ContikiSec: A Secure Network Layer for Wireless Sensor Networks under the Contiki Operating System

Lander Casado and Philippas Tsigas

Department of Computer Science and Engineering, Chalmers University of Technology,
SE-412 96 Göteborg, Sweden
landerc@student.chalmers.se, tsigas@chalmers.se

Abstract. In this paper we introduce ContikiSec, a secure network layer for wireless sensor networks, designed for the Contiki Operating System. ContikiSec has a configurable design, providing three security modes starting from confidentiality and integrity, and expanding to confidentiality, authentication, and integrity. ContikiSec has been designed to balance low energy consumption and security while conforming to a small memory footprint. Our design was based on performance evaluation of existing security primitives and is part of the contribution of this paper. Our evaluation was performed in the Modular Sensor Board hardware platform for wireless sensor networks, running Contiki. Contiki is an open source, highly portable operating system for wireless sensor networks (WSN) that is widely used in WSNs.

Keywords: Wireless Sensor Networks Security, Link Layer Security

1 Introduction

Wireless sensor networks (WSNs) are unique networked systems, composed of wireless sensor nodes deployed en masse. WSNs distinguish themselves from other traditional wireless networks by relying on extremely constrained resources such as energy, bandwidth, and the capability to process and store data [1]. The number of applications that make use of WSNs is constantly increasing. The range of these applications goes from military and societal security to habitat and environment monitoring.

For many applications it is essential to provide secure communications. In general, WSNs face the same security risks as conventional wired or wireless networks; eavesdropping, packet injection, replay and denial of service attacks are some of the common attacks in WSNs. Due to the inherent properties of sensor nodes, traditional security protocols are not suitable for WSNs. A set of different attempts to implement secure communication specifically for WSNs appeared recently in the literature, such as TinySec [2], SenSec [1], MiniSec [3], and TinyECC [4]. All of these are designed to run under TinyOS [5], a widely used operating system for sensor nodes.

In 2004 Contiki was presented. Contiki is an open source, highly portable, multi-tasking operating system for memory-efficient networked embedded systems and wireless sensor networks [6]. Contiki has become quite popular and is attaining a good position in the WSNs community. Contiki does not offer any confidentiality or authenticity services for the communication between nodes. The current version provides only cyclic redundancy checks (CRC-16) to achieve integrity.

In this paper we present ContikiSec, which is the first attempt, to the best of our knowledge, to implement a secure network layer for wireless sensor network architecture for Contiki. ContikiSec has a configurable design, providing three security modes. ContikiSec offers confidentiality, authentication, and integrity in communications under the Contiki operating system. We base our design on existing security primitives that have been proven to provide security properties. We designed a complete solution after careful performance characterization and analysis of the existing security primitives that we selected. Our design tries to balance low energy consumption and security for the Contiki OS. Our design was guided through a set of performance evaluations in the MSB-430 platform, a Modular Sensor Board hardware platform created by ScatterWeb. During this evaluation, which is part of the contribution of the paper and is introduced in Section 4, we present the performance and resource consumption of six different block ciphers in our network. The six block ciphers that we considered are: AES, RC5, Skipjack, Triple-DES, Twofish, and XTEA (please refer to Section 4 for the detailed definitions of these block ciphers). The parameters that we focused on were the encryption speed, memory usage, and energy consumption. Our results show that AES is the most suitable block cipher for the WSNs under consideration. Consequently, we evaluate the performance of the CBC–CS, CMAC, and OCB modes of operation (please refer to Section 4 for the detailed definitions of these modes of operation). Based on the findings of our evaluation results we then present and evaluate the design of ContikiSec.

The rest of the paper is organized as follows. In the next section, we describe the security properties of a secure network layer for wireless sensor networks and we review the related work in WSNs. In the same section, we briefly describe the characteristics of the Modular Sensor Board hardware platform and the Contiki OS. In Section 3 we describe the performance and measurement methodology that we used to evaluate and analyze security primitives for our networks. The evaluation of the selected security primitives (block ciphers and modes of operation) are described in Section 4. In Section 5 we present the ContikiSec design, guided by the evaluation presented in Section 4. Finally, the paper is concluded in Section 6.

2 Background

2.1 Security Properties

The security properties that should be provided by a secure network layer for wireless sensor networks are described below.

Confidentiality. Confidentiality is a basic property of any secure communication system. Confidentiality guarantees that information is kept secret from unauthorized parties. The typical way to achieve confidentiality is by using symmetric key cryptography for encrypting the information with a shared secret key. Symmetric key algorithms could be divided into stream ciphers and block ciphers. In the case of block ciphers, a mode of operation is needed to achieve semantic security.

