Comparing Block Cipher Modes of Operation on MICAz Sensor Nodes

Gernot R. Bauer, Philipp Potisk, and Stefan Tillich
Institute for Applied Information Processing and Communications
Graz University of Technology
Inffeldgasse 16a, A–8010 Graz, Austria
{*gernot.bauer,philipp.potisk*}*@student.TUGraz.at, Stefan.Tillich@iaik.tugraz.at*

Abstract

Wireless sensor networks are a key technology for "ubiquitous computing" applications. The challenges of securing such networks are tremendous. On the one side, sensor nodes are commonly deployed in potentially hostile environments, which requires additional protection in comparison to traditional computing systems. On the other side, the capabilities of sensor nodes in terms of computing power, memory, and available energy are severely limited, which makes it hard to adapt existing security solutions. In this paper, we examine different options for providing confidentiality and message authentication to sensor network communication. More specifically, we examine four modern block cipher modes of operation regarding their applicability in sensor networks. These are the Offset Codebook mode (OCB), the Counter Cipher Feedback with Header mode (CCFB+H), the EAX mode, and the Galois/Counter mode (GCM). Our practical evaluation targets the MICAz sensor node and accounts for the typically small packet size of sensor network traffic. Our results indicate that the CCFB+H mode is the best choice for a large range of applications.

1. Introduction

Wireless sensor networks (WSNs) constitute an interesting field of application where the benefits of digital computing are closely integrated with our every-day environment. Such networks typically consist of a number of low-cost sensor nodes, often called *motes*, which are connected to their environment through sensors and/or actuators, can perform computations and communicate wirelessly. A common feature of sensor nodes is an autonomous and limited power supply, usually in the form of batteries.

For most WSN applications, security is an important or even crucial requirement. The range of potential attacks faced by such networks is typically much broader than in traditional desktop or server computing systems. The broadcast nature of wireless communication makes it inherently prone to eavesdropping and message spoofing. Moreover, the deployed sensor nodes might be physically accessible to an attacker and therefore potentially subject to physical tampering.

The low-cost nature and resource constraints of sensor nodes make the implementation of adequate security measures a very challenging task. The overhead incurred through added protection must not "eat away" too much of the precious resources of computing power, program and working memory and energy. A base of strong cryptographic algorithms (primitives) is the cornerstone of any sound security solution. Efficient implementation of existing algorithms of constrained devices as well as the design of new cryptographic primitives suited for on such devices are a matter of ongoing research.

While cryptographic primitives are indispensable components, they alone cannot provide security. For example, a block cipher only determines the encryption and decryption of fixed-size data blocks, but it does not give any further provisions, e.g., on how to protect the authenticity of variable-sized messages. In order to achieve such security provisions, the correct utilization of cryptographic primitives is defined in so-called *modes of operation*. Various primitives in various modes can then be employed in cryptographic protocols to build up system-wide security measures.

Block ciphers are a versatile primitive with a large range of modes of operation defined for them. These modes predominantly deal with encryption and message authentication (MAC generation). Traditionally, a single mode only deals with a single security provision, e.g., CBC mode for encryption, and CBC-MAC for message authentication[1]. When several security provisions are required, these modes have to be combined in a secure fashion (through proper *generic composition* [1]). If the same cryptographic primitive is to be used in several different modes, it is generally problematic to exploit synergies. For example, using the same key for CBC mode encryption and CBC-MAC can break message authentication.

Nevertheless, in the face of scarce resources it is desirable to exploit all possible synergies. If a single key suffices for encryption and MAC generation, memory can be saved and computational overhead due to key expansion can be limited. This is one of the reasons that some modern modes of operation allow to encrypt and authenticate messages with a single key (*Authenticated Encryption (AE)*). Some flexible

1. Note that CBC-MAC is only secure for fixed-size messages.

modes even offer the possibility of adding data which is only authenticated, but not encrypted (associated data). Such functionality is commonly denoted as *Authenticated Encryption with Associated Data (AEAD)* [2]. The AEAD constructs are especially suited for the protection of network packets: The data payload of the packet (message) can be encrypted and authenticated while it is normally sufficient to authenticate the packet header[2].

In this paper, we analyze AEAD modes of operations regarding their suitability for implementation on *MICAz motes*. These modes are the *OCB* mode [3], the *EAX* mode [4], the *CCFB+H* mode [5], and the *GCM* mode [6].

The rest of this paper is organized as follows. Section 2 gives a brief description of our target platform. Section 3 covers the used modes, describing their most important parts as well as limitations and legal issues. We compare the implemented modes for their performance and suitability in Section 5. Section 6 contains some implementation optimizations which are applicable for our target platform. We discuss our results and their implications in Section 7.

