Performance and Energy Consumption Analysis of AES in Wireless Sensor Networks

Dan Dragomir and Cristina Panait

Abstract With WSN deployments increasing in popularity, securing those deployments becomes a necessity. This can be achieved by encrypting inter-node communications and/or using message authentication codes. AES is a well studied symmetric cipher, with no known practical vulnerabilities, that can be used to solve both problems. We provide an optimized implementation of AES, with five modes of operation for encryption (ECB, CBC, CFB, CTR and GCM) and two modes for authentication (CBC-MAC and GCM-MAC), that use the hardware accelerator available on the ATmega128RFA1 microcontroller, and compare it with the best known software implementation. We show that our hardware AES implementation is both faster and more energy efficient than a software implementation. This is not the case for previous sensor nodes and implementations, which show an improved execution speed, but with a higher energy consumption. We also show that our implementation of CTR is faster and more energy efficient than the unsecure fully hardware-supported ECB mode.

1 Introduction

Wireless sensor networks (WSNs) and their applications present an increased risk to a series of attacks, which can affect a network's operation. As a solution to these problems we present a performance and energy consumption analysis of AES-128 on WSN hardware. AES is a well studied block cipher with no known practical vulnerabilities, has a speed comparable with other symmetrical encryption algorithms and is supported on multiple WSN platforms through a hardware acceleration module. This work is an extended version of [12] and improves the previous study in two key areas. Firstly, it further optimizes the hardware-assisted implementation, leading

D. Dragomir (✉) · C. Panait
Faculty of Automatic Control and Computers—University POLITEHNICA of Bucharest, Splaiul Independentei 313, 060042 Bucharest, Romania
e-mail: dan.dragomir@cs.pub.ro

C. Panait
e-mail: cristina.panait19@gmail.com

M. Grzenda et al. (eds.), *Advances in Network Systems*, Advances in Intelligent Systems and Computing 461, DOI 10.1007/978-3-319-44354-6_11

to better performance and lower energy consumption and secondly, it adds two new algorithms to the analysis, that handle message authentication and verification.

As a general definition, a wireless sensor network is composed of a set of nodes which communicate through a wireless medium in order to perform certain tasks. A couple of examples where WSNs can be deployed, as stated in [6], are: fire extension detection, earthquake detection, environment surveillance for pollution tracking, intelligent building management, access restriction, detection of free spaces in parking lots and so on. WSNs bring a number of advantages in these situations, like enhanced flexibility and mobility, mainly because nodes are generally powered from an on-board battery and are thus self sufficient. This, however, is also their biggest weakness. The lifetime expectancy of a node depends on its usage. The constraints mainly come from the limited energy source, as data processing and transmission can be energy intensive.

The particular characteristics of these types of networks make the direct implementation of conventional security mechanisms difficult. The imposed limitations on minimizing data processing and storage space, and reducing bandwidth need to be addressed. The major constraints for WSNs, as presented in [2, 14, 17], are: energy consumption (which can lead to premature exhaustion of the energy source and to the denial of service), memory limitations (flash, where the application source code is stored, and RAM, where sensed data and intermediary computing results are stored), unreliable communication (the routing protocols used, collisions), latency (which can lead to synchronization issues and algorithms that cannot act correctly) and unattended nodes (an attacker could have physical access to the nodes).

In Sect. 2 we discuss some of the related work. Sections 3 and 4 present the algorithm design and modes of operation and the implementation with two methods, software and hardware. Then, in Sect. 5, we make a comparative analysis of the solutions, based on execution time and energy consumption, and select the encryption and authentication methods suitable for ATmega128RFA1-based platforms, taking also into consideration the provided security. Finally, we present the conclusions of our work.

2 Related Work

The problem of measuring the cost of encryption on wireless sensor node hardware has been addressed previously. In [9] Lee et al. analyze a range of symmetric-key algorithms and message authentication algorithms in the context of WSNs. They use the MicaZ and TelosB sensor nodes and measure the execution time and energy consumption of different algorithms. For AES they provide measurements for a hardware assisted implementation and conclude that it is the cheapest when either time or energy is considered. They do not however study this implementation on different plaintext lengths and instead rely on datasheets to extend to lengths longer than one block. However, this conclusion is not backed by Zhang et al. [19] which compares different AES implementations on the MicaZ nodes. They conclude that hardware

assisted encryption is faster, but also consumes more energy due to the external chip which handles the computation in hardware.

Compared to their work, we study only AES-128 which is a well known cipher also adopted by the National Institute of Standards and Technology (NIST) and which has been proposed as a viable alternative [8] to other less studied ciphers in WSN applications. This choice is also supported by the fact that multiple 802.15.4 transceivers offer a hardware accelerator for AES operations. We study the newer Sparrow v3.2 sensor nodes based on the ATmega128RFA1, which integrates the microcontroller with the radio transceiver and hardware encryption module, and show that AES-128 can be efficiently implemented reducing both execution time and energy consumption. We also provide hybrid implementations for modes of operation that are not natively supported by the hardware and show that they can still be efficiently implemented with the available primitives.

