SETTING INITIAL SECRET KEYS IN A MOBILE AD HOC NETWORK

Murat Cihan[1] and Çetin Kaya Koç[2,3]

[1] Işık University, Maslak 34398, İstanbul, Turkey,
mcihan@isikun.edu.tr,
[2] İstanbul Commerce University, Eminönü 34378, İstanbul, Turkey
[3] Oregon State University, Corvallis, Oregon 97331, USA,
koc@eecs.oregonstate.edu

Abstract. Mobile ad hoc networks require specialized authentication protocols due to the mobility of users and lack of always-available trusted servers. There are a variety of mobile ad hoc authentication protocols for creating session and group keys, once a subset of individual nodes are signed in, i.e., established their shared secret keys. In this paper, we address the issue of setting initial secret keys for the nodes in a mobile ad hoc network architecture. While there are some new approaches such as the *resurrecting duckling protocol*, we address more practical issues. We have developed new protocols for setting the initial keys and extending the circle of authenticated nodes as new nodes arrive to join the network. We created an experimental system for the Linux platform in the Java programming language using the Bluetooth as the networking architecture. Our protocols can be used to create a mobile ad hoc network of authenticated nodes of cell phones, palm and pocket PC computers, and laptop computers. In this paper, we describe the design and implementation details of the protocols and summarize the results of our benchmarking tests.

1 Introduction

Ad hoc networks are designed to work autonomously, requiring no previously configured static infrastructure. Currently, there are several ways to form an ad hoc network due to widespread usage of mobile computing and wireless communication technologies. As a result of these improvements, mobile ad hoc networks found many specific application areas in our daily lives.

Due to the mobility of users and lack of trusted servers in mobile ad hoc networks require specialized authentication protocols. There are several mobile ad hoc authentication protocols for creating session and group keys, once a subset of individual nodes have established their shared secret keys [15], [1], [2]. In this paper, we study on the issue of setting initial secret keys for the nodes in a mobile ad hoc network architecture. After the brief introduction, we give detailed information about ad hoc networking technologies. We have particularly considered the most used three ad hoc networking technologies: Wireless LAN, infrared, and Bluetooth.

An ad hoc network can be characterized as a dynamically reconfigurable network. The nodes are generally mobile and it is easy to join or leave the network. These properties require ad hoc networking to be examined in a different way from traditional networking concepts [10]. In an ad hoc network any node can join in or move outfrom the network without a special effort, except some security issues have to be considered. Within the network, members must not have higher precedence or more superior role than the others that form the network. In case where special duties assigned on particular mobile nodes, any node removal would create serious problems for thewhole ad hoc network. For example, removal of a server or a gateway node, which has a duty to the others, would create trouble since gateway or server would be missing. Recovering such problems may cause inefficient resource allocation. Therefore, it is not a good practice to load server applications or programs on the nodes unless some special algorithms and solutions have been designed for such problems [3].

Ad hoc networking is a fairly new concept, however, its application areas are very large. One particular application area is in military operations. The idea of an easily constructed network with mobile nodes is an attractive feature for battlefield computation and communication. Units could be traced in the field or a particlar group can be given specific orders. Another application of ad hoc networking is found in multi player computer games. Instantaneous information sharing in a business meeting between the presenter and the audience is yet another example. The same is valid for an instructor passing information to the students in a classroom. Furthermore, a patient's health information could be shared with the doctors and nurses who have a mobile devices in ad hoc network.

2 Ad hoc Networking Technologies

One can construct an ad hoc network using several different technologies. With the advances in mobile device and programming technologies, it is easy to create such a network at any time and in any place needed. By definition an ad hoc network is open to new nodes and no time limitation applies to the incoming or leaving nodes. To form such a network, one should select the appropriate technology. With wires, it is almost impossible to form an ad hoc network with the desired properties. For this purpose, wireless technologies are preferred. Infrared, wireless LAN, and bluetooth are the examples of these wireless technologies.

We can group these technologies into two general categories in terms of usage: Wireless local area networks (WLAN) , wireless personal area networks (WPAN) [16]. Infrared and bluetooth fall into the second category while the others, which are interested in broader areas, fall into the former one. In Figure 1, we give an example of an ad hoc network, with several members and different technologies used for connections.

