

Secure Ad Hoc Networking

Carlos Viescas Huerta^{1,*}

¹Linux for Embedded Objects 2, SDU Robotics

*cavie17@student.sdu.dk

ABSTRACT

This project was developed during the course of *Linux for Embedded Objects 2*, at SDU. The project consisted in the development of a method to secure mobile ad hoc networks. The first part of the project consisted on investigating the fundamentals of mobile ad hoc networking, such as routing, security issues, security solutions...

Then, a system was designed based on combination of asymmetric and symmetric key cryptographic algorithms (see [12](#)). The system consists of an authentication protocol and a key exchange protocol based on the RSA algorithm, and a private key encryption for network traffic based on Blowfish cipher. The implementation was done in C++ and using the help of the OpenSSL library for the Blowfish algorithm. Due to problems with hardware on Raspberry Pi's, the system was tested using a socket client-server program.

Contents

1	Introduction	2
1.1	Security challenges and threat models.	2
1.2	Reading guide.	3
2	Background	4
2.1	Operating principle of ad hoc networking.	4
2.2	Routing protocols for ad hoc networks.	5
	Proactive (Table-driven) routing. • Reactive (On-demand) routing. • Hybrid routing.	
2.3	Types of threats.	6
	Routing threats. • External threats. • Internal threats.	
2.4	Cryptography.	8
	Symmetric key cryptography. • Asymmetric key cryptography.	
2.5	Secure key exchange and Authentication.	14
3	Project Implementation	17
3.1	Ad Hoc network setup.	17
3.2	Problems with ad-hoc mode and hardware.	17
3.3	Testing cryptographic algorithms.	18
	RSA algorithm implementation. • Blowfish algorithm Implementation.	
3.4	Network security system.	19
4	Tests	21
5	Discussion and Conclusions	21
6	Future work	21
	References	24

1 Introduction

Wireless ad hoc networks (also referred as MANETs), are decentralized networks in which nodes (hosts) do not depend on a central infrastructure or access point to communicate. Each node participates in routing by forwarding data to the other nodes. This implies that nodes within a radio range can share messages directly using wireless links, whereas nodes in the network that are located far away, depend on the other nodes to forward these messages. When and which nodes have to route data is determined dynamically, depending on connectivity and the algorithms underlying the network^{1,2}. A representation of this changes in the topology of MANETs is shown at figure 1. In (a), host D is reachable directly by host A, so they communicate using a direct link. However, on (b) D moves outside the communication radio of A. Messages from A reach D because nodes C, E and F act as routing points.

Ad hoc networks are suitable for many applications including military and navy communications, search and rescue, hospital networking and smart city connections. In the field of robotics, ad hoc networks can be used for anytime-anywhere communication between robots without the need of a central unit. Usually, multi-robot applications use a centralized model of communications in which each robot has to talk to the controller in turns. Using MANETs, the problem can be converted to a distributed fashion communication between all the machines. Another application in the robotic field can be the use of MANETs in a drone network to quickly share information on the go between all drones.

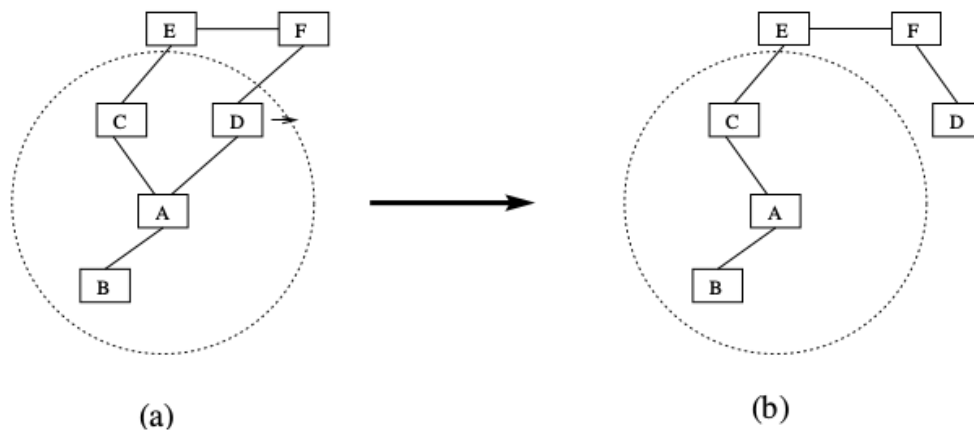


Figure 1. Change in topology in wireless ad hoc networks. This figure belongs to the article *Securing ad hoc networks*².

An special case (and major application) of ad hoc networks are **sensor networks**. These networks are built by bringing up together multiple sensors spatially dispersed. Sensor data can be transported wireless across the network towards monitoring nodes (computers). Also different sensors can have instant access to the data that other sensors have, if it is required by the application. Uses of sensor networks can cover from small-scale operations such as house or workspace security to large-scale and space purposes such as inter-city traffic monitoring or wild-life tracking, as examples.

In general, ad hoc networks are limited to very specific uses and purposes. There are many possibilities and variants of routing protocols, resource limitation and other issues concerning the development of the network. This problem causes that in most cases, security is in a lower level of priority when designing the network and, in some cases, these networks do not implement any access control or information encryption.

1.1 Security challenges and threat models.

Being able to secure ad hoc networks is critical since most of its applications involve private and often extremely confidential information. Security in this type of networking implies satisfying the following principles^{2,3}:

- **Availability:** Ensuring that all nodes have on demand access to routing information and network traffic at all times. This access to information has to be permanent even in the event of suffering a **denial of service** attack. This kind of attacks, which intend to destroy network services, can be launched on both the physical and access layers (causing interferences on the physical communication channels) or in the network layer (disrupting routing protocols and disconnecting the network).

- **Confidentiality:** Ensuring that information cannot be disclosed by anybody who is not authorized to see it, even if an attacker intercepts it. The process of hiding information is known as encryption.
- **Integrity:** Ensuring that a message transmitted across the network is never corrupted (due to network failure or attacks).
- **Authentication:** Ensuring the correct identity of the nodes inside the network. Authentication is used so that any node without specific authorization can join the network and get data from it.

1.2 Reading guide.

This document is structured as follows: Section 2 is focused on presenting and describing the fundamentals of the different concepts, algorithms, tools and methods involved in this project. Ad-hoc routing, cryptography, authentication... are topics discussed throughout the chapter. In section 3, the implementation of the project is described, being divided in subsections according to the action plan of each subpart in the project. Section 4 describes how the implementation was tested. Finally, sections 5 and 6 gives a discussion about the project and its results and possible future work to improve it. All the additional material (source code, configuration files) can be seen in⁴.

2 Background

2.1 Operating principle of ad hoc networking.

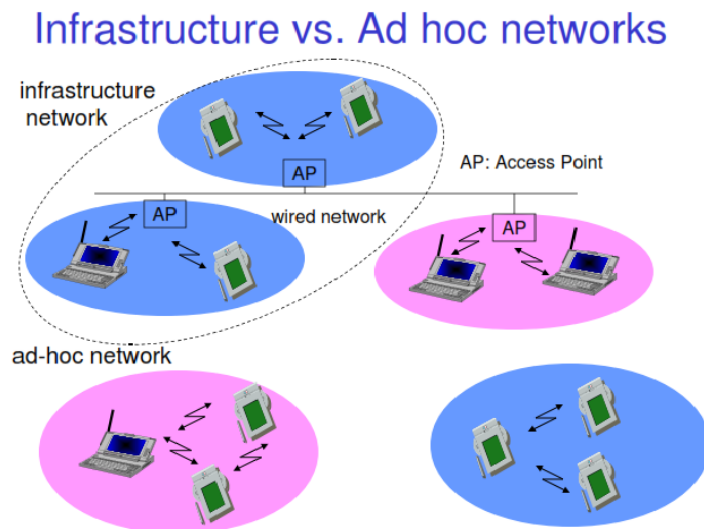


Figure 2. Representation of infrastructure networks vs ad hoc networks⁵.

The basic principle behind MANETs is directly defined by their name *ad hoc*. This latin expression has two different meanings: "impromptu", which means on the go, and "for a certain purpose". In the end, MANETs are designed to be build on the run, with limited resources (the ones available in the moment) and for an specific reason or use. These type of networks are infrastructure-less, that means, there is no central Access Point or no connection provider. In additions, some nodes (or all of them) can be mobile, thus continuously changing the topology (the routing shape) of the network. In general, nodes will be assumed to be bidirectional, that is, all nodes can receive and forward data. Networks can be heterogeneous - nodes with different characteristics, such as a combination of computers, smart-phones and Raspberry Pi's - or homogeneous (all nodes with the same characteristics, such as a network entirely made of Raspberry Pi's of the same model). Due to its characteristics, ad hoc networks must be self-configuring and adaptive, allowing spontaneous formation and deformation of the network.