Semantic Security. Semantic security guarantees that a passive adversary could not extract partial information about the plaintext by observing the ciphertext [3]. Block ciphers do not hide data patterns since identical plaintext blocks are encrypted into identical ciphertext blocks. Thus, a special mode of operation and an initialization vector (IV) are often used and are needed to provide some randomization. IVs have the same length as the block and are typically added in clear to the ciphertext.

Integrity. Integrity guarantees that the packet has not been modified during the transmission. It is typically achieved by including a message integrity code (MIC) or a checksum in each packet. The MIC is computed by calling a cryptographic hash function that detects malicious altering or accidental transmission errors. Checksums are designed to detect only accidental transmission errors.

Authenticity. Data authenticity guarantees that legitimate parties should be able to detect messages from unauthorized parties and reject them. The common way to achieve authenticity is by including a message authentication code (MAC) in each packet. The MAC of a packet is computed using a shared secret key, which could be the same key used to encrypt the plaintext. In addition to authenticity, MACs also provide integrity.

2.2 Existing Security Architectures for WSNs

In recent years, the increased need of security in WSNs has prompted research efforts to develop and provide security modules for these platforms. These efforts go from simple stream ciphers to public key cryptography architectures.

SPINS [7] is the first security architecture designed for WSNs. It was optimized for resource-constrained environments and it is composed of two secure building blocks: SNEP and µTesla. SPINS offers data confidentiality, two-party data authentication, and data freshness. However, SNEP was unfortunately neither fully specified nor fully implemented [2].

In 2004, TinySec [2] was presented as the first fully implemented link layer security suite for WSNs. It is written in the nesC language and is incorporated in the official TinyOS release. TinySec provides confidentiality, message authentication, integrity, and semantic security. The default block cipher in TinySec is Skipjack, and the selected mode of operation is CBC–CS. Skipjack has an 80-bit key length, which is expected to make the cipher unsecure in the near future [8]. In order to generate a MAC, it uses Cipher Block Chaining Message Authentication Code (CBC–MAC), which has security deficiencies [9]. It provides semantic security with an 8-byte initialization vector, but adds only a 2-byte counter overhead per packet. TinySec adds less than 10% energy, latency, and bandwidth overhead.

SenSec [1] is another cryptographic layer for WSNs, presented in 2005. It is inspired by TinySec, and also provides confidentiality, access control, integrity, and semantic security. It uses a variant of Skipjack as the block cipher, called Skipjack-X. It has been conjectured that Skipjack-X is more secure than Skipjack, because it is not vulnerable to brute force attacks. In addition, SenSec provides a resilient keying mechanism.

MiniSec [3] is a secure sensor network communication architecture designed to run under TinyOS. It offers confidentiality, authentication, and replay protection. MiniSec has two operating modes, one tailored for single-source communications, and the other tailored for multi-source broadcast communication. The authors of MiniSec chose Skipjack as the block cipher, but they do not evaluate other block ciphers as part of their design. The mode of operation selected in MiniSec is the OCB shared key encryption mechanism, which simultaneously provides authenticity and confidentiality.

TinyECC [4] is a configurable library for elliptical curve cryptography operations in WSNs. It was released in 2008 and targets TinyOS. Compared with the other attempts to implement public key cryptography in WSNs, TinyECC provides a set of optimization switches that allow it to be configured with different resource consumption levels. In TinyECC, the energy consumption of the cryptographic operations is on the order of millijoules, whereas using symmetric key cryptography is on the order of microjoules [10].

2.3 Sensor Network Platform

Our selected platform is composed of MSB-430 sensor nodes that run the Contiki operating system. The main features of the hardware and the operating system are described below.

Modular Sensor Board (MSB-430). Modular Sensor Board (MSB-430) is a hardware platform for wireless sensor networks created by ScatterWeb [11]. The MSB-430 contains the MSP430F1612 [12] microcontroller, and offers 60 KB of memory divided into 5 KB RAM and 55 KB Flash-ROM. The MSP430F1612 achieves ultra-low power consumption with its five software selectable low power modes of operation.

The MSB-430 is equipped with the Chipcon CC1020 [13] transceiver and a low-noise amplifier. The radio frequency can be selected separately for receiving and transmitting by software. This is an important feature for advanced routing schemes, where multiple radio channels are used. The maximum transmission power is 8.6 dBm, and can be adjusted to reduce power consumption. While the maximum transmission rate is 153.6 kbps, the board is optimized for channel conformity, which allows a typical data rate of 19.2 kbps when using Manchester encoding.

The MSB-430 is supported by the open-source ScatterWeb operating system [11]. In this project the selected operating system is Contiki, which has a port to the MSB-430 platform.

