AES Data Encryption in a ZigBee Network: Software or Hardware?

Geoffrey Ottoy[1,3], Tom Hamelinckx[2,3], Bart Preneel[1], Lieven De Strycker[2,3], and Jean-Pierre Goemaere[2,3]

[1] K.U. Leuven, COSIC research group, Kasteelpark Arenberg 10, bus 2446, 3001 Leuven-Heverlee, Belgium
[2] K.U. Leuven, TELEMIC research group Kasteelpark Arenberg 10, bus 2444, 3001 Heverlee, Belgium
[3] Catholic University College Ghent, DraMCo research group, Gebroeders Desmetstraat 1, 9000 Ghent, Belgium

Abstract. This paper describes the experiments which have been conducted to determine the optimal implementation concept for AES (Advanced Encryption Standard) data encryption in a ZigBee network [1,2]. Two concepts have been considered. The first one is a AES128-CBC hardware co-processor embedded on a Spartan 3A FPGA[1]. The second configuration implements the same cryptographic algorithm on the processor which controls the ZigBee nodes. The ZigBee modules in the network contain an 8-bit microcontroller which takes care of the ZigBee protocol stack –and the encryption calculations in the second case. Both approaches are examined and compared. In this paper we show that –in general– a software implementation is feasible in a ZigBee network, though a low-power hardware cryptographic co-processor could prove to be useful in some cases.

Keywords: software encryption, hardware encryption, AES, ZigBee.

1 Introduction

It is obvious that securing a wireless sensor network is a challenging task. Sensor nodes don't have the same computational power, memory and energy resources as e.g., a personal computer. Commonly, an 8-bit general purpose low power processor takes care of the software that controls the sensor node. A good portion of the program memory (typical 128 kB [3]) is required for controlling the sensors and the RF part of the module. The remaining memory space is available to implement a security algorithm. Batteries or energy scavengers can be used to power the nodes [4,5].

The ZigBee standard applies the Advanced Encryption Standard (AES) [6] for securing wireless transmissions. In this paper we evaluate the implementation of

[1] Field-Programmable Gate Array.

AES128-CBC, which is a symmetric key cryptographic algorithm. This algorithm is three to four orders of magnitude faster to compute, compared to a public key cryptographic algorithm with the same security level, such as RSA [7].

It is important to mention that, because of the aforementioned resource constraints, security in wireless sensor networks was originally considered exclusively through symmetric key cryptography. Symmetric cryptography is not as versatile as public key cryptographic techniques –especially when it comes to key management– which complicates the design of secure applications. Gaubatz *et al.* [8] show that special purpose ultra-low power hardware implementations of public key algorithms could be used on sensor nodes.

However, AES is widely used and it can be trusted as it gained confidence through the years. In addition, a lot of WSN-applications do not require the flexibility granted by the use of public key cryptography. Numerous papers discuss AES hardware implementations in various technologies which are very compact, energy efficient and with a high throughput (see e.g., [9,10,11]).

So we *know* that, by using hardware encryption we reduce the energy consumption. What we want to achieve in this paper however, is to point-out that, although being more efficient, a hardware encryption is not always the optimal solution. A balance needs to be found between the extra chip-cost and the gain or loss in energy consumption. To find this equilibrium, we have measured the share of each step in the data-transmitting process, for hardware as well as software.

In the following section we cover the important facts which need to be taken into account when implementing AES in a ZigBee network, in both soft- and hardware. In section 3 we give a general overview of the implementation setup. The software approach, as well as the hardware approach are elucidated. In section 4 we present the results of both implementations. And finally, in section 5 we state our conclusions and we give some opportunities for future work.