In [8] Law et al. conduct a thorough survey of the costs of different block ciphers, when implemented on sensor node hardware. They conclude that Rijndael (AES) is the second most efficient cipher, being surpassed only by Skipjack. However, their analysis is based on older hardware and does not consider any hardware accelerated implementations.

In [4] de Meulenaer et al. study the problem of key exchange and measure the cost of two key agreement protocols: Kerberos and Elliptic Curve Diffie-Hellman. They measure the energy consumption of the two protocols on MicaZ and TelosB sensor nodes and conclude that the listening mode is the principal factor in the energy efficiency of key exchange protocols, with Kerberos being the more efficient protocol. Compared to their work, we concentrate on encryption and authentication algorithms, and more specifically on AES, with key distribution left for future work.

In a previous work [12] we reported the performance and energy consumption of our implementation for 4 modes of operation that use the AES block cipher: ECB, CBC, CFB and CTR. The current work extends that analysis by also studying the cost of data authentication using CBC-MAC and adding a new mode of operation, GCM, that seamlessly supports both encryption and authentication using the same secret key. We also improve on the performance and energy consumption of our previous implementation for some of the modes, by making use of pipelining, in the hardware assisted case.

3 Design

AES is a block cipher encryption algorithm that uses symmetrical keys for encrypting a block of plaintext and decrypting a block of ciphertext [3]. The algorithm uses a series of rounds consisting of one or more of the following operations: byte-level substitution, permutation, arithmetical operations on a finite field and XOR-ing with a given or calculated key [15]. As a general rule, the operations are handled bytewise.

AES receives as input a plaintext of 16 bytes and the encryption key, which has a variable dimension of 16, 24 or 32 bytes. The input text is processed into the output

text (ciphertext) by using the given key and applying a number of transformations. Encryption and decryption are similar, except for the fact that decryption needs an extra step—it first runs a full encryption in order to obtain the modified key needed for decrypting data.

In [13], Schneier divides symmetrical encryption algorithms in two basic categories: block ciphers and stream ciphers. A block cipher encrypts a block of plaintext producing a block of encrypted data, whilst a stream cipher can encrypt plaintexts of varying sizes. This makes block ciphers prone to security issues if used to encrypt plaintexts (in a naïve way) longer than the block size, mainly because patterns in the plaintext can appear in the ciphertext.

A more secure way to encrypt data with a block cipher can be achieved by combining the resulting ciphertext blocks using a few basic operations, resulting in what is called a *mode of operation* for that block cipher. It is worth mentioning that the combining operations are not directly securing the data. This is the responsibility of the block cipher. The operations however are used to maintain the security of the cipher when it is operated on plaintexts longer than the block size.

Another use case for a block cipher is to compute a Message Authentication Code (MAC) for a piece of data, in order to provide data authentication. As for the encryption case, a naïve implementation would not provide the necessary guarantees for the generated MAC. Extra precaution must be taken when both encryption and authentication are required as not all combinations of encryption and authentication methods provide the desired properties. For a thorough treatment of the security properties of the different options we refer to Katz and Lindell [7].

3.1 Electronic Code Book (ECB)

The ECB mode of operation receives blocks of plaintext, respectively ciphertext, and a key and produces corresponding blocks of ciphertext, respectively plaintext. One property of this mode of operation is that two blocks of plaintext, encrypted with the same key, will result in two identical blocks of ciphertext. ECB is the most simple mode of operation. However, one major drawback is that it does not hide data patterns, meaning that identical ciphertext blocks imply the existence of identical plaintext blocks.

3.2 Cipher Block Chaining (CBC)

The CBC mode of operation takes as input parameters the plaintext, respectively the ciphertext, the key and an initialization vector (IV). One property of CBC is that two encrypted blocks are identical only if their respective plaintexts have been encrypted using the same key and the same IV. Unlike ECB, CBC has link dependencies, as its basic chaining mechanism makes the ciphertext blocks dependent on

previously encrypted data. This, coupled with a randomly chosen IV, ensures that identical plaintext blocks will be encrypted to different ciphertext blocks.

With slight modifications CBC can be transformed to provide authentication instead of encryption, resulting in CBC-MAC. For this, the initialization vector must be fixed to a constant value (usually 0) and only the last encrypted block must be kept, which is also the authentication tag. This scheme will provide a secure MAC, but only for messages of fixed length (agreed ahead of time by the communicating parties). A further modification of prepending the length of the message as the data in the first block of the encryption, can make this mode secure for varying length messages. A disadvantage of CBC-MAC is that when both encryption and authentication are required, separate keys must be used for the two steps in order to maintain security.

3.3 Cipher Feedback (CFB)

The CFB mode of operation is very similar to CBC regarding its input parameters and the operations it performs. The main difference between them lies in the fact that CBC works as a block cipher, while CFB can be used as a stream cipher. Unlike CBC, CFB can encrypt variable-length blocks (which are not restricted to 16 bytes). The properties of this mode of operation are similar with the ones of CBC. One key difference between the two can be observed at the implementation level: CFB uses only the encryption primitive of the underlying block cipher, both for encrypting and for decrypting data.