Contiki. Contiki is an open source, highly portable, multi-tasking operating system for memory-efficient networked-embedded systems and wireless sensor networks [6]. Contiki is written in the C language and is designed for microcontrollers with a small amount of memory. A common Contiki configuration uses 2 kilobytes of RAM and 40 kilobytes of ROM. Contiki has been ported to different hardware platforms, such as MSP430, AVR, HC 12, and Z80.

Compared with other operating systems developed for resource-constrained platforms, one of the most interesting features of Contiki is dynamic loading, the ability to load and replace individual applications or services at run-time. Moreover, Contiki offers a hybrid model with an event-driven kernel where preemptive multi-threading is implemented as an application library [6]. Thus, multi-threading is only used when the application needs it.

3 Performance and Measurement Methodology

Due to the extreme resource limitations of WSNs, it is essential to measure the memory use, encryption speed, and energy consumption of the implemented security primitives. In this section we describe accurate software-based methods to measure parameters such as throughput, ROM, RAM, and energy consumption of WSN applications. These methods are used in our evaluation later on.

3.1 Software Encryption Speed

In order to measure the encryption speed, one way is to use a high precision oscilloscope to check the digital output pin. Another method is to use the Contiki real-time timers. Measuring parameters externally is more complex and often less precise, because the margin of error of the measuring instrument must be taken into account. Therefore, we decided to use the Contiki real-time timers. Certainly, we achieved a very high accuracy using this method.

The auxiliary clock (ACLK) is sourced from a 32768 Hz crystal oscillator. By selecting this clock, we can choose different divisors, such as 1, 2, 4, or 8. Using 1 as a divisor, we obtained the maximum resolution of 30.5176 microseconds. After some experiments, we realized that we can achieve greater resolution by measuring the time that is needed to complete the assignment statement. Using a simple loop, we observed that, on average, a variable could store the same tick value 5.75 times. Based on this observation, we defined the concept of $\mu tick$, which is the time needed to do the assignment statement. Please note that the assignment is always an assignment of a 2-byte value. By using this observation we obtain a resolution of

5.3074 microseconds, an acceptable accuracy compared with related work [14], which has a resolution of 5 microseconds.

3.2 ROM

Due to the limited memory resources of sensor nodes, the ROM-Flash used is an important parameter that has to be considered in the design of a software security layer in WSNs. Conventional cryptographic libraries are too big for embedded devices, the code size of the selected cryptographic suite has to be on the order of a few kilobytes. We are interested in measuring the memory used only by the cryptographic layer and not by the whole program that includes the Contiki operating system. To achieve this, we first compile the code with all the cryptographic functions and measure the size using the *msp430-size* utility [15]. By repeating the same process but without the cryptographic suite we can then calculate the memory required by the cryptographic module by a simple subtraction. This value is the best estimate of the ROM requirement of the security module.

3.3 RAM

RAM memory is even more limited than ROM-Flash. In the MSB-430 nodes, the volatile memory is only 5 KB. This parameter is hard to measure because of the variable size of the stack. The typical way to measure RAM memory is using a real-time debugger and setting breakpoints in the code. Unfortunately, there is no debugger available for the MSB-430 platform. Therefore, we have used the *msp430-ram-usage* tool [15] that gives us an approximate value of the used RAM.

3.4 Energy Consumption

Energy is a very scarce resource in sensor nodes. Hence, it is crucial for the security architecture to retain a low energy overhead. The unique characteristics of sensor network applications make hardware-based energy measurement difficult [16]. In addition, the cost of a hardware-based mechanism for energy measurement is too high; the cost per hardware-unit is similar to the price of the sensor node [16].

In this project, we used *Energest* to evaluate the energy consumption of the cryptographic primitives. *Energest* is a software-based on-line energy estimation mechanism that estimates the energy consumption of a sensor node [17]. The mechanism runs directly on the sensor nodes and provides real-time estimates of the current energy consumption. *Energest* provides good estimates, but further study is needed to quantify the accuracy of the mechanism [17]. This tool maintains a table with entries for all components, such as CPU, radio transceiver, and LEDs. When a component starts running, a counter starts to measure the estimated energy consumption of this component. When the component is turned off, the timer is stopped.

4. Performance Characterization of Security Primitives under Contiki

4.1 Block Cipher Evaluation

The block ciphers that we have examined in this paper are briefly described below.

AES. Advanced Encryption Standard is a symmetric block cipher that can encrypt/decrypt data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. AES is based on the Rijndael algorithm, developed by Daemen and Rijmen in 1998 [18].

RC5. RC5 is a patented symmetric block cipher designed by Ronald L. Rivest of the MIT Computer Science and Artificial Intelligence Laboratory [19]. It is a parameterized block cipher with a variable block size, a variable key size, and a variable number of rounds.

Skipjack. Skipjack is an algorithm for encryption developed by the U.S. National Security Agency (NSA) and was declassified in 1998 [20]. It operates on data blocks of 64 bits with an 80-bit key.