2 Relevant Considerations

Fig. 1 represents the ZigBee protocol stack [1]. The Application Layer (APL) and Network Layer (NWK) along with the Security Service Provider (SSP) are defined by the ZigBee Alliance [12]. The two lower layers –the Medium Access Layer (MAC) and the Physical Layer (PHY)– are defined by the IEEE 802.15.4 standard [13]. The following paragraphs will only consider the APL, NWK and SSP. The security in ZigBee consists of methods for key establishment, key transport, frame protection and device management. It is clear that, in this paper, we will focus on the aspect of frame protection. We will not consider the topics key establishment and transport. Instead, we work with pre-programmed keys.

In a ZigBee network, a so-called *open trust model* is used; i.e. it is considered that the different layers and all the applications on the device/node can trust each other. This has some consequences. First of all, it implies that the same keys can be used in the different layers –which reduces the storage and setup

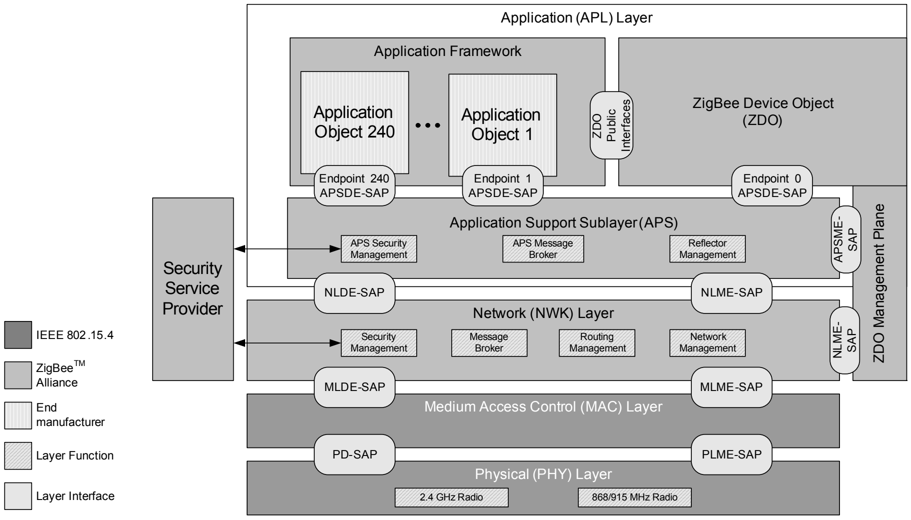

Fig. 1. ZigBee protocol stack

cost. Also, each layer is responsible for securing its own frames. For example, when the APL wants to send encrypted data, it has to do the encryption by itself because the NWK will not necessarily encrypt the frame. A major disadvantage of the open trust model is its weakness with regard to insider attacks. To simplify the communication and cooperation between different nodes, all the nodes in the network have the same security level. This means that each connection is as safe as any other connection in the network.

In general, we can consider two types of WSN. We will refer to them as centralized and distributed networks (Fig. 2). In the former, data flows from different nodes all over the network to one central sink node and, depending on each case, also back. In the latter, data flows between groups of nodes. An example of an application requiring a centralized network is a fire detection system. The smoke detectors are connected to a central unit which is able to e.g., control the fire alarm and call the emergency services. An example of a distributed network application is a lighting control system. A switch (or a couple of switches) is bound –to use the ZigBee terminology– to a light. Fig. 2 depicts the difference between the two types of WSN.

Generally, the nodes in these types of applications generate small amounts of data resulting in small data rates. However, different types of applications can exist in one WSN, increasing the total amount of data traffic. Moreover, as the number of nodes increases, the links closest to the central sink node can experience high data rates or packets can arrive with a larger delay [18]. We will try to study this situation by saturating a wireless link, thus testing the behavior of a node under stress.