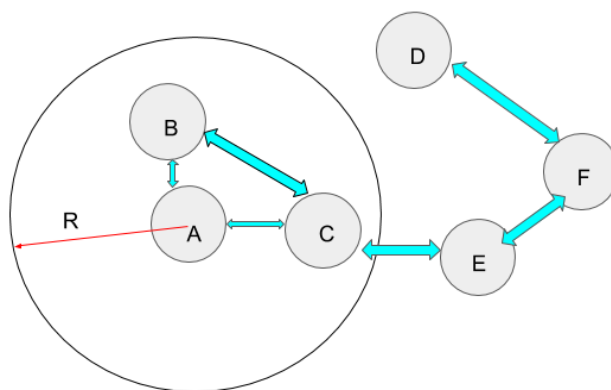


Figure 3. Bringing up MANETs.

Figure 3 shows an example of ad hoc network with different routings. In such networks, each node has a limited communication radio range. Within this range, nodes can bring up channels to communicate directly in peer-to-peer mode. If there is no such channel available, multi-hop communication is necessary. That is, a routing path between neighboring nodes has to be found in order to transmit network traffic (that is the case for transmitting messages from A to E in the figure, for example). This communication system implies intermediate nodes acting as routers.

To start an ad hoc network, at least two (for example A and B in figure 3) nodes initially have to broadcast their presence by continuously emitting packets with their address information (this step is called *beaconing*). If both nodes can verify each other's beaconing messages and establish direct connection, then their routing tables are updated and the network is created. Now, two different situations can occur when adding new nodes to the network. To begin with the simple case, we'll add a node that is reachable by both A and B: node C. Reachable means that all nodes in the network can establish a direct channel with C. In this situation, the topology updates, which consist on address and routing updates, are carried out in the three nodes A, B and C immediately and at the same time.

On the other hand, a fourth node (D) can be added, that is not reachable by all the nodes in the network. In the example of figure 3, only C can reach directly D. In this case, topology updates are carried in turns. First, the routes between C and D are updated. Then, the routes between C, B and A. Finally, the routes between C and D are updated again to confirm that A and B can communicate with D via C.

These configuration routines have to be made constantly, not only when adding/removing nodes from the network, but also when channels between nodes break and topology of the network changes due to mobility of nodes. When these changes occur, all nodes have to be aware of it and incorporate the new routing paths to their tables⁵.

2.2 Routing protocols for ad hoc networks.

A routing protocol is basically a procedure whose primary goal is to construct routing tables. A routing table is a database, stored in a router or in a networked computer, that contains a list of the paths to particular destinations in the network and distances (in some kind of metric) associated to these paths⁶. Routing tables can be considered maps of the topology of a network. To get a better idea they can be compared to route maps used in package delivery. When a node sends data to another node in the network, it checks first the destination. If there is not a direct connection to it, the message has to be sent via other nodes along a proper route until it gets to the final destination node. However, nodes are not deciding by themselves the route that sent data will follow. Instead, a node will send an IP packet to a gateway in the network, which will decide how to route the data correctly. Each gateway will need to keep track of the path used to deliver the data, and here is where routing tables come into scene. These databases store the paths, like a map, and allow gateways to provide this information to the node requesting it.

Transferring the concept of routing protocol to MANET fashion, it is possible to define ad hoc routing protocols as conventions to control how nodes will decide the way to route packets across the mobile network. The main difficulty in ad hoc networks is that here nodes are not familiar with the dynamically changing topology of the network. Instead, nodes have to discover it by detecting and adding new neighbors⁷. This particular method of packet forwarding is known as **hop-by-hop routing**.

Briefly explained, the traditional convention for packet forwarding, called **source routing**, allows the sender nodes to specify the complete route that the packet should travel across the network. This means that each sent packet will contain source and destination addresses and a list of intermediate nodes defining the path it has to take. In the routing tables, a source protocol stores, for each destination, a list of intermediate nodes that leads to it. When a packet needs to be sent, the route is extracted from the table and included in a special section of the packet itself⁸.

In hop-by-hop routing, packets only contain source and destination addresses. In the routing tables, for each destination the only address maintained is that of the next node to be visited. When a packet is received, if the destination address corresponds to that of the current node, the packet is accepted. If, however, the destination address corresponds to an address listed on the routing table of the current node, then the packet is forwarded to the next hop, whose address is taken from the table entry. If none of these two cases happen, the next hop has to be discovered dynamically using certain protocols. Today, there are three basic protocols accepted for ad hoc networking: The **reactive approach** (also known as *on-demand*). The **proactive approach** (also called *table-driven*) and the **hybrid routing approach**. There is still, however, a lot of research going on in this field, and there are many other newer and more sophisticated routing techniques for ad hoc networks.

An important consideration about these three ad hoc routing protocols is that they are designed to cope robustly with the problem of dynamic topologies of MANETs. However, they do not implement security measures against malicious attacks.

2.2.1 Proactive (Table-driven) routing.

This approach keeps nodes actively working in route discovery and maintenance of routes at any moment, even when there is no traffic in the network. The objective is to preserve routes well maintained and ready for eventual data traffic. Proactive protocols rely on keeping fresh lists of destinations and paths by constantly distributing routing tables throughout the network.

Consequently, this approach generates a lot of control traffic and then, a large amount of data for maintenance. However, as an advantage, packets experience low latency thanks to already discovered routes. Table-driven routing works fine in stable networks where nodes have low mobility and low failure rate^{7,8}. Some examples of algorithms implementing proactive routing protocols are the following:

- Optimized Link State Routing Protocol⁹.
- Destination-Sequenced Distance Vector routing¹⁰.
- BATMAN¹¹ (Better Approach To Mobile Adhoc Networking).

2.2.2 Reactive (On-demand) routing.

In reactive routing approaches, discovery of new routes is delayed until it is strictly necessary. If a packet needs to be routed to a destination and the path is not known, then the network is flooded with *Route Request* packets and the list of intermediate nodes is found on demand. This approach overcomes the problem excess of control traffic, but transmitted packets may experience a relatively big latency meanwhile waiting for the algorithm to find a route to the destination node. Some examples of on-demand routing algorithms are:

- ABR¹² (Associativity-based routing).
- Ad-hoc on-demand distance vector routing¹³.
- DSR¹⁴ (Dynamic Source Routing).

2.2.3 Hybrid routing.

Hybrid approaches combine features of both proactive and reactive routing. Roughly speaking, in general these algorithms initiate routing by establishing at the beginning some proactively explored routes. Then, it also serves on-demand from nodes that have been additionally activated in reactive way. One example of algorithm implementing this approach is the so called ZRP¹⁵ algorithm (Zone Routing Protocol). This method works on the idea that the largest amount of network traffic will take place between nodes that are geographically close together. Between this set of nodes, a proactive implementation is made to route this traffic. This implies heavy control traffic between small geographic areas. To reach nodes and destinations located far away, a reactive implementation is used.

2.3 Types of threads.

Ad hoc networks can suffer attacks at many levels and coming from very different malicious agents. This section will be divided into three categories: routing, external and internal threads. However, this is not the only way to make this division. For MANETs, potential attacks can be also classified in other ways, for example according to the layer in the stack in which the attacks are made. From the network point of view, one important consideration is the goal of the attack itself. In mobile ad hoc networks, two different goals can be the target¹⁶:

On the one hand, **control traffic**. Messages used for routing purposes and containing just information about the network topology. The distributed fashion of mobile ad hoc networks, in which there is no centralized authority that can enforce security, and the self-configuring capability of the network provoke that the formation of the network itself is based on trust on nodes participating on it. In many applications, nodes can join and leave the network as they please. For an attacker, the only requirement is to be inside the network. A malicious node can just join and disrupt it by hijacking the routing tables or altering valid routes, or can eavesdrop on the networks and get private data. Control traffic affects directly the routing protocols, which should be capable of detecting and defending against it.

On the other hand, attacks directed to **data traffic**, which implies targeting full data combinations. Attacks on this side focus on dropping packets passing through malicious nodes or delaying the forwarding of these packets.

An schema with examples of attacks according to traffic is shown in figure 4 below.