2. Target Platform (MICAz)

For our practical evaluations we have targeted the MICAz sensor nodes from Crossbow [7]. MICAz motes are a popular sensor node platform equipped with an Atmel ATmega128L microcontroller, a TI/Chipcon CC2420 radio module, and a battery pack containing two standard AA-size batteries. Various plugboards with sensors and/or actuators can be attached to the MICAz mote. Figure 1 depicts a typical MICAz mote.

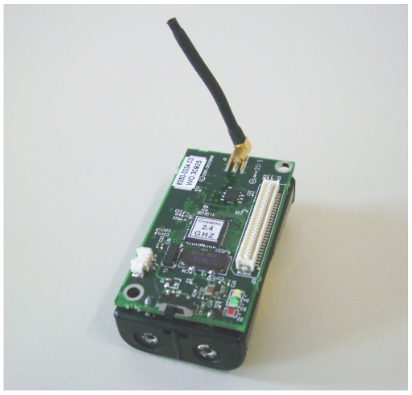

Figure 1. MICAz mote

The ATmega128L microcontroller has a simple yet powerful 8-bit RISC architecture (AVR) and features 128 KB program memory, 4 KB EEPROM and 4 KB working memory (SRAM). It is clocked at a rate of 16 MHz. The wireless interface works in the 2.4 GHz band and is able to send at rates of up to 250 Kbps.

2. In fact, encryption of header fields is not possible in many protocols.

An important development environment for MICAz motes is the TinyOS open-source operating system [8]. It is based on nesC language, which allows to develop sensor network applications by composition of various software components. These components are normally abstractions of existing hardware modules (e.g., radio interface, sensors, UART, LEDs).

3. Description of the Surveyed Modes of Operation

Table 1 compares the most important properties of the investigated AEAD modes.

An important distinction is between *one-pass* and *two-pass* modes. The first can provide confidentiality and message authentication with a single pass over the message. The latter takes two passes. One-pass modes are potentially faster, but the approach is covered by several patents. As a consequence, the research community has turned towards fast two-pass modes. We have also concentrated on unpatented two-pass modes and have included OCB mode as a typical representant of a one-pass mode for comparison.

Encryption overhead refers to a potential expansion of the ciphertext in comparison to the plaintext (message expansion). All surveyed modes have a minimal encryption overhead, i.e., the length of the ciphertext equals the length of the plaintext. This is a very desirable property for sensor nodes, where each additional bit sent over the wireless interface is very costly in terms of energy.

The encryption functionality of the examined AEAD modes consists of adaptations and expansions of traditional encryption modes of operations like Electronic Codebook mode (ECB), Cipher Block Chaining mode (CBC), Cipher Feedback mode (CFB), and Counter mode (CTR) [10]. Message authentication is achieved by the use of secure CBC-MAC variants (PMAC and OMAC) with the exception of the GCM mode, which uses an universal hash function construct (GHASH). The rest of the properties of the modes are largely similar.

For encryption, each AEAD mode takes as input an *Initialization Vector (IV)*, along with a *message* and a *header*. The message is then encrypted, resulting in the ciphertext. The IV must be different for every encryption in order to prevent identical messages from yielding identical ciphertexts.

A so-called *Message Authentication Code (MAC)* tag of a specific size is calculated from the message and the header. The MAC tag is then attached to the header and the ciphertext in order to allow verification of the authenticity. It is usually allowed to pick a part of the MAC tag to limit the amount of data to be transferred. For decryption, the original message is recovered from the ciphertext and the MAC tag is regenerated and compared to the received one. The exact order of these operations depends on the mode.

	OCB	**EAX**	**CCFB+H**	**GCM**
Type	One-pass	Two-pass	Two-pass	Two-pass
Encryption mode	Tweaked ECB (XEX construction [9])	CTR	CFB with CTR	CTR
Needs block cipher decryption	Yes	No	No	No
Authentication	PMAC	OMAC	OMAC	GHASH
Tag verification	After decryption	Before decryption	Before decryption	Before decryption
Supported tag length	0 to block size	0 to block size	0 to (block size - IV length)	0 to block size
Supported IV length	Block size	Any	0 to (block size - tag length)	Any (shortcut for 12-byte IV)
IV requirements	Non-repeating	Non-repeating	Non-repeating	Non-repeating
Known patents	In USA	None	None	None

Table 1. Main properties of the implemented AEAD modes.

In the following sections, we describe the principal functionality of each of the four surveyed modes of operation. We try to give a principal understanding of the most important mechanisms, and omit some of the finer details for the sake of brevity. For the complete description of each mode we refer the reader to the respective original publications which are cited in each section.