2.1 Wireless LAN

Wireless LAN protocols, such as IEEE802.11 standard, are generally used in wide area networking [7]. Wireless routers and switches are used to connect the

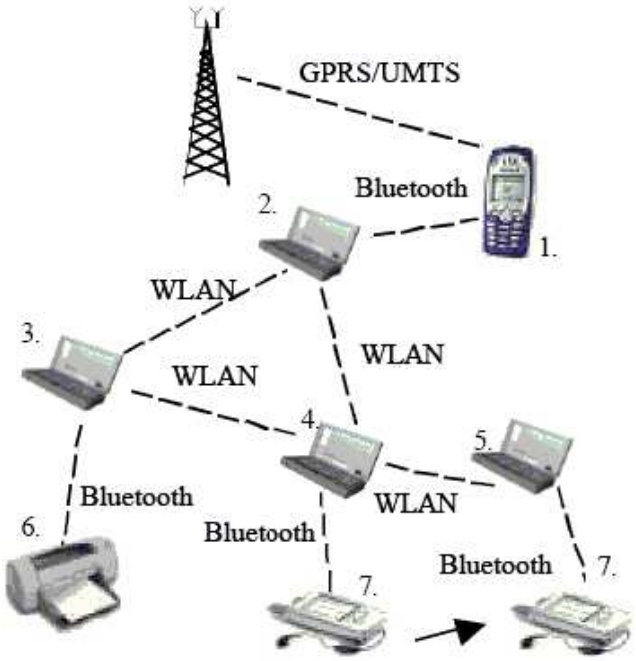

Figure 1: An example of Ad Hoc Network (Source: [10])

nodes. These common connection points are connected to some device with wire to establish a connection with the wireless nodes. No two nodes can communicate with each other without being connected to the switch or the router. These devices play the role of intermediary access point for all of the nodes. The number of nodes connected to the network is limited to the capacity of access points. This technology is used, for example, in a building to let the mobile users connect to the LAN, thus to the Internet.

2.2 Infrared

Infrared is used in fairly smaller areas for passing instantaneous information between devices. It is designed for short range, easy, interoperable, and low cost information sharing [18]. Interoperable means that many platforms could support this technology and this standard. It is a low-cost solution compared to the cost of the device on which infrared device is built. With the range of 1 meter with 15 degree half-angle, it is designed for point-and-shoot place-and-play communication [8]. This feature makes infrared a replacement for the serial cable between two parties. Instead of connecting two devices with a cable for a small communication, infrared could be used. Thus, within a previously known short range [18], two parties could make use of infrared. Since the working range

is limited, infrared is not a good solution for large ad hoc networks. Exchanging business cards via mobile devices and getting print out via infrared enabled printer and laptop could be some examples for use of infrared technology in ad hoc networking.

2.3 Bluetooth

Bluetooth is another personal area networking technology. It is a short range wireless radio network designed for connecting peripheral devices [6]. This is again a cable replacement technology like infrared. However; with its profiles, bluetooth is able to give much more than that [5]. Bluetooth is used not only for peripheral communication, but also for communicating mobile phones, modems, Personal Digital Assistants (PDA), computers, printers, local area networks, and other similar devices. Each device can communicate up to seven other nodes at a time and can be a member of several different ad hoc networks. Any bluetooth enabled device could search for other bluetooth devices and could communicate with them [9]. A *piconet* is constructed when two or more units that share the same area and the same communication channel, whereas in the same area, different communication channels make up a *scatternet*. A scatternet could also be described as a group of piconets [4]. With these characteristic, bluetooth is much effective when there are many nodes communicating.

In a bluetooth ad hoc network, all nodes assumed to have the same hardware capabilities. One of the nodes must be the master to regulate the traffic on the piconet while others are slave nodes. However, the members may change roles if necessary and this property provides homogenousness among the nodes [9]. A bluetooth ad hoc network is open for new nodes, join or leave operations are easy [17]. As an example of how bluetooth is used in ad hoc networking, a bluetooth enabled mobile phone with a bluetooth enabled headset could be given. In a short range, one may not even need to touch the the mobile phone to set up a talk session just using the bluetooth enabled headset.