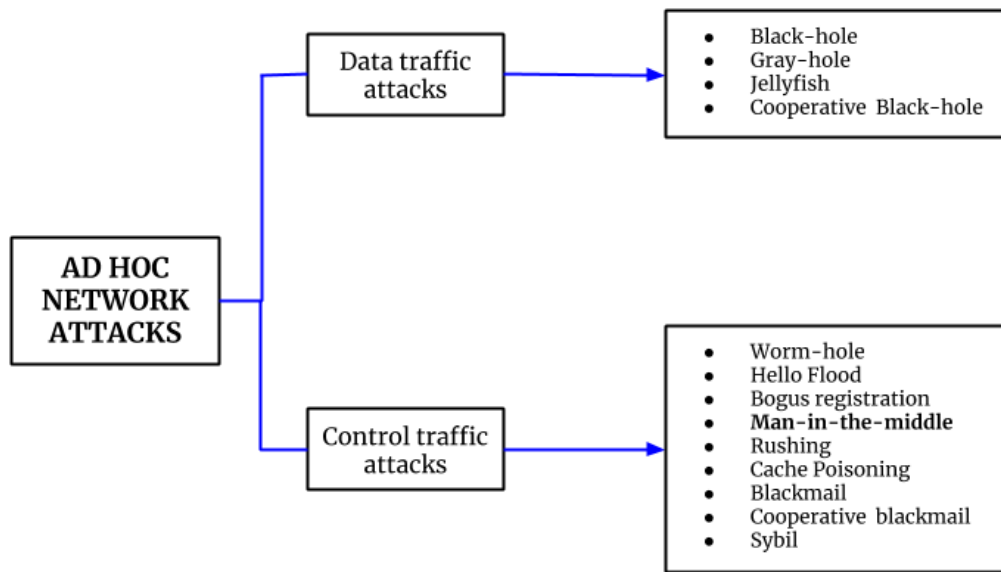


Figure 4. Classification of attacks according to type of target¹⁶.

2.3.1 Routing threats.

Routing threats are basically meant to destroy the network by altering or injecting erroneous routing information and breaking communication between nodes. To achieve secure routing, confidentiality and authentication are basic requirements. On the one hand, encrypting the routing information and making it confidential in the same way as traffic data, is essential to prevent from attackers to access it and therefore, modify the existing (correct) routes across the network. On the other hand, authentication is essential to prevent any external adversary to join the network and introduce erroneous data. There exist methods and algorithms to ensure confidentiality of the routing information as well as to perform node authentication. However, one property of ad hoc networks can also be considered to cope with these threats, in the event of compromise. The mobile nature of MANETs and its changing topology should be able to handle outdated routing information. In that way, false routing data can be treated as outdated information. Therefore, while there are still enough correct nodes, the routing protocol should be able to find alternative paths that avoid the compromised nodes.

2.3.2 External threats.

External threats are potential attacks coming from entities outside the network. In ad hoc systems that have authentication protocols to block the addition of unauthorized nodes to the network, external threats typically focus on attacking the data link and physical layers of the network or intercepting messages.

One of the most basic and common types of external threats are the so called **man-in-the-middle attack**. The attacker secretly sniffs and possibly alters the communication between two nodes who believe that they are talking directly to each other. To explain the principle of a man-in-the-middle attack we will use the typical example of Alice and Bob illustrated in figure 5.

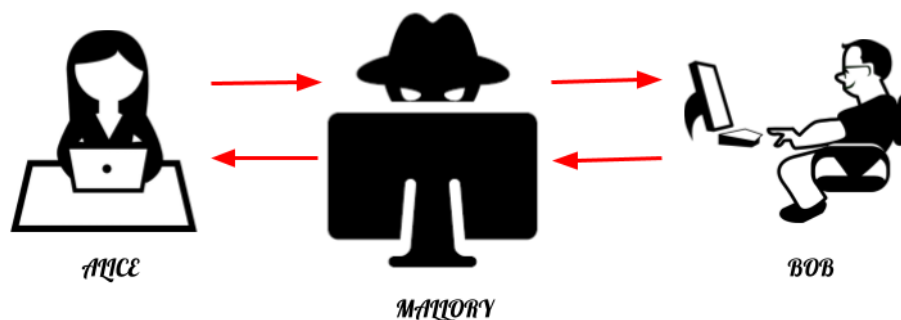


Figure 5. Representation of a man-in-the-middle attack with the example of Alice and Bob.

The typical outline would be as follows¹⁷:

1. Initially, Alice sends a message (*"Hello Bob, it's Alice, can you give me your key?"*) to Bob which is intercepted by Mallory.
2. Then, Mallory establishes communication with Bob (who does not know it) and forwards the message.
3. Bob, believing that he is talking to Alice, replies with his encryption key, which is now in Mallory's possession.
4. Mallory alters the response, and changes Bob's key with that one of himself, to send it to Alice.
5. Now, Alice encrypts her new message (*"Shall we meet in the library?"*) with that key, believing it belongs to Bob.
6. Mallory intercepts again this message. Then, he decrypts it with his key, alters it (*"Shall we meet by the river walk?"*) and forwards it to Bob.
7. Bob, thinking it was a secure communication, goes to the river walk and gets robbed by Mallory.
8. Alice does not know that Bob was robbed and, therefore, until enough time to suspect is passed, she cannot determine if something happened to Bob or if he is just late.

In centralized networks, there are many ways to prevent this simple kind of attack. However, it reveals two major problems in ad hoc networking security compared to centralized networks: **authentication** and **encryption**. To perform man-in-the-middle in an ad hoc network, the attacker needs to be part of the route. Protection by encryption of all transmitted packets brings the problem of distributing encryption keys throughout the rest of the good nodes of the network. To be sure which nodes should be in possession of that key, authentication of authorized nodes is required. However, the dynamic nature of MANETs and, in most cases, its sudden application (built on the go) require very sophisticated and difficult authentication techniques.

Man-in-the-middle and other eavesdropping attacks may only be focused on listening to the transmitted data and stealing information. In addition, there are many other attacks whose target is not data but **denial of service**. As the name properly indicates, these threats' aim is to break communications and, in general, their effectiveness is proportional to the duration of the attack and, specially, the routing protocol. For reactive routed networks, denial of service attacks may focus on intercepting and dropping packets at a new discovered node. To overcome the attack, the protocol must decide to look for alternative paths avoiding denying nodes. Contrarily, proactive protocols might not be able to react immediately to this packet dropping, and wait for connection time-out. Therefore, the effect of the attack is greater in a network routed proactively³.

One example of denial of service attack is the **sleep derivation torture attack**. This technique focuses on wasting node energy until it runs out of power and is forced to abandon the network. MANETs built with limited power and resources are the most vulnerable to this attack.

2.3.3 Internal threats.

In general, attacks carried by external entities are the easier ones to detect and refuse. The biggest threat comes from authorized nodes in the network that become compromised and might advertise incorrect information to the rest of the network. Protection against compromised nodes is difficult to detect and defend and, cryptography and authentication protocols have no longer effect, since they also use the same keys and signatures. Compromised nodes can be classified into four categories: failed nodes, badly failed nodes, selfish nodes and malicious nodes. However, any threatened node can have behaviors of different categories and the potential consequence of this behaviors vary a lot depending on the routing protocol, degree of mobility and many other considerations.

2.4 Cryptography.

Cryptography is the study of mathematical techniques to protect information, shared during the communication process, from third parties called adversaries, that try to access it without permission. The goal of cryptography is to transform messages into unintelligible strings, so that only the parties that have access to the unlocking key can read those messages. The initial message transformation is called **encryption**, while posterior recovery of the message is called **decryption**. The use of cryptography is thousands of years old. Ancient Egypt, Greek or Rome already used cryptographic techniques, mostly based on *substitution* or **transposition**¹⁸. In an encrypted communication, the sender of the secret message has to provide the receiver with the decryption technique that decodes the message. This *technique* is called **key**. In modern cryptography, there exist two types of encryption algorithm families depending on the nature of this key: Symmetric and Asymmetric key cryptography.

On the contrary, the opposite technique to data encryption is called **cryptanalysis**. It is used to break cryptographic security systems and snatch information when the secret key is unknown.

2.4.1 Symmetric key cryptography.

Symmetric key algorithms, also called *private key*, use the same key to encrypt and decrypt information. Therefore, symmetric cryptography involves a **shared secret** between the communicating parties. Key sharing and management is critical for the success of these algorithms. This type of cryptography was the first one implemented and many of its algorithms still in use and recommended by encryption suites nowadays. Some of the common symmetric key algorithms are DES, AES or Blowfish, but there exist many more. In this project in particular, we will implement the Blowfish algorithm, described below.

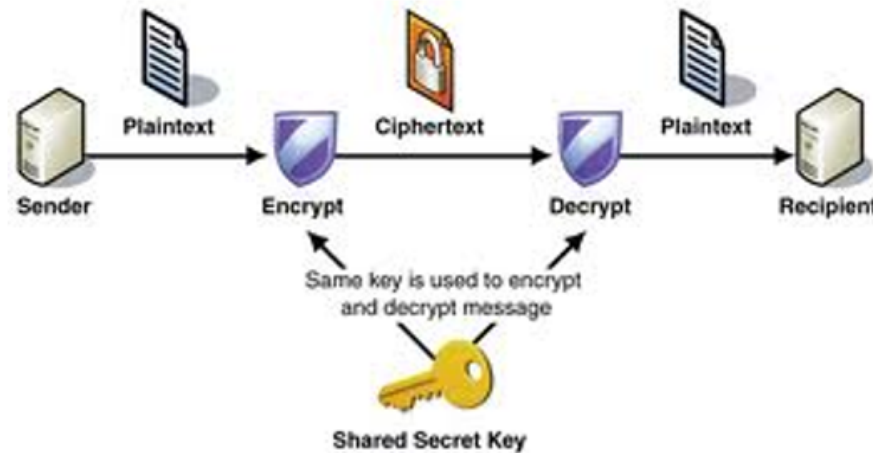


Figure 6. Symmetric key cryptography representation¹⁹.

