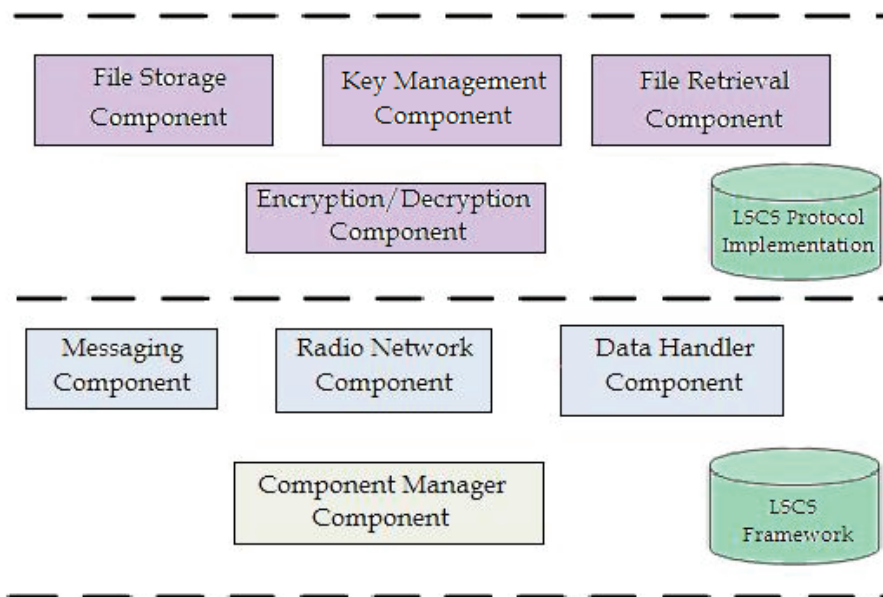


Figure 4-2. The layered components approach used for our system design. All components are maintained locally in the mobile router

At the bottom layer, we provide a set of components that essentially form a framework onto which our protocols and algorithms are built. Using the services provided by the framework components, we defined a new set of higher-level components that implement our core system functionalities. These two component layers form the LSCS system to be integrated into the LOTUS project.

4.2.2 LSCS Framework Components

The Data Handler component.

This is probably one of the most important components in our framework. This component provides the necessary glue for interfacing with the sensors embedded in the mobile router and contains code that facilitates the passing of the given measured data for encryption and storage. *The component manages the data provided by the sensors in a common and predefined format.* Also, it manages connections to the data processing unit (file storage and retrieval components) that are allocated on-demand to requesting modules. For example, if the central system sends a data retrieval request, the data handler will create a new connection to the file retrieval component. Similar steps are followed in the case of data storage.

The Messaging component

The main purpose of this component is to serve as a dispatcher of message requests that are received from the central system. Possible requests may be RESTART

commands, TRACK ON/OFF commands, DATA RETRIEVAL commands, etc. Modeled after the observer design pattern, this component provides a generic interface so that other components may register themselves to be asynchronously notified of particular types of messages that are received over the network. Control and management requests are designated for the sensors whereas data retrieval requests are designated for the data handler component. The component itself also registers a set of message classes that implement each supported request.

The Component Manager

This component is responsible for the asynchronous communication between other participating components. It maintains a list of all active components and serves as the mediator in case a component wants to communicate (eg. exchange parameters) with another component. If this is the case, all data exchange between components is done through the component manager.

The Radio Network component

This component handles secured data transmission and reception. It is responsible for connecting to the network by providing interfaces that access the underlying GSM commercial network. This component provides the necessary interfaces for sending and receiving measured data and message requests. Note that this component may use compression and/or serialization in order to ensure that data is highly packed in all outgoing messages so that network traffic is minimized.

4.2.3 LSCS Protocol Components

The File Storage component

This component handles encrypted file and metadata storage into the data processing unit of the mobile router. It is used for storing at least 24 hours of measurements to the local storage of the mobile router. Once the data is provided by the sensors and processed in the data handler (compressed and encrypted using the symmetric session key), it is forwarded to the file storage component.

Initially, this component creates a file metadata header which contains several fields that describe the given data. Such fields may be time period of measurement, type of sensor etc. Then, the metadata is encrypted using the symmetric session key and appended to the corresponding data. Finally, all measured data along with the file metadata are stored in the data processing unit.

According to the LOTUS project requirements, the required data space is 1380 MB for 24 hours – with a frequency of one measurement per 1 second and a detailed measurement size of 16 kB. Thus, this component also supports a mechanism for keeping track of the stored files and deleting the ones that exceed the required time period. The names of all the files are maintained internally in a list which gradually fills up as new data get submitted for storage. The job of the “cleaning” mechanism is to manage the submitted data storage requests. It traverses the list and checks to see whether it is full. If this is the case, it changes the status of the expired stored files to STATE_EXPIRED and deletes them.

The File Retrieval component

This component is the one that provides to the LOTUS central system the capability to request one or more locally stored files. If the messaging component receives a data retrieval request, the data handler creates a new connection with the file retrieval component which is responsible for locating the requested file data. In order to do this, it uses the list of stored file names and the created metadata. When it finds the requested data, it forwards them (along with the corresponding metadata) to the radio network component for transmission.