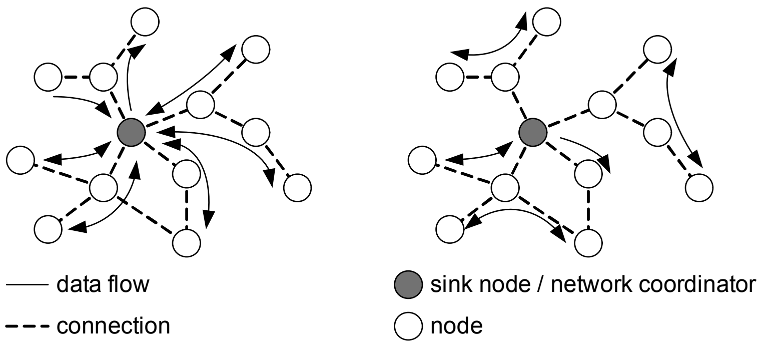

Fig. 2. A centralized WSN (left) versus a distributed WSN (right)

As for implementing AES, a very nice step-by-step overview of how to make a hardware implementation the algorithm is given in [14]. Each design decision is clarified and described in detail for each building block of the architecture. Here it is also stated that for low-end applications (like those in a WSN), high throughputs are not required. Techniques like loop-unrolling and pipelining – which lead to complex and expensive designs– can be omitted.

A detailed list of different AES implementations can be found in [15].

3 Implementation Concept

To illustrate the implementation of AES128-CBC in a ZigBee network, we set up a small network with a ZigBee coordinator and two ZigBee routers. The modules contain an 8-bit microcontroller ATmega1281 [16] and a Chipcon CC2420 transceiver [17]. The three nodes are connected to three different laptops to visualize what is happening in the respective nodes, as one can see in Fig. 3. The laptops are not doing any cryptographic calculations. These are performed by the nodes –possibly extended with an FPGA (see subsection 3.1).

In order to measure calculation speed, throughput and energy consumption of the implementation, we send a picture (282 kB) through the network. By sending this large amount of data[2], we put a maximum load on the wireless link, so we are able to see the effects of the security measures more distinctly. The sending node is able to encrypt the data using AES128-CBC and transmit the encrypted picture to the other node. This one contains the decryption algorithm. The laptop receives the decrypted photo and displays it on the screen. To prove the data was really encrypted and for demonstration purposes, an eavesdropper node is able to collect the data. But this node does not "know" the secret key, so the laptop shows an unreadable picture.

[2] In a "real-life" ZigBee network, nodes are not permanently sending data to reduce energy consumption.

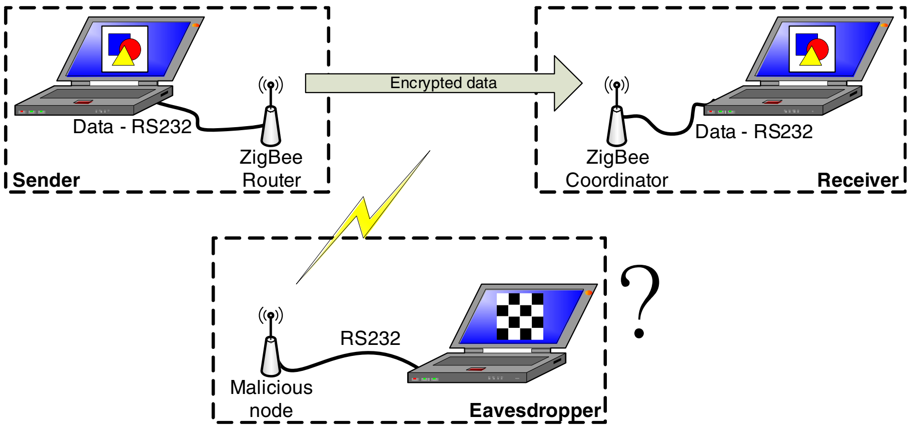

Fig. 3. Demonstration setup

3.1 The Hardware Implementation Setup

A first mechanism to achieve the aforementioned setup, is a hardware implementation of the AES algorithm in a Spartan 3 FPGA. The ZigBee module is now extended by the FPGA, which acts as some kind of co-processor –see Fig. 4. The photo is transmitted via an RS232 connection, encrypted by the FPGA and sent to the ZigBee module. The AES128-CBC algorithm is written in VHDL and is optimized for speed. For every 16 bytes, the algorithm calculates ten rounds in which the data is encrypted [9]. This requires 15 clock cycles, i.e. one clock cycle for each round and five cycles for control signaling.