In order to facilitate the understanding of the following sections, we give a very brief description of the basic encryption modes of operation (ECB, CBC, CFB, and CTR) by summarizing their encryption functionality. In ECB mode, the plaintext blocks are encrypted independently with the block cipher to yield the ciphertext blocks. In CBC mode, the previous ciphertext block is XORed to the current plaintext block prior to encryption with the block cipher. The first plaintext block is XORed with the IV. The outputs of the block cipher are the ciphertext blocks. In CFB mode, the previous ciphertext block is encrypted with the block cipher and the output is XORed to the current plaintext block to yield the ciphertext block. The first ciphertext block is produced by XORing the first plaintext block with the encrypted IV. In CTR mode, a non-repeating sequence of blocks is encrypted with the block cipher and the output is XORed to the plaintext blocks in order to create the ciphertext blocks. Usually, the IV is used to create the first input block to the block cipher and its value is incremented to produce the subsequent input blocks.

CBC-MAC uses CBC mode encryption with an IV of all zeros and uses the last ciphertext block as MAC tag. However, CBC-MAC is not secure for variable-sized messages. There are slight tweaks (e.g., PMAC and OMAC) which make the MAC tag dependent on the message size and thus secure for messages of arbitrary size.

3.1. Offset Codebook Mode (OCB)

The Offset Codebook mode is a one-pass AEAD scheme conceived by Rogaway et al. [11]. The original version from [11] (now denoted OCB 1.0) could not deal with associated data (i.e., data that is only authenticated). This feature was added in an updated version, called OCB 2.0, which is described in [3]. In this paper we will use the term OCB to stand for OCB 2.0.

OCB encrypts arbitrary length plaintexts without overhead. The IV must be non-repeating, but can be predictable. It is based on computations of offsets in GF(2^{128}). These offsets are XORed with each message block before and after encryption. For the last block, the block cipher is used to generate a key stream which is then XORed with the plaintext. Thus, the ciphertext length equals the length of the plaintext. The offset computation is dependent on the key and IV. Prior to the availability of the message, no extensive precomputations can be done.

The header blocks are XORed with offsets, encrypted and then XORed together. The message blocks are XORed together where the length of the last block is specially taken into account. The final MAC tag is the XOR of the tags resulting from both the processing of the header and the message.

Phillip Rogaway offers an implementation of OCB on his web page[3]. Due to this implementation and the description in the paper, we found OCB to be fairly easy to implement. According to the draft on OCB 2.0 in [3], there are pending US patents that only allow the use of OCB in projects conforming to the GNU General Public License[4].

3.2. EAX Mode

The EAX mode of operation is a two-pass scheme defined by Bellare et al. [4]. The encryption is performed in counter mode (CTR), which allows the encryption of arbitrary length data without overhead. In addition, only the encryption function of the underlying block cipher has to be implemented.

3. http://www.cs.ucdavis.edu/~rogaway/ocb/

4. Note that Phillip Rogaway explicitly states that other royalty-free uses might be granted on request.

EAX uses OMAC for authentication. This MAC algorithm is based on CBC-MAC, padding the last block with one of two constants (depending on its length). These constants are key-dependent and can be precomputed during an initial setup phase. The authors also provide a little tweak to MAC computation by prepending the data with a constant block before performing the OMAC operation. This variant of OMAC is called OMAC^t in the specification.

The IV can be of arbitrary size. To ensure security, it must be non-repeating, but it can be predictable. The resulting tag length is customizable with the maximum size being the block cipher's block length. If no message is specified, the EAX mode can be used just for authentication of the header.

EAX uses the computed OMAC value of the IV as initial counter value for the encryption. This OMAC value as well as the OMAC values of the ciphertext and the plaintext header are XORed to compute the authentication tag. Thus, the tag can be verified by the receiver before decrypting the ciphertext.

The EAX mode has a clear specification and the authors provide test vectors for use with AES-128 as well as a recommended software interface. Moreover, there are quite a few implementations of EAX available. Another advantage is that EAX is in the public domain and, according to the authors, there are no patents covering its use.

3.3. Counter Cipher Feedback with Header Mode (CCFB+H)

CCFB+H mode is a minimal-expanding AEAD scheme designed by Stefan Lucks [5][5]. Even though the author states that it is a two-pass scheme, it has a strong resemblance to a one-pass mode.

CCFB+H uses the block cipher for both encryption and authentication. The input of the block cipher is assembled from previous ciphertext blocks and an incrementing counter value. Likewise, the output of the block cipher is partly used to generate a keystream which is XORed to the message for encryption and as a contribution to the MAC tag. The exact sizes of these two parts are determined by the size of the IV and the MAC tag respectively. Therefore, the size of these parameters is important in terms of performance. A longer tag increases the number of block cipher calls, while a tag of size 0 effectively reduces CCFB+H to traditional Cipher Feedback mode (CFB). When a header is present, CCFB+H calculates the OMAC of the header and uses it, together with the IV, as input for the first block cipher call.