Blowfish cipher

Designed in 1993 by Bruce Schneier (USA). Although nowadays it has a modern successor (the **Twofish**), there is no effective cryptanalysis against it. As Schneier himself stated when he first published it, the algorithm is unpatented and freely available for public use²⁰.

The Blowfish algorithm is a *block cipher*. It means that it divides a message into fixed length blocks during the encryption and decryption processes. In particular, block length is fixed to 64 bits. For messages whose length is not a multiple of 8, the message requires being padded²¹. With respect to the key, Blowfish works with keys up to 448 bits of length.

In the encryption process, the input is the 64-bit message in plaintext. The message is divided into two 32-bit halves. Then, the left part is XORed with the first element of a P-array, giving a new value denoted in the diagram in figure 7 (left) as P'. After that, P' is run through a transform function F and XORed again, this time with the 32-bit right part of the message. Its output is denoted as F' in figure 7 (left). Finally, F' replaces the left half of the message and P' the right half. This process is repeated 15 times with the successive entries of the P-array. To finish the process, the resulting P' and F' are XORed with the two last entries of the P-array (entries 17 and 18) and recombined to produce the final 64-bit encrypted text.

The transform function F is represented graphically in figure 7 (right). The 32-bit input is divided into four 8-bit pieces that are used as indexes into 32-bit S-boxes. The output of the four boxes are then added and XORed together to produce F value that will be later XORed with the right half of the message.

Both the P-array and the S-box values of the Blowfish are precomputed, based on the secret key used. Their values are computed as follows:

1. P is defined as an array of 18 integers of 32 bits.
2. S is defined as a 2D array of size 4x256, containing 32-bit integers.
3. Both arrays are initialized with constant values, which are hexadecimal digits of π .
4. The key is divided into blocks of 32 bits. Then, it is XORed with P and S and the results overwrite both arrays.

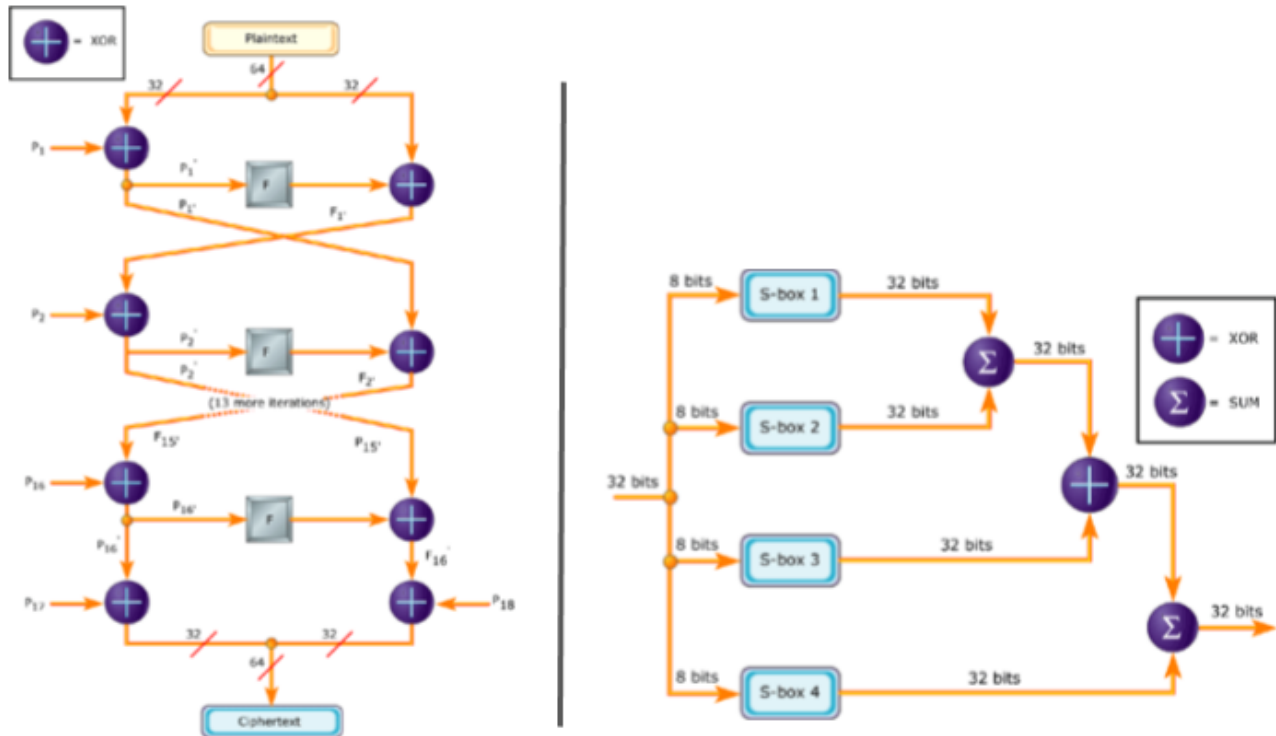


Figure 7. On the left, graphical representation of the Blowfish algorithm. On the right, graphical representation of blocks F in the left diagram²¹.

5. Finally, a message with all zeros is encrypted and the results overwrite again the values of P and S.
6. P and S arrays are now ready to be used.

Finally, the decryption process is done equal to the encryption, but reversed. Below, there is a pseudocode implementation of the blowfish algorithm, from²²:

```

// The Blowfish Algorithm
uint32_t P[18];
uint32_t S[4][256];

// F Transform Function
uint32_t f (uint32_t x)
{
    uint32_t h = S[0][x >> 24] + S[1][x >> 16 & 0xff];
    return ( h ^ S[2][x >> 8 & 0xff] ) + S[3][x & 0xff];
}

// Encryption routine
void encrypt (uint32_t & L, uint32_t & R)
{
    for (int i=0 ; i<16 ; i += 2)
    {
        L ^= P[i];
        R ^= f(L);
        R ^= P[i+1];
        L ^= f(R);
    }
    L ^= P[16];
    R ^= P[17];
    swap (L, R);
}

// Decryption routine
void decrypt (uint32_t & L, uint32_t & R)
{
    for (int i=16 ; i > 0 ; i -= 2)
    {
        L ^= P[i+1];
        R ^= f(L);
        R ^= P[i];
        L ^= f(R);
    }
    L ^= P[1];
    R ^= P[0];
    swap (L, R);
}

// Computation and Initialization of the P-array and S-array from Pi.
void key_decomp ()
{
    for (int i=0 ; i<18 ; ++i)
        P[i] ^= key[i % keylen];
    uint32_t L = 0, R = 0;

    for (int i=0 ; i<18 ; i+=2)
    {
        encrypt (L, R);
        P[i] = L; P[i+1] = R;
    }
    for (int i=0 ; i<4 ; ++i)
        for (int j=0 ; j<256; j+=2)
        {
            encrypt (L, R);
            S[i][j] = L; S[i][j+1] = R;
        }
}

```

2.4.2 Asymmetric key cryptography.

Private key algorithms present one main drawback or weak point: Secret key management. If a eavesdropping agent is able to intercept the secret key, the whole communication is compromised and the authorized agents would not notice it. To overcome this problem, a new modern family of encryption algorithms appeared: asymmetric cryptography, also known as **public-key** cryptography. This algorithms use a pair of keys to secure information: one public key available for anyone, and one private key just in possession of its owner, who does not have to share it.

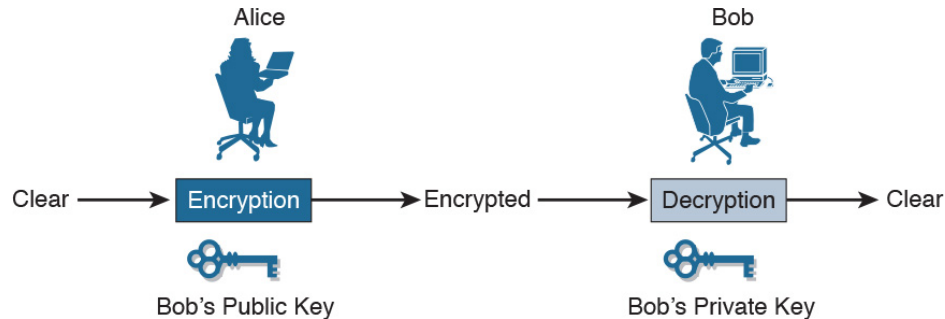


Figure 8. Asymmetric cryptography diagram²³.