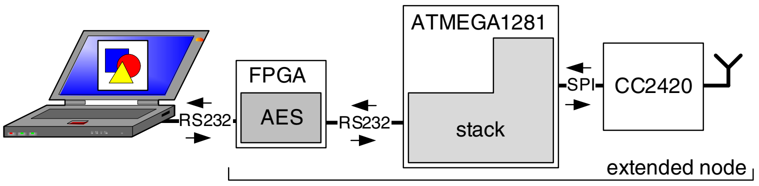

Fig. 4. Hardware implementation block diagram

Fig. 5 presents an overview of the different blocks used in the FPGA-design. A PicoBlaze (softcore) processor is implemented to control every hardware block. Upon receiving a data byte (serial), the byte is clocked into a shift register. When all 16 bytes (of a data block) are received, the whole block is clocked into the AES calculation block. After encryption, the data is sent to the ZigBee module over a serial line.

A hardware implementation could solve any limitation in program memory. Moreover, with a hardware implementation, the calculation speed promises to

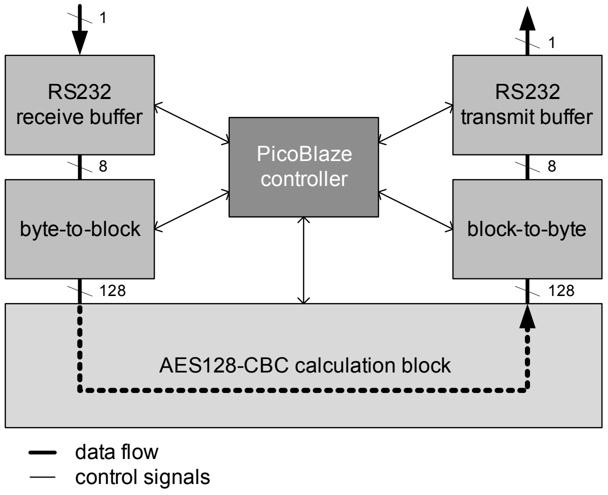

Fig. 5. FPGA-design block diagram

be much faster than any software implementation. Because the ZigBee module does not have to take care of the encryption calculation anymore, we expect the communication to go faster.

3.2 The Software Implementation Setup

Fig. 6 shows an overview of the software implementation. Instead of using a co-processor, the AES encryption/decryption is embedded on the ZigBee module's microcontroller. This controller already contains the code of the ZigBee protocol stack, so the remaining program memory space[3] and computation time are limited. The algorithm is implemented in C and optimized for 8-bit Atmel controllers. A node contains an encryption-only or a decryption-only implementation, utilizing the total available memory resources of the provided platform. This means that this concept is usable in an application where the nodes in the network do not have to decrypt, but send their encrypted data to a more powerful node (e.g., a sink node) which can take care of both.

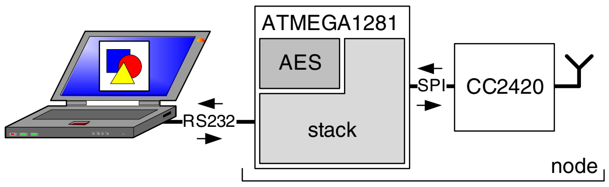

Fig. 6. Software implementation block diagram

[3] Our platform provides 128K program memory and 8K + 32K data RAM.

4 Implementation Results

In this section we give the results of the conducted measurements. These include power measurements for the software and hardware implementation as well as the influence of the encryption on the data rate. To interpret the results correctly, some additional data are given (Table 1). In Table 2 the measurements, necessary to calculate the energy consumption of both approaches, are shown.