3.4 Counter (CTR)

The CTR mode of operation [10] also produces a stream cipher. The IV used in CBC and CFB is now associated with the starting value of a counter, which is incremented and used to encrypt each block in turn. In this mode, the output from an earlier block is not used for computing the current block, as the previous two modes of operation. For the described system to work, a generator is needed on each side of the communication. The generators have to remain synchronized in order to produce the same stream of counter values on both sides. A disadvantage of this mode of operation is the possible desynchronization of the communicating entities. This results in the incorrect decryption of all subsequently received data.

A closely related mode of operation is Counter with CBC-MAC (CCM) [18], which provides both encryption and authentication of the plaintext data. In this mode, CBC-MAC is initially run over the message in order to obtain a authentication tag, and then, CTR mode is run on both the plaintext data and the authentication tag to obtain the ciphertext, which now provides both encryption and authentication. A

slight variation of CCM, called CCM*, is part of the IEEE 802.15.4 standard [5] for wireless personal area networks.

3.5 Galois/Counter Mode (GCM)

The GCM mode of operation [11] combines the Counter mode with operations on a Galois field in order to produce another mode of operation which provides both encryption and authentication of a piece of data using a single secret key. The key operation is a multiplication in $GF(2^{128})$, defined by the polynomial $x^{128} + x^7 + x^2 + x + 1$, which is used to define a hashing function that generates the authentication tag. The algorithm supports additional authenticated data (AAD), which is data protected against tampering by the authentication tag, but left unencrypted. This additional data is useful in networking protocols, where source and destination information must be left in cleartext for the purpose of routing. The algorithm can be easily converted to an authentication only mode of operation by providing only AAD and no encryption payload.

4 Implementation

A practical example would be a wireless sensor network, which transmits data gathered from three types of sensors: temperature, humidity and luminosity. Because of privacy and integrity concerns all data must be encrypted during transmission and routing information must be authenticated. The working platform for this scenario is based on the Sparrow v3.2 node [16]. Its technical specifications are:

- CPU: ATmega128RFA1, 16MHz
- Memory: 128KB flash, 16KB RAM
- Bandwidth: up to 2Mbps
- Programming: C/C++

The ATmega128RFA1 microcontroller is actually a SoC (System on Chip) which incorporates a radio transceiver compatible with the IEEE 802.15.4 standard [1]. It offers, among other things, a relatively low energy consumption (mostly in sleep states), a FIFO buffer of 128 bytes for receiving and transmitting data, a partial hardware implementation of the MAC layer and support for AES-128.

This microcontroller facilitates secured data transmissions by incorporating a hardware acceleration module which implements the AES algorithm. The module is capable of both encrypting and decrypting data, with most of the functionality implemented directly in hardware. It is compatible with the AES-128 standard (the key is 128 bits long) and supports encryption and decryption for ECB mode, but only encryption for CBC mode. The input to these operations consists of the plaintext/ciphertext block and the encryption key. Note that for decryption, the extra round

needed by AES to compute the decryption key is performed automatically. Other modes of operation are not supported by the hardware.

As we already stated in the previous sections, energy consumption is the main issue and challenge for WSNs. In order to obtain the best approach for ensuring confidentiality and integrity with minimal energy consumption, we implemented and compared AES-128, coupled with the ECB, CBC, CFB, CTR and GCM modes of operation. All five modes have both a hardware and a pure software implementation. Since only ECB has a full hardware implementation, for the other modes we used a hybrid approach, combining the hardware part from ECB with software implementations for the remaining operations. We also refer to these hybrids as hardware implementations. Were possible we pipelined the algorithm's execution so that the extra software steps not implemented in hardware were overlapped with the encryption of the next data block, thus achieving better performance than the serial solution employed in [12]. For the pure software implementation we used an optimized version of AES, called TableLookupAES [19].

5 Evaluation

5.1 Experimental Setup

To measure the energy consumption of our implementation, we perform two kinds of measurements: the time required (t) and the current drawn by the node (I) while encrypting/decrypting. Using the formula $E = P \cdot t$, where $P = U \cdot I$ is the power required by the node, we can compute the energy consumed by the algorithm, be it implemented in software, in hardware or using a hybrid approach. We ensure a constant voltage U using a voltage regulator, as explained in the next subsection.

In certain applications, the latency of encrypting/decrypting a given payload might be more important than the energy consumed. For this reason, this section also presents the timing results of the different solutions, independent of the energy measurements. As we later show, the current drawn by the node using both software and hardware security approaches is practically the same. Thus, the time taken is a sufficient metric for relative comparisons between the different solutions.

5.1.1 Current Measurement

For the purpose of measuring the energy consumption of the Sparrow sensor node during our experiments, we built a current sensing circuit based on the INA 193 current shunt monitor.