In general, the public key is the one used to encrypt the message, while the private key is used to decrypt it. Figure 8 shows the typical flow of a communication ciphered by a public key algorithm. A user that wants to receive encrypted (Bob) data makes its public key available. Then, the sender (Alice) just has to encrypt the message with such key and forward it back. Since the first user is the only one in possession of the matching private key, he is the only one that can read that message.

Some examples of public key algorithms in common use nowadays are¹⁸:

- **DSA/DSS.** The Digital Signature Standard (DSS), developed by the US National Security Agency, based on the Digital Signature Algorithm (DSA). It can be used just for digital signatures or for both signatures and encryption.
- **RSA.** This algorithm, explained below, was one of the first public key techniques to be published. Developed in 1977 at the MIT, it can be used for encryption and as a basis for digital signature systems.
- **Elliptic curves.** While many encryption systems are mathematically based on logarithmic and factoring functions, this type of public key cryptographic methods are based solutions to the ellipse equation: $y^2 = x^3 + ax + b$. Elliptic curves offer a high degree of security and privacy with keys considerably shorter than those used by other algorithms.

To this point, we can summarize the comparison as that asymmetric cryptography offers a higher degree of security than symmetric cryptography. However, public key algorithms are usually computationally expensive and much slower. In ad hoc networking, private key algorithms still the most used today, mainly because the little development until now for this type of networking (compared to other network schemes) and ease of implementation.

RSA algorithm.

As it was previously said, RSA was one of the first cryptographic algorithms that used public and private key duality. The name is an acronym of the surnames of its first publishers: Ron Rivest (USA), Adi Shamir (Israel), and Leonard Adleman (USA). The strength of this algorithm relies on the so called **factoring problem**, which is, roughly said, the difficulty of factorizing the product of two large prime integers²⁴. This problem has always been a mathematical enigma that even today remains unclear. It is not proved that efficient factorizing algorithms do not exist, but it is suspected that they do not. When a composite number is sufficiently large, there is no efficient procedure to decompose it into prime numbers. Lots of efforts has been made over the last years to find an algorithm providing a solution to it, but with no result at all. Not only RSA, but other cryptographic algorithms rely on this paradigm, which remains being of special interest to mathematicians and computer scientists. But, in the end, going back to RSA, it all comes that, to decrypt it without the private key, the only way is to know the exact numbers that were use to generate the public one.

The RSA algorithm can be split in four stages²⁵:

1. Key Generation

2. Key Distribution
3. Encryption
4. Decryption

One basic fundamental for RSA to be successful, is to select three large numbers (e , d and n) such that the following relationship, called modular exponentiation, holds.

For all integer m , with $0 < m < n$:

$$(m^e)^d \equiv m \pmod{n} \quad (1)$$

The modular exponentiation is an operation very useful in computer science. It consists on calculating the remainder when a given integer (base) is raised to the power of another integer and then divided by a third last integer called the modulus. If it is satisfied, then even knowing e and n , it remains extremely difficult to find d . To give an example, lets set:

$$base = 5; \quad e = 3; \quad m = 13; \quad c \equiv base^e \pmod{m}; \quad c \equiv 5^3 \pmod{13}; \quad c \equiv remainder(125/13) \equiv 8. \quad (2)$$

Steps of the RSA algorithm

- Select two distinct prime numbers p and q . They should be of similar magnitude, but differ in length by few digits to improve security.
- Compute n by multiplying p by q . It would be the modulus for the public and private keys. Its length, measured in bits, will be the key's length.
- Compute R , the Carmichael's totient function of n , which is:

$$R = \lambda(n) = LCM(p-1, q-1) \quad (3)$$

where LCM stands for *Least Common Multiple*.

- Choose a small integer e (modular exponent), that should not be a factor of n and should be smaller than R .
- Compute d , the modular multiplicative inverse of e as:

$$d \equiv e^{-1} \pmod{\lambda(n)} = e^{-1} \pmod{R} \quad (4)$$

- Finally, the **public key** is: (n , e). The encryption function for a given message is:

$$C(message) = message^e \pmod{n} \quad (5)$$

- And the **private key** is: (n , d). The decryption function:

$$message = C^d \pmod{n} \quad (6)$$

The entire algorithm is presented in pseudocode below:

```

// RSA ALGORITHM
// Key Generation
int totient(p, q){
    int R = lcm(p-1, q-1);
    return R;
} // totient()

int coprime (a, b){
    int t;
    while(1){
        t = a/b;
        if (t == 0) {return b;}
        a = b;
        b = t;
    } // while
} // coprime()

vector keyGen(key_length){
    // Generate p, q and n
    int p = int.Rand(key_lenght/2);
    int q = int.Rand(key_lenght/2);
    int n = p*q;
    // Generate R using Carmichael's totient function
    int R = totient(p, q);
    // Get e
    int e = 2;
    while( e < R )
    {
        if( coprime(e, R) == 1 )
            {break;}
        else
            {e++;}
    } // while
    // Public key = (n, e)
    // Get d
    int d = e.modInv(R);
    // Private key = (n, d);
    vector keys.push_back(n, e, d);
    return keys;
} // keyGen()

// Encryption
int encrypt(message, n, e){
    int ciph = mod(message, n e);
    return ciph;
} // encrypt()

// Decryption
int decrypt(ciph, n, d){
    int message = mod(ciph, n, d);
    return message;
} // decrypt()

```

The RSA is a popular and secure algorithm that has one main disadvantage: it is computationally slow. Often, it is used as a way to secure private key distribution for symmetric cryptosystems, which are quite faster.

2.5 Secure key exchange and Authentication.

The main problem of key exchange, which becomes harder in the case of mobile ad hoc networks, is to ensure the identity of the parties that are communicating. MANETs are rapidly-expanding networks that are in permanent seek of partners grow by

adding new nodes (devices) to the network. It can be easy to enter the network, just simply asking to join or using *spoofing* methods, and once inside, attack its structure to weaken or break it. Authentication, which is a way of providing identity information, is crucial as a previous step before sharing the cryptographic keys that the nodes in the network use to hide the information. Ad-Hoc networks present also another degree of difficulty in the identity issue: the lack of online server. There is no way to connect to an online server to obtain a certificate of authority of the new unidentified peer node that it is joining the network.

One possibility was already introduced, using asymmetric cryptography to exchange a symmetric private key which is the one used to encrypt and decrypt the data transmitted on the network. In that way, the public key would not only hide the network's common secret one, but also would work as a digital signature.

Signatures, in computing, are mathematical methods for presenting the authenticity of messages and documents. Signatures ensure authentication, non-repudiation and integrity of the message. Digital signature schemes consist of many elements. First, a set K of possible public-private key pairs (randomly generated). Then, a finite set P of possible messages to send and a finite set A of signatures. Finally, 3 basic algorithms:

- A key generation algorithm, used to get a private key from the set K (chosen randomly).
- A signing algorithm, that given a message and a key produces a signature.
- A verifying algorithm, given the message, the signature and the public key, decides whether to accept or reject the authenticity of the message.

In that context, the RSA algorithm can be used as a signature scheme. The integers p and q, with the private key (n, d) are kept secretly, while the public key (n, e) is the one shared in the network. Then, a message M is generated and signed using the RSA decryption function - with the private key (based on Carmichael's totient):

$$Sig(M) \equiv M^d \pmod{n} \quad (7)$$

The final verification function is built using the encryption function of the algorithm - with the public key - and returning a boolean that indicates the acceptance status of the message origin.

$$M \equiv Sig(M)^e \pmod{n} \quad (8)$$

Below there is a pseudocode of the procedure:

```

// RSA algorithm for digital signature
struct publicKey(int N, int E);
struct privateKey(int N, int D);
struct keyPair( publicKey, privateKey);

// Public and Private keys
vector< publicKey(n, e) > PB;
vector< privateKey(n, d) > PV;

// Key-Pair selection
keyPair keySelect(vector< publicKey(n, e) > PB, vector< privateKey(n, d) > PV){

    int c = rand();
    keyPair = KP;
    while(1) {
        if (c<PB.size())
            {return KP( PB(c), PV(c) );}
        c = rand();
    }// while
}// keySelect()

// Signing function
int sign(M, privateKey(n, d) PVi){
    sigM = mod(M, PVi);
    return sigM
}// sign()

// Verification function
bool verify(sigM, privateKey(n, e) PBi){
    int original;
    M = mod(sigM, PBi);
    if(M == original)
        {return true;}
    else
        {return false;}
}// verify()

```

Authentication is the first requirement for secure key exchanging over the network. But there are also some other requirements to maintain the security of it when using a common secret key. One is, what to do with that secret key when nodes are coming and going in the network. That is, when to update the security means and change the key. This problem happens when a node leaves the network. This node is in possession of the current secret key of the system, and it can be stolen without the rest of the nodes knowing. When that happens, a new global symmetric key should be generated and broadcast over the network before other nodes are added.