3 Security Considerations

Information security is a fundamental issue with ad hoc networks and certainly security is an essential problem for all networks. With a strong infrastructure this problem could be diminished. However, with mobile nodes and wireless links, it is not easy to provide such a secure communication [19]. In the case of only two nodes talking to each other with no possibility of being traced, there would be no need for security specific applications. However, this is not the case. Most of the time the ad hoc network has more than two parties and information must pass over some intermediate stations, which are actually the other nodes in the network. In this context, information flow must be secured from the third party devices, even if it is on the way of flow [12]. Furthermore, since ad hoc networking is generally on air communication, it is also open for eavesdropping or interference. An ad hoc network should be secured from these kinds of attacks.

While there are variety of security issues to be concerned about, we address a particular problem in ad hoc network security: the placemet of setting of the initial secret keys. The most remarkable work in this area is the *Resurrecting Duckling Protocol* [14]. The protocol assumes that the devices are manufactured by its manufacturer to allow it to be implanted with a secret key the first time device is turned on. Once the device (duckling) is purchased by its owner, it is powered on in a physically secure area and device "looks" for another device (mother duck) to connect to and accept a secret key from the mother and from there on, it recognizes its mother as a secure communicating partner.

The resurrecting duckling protocol is simple and very powerful. As the devices are made to confirm this protocol, either to act as duckling or as mothers (of course, roles can be interchanged: a mother duck can be a duckling to another duck). however, this protocol has not been embraced by manufacturers yet. Until then, we need practical protocols to set up initial secret keys. This is the subject of our investigation.

We propose four particular protocols for setting up initial keys. We briefly describe these four protocols for setting initial secret keys for the nodes in a mobile ad hoc network architecture and give benchmarking results obtained from our reference implementation. These protocols are:

1. *PlainText:* The new key is passed as a plain text to the device.
2. *SimpleEncryption:* The new key is passed after encrypting with previously known key and decrypted at receiver side.
3. *DiffieHellmann:* Two devices agree on the key using the classical (mod p) Diffie-Hellmann key exchange method.
4. *EllipticCurveDH:* Two devices agree on the key using the elliptic curve Diffie-Hellmann key exchange method.

4 Reference Implementation Platform

We created a laboratory ad hoc network system in order to implemented and test our protocols. The following nodes and network hardware and protocols are used:

- Laptop PC, HP NX9000, Intel P4 Mobile 2200 MHz processor, 256 MB memory [C1]
- Desktop PC, Intel P4 1400 MHz processor, 256 MB memory [C2]
- Desktop PC, AMD Duron 1200 MHz processor, 256 MB memory [C3]
- Pocket PC, Qtek 2020, Bluetooth Enabled [C4]
- Bluetooth enabled Mobile phone, Siemens S55 [P1]
- Bluetooth enabled Mobile phone, Sony-Ericsson T630 [P2]
- Class 2 USB Bluetooth adapter, Billionton [A1]
- Class 1 USB Bluetooth adapter, Billionton [A2]

The first computer [C1]is set to be *Client* device, having attached the first Bluetooth adapter [A1] with a 10 meters range. Next, we set the [C2] computer as

Server device, having attached the other Bluetooth adapter [A2] providing a 100 meters range. The client and the server devices are set for running `BlueClient` and `BlueServer` applications, respectively. The other devices [C3, C4, P1, P2]are used for discovery and noise purposes.

In Table 1, we have summarized the code space requirements of the implementation. Each class in the project is included this table, indicating; number of lines in the source code, the size of the source code in bytes, and the size of compiled class files in bytes.

Table 1: Code space analysis of the program

Class Name	Number of Lines	Source Code (bytes)	Compiled Class (bytes)
BlueClient	1,132	41,488	27,002
BlueServer	680	26,379	19,344
EC/ECOperations	473	16,263	9,954
DH/diffieHellmann	297	10,461	6,413
Aes/AES	1,260	43,180	43,171
Des/Encrypt	195	4,039	4,547
Des/Decrypt	190	3,918	4,461
Des/desHelper	226	5,563	7,917
Des/Subkeys	70	1,306	1,622
Total	4,523	152,592	124,431

Table 1, also gives us the total amount of space needed for hosting the whole implementation, requiring a 124kB disk space. In case only one of the four methods to be used, this space requirement falls. For example, `diffeHellmann Key Exchange Method` requires only 6 kB, which yields only 50 kB with the two driver classes, namely *BlueClient* and *BlueServer*. It is not a huge consumption of disk scape even if small mobile devices with small disk spaces are concerned.