Figure 1 presents the circuit we designed. Power is provided by a $3.3V$ voltage regulator, which ensures a constant voltage regardless of the current drawn by the circuit. A shunt resistor connected in series with the Sparrow node acts as a cur-

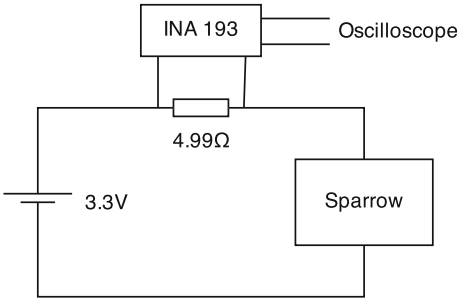


Fig. 1 Current measurement setup

rent sensor. The voltage drop on the resistor is directly proportional with the current drawn by the circuit. This has two implications. On the one hand, the chosen resistor value must be small enough not to disturb the rest of the circuit (e.g. by incurring a big voltage drop). On the other hand, the same value has to be big enough so that the expected currents register a voltage drop that can be sensed with enough precision. In order to improve the measurement precision and sensitivity, without the drawbacks of a big resistor value, we employ a INA 193 current shunt monitor, which provides a constant gain of 20 V/V on the input voltage drop, and a 4.99 Ω precision resistor with a tolerance of 0.01 %. The output of the current sensing circuit is connected to a Metrix OX 5042 oscilloscope which we used to monitor the current drawn by the node during the different encryption/decryption operations. Determining the current is as simple as dividing the voltage shown on the oscilloscope by the current shunt monitor gain (20 V/V) and the shunt resistor value (4.99 Ω).

5.1.2 Time Measurement

Using the oscilloscope, we also measure the time required for each encryption/decryption operation. The oscilloscope has a function that accurately measures pulse duration. We create a pulse lasting for the duration of the operation by setting a GPIO pin before the start of the operation and clearing it after it ends. Using this method, we can measure the duration of an operation with minimal overhead: 1 bit set instruction and 1 bit clear instruction, each taking 2 cycles.

Although the proposed measurement scheme is precise, it has the disadvantage of requiring manual intervention. The available oscilloscope cannot be interfaced with a PC, so a measurement point is obtained by uploading a program which encrypts a hardcoded message length in a loop, reading the information from the oscilloscope and repeating the process for all message lengths.

In order to automate the time measurements, we resorted to a software implementation running along side the encryption/decryption operation, that measures the time required. To keep overhead to a minimum, our solution employs the hardware timer module available on the ATmega128RFA1 to count the number of cycles taken by the operation. Each operation is measured by sampling a counter before and after the

operation and taking the difference of the two values. The count is then converted to a time value given that the microcontroller operates at $16MHz$.

This time measurement solution allowed us to automate the whole process of evaluating the algorithms for different message sizes. A small overhead can be observed between the software based time measurement and the oscilloscope based one, but the relative difference between the algorithms is unaffected. If absolute numbers are required, the software-based measurements can be corrected by noticing that the overhead increases linearly with the message size when compared with the oscilloscope measurements.

5.2 Results

We conducted multiple experiments, to evaluate both the time taken and the energy consumed by AES encryption/decryption and authentication/verification operations. We measured our hardware assisted implementation against the pure software implementation based on look-up tables.

5.2.1 Time Experiments

We started of with measuring the difference between the optimized software implementation and our hardware assisted implementation for each of the five studied modes of operation. For each type of implementation and operation mode, we measured the time taken by an encryption operation and a decryption operation on varying message lengths. We used message lengths from 1 byte to 127 bytes, which is the maximum packet size allowed by the transceiver and the 802.15.4 standard.

As can be seen in Fig. 2, the hardware assisted implementation easily outperforms the optimized software implementation for 4 of the 5 modes. For GCM the difference is not as pronounced as the other modes, but the hardware assisted implementation is still faster for all message lengths. The reason the difference is less pronounced is that, compared to the other four modes, GCM takes a longer time to compute. That time is used by the software implementation of the $GF(2^{128})$ multiply operation, which cannot be accelerated in hardware on the ATmega128RFA1. The time saved for doing the block cipher in hardware is small compared with the multiply operation, which leads to a less pronounced speed-up. The staircase shape of the graph is easily explained by the requirement of every block cipher, including AES, to operate on multiples of the block size. Plaintext sizes that are not a multiple of the block size need to be padded in most cases, thus still incurring the cost of an entire block.