Typically, some systems are configured so that a new key needs to be broadcast periodically. Ad hoc systems are sensitive to these interactions because of the limited channel broadcast capacity and the mobile nature of the devices that form it, that might also use a consumable battery as power source. In addition, there exists the problem that, when updating the key, sensitive information leaks may occur. Therefore, the time spent in sending and receiving keys should be kept to a minimum⁸.

3 Project Implementation

To implement a system in which the above concepts are put into practice, the intention was to create a network of Raspberry Pi's, running Raspbian OS (version 29-11-2017) connected in ad-hoc mode. Raspbian offers a easy interface to set up ad hoc connections using a USB adapter. The idea of the project is to connect this devices and make a series of experiments transmitting packets at TCP/IP level. In this session, different methods for securing the transmission with cryptography would be tried, while the conversation between Pi's is being eavesdropped from the laptop.

3.1 Ad Hoc network setup.

There are two methods to set Raspberry Pi's to broadcast in an ad-hoc network, depending on if its required to be launched temporary (from command line) or not. The permanent case basically consists on changing the following configuration files:

- */etc/rc.local*
- */etc/network/interfaces*
- */etc/dhcp/dhcpd.conf*
- */etc/default/isc-dhcp-server*

For other Raspberries to be able to join the ad-hoc network that started by one, they require being assigned an IP by the starting node. Hence the need of setting the Pi that is starting to broadcast first as a DHCP server.

In direct mode, which would allows Pi's to have static IPs, the following commands have to be used in both machines:

```
$ sudo ifconfig wlan0 down
$ sudo ifconfig wlan0 <desired_static_IP> netmask 255.255.255.0
$ sudo iwconfig wlan0 channel 1 essid leo2 mode ad-hoc
$ sudo ifconfig wlan0 up
```

Which are used to set ad-hoc communication on channel 1 under the name *leo2*. To confirm that network is successfully created and both nodes are in range, from one of the Pi's the WLAN0 channel can be scanned reporting something like the following result:

```
$ iwlist wlan0 scan

wlan0    Scan completed :
Cell 23 - Address: 02:11:87:F0:E0:00
          ESSID:"leo2"
          Protocol:IEEE 802.11bg
          Mode:Ad-Hoc
          Frequency:2.412 GHz (Channel 1)
          Encryption key:off
          Bit Rates:54 Mb/s
          Quality=0/100 Signal level=74/100
```

If each other's cell can be seen in both Pi's (ore more if the network is has more than two nodes), then it should be possible to ping between the two devices.

3.2 Problems with ad-hoc mode and hardware.

When testing the connection, although it was possible to find the cell of the Raspberry Pi's through *iwlist wlan0 scan*, it was not impossible to ping between Pi's, neither connect via socket or SSH from one to the other. This problem is due to the fact that the ad hoc connection is not supported for the driver of the USB dongle used, EDIMAX EW-7811UN, in Raspberry Pi 2 models²⁶.

Since no feasible solution to this problem was available, it was decided to test the security system using a client-server setup which will be described in section 4.

3.3 Testing cryptographic algorithms.

3.3.1 RSA algorithm implementation.

The RSA algorithm, self implemented, has been done in C++, following the pseudo code described in section 2.4.2. The code had been structured in a Class named RSA, created as a shared library, so that before using it some tests can be ran just calling the created functions. The RSA class contains the following functions:

- ***bool isPrime(long int pr);*** This function checks if a given number is a prime number, returning a boolean as result. It is used to check the input values p and q which serve to generate the keys.
- ***long int LCM(long int a, long int b);*** This function calculates the Carmichael's totient for the given p and q (see section 2.4.2).
- ***void compute_e(long int p, long int q, long int R);*** Calculates the value of the e exponent, component of the public key.
- ***long int compute_d(long int x, long int R);*** Calculates the value of the d exponent, component of the private key.
- ***void encrypt(int ind, long int n, char msg[100]);*** Function that encrypts the message to be sent.
- ***void decrypt(int ind, long int n);*** Function that decrypts the message received.

The input to the program is just the message. The prime integers p and q can be either set as a user input (this is just for testing the program) or randomly generated (the effective policy). The first test was simply made to verify that the mathematical operations work fine and the class is able to encrypt and decrypt *char* type messages. Here is an example of the output of the testings:

```
charlie@Asgard:~/workspace/LE02/cryptography/RSA$ ./bin/basicTest
ASYMMETRIC CRYPTOGRAPHY - RSA ALGORITHM - TEST EXAMPLE
LE02 - SDU Robotics

Enter the first prime number (p) to generate the key-pair: 41
Enter the second prime number (q) to generate the key-pair: 67
Enter the message to send:
thisisdenmark

Generating Keys...

Possible Key-Pairs:
0: Public Key ( n = 2747, e = 7 ) | Private Key ( n = 2747, d = 943 )
1: Public Key ( n = 2747, e = 13 ) | Private Key ( n = 2747, d = 1117 )
2: Public Key ( n = 2747, e = 17 ) | Private Key ( n = 2747, d = 233 )
3: Public Key ( n = 2747, e = 19 ) | Private Key ( n = 2747, d = 139 )
4: Public Key ( n = 2747, e = 23 ) | Private Key ( n = 2747, d = 287 )
5: Public Key ( n = 2747, e = 29 ) | Private Key ( n = 2747, d = 1229 )
6: Public Key ( n = 2747, e = 31 ) | Private Key ( n = 2747, d = 511 )
7: Public Key ( n = 2747, e = 37 ) | Private Key ( n = 2747, d = 1213 )
8: Public Key ( n = 2747, e = 43 ) | Private Key ( n = 2747, d = 307 )
9: Public Key ( n = 2747, e = 47 ) | Private Key ( n = 2747, d = 983 )
10: Public Key ( n = 2747, e = 53 ) | Private Key ( n = 2747, d = 797 )
11: Public Key ( n = 2747, e = 59 ) | Private Key ( n = 2747, d = 179 )
12: Public Key ( n = 2747, e = 61 ) | Private Key ( n = 2747, d = 541 )
13: Public Key ( n = 2747, e = 71 ) | Private Key ( n = 2747, d = 911 )
14: Public Key ( n = 2747, e = 73 ) | Private Key ( n = 2747, d = 217 )
15: Public Key ( n = 2747, e = 79 ) | Private Key ( n = 2747, d = 919 )
16: Public Key ( n = 2747, e = 83 ) | Private Key ( n = 2747, d = 827 )
17: Public Key ( n = 2747, e = 89 ) | Private Key ( n = 2747, d = 89 )
18: Public Key ( n = 2747, e = 97 ) | Private Key ( n = 2747, d = 313 )
19: Public Key ( n = 2747, e = 101 ) | Private Key ( n = 2747, d = 941 )
20: Public Key ( n = 2747, e = 103 ) | Private Key ( n = 2747, d = 487 )
21: Public Key ( n = 2747, e = 107 ) | Private Key ( n = 2747, d = 1283 )

Select Key Pair:
17

Encrypted Message:
t[0][0]n[0]*

Decrypted Message:
thisisdenmark
charlie@Asgard:~/workspace/LE02/cryptography/RSA$ ls -l
```

Figure 9. Capture from console output of the RSA test program.

The RSA Class is used to implement both the encrypted network-key exchange program and the authentication one. For the first use, there are three executables:

1. **randomKey**: Generates the public and private key-pair. It uses two random prime integers as inputs, and it also selects randomly the key pair among all the possible options for the given primes. Both keys are stored in separated files.
2. **encryptionRSA**: Uses the public key to encrypt the blowfish private key, which is stored in a file. Then, it saves the encrypted message and other parameters into files.
3. **decryptionRSA**: Reads the encrypted message and necessary parameters from the above mentioned file. Then, with the private key decodes the message.

The authentication part is implemented as described in 2.5. A set of 4 possible authentication messages, paired with a set of 4 possible key-pairs are used. The program selects a message-key duo randomly. Then, the message is encrypted using the private key and shared alongside with the public key with the other device that it is trying to verify the identity.

```

Correct Authentication. Device accepted
charlie@Asgard:~/workspace/LE02/cryptography/bin$ ./sign
Signing random CA
../Keys/Signature PublicKey_4.txt
Key to export: 4559, 29
Authentication encoding key: (4559, 533)
Signed 'CA':
369318219741002313707327318213642231

charlie@Asgard:~/workspace/LE02/cryptography/bin$ ./authentication
Decoding CA for authentication
Cipher Length = 10
Message to decode:
369318219741002313707327318213642231
Authentication decoding key: (4559, 29)

Decrypted 'CA':
realoviedo
charlie@Asgard:~/workspace/LE02/cryptography/bin$ ./verification
Received CA: realoviedo
Authentication Status: 1
Correct Authentication. Device accepted
charlie@Asgard:~/workspace/LE02/cryptography/bin$ ./sign
Signing random CA
../Keys/Signature PublicKey_1.txt
Key to export: 1769, 79
Authentication encoding key: (1769, 319)
Signed 'CA':
198975191655170017999561757165597

charlie@Asgard:~/workspace/LE02/cryptography/bin$ ./authentication
Decoding CA for authentication
Cipher Length = 10
Message to decode:
198975191655170017999561757165597
Authentication decoding key: (1769, 79)

Decrypted 'CA':
california
charlie@Asgard:~/workspace/LE02/cryptography/bin$ ./verification
Received CA: california
Authentication Status: 0
Wrong credentials. Device rejected
charlie@Asgard:~/workspace/LE02/cryptography/bin$

```

Figure 10. Capture from console output of the authentication test.