5 Device and Service Discovery

Device discovery is the first touch of the local device with the other devices around, in which it determines the initial information about them, such as device type, Bluetooth address, and friendly name. Since it is the first step in the two sided application, we shall start with it. In Table 2, we tabulate the elapsed time in device search in several attempts. We set four remote devices around ([C2, C4, P1, P2]) and decreased the number of remote devices one at a time until the server device is left. We made five device discovery attempts for each state and averaged the elapsed time values. The symbols in this table have the following meanings:

– C: One computer is discovered.
– P: One cellular phone is discovered
– C-C-P-P: Two computers and two cellular phones are discovered.
– C-C-P: Two computers and one cellular phone are discovered
– C-C: Two computers are discovered
– C: Only one computer is discovered.

Table 2: The elapsed time for device discovery in milliseconds.

Trial	C-C-P-P	C-C-P	C-C	C
1	16,329	12,632	10,540	10,742
2	12,737	11,939	14,192	10,378
3	16,052	14,538	13,002	10,380
4	14,436	14,436	13,012	10,405
5	11,939	11,939	14,470	10,783
Average	15,355	13,096	13,043	10,537

We assumed that we would have the most accurate results when the link quality is high, as they are closer (within the range of almost 1 meter). Therefore, the measurements were made while the local client device is pretty close to the other devices. The average elapsed time values showed us that the number of devices around affects the time consumed for device discovery since local device tries to gather some data on each and every remote device around. We can conclude that with the most optimistic assumption, device discovery process takes 10 seconds in average.

Service discovery process is another time consuming work in the implementation. We measured the time passed in service discovery process on two different computers, on of which is was our server device ([C2]) hosting only *KeyExchangeMethods* and the other one ([C4]) having several services available to be discovered. we averaged the elapsed time after five trials, as given in Table 3.

Table 3: Time passed for service discovery in milliseconds.

Trial	Server Device	Other Computer
1	1,318	1,174
2	1,525	1,139
3	1,140	1,136
4	1,282	1,143
5	1,428	1,153
Average	1338	1149

The client device queries service discovery database of the remote device in service discovery process. The list of services to be searched are specified as a

list in the client application, which in turn increases the work load in service discovery process. We searched the same list of services on the two remote devices. The average elapsed time values were not much different than each other. This showed us that the number of services provided in two devices does not cause a huge gab between two. In addition, according to the results, service discovery process does not tend to be a much time consuming process, but finishes in almost 1 second.

6 Running Time of the Protocols

6.1 PlainText Key Exchange Method

Once the two devices are connected on `KeyExchangeMethods` service, they select one of the four methods to run. The first method in hand is *plainText Key Exchange Method*. Here, we demonstrate the time passed in running this method between the two devices. In Table 4, we give the elapsed time in 7 trials and average of them, for both of client and server applications.

The key generation process was the interactive part of this method. We did not want to reflect the time passed in user input to total elapsed time. Thus, we used a constant input for key generation instead of requesting from the user as follows:

```
String key = "0123456789ABCDEF";
 key = getDigest(key,modeMD5);
```

Table 4: Elapsed time during `plainText Key Exchange Algorithm` by Client and Server applications, in milliseconds.

Trial	Client Application	Server Application
1	6	4
2	10	3
3	3	7
4	5	6
5	6	6
6	6	6
7	3	6
Average	5.6	5.3

Since *plainText Key Exchange Method* consists of generating a key at client an simply passing it to server application, we see pretty small amount of elapsed time. Total of almost 5 milliseconds were enough for this method to be accomplished at both sides.

6.2 SimpleEncryption Key Exchange Method

The second method has two interactive parts. As in the previous method, we did not want to include the time passed in user input for both session key and temporary key generations processes. Instead, we again used constant inputs in the program. Since we included four options in this method, we examined each of them separately. The elapsed time values on the client side are demonstrated for each of four options in Table 5.

Table 5: Elapsed time during `SimpleEncryption Key Exchange Algorithm` by Client application, in milliseconds.

Trial	DES-MD5	DES-SHA	AES-MD5	AES-SHA
1	77	43	59	65
2	42	45	59	65
3	53	48	68	69
4	46	67	61	68
5	56	54	71	74
6	50	59	76	88
7	38	49	66	71
Average	51.7	52.1	65.7	71.4

We noticed that the options with *AES* took relatively more time than the options with *DES* when selected as the encryption algorithm. Moreover, hash algorithm was not making much difference in running time with respect to the other elements. In addition to client side analysis, we included the elapsed time values on the server side demonstrating each of four options in Table 6.