Table 1. Additional data

Metric	***Value***	***Additional Info***
MCU clock	8 MHz	
ZigBee V_{supply}	9.04 V	we use a standard 9-Volt battery
P_{TX}	-15 dBm	the ZigBee transmission power
FPGA clock	8 MHz	the same as the MCU clock to make an "honest" comparison between both approaches
FPGA V_{int}	1.2 V	the internal core supply voltage
FPGA V_{aux}	3.3 V	the auxiliary supply voltage

Table 2. Current measurements

Metric	***Value***	***Metric***	***Value***
t_{enc_sw}	2.64 ms	I_{enc_sw}	31.5 mA
t_{enc_hw}	1.88 μs	$I_{enc\_hw\_int}$	16.0 mA
		$I_{enc\_hw\_aux}$	10.4 mA
t_{dec_sw}	3.56 ms	I_{dec_sw}	31.5 mA
t_{send}	1.80 ms	I_{send}	122.5 mA
$t_{receive}$	360 μs	$I_{receive}$	122.5 mA
		I_{sleep}	1 μA
t_{stack}	4.56 ms	I_{stack}	31.5 mA

Table 3. Energy consumption for manipulating 128-bit data blocks

Energy	***Software***	***Hardware***
E_{send}	2.01 mJ	2.01 mJ
$E_{receive}$	343 μJ	343 μJ
E_{enc}	782 μJ	0.1 μJ
E_{dec}	949 μJ	*

*We have not yet tested the hardware decryption, however if we extrapolate our results, we assume that E_{dec_hw} will not exceed 0.2 μJ.

The first striking result is the current drawn when the ZigBee module is transmitting (122.5 mA). The time needed to send a 128-bit data packet along with the necessary headers is 1.80 ms. This results in an energy consumption of 2.01 mJ (Table 3). The $I_{receive}$ –also 122.5 mA– is in fact the required current to send an acknowledgment frame to the sending node. The total time to receive a packet is approximately the same as for transmitting –with a current of 31.5 mA.

When the MCU[4] is performing the encryption or handling stack-related tasks (t_{stack}), the current consumption is the same (31.5 mA). This means that if we want to reduce E_{enc_sw} we need to reduce the encryption time. This is confirmed by the results of the hardware encryption. With a slightly lower current consumption, the small value of E_{enc_hw} is primarily due to the faster encryption (1.88 μs or 15 clock cycles).

We take the period between encryption and transmission as being t_{stack}. For the ease of programming, the MCU always stays in his active mode, but the rest of the time, the microcontroller could be in sleep mode (1 μA).

If we assume the node to be in sleep mode when not processing any data, the energy to transmit a secured data packet consists of: the energy needed to encrypt the data (E_{enc}), the energy needed to perform stack-related tasks E_{stack} and the energy required to send the data E_{send} –see Fig. 7. This figure is based on Fig. 8 (to be found in the appendix) and represents the power consumption of an encrypting ZigBee node.

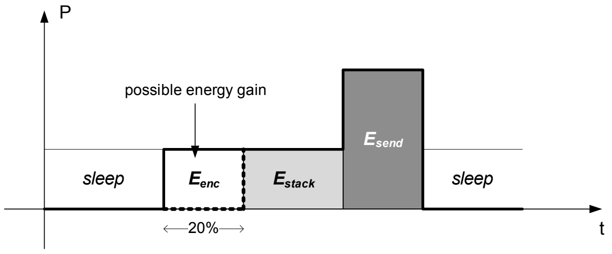

Fig. 7. Representation of possible energy gain when using hardware encryption

In our case, we believe that by eliminating the software encryption, we can save about 20% of energy. It is clear to see that the relative amount of energy gained, is highly dependend on P_{TX}.