Finally, a verification program (run in the network node) checks the validity of the identifying message sent the foreign device. To test it, one of the 4 possible signatures has been left out, just to verify that it actually accepts and rejects valid and wrong authentications.

3.3.2 Blowfish algorithm Implementation.

The Blowfish algorithm was implemented using OpenSSL C++ toolkit. This is one of the most popular libraries for TLS and SSL security available. Following the same idea as in the RSA implementation, using this tools a Blowfish shared library class was implemented so that several small programs can use it. The implementation is able to read and write ciphered data into files as well. Numerical keys are generated from *char type* passwords, using the OpenSSL function **BF_set_key()**. Then, the encryption and decryption functions use OpenSSL **BF_ecb_encrypt()** to hide the message that will be shared. Figure 11 shows a screenshot of one of the several tests made.

OpenSSL blowfish functions only work with blocks of 8 bytes of data, therefore messages require being multiples and divided into exact 8-byte block. The implementation operates with keys of 10 bytes.

3.4 Network security system.

The security system proposed for the ad hoc network consists of several steps. The internal communication inside the ad-hoc network will be held with a symmetric key security method. The private key is only disclosed to authorized nodes. To share

```

charlie@Asgard: ~/workspace/LEO2/cryptography/Blowfish
charlie@Asgard:~/workspace/LEO2$ cd cryptography/Blowfish/
charlie@Asgard:~/workspace/LEO2/cryptography/Blowfish$ ./bin/basicTest
SYMMETRIC CRYPTOGRAPHY - BLOWFISH ALGORITHM - TEST

Enter key psswd:
ciudadlago

Generated Blowfish's private key:
Enter message to send (64-bits / 8 bytes):
rivendel
Encrypted message
X
g♦♦=@
Decrypted message
rivendel
charlie@Asgard:~/workspace/LEO2/cryptography/Blowfish$

```

Figure 11. Capture from console output of the Blowfish test program.

this key with them, the authentication and the secure exchange are held over RSA algorithm. The flow diagram can be seen in figure 12.

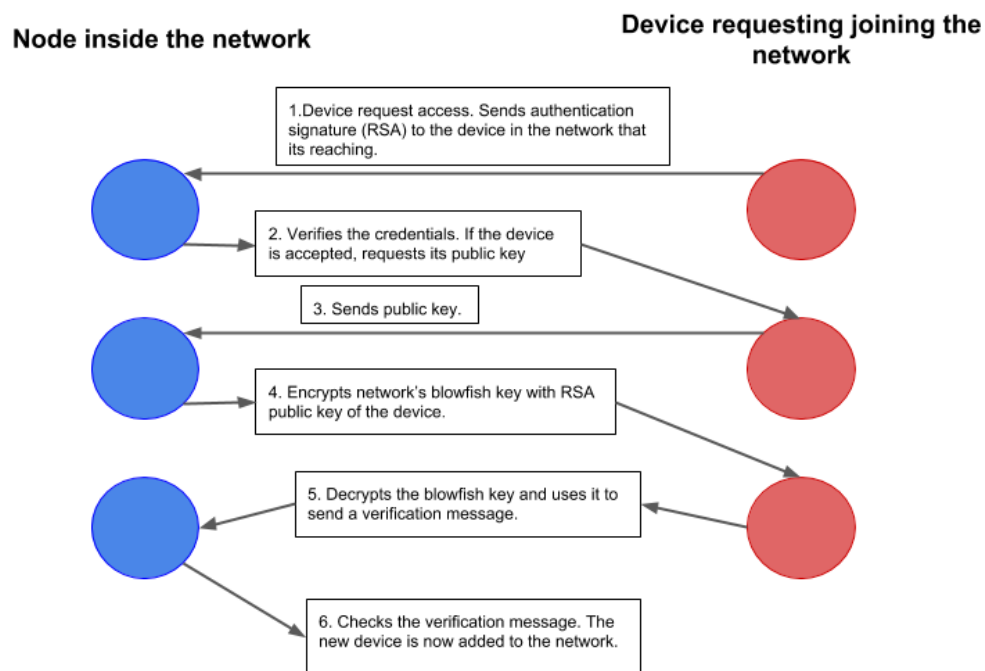


Figure 12. Flow diagram of the security system for the ad hoc network.

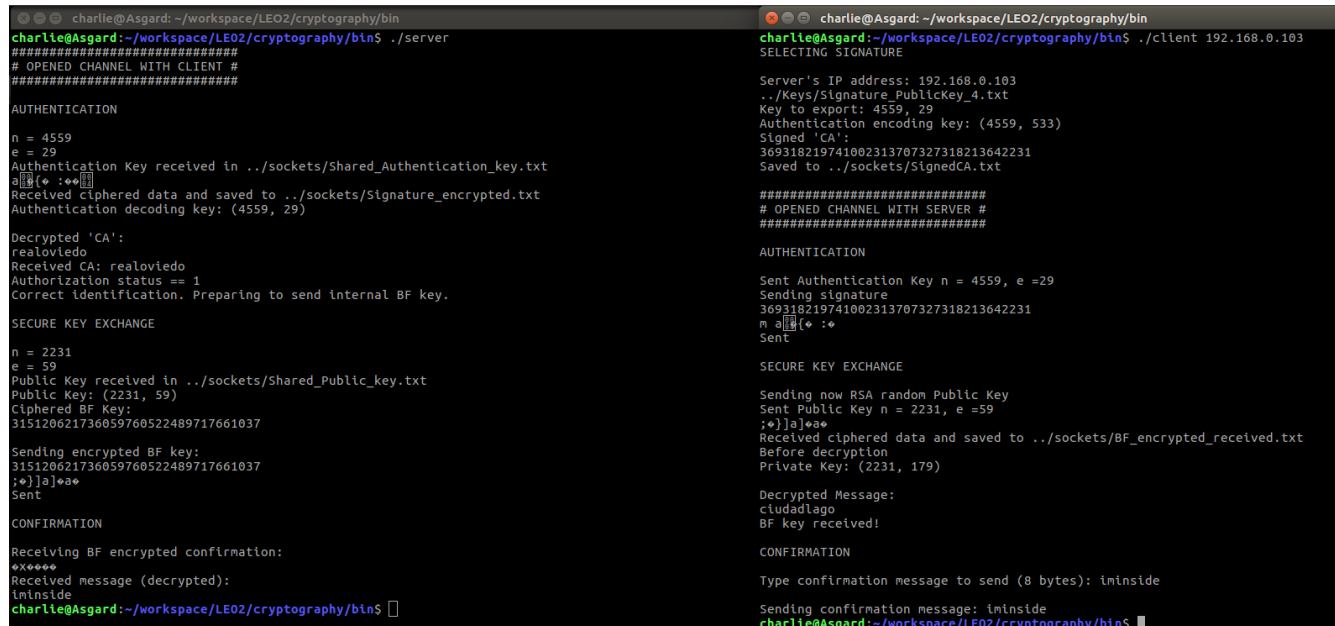
When a new node reaches the network and asks for permission to join, it has to send the correct authentication signature. Once it is verified, the node inside the network requests its RSA private key to encrypt the Blowfish key with it. The foreign node decrypts it and then sends a verification message ciphered with the received key. Then, this new node is now fully inside the network and ready to read data traffic.

4 Tests

Since it was not possible to test the algorithm in a real ad hoc network formed with the Raspberry Pi's, the system was tested using a *Server-Client* method implemented also in C++. The test conducts the process flow described in figure 12. The client program acts as a device requesting to be added to the network, while the server program acts as the node inside the network.

The client starts by sending its authentication public key and its signature so that the server can verify its identity. If the credentials are wrong, then the server cuts the communication. If not, the process continues with the client sending its general public key. Then the server uses it to encrypt the blowfish private key, which then is forwarded to the client.