Table 6: Elapsed time during `SimpleEncryption Key Exchange Algorithm` by Server application, in milliseconds.

Trial	DES-MD5	DES-SHA	AES-MD5	AES-SHA
1	24	6	13	29
2	7	7	14	34
3	22	5	33	25
4	10	5	13	27
5	24	5	30	33
6	11	11	19	31
7	13	5	31	25
Average	17.3	6.3	21.9	29.12

We see that encryption algorithm selection has the same effect on the server side applications. The options with *AES* took more time relative to the options with *DES*. As a result, we may conclude that in `SimpleEncryption Key Exchange Method`, the options with *DES* are less time consuming than those with *AES*. In general, all of the four options have elapsed time values less than 50 milliseconds at client side and less than 30 milliseconds at server side. This method may seem to consume longer time with respect to the first method but has pretty short time intervals.

6.3 DiffieHellmann Key Exchange Method

Different from previous two methods, `DiffieHellmann Key Exchange Method` has no interactive operations. The whole process goes in between the client and the server applications. The client application is responsible for generating entities that are necessary for the algorithm to start. Therefore, it is supposed that client application would consume more time than server application does. In Table 7, we give the elapsed time values in this method, at both client and server sides.

Table 7: Elapsed time during `diffieHellmann Key Exchange Algorithm` by Client and Server applications in milliseconds.

Trial	Client Application	Server Application
1	83	31
2	62	10
3	84	11
4	114	34
5	75	25
6	93	28
7	95	31
8	91	30
Average	87.1	25

This method is based on consecutive multiplications at both sides and passing the results to each other. At the end, we clearly see that the client application needs more time to accomplish its tasks than the server application does. In addition, the results are almost at the same level with the `SimpleEncryption Key Exchange Algorithm`, with average of 90 milliseconds at client side and 25 milliseconds at server side.

6.4 EllipticCurveDH Key Exchange Method

The most time consuming phase in `EllipticCurveDH Key Exchange Method` is to generate a special curve to agree on. Instead of generating an elliptic curve

from scratch, we decided to use one of the recommended elliptic curve by NIST [11]. Therefore we minimized the time for preliminary steps of the algorithm. As in `diffieHellmann` algorithm, the algorithm is based on multiplicating a point on the curve with a scalar at both sides with special rules. In our implementation, we included a special function that recursively add two points on the curve. Thus, we also minimized the intermediate steps. In Table 8, we give the elapsed time values in this method.

Table 8: Elapsed time during `EllipticCurveDH Key Exchange Algorithm` by Client and Server applications, in milliseconds.

Trial	Client Application	Server Application
1	436	490
2	379	428
3	268	358
4	364	390
5	279	421
6	267	298
7	282	452
8	353	378
Average	328.5	401.9

We notice that, `EllipticCurveDH Key Exchange Algorithm` causes more time consumption than the previous three methods. On average, the client side needs 330 milliseconds as the server side consumes 400 milliseconds to accomplish their tasks.

6.5 Summary Timing Results

We summarize the results obtained from running time measurements. Table 9 demonstrates the elapsed time values for each algorithm, as average of those obtained from client and server sides. *plainText Key Exchange Algorithm* seems to be the fastest method, which lasts less than 10 milliseconds. However, we do not consider it as safe since it is very open to passive attacks such as traffic analysis.

The second fastest method is *SimpleEncryption Key Exchange Algorithm* with less than 40 milliseconds of elapsed time in average. This is a stronger form of *plainText* algorithm, however we emphasize a weak in this algorithm: How secure is the key that is used for encrypting and decrypting the secret key. We propose an instant communication between the users that results a PIN-like seed for the first key.

The third fastest method seems to be *diffieHellmann Key Exchange Algorithm*, giving a less than 60 milliseconds of average running time. In addition,

Table 9: The average running times for each algorithm in milliseconds.

Method	Average running-time
plainText	5.43
SimpleEncryption	39.45
diffieHellmann	56.06
EllipticCurveDH	365.19

EllipticCurveDH Key Exchange Algorithm processes slower than the other algorithms, with the average running time of 365 milliseconds. However, we consider these two slowest algorithms as providing rather stronger security than the previous two algorithms, since due to their unique structure which is they are based on public-key infrastructure and they do not leak intermediate results to the adversaries.