Triple-DES. Triple-DES, also known as Triple Data Encryption Algorithm, is a variant of the Data Encryption Standard (DES) algorithm that executes the core DES algorithm three times [21]. It uses a 64-bit block size and a 168-bit key.

Twofish. Twofish is a symmetric block cipher with a block size of 128 bits and key sizes of 128, 192, and 256 bits [22]. It was designed by Schneier *et al.* for the AES contest.

XTEA. Extended Tiny Encryption Algorithm is a symmetric block cipher designed by Wheeler and Needham of the Cambridge Computer Laboratory [23]. XTEA is considered one of the fastest and most efficient algorithms. It operates on data blocks of 64 bits with a 128-bit key.

We focused on C language implementations of these ciphers since our objective is to integrate them into the Contiki operating system, which is designed in C. Due to the computational and memory limitations of sensor nodes, we used the most optimized, publicly available versions of each cipher.

In Figure 1, we show the time needed for encrypting and decrypting a single block of plaintext. Additionally, we show the time needed to carry out the key expansion process. For the context of these measurements it is important to emphasize the block size with which each cipher operates: XTEA, Skipjack, and Triple-DES operate with 64-bit blocks; in contrast, AES and Twofish operate with 128-bit blocks.

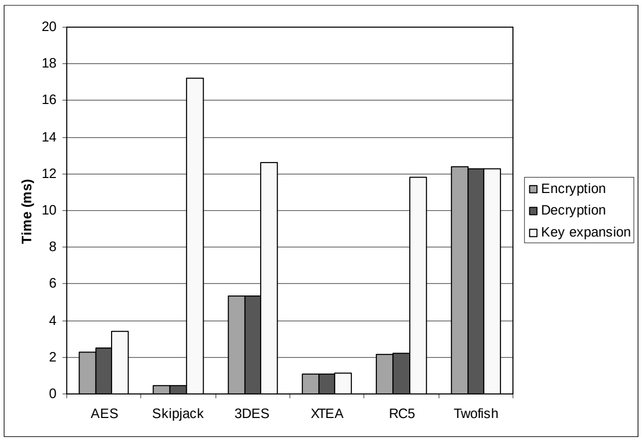

Fig. 1. Encryption, decryption, and key expansion time.

From the figure above we can observe that Skipjack is the fastest cipher for encrypting/decrypting a single block of plaintext. However, it is the slowest cipher for the key expansion process. In our design, which we describe in the next section, we can have the key expansion process executed only when the node is initialized. Therefore, the time needed to do the key-setup process is not very significant. In order to simplify the analysis while taking into account the block size, we show the encryption throughput and the key length of each cipher in Table 1.

Table 1. Key length and throughput.

Cipher	Key length	Throughput (Kbps)
AES	128	56.39
Skipjack	80	145.45
3DES	168	12.01
XTEA	128	59.26
RC5	128	29.49
Twofish	128	10.31

As mentioned, Skipjack has by far the best throughput. Nevertheless, if greater security is needed, we should choose a 128-bit key length cipher. In that case, AES and XTEA are the most suitable ciphers; both have similar encryption throughput. Their respective throughput is nearly three times smaller than the throughput of Skipjack.

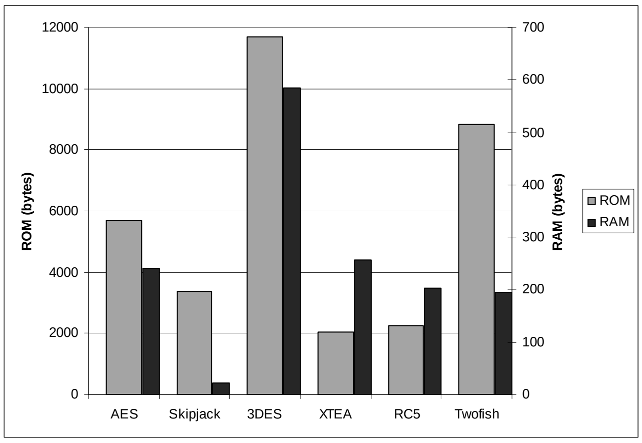

Fig. 2. ROM and RAM usage.

In Figure 2 we show the memory requirements for the ciphers under consideration. We observe that Skipjack uses only 21 bytes of RAM, which is not a typical value. As we mentioned, the obtained RAM usage is an approximation. When analyzing the ROM usage, we see that XTEA uses the least ROM, followed by RC5 and Skipjack.

Energy is an extremely scarce resource in sensor nodes. Energy consumption of the cryptographic primitives must be taken into account when designing a security layer for WSNs. As expected, faster ciphers consume less energy. In Figure 3 we illustrate the estimated energy consumption of each block cipher.