The difference in performance varies between $\sim$7.0$\times$ for the ECB mode, which is fully supported in hardware, down to $\sim$1.15$\times$ for the GCM mode, which is only partially supported in hardware through the AES single block encryption primitive. The MAC version of GCM has an even lower speed-up, of $\sim$1.01$\times$, which is explained by the fact that in a pure authentication mode, GCM only performs one block encryp-

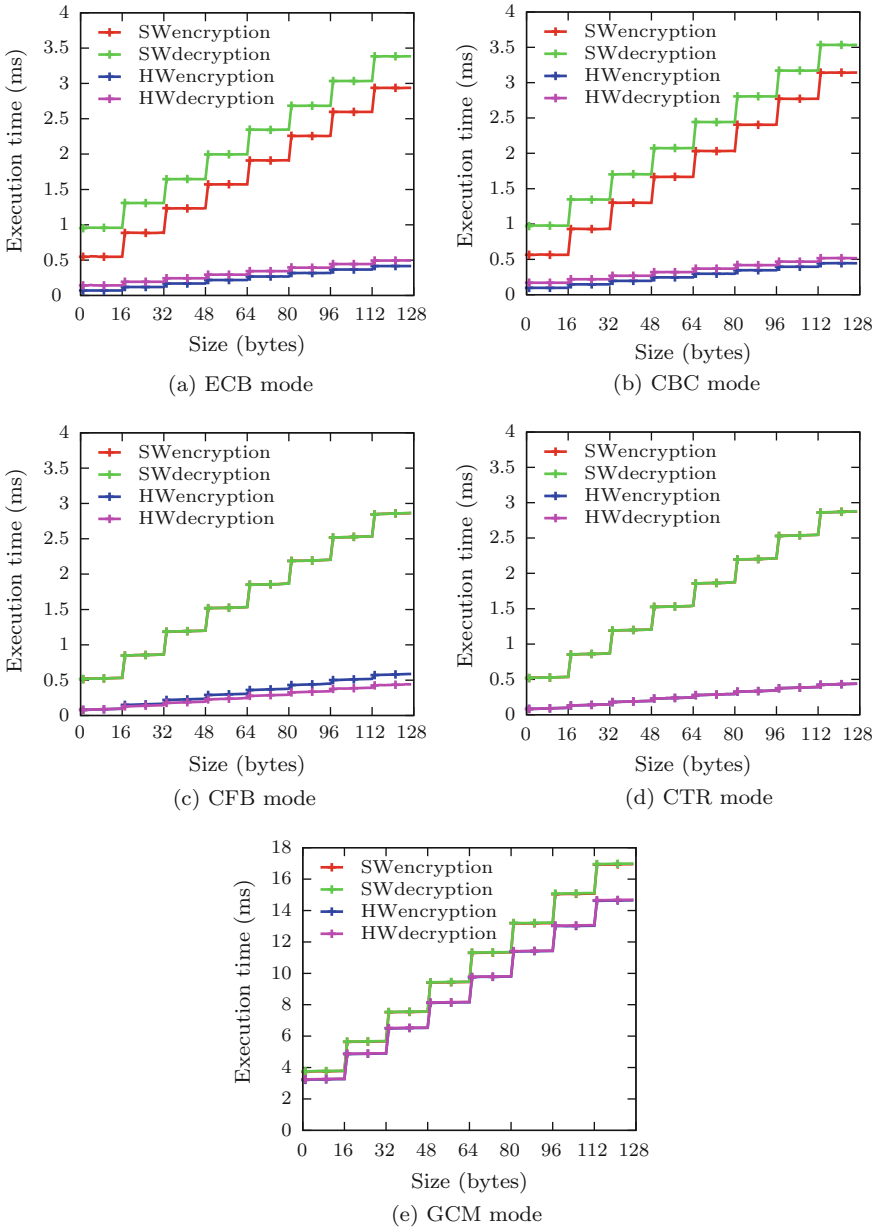


Fig. 2 Comparison between software and hardware implementations of AES encryption modes

Table 1 Execution speed-up hardware versus software

	Encryption	Decryption	Authentication	Verification
ECB	7.01×–7.94×	6.44×–6.84×	–	–
CBC	5.78×–7.07×	5.75×–6.82×	5.40×–7.72×	4.82×–7.46×
CFB	4.86×–6.51×	5.55×–6.69×	–	–
CTR	5.45×–6.75×	5.47×–6.75×	–	–
GCM	1.15×	1.15×	1.01×–1.08×	1.01×–1.08×

tion regardless of the message length. The difference in performance between the optimized software implementations and our hardware assisted implementations is further summarized in Table 1.

For the ECB and CBC modes we can observe (Fig. 2a, b) the extra preparation step needed by the single block decryption primitive, which makes decryption slightly more time consuming than encryption. No difference can be observed (Fig. 2c, d) between encryption and decryption for the CFB and CTR modes in the software implementation, because they use the same encrypt primitive of AES for both encryption and decryption, albeit with some extra processing. In the hardware assisted case CTR maintains equal performance between encryption and decryption, however CFB encryption gets progressively slower as the message length increases. This is caused by the extra software processing step, which cannot be hidden by pipelining, as is done for CTR, because of data dependencies in CFB encryption. Again, no performance difference can be observed (Fig. 2e) in the GCM case between encryption and decryption. What is relevant for GCM though, is the time required for either the software or hardware implementations which is almost five times longer than even the slowest of the other modes.