The Key Management component

This component is responsible for the maintenance and update of the symmetric session key shared between the mobile router and the central system. This session key is used for encryption/decryption of all network traffic. When a communication session needs to be established, this component initiates the key establishment phase in order for the mobile router and the central system to agree on a private symmetric key. This is accomplished via the help of the public key held by the CS. Once this is complete, the key management component provides this key to the encryption/decryption mechanism, through suitable interfaces, and also updates it through periodic invocations of the key refreshing procedure.

The Encryption/Decryption component

This component is responsible for providing a high level of security. It incorporates advanced encryption/decryption techniques and provides relevant interfaces. All outgoing network traffic is encrypted using the symmetric session key that is maintained in the key management component. Also, all measured data is encrypted before saving it in the data unit of the mobile router. Security measures shall prevent threats like eavesdropping, integrity violations, masquerading and traffic analysis.

5 Conclusions

Data security in the LOTUS project depends on the security of the communication and data handling framework that will allow for

- implementation of an underlying secure transmission scheme for communication between mobile sensors and the central system,
- design of a key establishment procedure for allowing various levels of security,
- definition of encryption/decryption interfaces that can be used on the measured data, and
- simple integration of mobile sensors data unit and secure data storage/retrieval algorithms.

Based on this set of requirements, we presented an overview of known techniques related to the establishment of a secure channel among two different entities using techniques that are based on both symmetric and asymmetric cryptography. We reviewed the basic notions and primitives that exist in each setting and we gave representative protocols for both key establishment and key transport that can be used as a basis for the LOTUS secure communication framework. Additionally, we covered the issues surrounding the concept of secure storage since data must be stored and maintained locally, in the mobile router, in encrypted form.

Finally, we introduced the LOTUS Secure Communication Scheme (LSCS), a framework that will facilitate the use of networking mobile sensors for reporting measurements to the central system over a radio network in a secure and authenticated manner. We presented a high level architecture of the system and a set of specific components that will enable both Data handling and Network Communications practices. Exact instantiation and more detailed description of these components will take place in a subsequent *confidential* report.

6 References

- [AN94] Martín Abadi and Roger Needham. *Prudent Engineering Practice for Cryptographic Protocols*, In IEEE Symposium on Research in Security and Privacy, pp. 122-136, 1994.
- [BCK96] Mihir Bellare, Ran Canetti and Hugo Krawczyk, *Keying Hash Functions for Message Authentication*, CRYPTO 1996, pp. 1-15
- [BM92] Steven M. Bellovin and Michael Merritt. *Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks*, In IEEE Symposium on Research in Security and Privacy, 72-84, 1992.
- [BM03] Protocols for Authentication and Key Establishment, by Colin Boyd and Anish Mathuria, Springer, 2003
- [Bro96] Charles Brookson, Information Management & Computer Security 4/2 [1996] 7-10
- [Bro01] Brookson C., *GSM (and PCN) Security and Encryption*, Technical Report, August 2001, <http://www.brookson.com/gsm/gsmdoc.doc>
- [BSW00] A. Biryukov, A. Shamir and D. Wagner, *Real Time Cryptanalysis of A5/1 on a PC*, Fast Software Encryption Workshop 2000, New York, USA, April 2000
- [CB03] John Chirillo and Scott Blaul. *Storage Security: Protecting, SANs, NAS and DAS*. Wiley Publishing, Inc. 2003.
- [CJ77] John Clark and Jeremy Jacob. A Survey of Authentication Protocols, Version 1.0, 1977. Available at: <http://www.cs.york.ac.uk/~jac/papers/drareview.ps>.
- [DH76] W. Diffie and M. E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976, pp: 644-654.
- [DOW92] W. Diffie, P.C. van Oorschot and M. Wiener. *Authentication and authenticated key exchange*. In *Designs, Codes and Cryptography*, 2:107-125, 1992.
- [DS81] D. E. Denning and G. M. Sacco, *Timestamps in key distribution protocols*. Comm. of the ACM, 24(8):533-536, August 1981.
- [DSA] Federal Information Processing Standard 186-3, Digital Signature Standard (DSS), (Revision of FIPS 186-2, June 2000)
- [DY81] D. Dolev and A.C. Yao, *On the security of public key protocols*. In Proc. of the 22nd Annual Symposium on Foundations of Computer Science, PP. 350-357, 1981.
- [ETSI] European Telecommunications Standards Institute (ETSI), *Security Related Network Functions*, Release 7, 1998.