These experimental observations show that Skipjack is the most efficient block cipher for our platform; in fact it is the best cipher with respect to throughput, RAM usage, and energy consumption. However, with the rapid advancement in computing, the 80-bit key in Skipjack is not completely secure. According to the claims of RSA Security Labs, 80-bit keys would become *crackable* by 2010 [8]. Therefore, we believe that Skipjack should not be the default block cipher in security architectures.

The experimental results also show that XTEA is also a very efficient block cipher for WSNs. It achieves an acceptable throughput and it needs less ROM than any other cipher. Although it has a 128-bit key, a related-key differential attack can break 27 out of 64 rounds of XTEA [24].

After Skipjack and XTEA, AES achieves the best results in our experiments. It obtains a data throughput greater than radio rate, and reasonable memory usage and energy consumption. Additionally, at present time AES is the block cipher approved as a standard and recommended by NIST [25]. Moreover, Didla *et al.* demonstrated that AES could be highly optimized for WSNs [14]. Unfortunately, their optimized code is not in the public domain and is currently under a patent filing. Vitaletti and Palombizio also showed that AES can be effectively used in WSNs, and they have developed a module with AES for TinyOS [26].

Based on the results described above and taking into account the trade-off between security and resource consumption, we selected AES as the most appropriate block cipher for the WSNs under consideration.

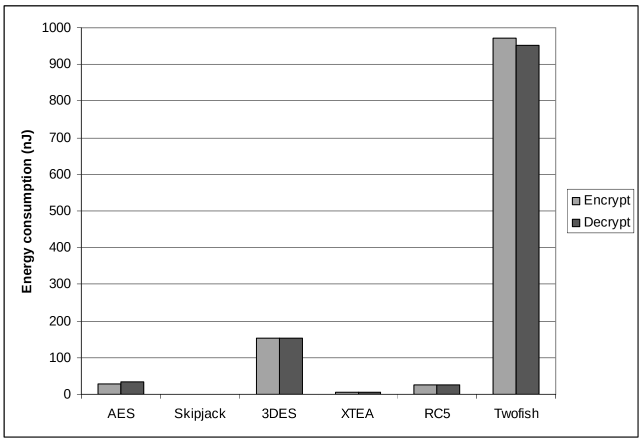

Fig. 3. Energy consumption for encrypting and decrypting one data block.

4.2 CBC-CS Evaluation

Cipher Block Chaining–Ciphertext Stealing (CBC–CS) is a mode of operation proposed by NIST [27]. A mode of operation is an algorithm that features the use of a symmetric block cipher. A mode of operation requires an initialization vector (IV), which is a random block of data, to achieve the semantic security property. In Figure 4, we evaluate the mode CBC–CS with IVs of different sizes, using AES as the underlying block cipher.

From our observations, the overhead produced by padding the plaintext to the cipher block size is very important. Consequently, we should try to avoid encrypting very short packets. In contrast, we observed that the power consumption for encrypting messages larger than the block size is nearly zero. In addition, sending an IV in each packet produces a significant increase in the energy consumption. The length of the IV should be defined depending on the security level required. In our opinion, a 2-byte IV makes a good balance between security and power consumption. In typical WSN scenarios, few packets per minute are sent, and thus the time to exhaust all the possible 2^{16} IVs is quite long [26].

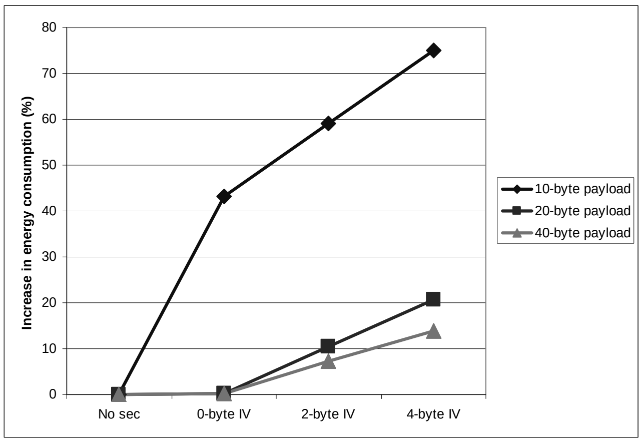

Fig. 4. The increase in energy consumption using CBC–CS–AES with different IVs.

4.3 CMAC Evaluation

Cipher-based Message Authentication Code (CMAC) is a cryptographic algorithm that provides assurance of the authenticity and, hence, the integrity of binary data [9]. In order to evaluate the impact of including a MAC in each message we have to define the length of the tag. Our experiments show that sending one byte has nearly the same power consumption as calling the AES encryption function six times. Thus, we have to limit the length of the MAC as much as possible. In TinySec [2] a 4-byte MAC is selected to provide authentication and integrity. The authors claim that a 4-byte MAC is appropriate for WSNs. In their paper they state: "*Adversaries can try to flood the channel with forgeries, but on a 19.2kb/s channel, one can only send 40 forgery attempts per second, so sending 2^{31} packets at this rate would take over 20 months.*" Furthermore, MiniSec and SenSec also use a 4-byte MAC. We also implement a 4-byte MAC in each message and we evaluate the influence on power consumption.