The behavior of the authentication modes is shown in Fig. 3. Both modes have an almost equal performance between generating the MAC and verifying it, in both

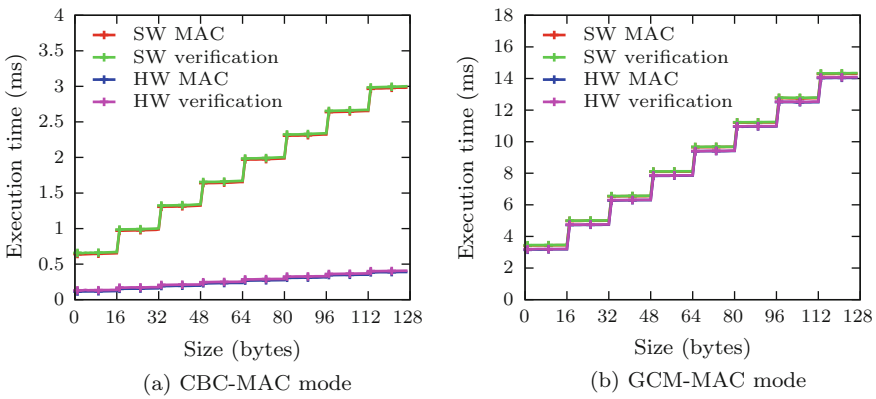


Fig. 3 Comparison between software and hardware implementations of AES based MACs

the software and the hardware implementations. This was to be expected as the verification step implies recomputing the authentication tag and comparing it with the received one. What is notable is that the comparing step adds minimal overhead compared with the tag calculation. Like in the encryption case, the hardware implementation is again much faster for CBC (Fig. 3a), while for GCM (Fig. 3b) the speed-up gained from the hardware assist is dwarfed by the time required for the $GF(2^{128})$ multiplication.

Figure 4 compares the hardware assisted implementations of 4 of the modes (ECB, CBC, CFB and CTR) against each other, during encryption and decryption. GCM was left out of this comparison as its hardware assisted performance was poor compared with the other modes. For encryption, ECB has the lowest runtime for all sizes, which was to be expected, as it does no extra operations on the output of the encrypt primitive to mask patterns in the plaintext. CTR is slightly worse, but not by much. The cost of the extra XOR operation required by this mode is mostly hidden by our pipelined implementation. This is not the case in the non-pipelined implementation measured in [12]. CBC is slightly worse, as its extra XOR operation cannot be

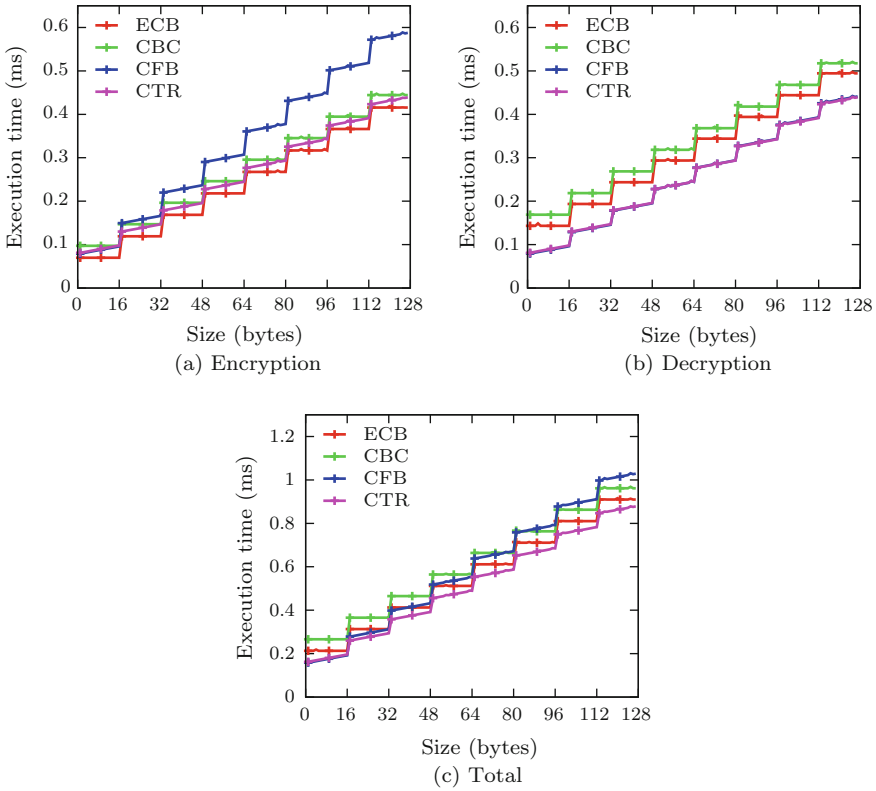


Fig. 4 Comparison between modes of operation with hardware acceleration

pipelined. The difference remains constant though, as the message length increases because only for the first block the XOR operation is emulated in software. For all the other blocks it is implemented by the hardware accelerator. Finally, CFB has the worst performance of the 4 modes mostly attributed to its extra XOR operation that cannot be pipelined like in the CTR case, because of data dependencies in the the algorithm. The cost of the extra operations increases as the encrypted message gets longer.