7 Conclusion

Our code space requirement analysis show that a total of 124 kilobytes of disk space is enough for installing the whole application on a device. This requirement decreases as only one of the algorithms is hired. Currently, mobile devices (mobile phones, portable computers, etc.) are produced providing memory in megabytes level, which constitutes a valid infrastructure for installation. Therefore, our solution is applicable to the current consumer products.

Running time analysis show that the fastest algorithm is also the least secure one. The other algorithms, which we consider as being more secure, consume more time to accomplish their tasks. However, it important to notice that all four protocols work and finish their processes in less than half of a second. Considering the memory constraints, running time requirements, and security levels of these algorithms, we propose to select the algorithm with respect to the security needs of the system in consideration. In addition, *diffieHellmann* and *EllipticCurveDH* methods are able to establish variable sizes of secret keys, which in turn can be applied to the different systems with various sizes of keys.

In this paper, we introduced and implemented four methods for exchanging initial secret keys in ad hoc networking environment. Our reference implementation was based on Bluetooth technology and we used Java programming language on Linux platform. Since ad hoc networking has many applications, reference implementations can be tested in variety on applications. We are planning to embed our methods into real-life applications and discover which of them is most suitable for a given application.

References

1. N. Asokan and P. Ginzboorg: Key Agreement in Ad hoc Networks, Computer Communications, Vol.23, No.17, (November 2000)

2. S. Capkun, L. Buttyan, and J. Hubaux: Self-organized Public-Key Management for Mobile Ad Hoc Networks, IEEE Transactions on Mobile Computing, Vol.2, No.1, (January-March 2003)
3. J. Costa-Requena: Mobility and Network Management in Ad Hoc Networks, IASTED International Conference, Communication Systems and Networks (CSN 2002), September 9-12, 2002, Malaga, Spain.
4. L. Dao-Hui, L. Gang, and G. Bao-Xin: The Radio Networking of Bluetooth, Proceedings of 3rd International Conference on Microwave and Millimeter Wave Technology, 2002
5. D. A. Gratton: Bluetooth Profiles The Definitive Guide, Prentice Hall PTR, 2003
6. J.C. Haartsen: Bluetooth - The Universal Radio Interface for Ad Hoc, Wireless Connectivity, Ericsson Review, Vol. 75, No.3, pp.110-117 (1998)
7. IEEE Std. 802.11: Wireless LAN Specifications, IEEE, 1999
8. I. Millar, M. Beale, B. J. Donoghue, Kirk W. Lindstrom, and Stuart Williams: The IrDA Standards for High-Speed Infrared Communications, The Hewlett-Packard Journal, February-1998
9. R. Morrow: Bluetooth Operation and Use, McGraw-Hill Publishing, 2002
10. K. Nieminen: Introduction to Ad Hoc Networking, Networking Laboratory, Helsinki University of Technology
11. National Istitute of Standards and Technology (NIST): Recommended Elliptic Curves For Federal Government Use, July 1999, `http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/INISTReCur.pdf`
12. P. Papadimitratos, Z. J. Haas: Secure Routing for Mobile Ad Hoc Networks, Proceedings of SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002), January-2002
13. M. Särelä : Military Communication Systems, Experience and Applications, The European Physical Journal B, Vol. 10, pp.99-103, (1999)
14. F. Stajano: Security for Ubiquitous Computing, John Wiley, 2002
15. G. Steel, A. Bundy, and M. Maidl: Attacking the Asokan-Ginzboorg Protocol for Key Distribution in an Ad-Hoc Bluetooth Network Using CORAL, Proceedings of the 23rd IFIP TC6/WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'03, Berlin, Germany), pp.1-10, September 2003
16. S. Uskela: Link Technology Aspects in Multihop Ad Hoc Networks, Networking Laboratory, Helsinki University of Technology (2002)
17. J. Välimäki: Bluetooth and Ad Hoc Networking, `http://citeseer.ist.psu.edu/540179.html`
18. S. Williams and I. Millar: The IrDA Platform, HP Laboratories, Technical Report, HPL-95-29, (1995), `http://www.hpl.hp.com/techreports/95/HPL-95-29.pdf`
19. L. Zhou, Z. J. Haas: Securing Ad Hoc Networks, IEEE Network - Special Issue on Network Security, (November/December 1999), pp24-30.