Once the client is in possession of the blowfish key, and decrypts it, it sends a confirmation message (of 8 bytes) that, in the test, it is set to be typed by the user. Below in figure 13, there is a screenshot of the output of this client-server test.



```
charlie@Asgard: ~/workspace/LEO2/cryptography/bin
charlie@Asgard:~/workspace/LEO2/cryptography/bin$ ./server
#####
# OPENED CHANNEL WITH CLIENT #
#####
AUTHENTICATION
n = 4559
e = 29
Authentication Key received in ../sockets/Shared_Authentication_key.txt
a[0](* :00[0]
Received ciphered data and saved to ../sockets/Signature_encrypted.txt
Authentication decoding key: (4559, 29)
Decrypted 'CA':
realoviedo
Received CA: realoviedo
Authorization status == 1
Correct identification. Preparing to send internal BF key.
SECURE KEY EXCHANGE
n = 2231
e = 59
Public Key received in ../sockets/Shared_Public_key.txt
Public Key: (2231, 59)
Ciphered BF Key:
315120621736059760522489717661037
Sending encrypted BF key:
315120621736059760522489717661037
[0]a]00
Sent
CONFIRMATION
Receiving BF encrypted confirmation:
*****
Received message (decrypted):
linside
charlie@Asgard:~/workspace/LEO2/cryptography/bin$

charlie@Asgard: ~/workspace/LEO2/cryptography/bin
charlie@Asgard:~/workspace/LEO2/cryptography/bin$ ./client 192.168.0.103
SELECTING SIGNATURE
Server's IP address: 192.168.0.103
../Keys/Signature_PublicKey_4.txt
Key to export: 4559, 29
Authentication encoding key: (4559, 533)
Signed 'CA':
369318219741002313707327318213642231
Saved to ../sockets/SignedCA.txt
#####
# OPENED CHANNEL WITH SERVER #
#####
AUTHENTICATION
Sent Authentication Key n = 4559, e = 29
Sending signature
369318219741002313707327318213642231
n a[0](* :00[0]
Sent
SECURE KEY EXCHANGE
Sending now RSA random Public Key
Sent Public Key n = 2231, e = 59
[0]a]00
Received ciphered data and saved to ../sockets/BF_encrypted_received.txt
Before decryption
Private Key: (2231, 179)
Decrypted Message:
ciudadlago
BF key received!
CONFIRMATION
Type confirmation message to send (8 bytes): linside
Sending confirmation message: linside
charlie@Asgard:~/workspace/LEO2/cryptography/bin$
```

Figure 13. Console output of the client-server test.

5 Discussion and Conclusions

The project started with a research about the fundamentals of mobile ad hoc networks, comprising design, functionality, routing protocols and security problems, to understand the theory behind it. Then, regarding the security issue, research continued to learn about cryptography, focusing on learning symmetric and asymmetric algorithms, and methods for authentication and key exchange that can be used in MANETs.

Since some technical problems appear during the practical implementation of the project, more emphasis was put on the cryptography side. The final part of the project consisted on building a system that used both public key (RSA) and private key (Blowfish) algorithms and that can be used to secure a MANET.

Although it was not possible to test it using the original idea of building an ad hoc network of multiple Raspberry Pi's, the alternative tests carried were satisfactory and therefore, it could be concluded that the academic learning goals of this project, proposed in the beginning, both in terms of research and practical work, have been accomplished.

6 Future work

In terms of future work, a clear improvement would be to be able to use the required hardware for the Raspberry Pi's to communicate in ad hoc mode. Then, it could be possible to test the system as originally thought.

From a conceptual point of view, another implementation improvement could be the use of more modern cryptography techniques. Blowfish and RSA, although they are still effective, are considered *classics*. They both have their successors. There exist a lot of modern approaches, some of them very interesting, such as the ones involving elliptic curves.

As a final improvement, one last idea could be also to use an authentication algorithm also different from the public key encryption one, such as the DSA.

Project's Code development

The implementation of the project is organized in the attached repository in the following way:

- **Ad Hoc Network Configuration Files.** Contains the necessary configuration files to set a Raspberry Pi to broadcast in ad hoc network (with the correct hardware).
- **Cryptography.** This directory contains all the code implementing the security algorithms and tests. It is organized as follows:
 - *Blowfish.* Implementation of the Blowfish class using OpenSSL tools, with all the functions to generate keys, encrypt and decrypt. Built as a shared library (*Blowfish.cpp*).
 - *RSA.* Implementation of the RSA class, with all the functions to generate keys, encrypt, decrypt and produce signatures for authentication. Built as a shared library (*RSA.cpp*).
 - *Security.* Class that contains useful functions to create sockets, read and write messages into it or other kind of functionalities required in the project. Built as a shared library (*Security.cpp*).
 - *SRC.* Contains the *server.cpp* and *client.cpp* that test system.
 - *Keys.* Directory in which all the generated keys and signatures are stored.
 - *Sockets.* Directory in which all the keys and data shared during socket communication are stored.
 - *Tests* and *genfiles.* Directories with test files, parameters and results useful to debug some parts of the code

Apart from the main client-server test, there are individual tests for each of the functions implemented in RSA and Blowfish classes.

References

1. Wikipedia.com. Wireless ad hoc network. *Wikipedia* (2018). URL https://en.wikipedia.org/wiki/Wireless_ad_hoc_network.
2. Zhou, L. & Haas, Z. Securing ad hoc networks. *IEEE Netw.* **13**, 24–30 (1999).
3. Derveloy, R. Security issues of ad hoc networks. *The Univ. Tenn. at Chatt.* (2012).
4. Huerta, C. V. Leo2 course project. *GitHub* URL <https://github.com/CVCopenhaguen/Damascus.git>.
5. Awerbuch, B. & Mishra, A. Introduction to ad hoc networks. *Dep. Comput. Sci.* (2008).
6. Wikipedia.com. Routing table. *Wikipedia* (2018). URL https://en.wikipedia.org/wiki/Routing_table.
7. Wikipedia.com. List of ad hoc routing protocols. *Wikipedia* (2017). URL https://en.wikipedia.org/wiki/List_of_ad_hoc_routing_protocols.
8. Barbeau, M. & Kranakis, E. *Principles of Ad Hoc Networking* (Wiley, 2007), 1 edn.
9. Wikipedia.com. Optimized link state routing protocol. *Wikipedia* (2018). URL https://en.wikipedia.org/wiki/Optimized_Link_State_Routing_Protocol.
10. Wikipedia.com. Destination-sequenced distance-vector routing. *Wikipedia* (2017). URL https://en.wikipedia.org/wiki/Destination-Sequenced_Distance_Vector_routing.
11. Wikipedia.com. B.a.t.m.a.n. *Wikipedia* (2018). URL <https://en.wikipedia.org/wiki/B.A.T.M.A.N>.
12. Wikipedia.com. Associativity-based routing. *Wikipedia* (2018). URL https://en.wikipedia.org/wiki/Associativity-based_routing.
13. Wikipedia.com. Ad-hoc on-demand distance vector routing. *Wikipedia* (2017). URL https://en.wikipedia.org/wiki/Ad_hoc_On-Demand_Distance_Vector_Routing.
14. Wikipedia.com. Dynamic source routing. *Wikipedia* (2017). URL https://en.wikipedia.org/wiki/Dynamic_Source_Routing.
15. Wikipedia.com. Zone routing protocol. *Wikipedia* (2016). URL https://en.wikipedia.org/wiki/Zone_Routing_Protocol.
16. Bhattacharyya, A., Banerjee, A., Bose, D., Saha, H. N. & Bhattacharyya, D. Different types of attacks in mobile adhoc network. *CoRR* **abs/1111.4090** (2011).
17. Wikipedia.com. Man-in-the-middle attack. *Wikipedia* (2018). URL https://en.wikipedia.org/wiki/Man-in-the-middle_attack.
18. eTutorials.org. Cryptography basics. *eTutorials Unix Internet Security Part II, Chapter 7* (2018).
19. Vejayon, J. Main differences between symmetric and public key cryptography. *Jay IT Secur.* (2013). URL <http://www.jayitsecurity.com/2013/01/main-differences-between-symmetric-and.html>.
20. Schneier, B. The blowfish encryption algorithm. *Schneier on Secur.* URL <https://www.schneier.com/academic/blowfish/>.
21. Gatliff, B. Encrypting data with the blowfish algorithm. *embedded.com* (2003). URL <https://www.schneier.com/academic/blowfish/>.
22. Wikipedia.com. Blowfish (cipher). *Wikipedia* (2018). URL [https://en.wikipedia.org/wiki/Blowfish_\(cipher\)](https://en.wikipedia.org/wiki/Blowfish_(cipher)).
23. Cryptographic technologies. *Pearson* URL http://ptgmedia.pearsoncmg.com/imprint_downloads/cisco/digitalstudyguide/9780134424064/alt_ch29.html.
24. geeksforgeeks.org. Rsa algorithm in cryptography. *Geeks for Geeks* URL <https://www.geeksforgeeks.org/rsa-algorithm-cryptography/>.
25. Wikipedia.com. Rsa (cryptosystem). *Wikipedia* (2018). URL [https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem)).
26. Ad hoc network with edimax ew-7811un - ping not working. *RaspberryPi.org* URL <https://www.raspberrypi.org/forums/viewtopic.php?t=103014>.