For decryption, CFB and CTR have a considerable advantage over ECB and CBC, as they only use the encrypt primitive, which has a smaller setup time than the decrypt primitive. Also, in the decryption case, the cost of the emulated XOR operation required by CBC, CFB and CTR is hidden by our pipelined implementation, thus giving a constant difference over all message lengths.

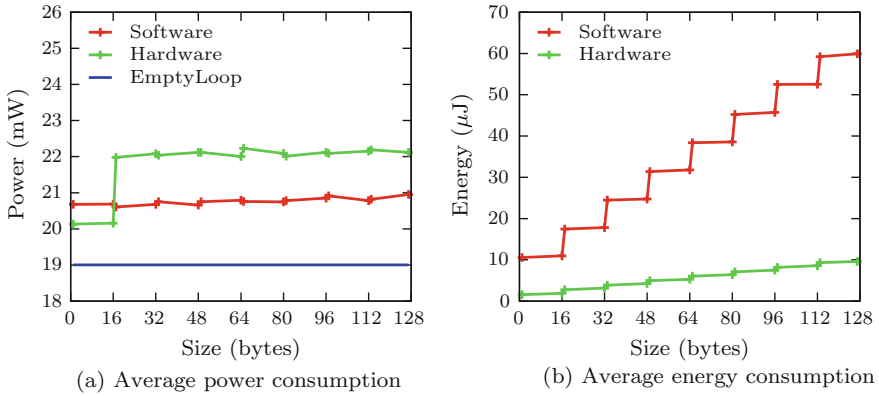
Another performance characteristic shown by both plots is the streaming nature of CFB and CTR. They can be optimized to reduce the performance cost when the message only covers part of a block. This can be seen as the slanting portions of the lines for CFB and CTR in both encryption and decryption. No such behavior is visible for ECB and CBC, which must fully process a whole block, even if only one byte of the message is contained in the block.

If we look at the cumulated time of both encryption and decryption (Fig. 4c), CTR holds a consistent advantage over all the other modes. CFB also holds an advantage over the unsecure ECB up to messages of 40 bytes. Compared with CBC, CFB is faster up to messages of 80 bytes. Thus, even if most WSN hardware offers accelerated support for AES-CBC, CFB and especially CTR can be better alternatives even if they are not completely accelerated in hardware.

5.2.2 Energy Experiments

For energy consumption we concentrated our efforts on determining the cost of using AES in CTR mode. We chose this mode based on the fact that the timing measurements showed it to be the best encryption/decryption mode for all message sizes. We only performed measurements for message encryption, as decryption is identical in terms of the code which is ran. We measured the cost of doing the encryption in software as well as the cost of using our hardware accelerated implementation. For completeness, we also measured the cost of an empty processing loop to compare against the two encryption implementations.

In our experiments, we used the measurement circuit described in Sect. 5.1.1 to measure the average current drawn during encryption, as well as the voltage and duration of the operation, as reported by the oscilloscope. As with the timing measurements, we performed the experiment for different message sizes, from 1 byte to 127 bytes. The oscilloscope was configured to report the mean over 16 samples in order to obtain the average energy consumption of the device. An instantaneous energy consumption is hard to obtain and is irrelevant when considering the long time operation of the node.



(a) Average power consumption (b) Average energy consumption

Fig. 5 Power and energy consumption of AES encryption in CTR mode

Using the raw current and voltage measurements, we plot the average power drawn with respect to the encryption size. As can be seen in Fig. 5a, for message sizes larger than 16 bytes the hardware implementation consistently draws more power than the software solution. This is in contrast with our previous measurements [12] which showed mostly equal amounts of power drawn by the two implementations. The difference though is that this new hardware implementation uses pipelining to overlap the hardware accelerated block encryption with the emulated XOR operation, thus using more of the transistors on the chip at a given time. This can also be seen in the first part of the graph, for message lengths less than 16 bytes, where a single block exists and pipelining is not possible. In this case the power drawn by the two solutions is more or less equal.

If we plot the average energy consumed by the encryption operation (Fig. 5b), we see a mostly linear increase in energy consumption with increasing plaintext size. The higher power needed by the pipelined hardware assisted implementation is more than offset by the lower running time, thus leading to a lower overall energy consumption. We believe the same conclusion holds for other operation modes, like CBC and CFB, as they mostly use the same operations as CTR, but in a slightly different order.

6 Conclusion

In this paper an updated evaluation of the cost of adding AES-128 encryption to WSN communications has been presented. We expand the work in [12] with more modes of operation and an improved pipelined implementation for some of the modes. Both the time penalty as well as the more important (from the point of view of a WSN) energy penalty have been analyzed for: ECB, CBC, CFB, CTR and GCM and for two implementations: a pure software implementation, based on the opti-

mized table lookup AES and the hardware accelerated implementation, that uses the AES hardware module of the ATmega128RFA1 microcontroller.