The maximum throughput attainable with our ZigBee network (no encryption) is one packet of 16 byte every 61 ms. This results in a net data rate of 2.1 kbps. Consequently, in the approach with the hardware implementation we obtain the same data rate. The software encryption slightly lowers the data rate –16 bytes every 63.6 ms or 2.0 kbps– however, this is a negligible performance loss.

5 Conclusions and Future Work

5.1 Conclusions

We have shown that AES128-CBC can be used to integrate security in a ZigBee network. The algorithm can run in both hardware or software. A hardware

[4] Micro Controller Unit.

approach can help to counter program memory issues and in some extent, speed-up the processing of data. However the data rate is not increased greatly because the main bottleneck is not the software encryption itself, but the sluggishness of the network. Moreover, using extra hardware –which requires a redesign of the board layout– will increase the cost and complexity of a ZigBee module. On the other hand, the use of hardware encryption can help to reduce the energy required to send encrypted data over the network. However, this gain depends largely on P_{TX}. This implies that attempting to make a low-power application, and therefore also lowering P_{TX}, the software encryption will be responsible for a relatively large share in the total power consumption. In this case it could be more advantageous to use a hardware implementation. We must also note that, in our measurements, the transmission power is already at a very low level, so the relative gain (of 20 %) is almost maximized.

Altogether, we can state that implementing AES in software is feasible in most cases. However, when more processing power is needed –e.g., on sink nodes– or when designing very low-power devices, a hardware co-processor could prove to be useful. So for each application a thorough analysis needs to be made before taking a decision.

5.2 Future Work

Further research should be done to determine which of the proposed solutions (hard– or software) is the best performing in a high density network. In these networks, the central node could act as a bottleneck if the encryption is too slow. It could be useful to measure the performance of the two implementations and see if the results presented in this paper are still applicable.

Another interesting path is implementing a hardware cryptocore for the processor. When this hardware block is embedded in the processor, it could possibly combine the advantages of both solutions, thus creating an interesting new component that is fast, configurable without consuming more power than a non-encrypting node.

Acknowledgments. We would like to thank our former colleague Bert Dufraimont for his preceding work on this subject. Also thanks to the COSIC group at K.U.Leuven[5] for their VHDL implementation of the AES encryption algorithm.

References

1. ZigBee Alliance: ZigBee Specification – Document 053474r17 (2007)
2. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography, p. 230. CRC Press, Boca Raton (1996) ISBN: 0-8493-8523-7
3. Zia, T., Zomaya, A.: An Analysis of Programming and Simulations in Sensor Networks. School of Information Technologies, University of Sydney (2006)

[5] `http://www.esat.kuleuven.be/cosic/`

4. Mathews, M., Song, M., Shetty, S., McKenzie, R.: Detecting Compromised Nodes in Wireless Sensor Networks. In: Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (2007)
5. Kaps, J.P.: Cryptography for Ultra-Low Power Devices. A Dissertation Submitted to the Faculty of the Worcester Polytechnic Institute (2006)
6. Federal Information Processing Standards Publication 197: Specification for the Advanced Encryption Standard (AES) (November 2001)
7. Shi, E., Perrig, A.: Designing Secure Sensor Networks. IEEE Wireless Communications, 38–43 (December 2004)
8. Gaubatz, G., Kaps, J.P., Öztürk, E., Sunar, B.: State of the Art in Ultra-Low Power Public Key Cryptography for Wireless Sensor Networks. Cryptography Information Security Lab, Worcester Polytechnic, Institute (2005)
9. Park, S.J.: Analysis of AES Hardware Implementations. Department of Electrical and Computer Engineering, Oregon State University (2003)
10. Mangard, S., Aigner, M., Dominikus, S.: A highly regular and scalable AES hardware architecture. IEEE Transactions on Computers 52(4), 483–491 (2003)
11. Gielata, A., Russek, P., Wiatr, K.: AES hardware implementation in FPGA for algorithm acceleration purpose. In: Proceedings of the International Conference on Signals and Electronic Systems, ICSES 2008, pp. 137–140 (2008)
12. ZigBee Alliance, `http://www.zigbee.org`
13. IEEE Computer Society, IEEE Std 802.15.4-2006 (2006)
14. Rijmen, V., Pramstaller, N.: Cryptographic Algorithms in Constrained Environments (Chapter 6). In: Skalvos, N., Zhang, X. (eds.) Wireless Security and Cryptography, pp. 186–195. CRC Press, Boca Raton (2007) ISBN: 978-0-8493-8771-5
15. AES Lounge, `http://www.iaik.tugraz.at/content/research/krypto/AES/IAIK-TUGraz`
16. Atmel: ATmega640/1280/1281/2560/2561 [Datasheet] (2005)
17. Chipcon: CC2420 ZigBee-Ready Transceiver [Datasheet] (2003)
18. Sun, J., Zhang, X.: Study of ZigBee Wireless Mesh Networks. In: International Conference on Hybrid Intelligent Systems, pp. 264–267. IEEE Computer Society, Los Alamitos (2009)