- [ETSI-2] European Telecommunications Standards Institute (ETSI), *Security Aspects, GSM*, Release 7, 1998.
- [FIPS197] Federal Information Processing Standard 197, Advanced Encryption Standard (AES), November 2001.
- [GM84] S. Goldwasser and S. Micali, *Probabilistic encryption*. Journal of Computer and System Sciences, 28:270-299, 1984
- [HAC96] A. J. Menezes, P. C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press LLC, 1996
- [IEEE P1363] IEEE P1363 Standard Specifications for Public key Cryptography, January 2000.
- [IKE] RFC 4306: *Internet Key Exchange (IKEv2) Protocol*, Internet Engineering Task Force (IETF), 2005.
- [ISO11770-2] ISO/IEC 11770 Part 2: Mechanisms using symmetric techniques, 1996
- [LV01] Arjen K. Lenstra and Eric R. Verheul, *Selecting Cryptographic Key Sizes*, Journal of Cryptology, 14, pp. 255-293, 2001.
- [Low96] G. Lowe, *Some new attacks upon security protocols*. In 9th IEEE Computer Security Foundations Workshop, pp 162-169, June 1996.
- [Mog05] Mogull, Rich (2005). *Management update: use the three laws of encryption to properly protect data* (Gartner Research Article G00130839)
- [MQV95] A. Menezes, M. Qu, S. Vanstone. *Some new key agreement protocols providing mutual implicit authentication*. In Workshop on selected areas in cryptography (SAC'95), 1995.
- [NIST800-38] Special Publication 800-38, Recommendation for Block Cipher Modes of Operation: 38A, Methods and Techniques, December 2001; 38B, The RMAC Authentication Mode, November 5 2002 draft; 38C, The CCM Mode for Authentication and Confidentiality, May 2004.
- [NIST800-57] NIST Special Publication 800-57, Recommendation for Key Management - Part 1: General
- [NIST800-67] Special Publication 800-67, Recommendation for Triple Data Encryption Algorithm Block Cipher, May 2004.
- [NS78] R. Needham and M. Schroeder, *Using encryption for authentication in large networks of computers*. In Comm. of the ACM, 21(12):993-999, December 1978.
- [NT94] B. C. Neuman and T. Ts'o, *Kerberos: An authentication service for computer networks*, In IEEE Comm. Magazine, 32(9):33-38, September 1994.
- [Pat97] Sarvar Patel, *Number Theoretic Attacks On Secure Password Schemes*, In Proceedings of the IEEE Symposium on Security and Privacy, 236-247, 1997

- [Poc03] Anilkumar Pochiraju, *Securing Networked Storage using Defense in Depth*, SANS Institute, 2003.
- [RSA78] Rivest, R.; A. Shamir; L. Adleman (1978). *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. *Communications of the ACM* 21 (2): 120-126.
- [SHS] Federal Information Processing Standard 180-2, Secure Hash Standard (SHS), August 2002.
- [Sin99] S. Singh. *The code book*. Fourth Estate, 1999.
- [STM95] Michael Steiner, Gene Tsudik and Michael Waidner, *Refinement and Extension of Encrypted Key Exchange*. In *ACM Operating Systems Review*(29), pp. 22-30, 1995.
- [SNIA] Storage Networking Industry Association. *A Dictionary of Storage Networking Terminology*. URL:
http://www.snia.org/education/dictionary/n/#network_attached_storage
- [SPICE] SPICE (2005, December). *Encryption for data at rest*. Retrieved from <http://security.health.ufl.edu/eduguides/index.shtml>
- [SSS07] Karen Scarfone, Murugiah Souppaya and Matt Sexton. *Guide to Storage Encryption Technologies for End User Devices*, 2007
- [WGB98] Wagner, Goldberg, Briceno. *GSM Cloning*, Technical Report, 1998, <http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html>
- [WRZ06] Y. Wang, B. Ramamurthy, X. Zou, *The Performance of Elliptic Curve Based Group Diffie-Hellman Protocols for Secure Group Communication over Ad Hoc Networks*, In *Proceedings of the IEEE International Conference on Communications (ICC '06)*, 2006, pp. 2243-2248
- [Wu98] Thomas Wu, *The secure remote password protocol*. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pp. 97-111, 1998.

LOTUS has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement No 217925.

The overall objective of the LOTUS project is to develop a new anti-terrorism tool for law enforcement agencies, in the form of an integrated surveillance system for continuous chemical background monitoring with fixed site and/or mobile detectors to identify "chemical hotspots" such as bomb or drug factories.

The LOTUS project aims to create a system by which illicit production of explosives and drugs can be detected during the production stage rather than preventing terrorist attacks while they are already in motion, which is extremely difficult.

The LOTUS concept is aimed at detecting chemical signatures over a wide urban area. The detectors may be placed at fixed positions although most detectors should be mobile. These distributed detectors continuously sample air while its carrier performs its daily work. When a suspicious substance is detected in elevated amounts, information about the type, location, amount and time is registered and sent to a data collection and evaluation centre for analysis. Several indications in the same area will trigger an alert, enabling law enforcement agencies to further investigate and respond.

LOTUS is a collaboration between:

FOI | AIT | Bruhn Newtech | Bruker | Portendo | Ramem | SAAB | Secrab | TNO | Universitat de Barcelona

Coordinator
FOI, Swedish Defence Research Agency
Department of Energetic Materials
Grindsjön Research Centre
SE-147 25 Tumba
SWEDEN

Website
www.lotusfp7.eu