We showed how the AES hardware module in the ATmega128RFA1 microcontroller can be used to implement other modes of operation than the ones supported natively. Our solution uses a hybrid approach that runs some operations in hardware and emulates the missing ones in software. Where possible we pipeline the algorithm's execution to completely hide the cost of the emulated operation in terms of processing time. Using this approach, we implemented CBC decryption, as well as three full modes of operation for AES, CFB, CTR and GCM which do not have direct hardware support.

We presented a methodology of accurately measuring the power consumption using low cost components and a way of determining the encryption/decryption duration using only the wireless node itself. We compared the different modes of operation and concluded that a pipelined and hardware assisted implementation of CTR can be faster than even the unsecure and completely hardware accelerated ECB mode. CFB is also a better overall alternative to CBC for message sizes smaller than 80 bytes. This is true even though the hardware accelerator has native support for the CBC mode and it relates to the way decryption works for CBC. In contrast, GCM has performed very poorly in the hardware assisted case, even though a small speedup was obtained when compared to a pure software implementation. If authenticated encryption is required, using CTR for encryption followed by CBC-MAC for authentication represents the best combination in terms of both time performance and energy consumption.

We also built on the work of Zhan [19] and showed that the newer ATmega128 RFA1 microcontroller with an integrated transceiver, used in the Sparrow v3.2 node, can reduce both the duration and the energy consumption of AES operations. This is in contrast to work done on previous sensor nodes, that used a separated microcontroller and transceiver and which had a higher energy cost when running the encryption in hardware as opposed to using a pure software implementation.

References

1. Atmel: 8-bit AVR Microcontroller with Low Power 2.4 GHz Transceiver for ZigBee and IEEE 802.15.4
2. Carman, D.W., Kruus, P.S., Matt, B.J.: Constraints and approaches for distributed sensor network security (final). Technical Report 1, NAI Labs, Cryptographic Technologies Group, Trusted Information System (2000)
3. Daemen, J., Rijmen, V.: The block cipher Rijndael. In: Smart Card Research and Applications, pp. 277–284. Springer (2000). doi:10.1007/10721064_26
4. De Meulenaer, G., Gosset, F., Standaert, O.X., Pereira, O.: On the energy cost of communication and cryptography in wireless sensor networks. In: IEEE International Conference on Wireless and Mobile Computing Networking and Communications, 2008. WIMOB'08, pp. 580–585. IEEE (2008). doi:10.1109/WiMob.2008.16
5. IEEE 802 Working Group: IEEE standard for local and metropolitan area networks - Part 15.4: Low-rate wireless personal area networks (LR-WPANs). IEEE Std 802, 4–2011 (2011)

6. Karl, H., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. Wiley (2007). doi:10.1002/0470095121
7. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. CRC Press (2014)
8. Law, Y.W., Doumen, J., Hartel, P.: Survey and benchmark of block ciphers for wireless sensor networks. ACM Trans. Sensor Netw. (TOSN) **2**(1), 65–93 (2006). doi:10.1145/1138127.1138130
9. Lee, J., Kapitanova, K., Son, S.H.: The price of security in wireless sensor networks. Comput. Netw. **54**(17), 2967–2978 (2010). doi:10.1016/j.comnet.2010.05.011
10. Lipmaa, H., Wagner, D., Rogaway, P.: Comments to NIST concerning AES modes of operation: CTR-mode encryption (2000)
11. McGrew, D., Viega, J.: The Galois/Counter mode of operation (GCM). Submission to NIST. http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec.pdf (2004)
12. Panait, C., Dragomir, D.: Measuring the performance and energy consumption of AES in wireless sensor networks. In: Proceedings of the 2015 Federated Conference on Computer Science and Information Systems, pp. 1261–1226 (2015). doi:10.15439/978-83-60810-66-8
13. Schneier, B.: Applied Cryptography: Protocols, Algorithms, and Source Code in C. Wiley (1996)
14. Sen, J.: Routing security issues in wireless sensor networks: attacks and defenses. In: Seah, W., Tan, Y.K. (eds.) Sustainable Wireless Sensor Networks, pp. 279–309. InTech (2010). 10.5772/663
15. Stallings, W.: Cryptography and Network Security—Principles and Practice, 5th edn. Pearson Education (2011)
16. Voinescu, A., Tudose, D., Dragomir, D.: A lightweight, versatile gateway platform for wireless sensor networks. In: Networking in Education and Research, 2013 RoEduNet International Conference 12th Edition, pp. 1–4. IEEE (2013). doi:10.1109/RoEduNet.2013.6714202
17. Wang, Y., Attebury, G., Ramamurthy, B.: A survey of security issues in wireless sensor networks (2006). doi:10.1109/COMST.2006.315852
18. Whiting, D., Housley, R., Ferguson, N.: AES encryption & authentication using CTR mode & CBC-MAC. IEEE P802, 11 (2002)
19. Zhang, F., Dojen, R., Coffey, T.: Comparative performance and energy consumption analysis of different AES implementations on a wireless sensor network node. Int. J. Sensor Netw. **10**(4), 192–201 (2011). doi:10.1504/IJSNET.2011.042767