To determine the energy overhead of computing and adding a MAC in each message we performed several experiments with different data length. In Table 2 we show the obtained results.

Table 2. Energy consumption of sending packets with a MAC.

Payload (bytes)	Mode	Energy (uJ)	Increase
10	Default	47.13	-
10	CMAC	53.43	13.37%
20	Default	82.48	-
20	CMAC	91.32	10.72%
40	Default	182.19	-
40	CMAC	195.75	7.44%

We observe that the energy cost of computing the MAC is negligible compared with the energy needed to send the MAC. The results demonstrate that the fixed overhead of sending each message (turning on the radio, sending the preamble and sync word) generally discourages short messages [2].

4.4 OCB Evaluation

Offset Codebook Mode (OCB) is a mode of operation that simultaneously provides confidentiality and authenticity [28]. When the application under consideration demands confidentiality and authenticity, OCB is claimed to be the most appropriate mode for WSNs in [3] and [8]. To evaluate the efficiency of OCB in our platform, we implemented this authenticated-encryption algorithm and compared it with the CBC–CS and CMAC modes.

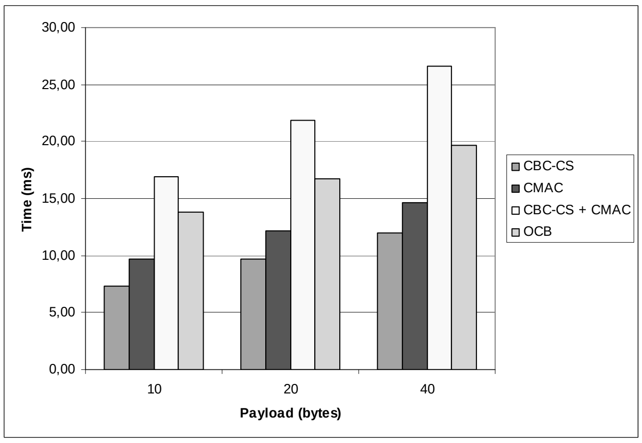

Fig. 5. Comparison of CBC–CS, CMAC, and OCB.

As can be seen in Figure 5, OCB is faster than CBC–CS encryption combined with the CMAC. The results suggest that OCB is even more efficient with larger data lengths. Therefore, our analysis indicates that OCB is the best choice to select from when confidentiality and authenticity are required in the WSNs under consideration.

5. ContikiSec Design

Guided by the study, as presented in the previous section, we are now ready to put all the pieces together and describe the complete design of our secure network layer for wireless sensor networks under the Contiki operating system.

The Contiki protocol stack is divided in four layers: the physical layer, the data link layer, the communication layer, and the application layer. In WSNs, the typical traffic pattern is the many-to-one communication pattern. Because of that, in order to avoid routing packets injected by an adversary to waste energy and bandwidth, security should be implemented at the link layer [2]. Contiki provides three media access control protocols for our platform: X-MAC, LLP, and NULLMAC. Currently, we have implemented ContikiSec under NULLMAC, but it could be ported to run under the other two protocols. In Figure 6 we show the Contiki protocol stack including ContikiSec.

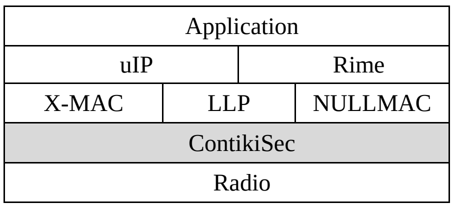

Fig. 6. The Contiki stack with ContikiSec.

ContikiSec supports three security modes: confidentiality-only (ContikiSec-Enc), authentication-only (ContikiSec-Auth), and authentication with encryption (ContikiSec-AE). We believe that a configurable design is especially desirable for WSNs, as WSNs are expected to support many different application scenarios with different security requirements. ContikiSec offers the programmer the choice to select between three security levels depending on the needs of the application on hand. In Figure 7, we show the default Contiki packet format for the CC1020 radio and the packet format with different security configurations.