Appendix: Graphs

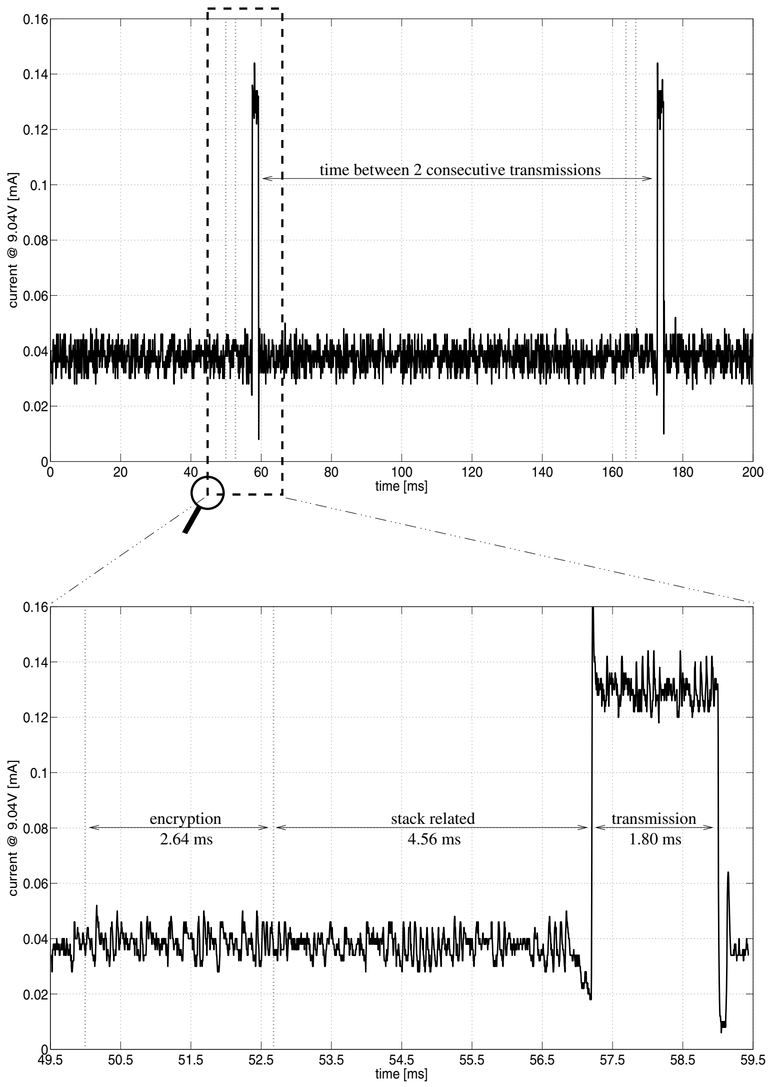

Fig. 8. Current consumption of an encrypting ZigBee module (top); current consumption in detail (bottom)