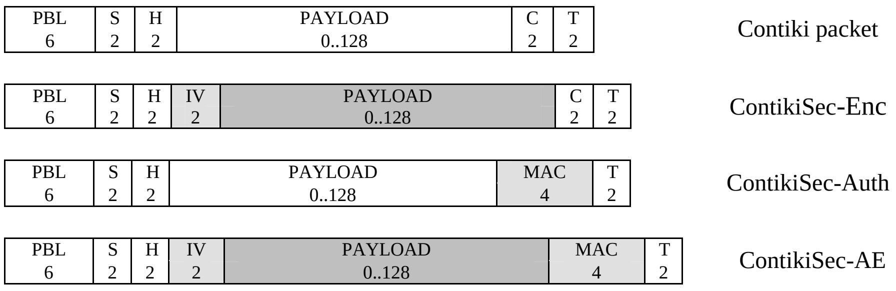

Fig. 7. ContikiSec packet formats compared to the Contiki packet format.

As we can observe, ContikiSec-Enc includes a 2-byte initialization vector (IV) in each packet, producing a 2-byte overhead. The length of the IV has a great impact for security and energy consumption. Longer IVs are more secure, but the energy cost of sending them is too high for WSNs. In Section 4 we argue that a 2-byte IV makes the best balance between security and energy consumption. The IV is created using the Contiki library that generates random numbers. For the seed value to initialize the random generator, we used the node identifier. Hence each node initializes the random generator differently. The payload is encrypted using the CBC–CS mode of operation with AES as the underlying block cipher. Moreover, we assume that all the nodes in the network are provided with a single 128-bit key. The key-expansion process is performed when the node is initialized and then stored in the RAM. In addition, the CC1020 radio driver adds a 2-byte checksum field to each packet. Consequently, ContikiSec-Enc supports confidentiality and integrity.

ContikiSec-Auth is designed for applications where confidentiality is not critical, but where it is crucial to know the originator of each message. Contiki provides a 16-bit cyclic redundancy check (CRC) function to calculate a 2-byte checksum and achieve integrity. However, the checksum is designed to detect only accidental modifications of the data, whereas a MAC also detects intentional unauthorized modifications of the data. Hence, ContikiSec-Auth provides authentication and integrity by removing the checksum field from the packet and instead including a MAC. To generate a MAC, ContikiSec-Auth uses the CMAC algorithm benchmarked in Section 4, which is currently the mode for authentication recommended by NIST [9].

Finally, ContikiSec-AE provides the highest level of security, achieving confidentiality, authentication, and integrity. As we demonstrate in Section 4, the OCB mode is the most efficient algorithm when those properties are required. Consequently, ContikiSec-AE uses the OCB mode with AES as the underlying block cipher, using a single shared-key for encryption and authentication. ContikiSec-AE increases the packet length by 4 bytes. In Figure 8 we show the energy consumption of the three modes of ContikiSec.

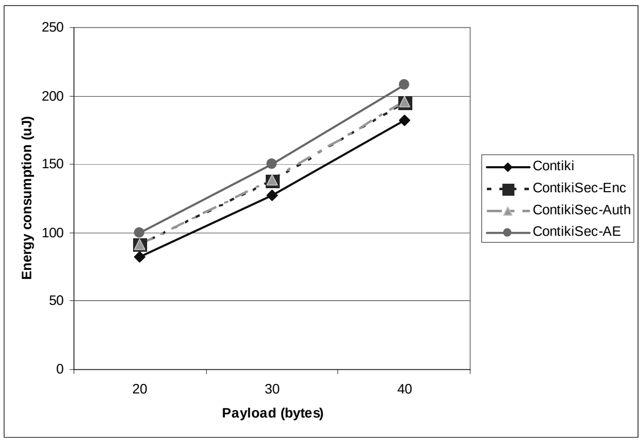

Fig. 8. Energy consumption of ContikiSec modes compared with the default Contiki configuration.

As mentioned, and shown in Figure 4, the energy consumption for cryptographic operations is negligible compared to the energy needed for sending each extra byte. Hence, as the size of the payload is increased, the impact of sending extra bytes decreases.

Without security, the energy consumption ranges from 82 µJ to 182 µJ for packets with payloads of 20 to 40 bytes. Despite the fact that ContikiSec-Enc and Contiki-Auth have different computational costs, we observe the same energy consumption because both incur a 2-byte overhead. Finally, ContikiSec-AE is the mode with the highest energy requirement, consuming around 15% more than Contiki in default mode for data messages with payloads of 40 bytes.

6. Conclusion

We present ContikiSec, which is a secure network layer for wireless sensor networks under the Contiki operating system. We have designed ContikiSec as a complete and configurable solution, providing three security modes, starting from confidentiality and integrity, and increasing to confidentiality, authentication, and integrity. Our design was governed by a careful selection and performance analysis of existing security primitives. Our design tries to achieve low energy consumption without compromising security. Our evaluation was carried on the MSB-430 platform, a Modular Sensor Board hardware platform created by ScatterWeb.

In the future we plan to study the boundaries of using asymmetric keys in our framework and also examine the effect of compromised nodes in secure network layer architectures for wireless sensor networks.

References

1. Li T., Wu H., Wang X., Bao F.: SenSec Design. Technical Report-TR v1.1. InfoComm Security Department, Institute for Infocomm Research (2005)
2. Karlof C., Sastry N., Wagner D.: TinySec: a Link Layer Security Architecture for Wireless Sensor Networks. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), pp. 162--175. Baltimore (2004)
3. Luk, M., Mezzour, G., Perrig, A., Gligor, V.: MiniSec: a Secure Sensor Network Communication Architecture. In: Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN 2007), pp. 479-488. ACM, New York (2007)
4. Liu A., Ning P.: TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In: Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008), SPOTS Track, pp. 245--256, (2008)
5. TinyOS, http://www.tinyos.net
6. Dunkels A., Grönvall B., Voigt T.: Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, pp. 455--462. IEEE Computer Society, Washington (2004)
7. Perrig A., Szewczyk R., Wen V., Culler D., Tygar J. D.: SPINS: Security Protocols for Sensor Networks. In: Proceedings of the 7th annual international conference on Mobile computing and networking, pp. 189--199. ACM, New York (2001)
8. Jinwala D., Patel D., Dasgupta, K. S.: Optimizing the Block Cipher and Modes of Operations Overhead at the Link Layer Security Framework in the Wireless Sensor Networks. In: Proceedings of the 4th International Conference on Information Systems Security. LNCS, vol. 5352, pp. 252--272. Springer (2008)
9. National Institute of Standards and Technology, Computer Security Division. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, Special Publication 800-38B (2005)

10. Chang, C., Nagel, D.J., Muftic, S.: Measurement of Energy Costs of Security in Wireless Sensor Nodes. In: Proceedings of 16th IEEE International Conference on Computer Communications and Networks (ICCCN 2007), pp. 95--102. (2007)
11. ScatterWeb, MSB-430datasheet, http://www.scatterweb.com/content/products/MSB_en.html
12. Texas Instruments, MSP430F1612 datasheet, http://focus.ti.com/mcu/docs/mcuprodoverview.tsp?sectionId=95&tabId=140&familyId=342
13. Chipcon, CC1020 datasheet, http://focus.ti.com/docs/prod/folders/print/cc1020.html
14. Optimizing AES for Embedded Devices and Wireless Sensor Networks. In: Presented at 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom) , Innsbruck, Austria (2008)
15. MSPGCC, http://mspgcc.sourceforge.net/
16. Jiang X., Dutta P., Culler D., Stoica, I.: Micro Power Meter for Energy Monitoring of Wireless Sensor Networks at Scale. In: Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN'07), pp. 185--195. ACM, New York (2007)
17. Dunkels A., Österlind F., Tsiftes N., He Z.: Software-based Sensor Node Energy Estimation. In: Proceedings of the 5th international conference on Embedded networked sensor systems (SenSys 2007), pp.409--410. ACM, New York (2007)
18. Daemen J., Rijmen V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer-Verlag (2002)
19. Rivest R.: The RC5 Encryption Algorithm. In: Proceedings of the 1994 Leuven Workshop on Fast Software Encryption, pp. 86-96. Springer (1995)
20. National Institute of Standards and Technology, Computer Security Division. SKIPJACK and KEA Algorithm Specifications. (1998)
21. National Institute of Standards and Technology, Computer Security Division. Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, Special Publication 800-67, Version 1.1 (2004)
22. Schneier B., Kelsey J., Whiting D., Wagner D., Hall C., Ferguson N.: The Twofish Encryption Algorithm. John Wiley & Sons (1998)
23. Needham R., Wheeler D.: Tea extensions. Technical report, Computer Laboratory, University of Cambridge (1997).
24. Ko Y., Hong S., Lee W., Lee S., Lim J.: Related Key Differential Attacks on 27 Rounds of XTEA and Full-Round of GOST. In: Proceedings of FSE 2004. LNCS, vol. 3017, pp. 299--316. Springer (2004)
25. National Institute of Standards and Technology, Computer Security Division, AES standard, http://csrc.nist.gov/archive/aes/index.html
26. Vitaletti A., Palombizio G.: Rijndael for Sensor Networks: Is Speed the Main Issue?. In: Proceedings of the 2nd Workshop on Cryptography for Ad-hoc Networks (WCAN 2006). ENTCS, vol. 171, pp. 71--81. (2007)
27. National Institute of Standards and Technology, Computer Security Division. Proposal To Extend CBC Mode By “Ciphertext Stealing”. (2007)
28. Rogaway, P., Bellare, M., Black, J.: OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. In: ACM Transactions on Information and System Security (TISSEC), pp. 365-403. ACM, New York (2003)