At the current time, the CCFB+H mode does not seem to be in significant use, indicated by an absence of implementations. A further obstacle for implementers is the non-availability of any test vectors for this mode of operation. Nevertheless, CCFB+H is relatively easy to implement and is unencumbered by patents.

5. Note that there is also a CCFB mode specified in case no header is present.

3.4. Galois/Counter Mode (GCM)

GCM is a two-pass scheme by McGrew et al. [6]. This mode is—with minor modifications but essentially equivalent—also a recommendation from NIST [12]. In this paper, we mainly refer to the original specification from [6].

GCM is based on counter mode for encryption and a polynomial universal hash function called *GHASH* for authentication. It is the only one of our surveyed modes that does not rely on the block cipher for the tag calculation. Counter mode is employed for encryption, which allows for parallelization and prevents ciphertext expansion. A zero length message can be used if there is only the need for authentication of the header.

The GHASH function consists of additions and multiplications in GF(2^{128}). Each block is multiplied by a special, key-dependent *hash subkey*. Except for the creation of the hash subkey, no block cipher encryption is required for GHASH. GCM is similar to the Carter-Wegman Counter mode (CWC), with the only difference that CWC defines the universal hash function over a prime field GF(p) [13].

The IV is used to create the initial counter for the encryption. There are no limitations on the length of the IV. For a length of 12 bytes, a substantial shortcut is provided by padding the IV instead of calculating the GHASH of it. The IV must be a non-repeating value for every key.

The MAC tag can have arbitrary length of up to the underlying block cipher's block length. It is calculated by computing the GHASH of the header and the encrypted message, complemented with a final XOR with the initial counter. As the encrypted message is authenticated, the tag can be verified before decryption.

The authors of GCM provide test vectors including intermediate values as well as optimization suggestions. The NIST specification [12] presents GCM in a more concise way, thus facilitating implementation. The bit-ordering used in the GF(2^{128}) computations is little endian, which is a bit unusual. As further complication, big-endian bit-ordering is used when incrementing the counter. Furthermore, the original specification seems to contain some errors in the optimization section. Thus, for an implementer GCM contains several pitfalls that have to be avoided.

4. Discussion of Security and Efficiency

Regarding the protection of sensor network packets, Karlof et al. provide an excellent discussion of the relevant issues like MAC tag length and IV structure in the context of the TinySec link-layer security architecture [14]. Applying one of the analyzed modes of operation, most of TinySec's parameters could be used to implement security for sensor network communication. In the following, we will briefly discuss the significant differences between a scheme using one of the surveyed modes and TinySec.

TinySec is a generic composition of CBC mode for encryption and a CBC-MAC variant for authentication. It requires two separate block cipher keys in order to ensure both confidentiality and authenticity of network packets. On the other hand, our surveyed AEAD modes of operation are designed to work with a single cipher key. Having to handle only a single key can reduce memory requirements; especially when key schedules are precomputed and stored for performance reasons.

The security of any packet protection will be dependent on the security of the underlying block cipher. TinySec uses the Skipjack block cipher, which has a block size of 64 bit. However, TinySec could be used with any block cipher[6]. In our evaluations, we have employed AES, which has a block size of 128 bit. Advantages of AES are that is has an arguably higher security margin than Skipjack and that it can also be implemented faster on the target platform [7].

The block size of the block cipher plays a role for the message expansion in TinySec. TinySec encrypts in CBC mode and uses so-called *ciphertext stealing* to prevent message expansion. However, ciphertext stealing requires the availability of at least one complete ciphertext block. Therefore, the minimal size of the ciphertext is the block size and any smaller plaintexts get expanded under encryption. In contrast, our surveyed modes of operation all produce ciphertext of the same size as the corresponding plaintext, i.e., they effect no message expansion.

Another important issue is the reuse of IVs. As a general rule, IVs should never be used twice under the same key. Normally, the size of IVs is sufficient, so that reuse will not occur within longer periods of time. However, Karlof et al. argue that sensor networks cannot afford to include IVs of such size due to the energy overhead of wireless communication [14]. They propose an IV structure using existing packet header fields and with only a 16-bit counter field to guarantee uniqueness of IVs (cf. Figure 2). In the worst case, IVs will therefore repeat after 2^{16} packets[8]. TinySec encryption is designed in a way that an attacker will gain just minimal information about the two plaintexts in the case that IVs repeat. More precisely, only the length (in blocks) of the longest shared prefix of the two plaintexts will leak.

Our targeted modes of operation do not exhibit such a graceful degradation in the face of repeating IVs. The most catastrophic consequences are exhibited by the EAX mode and GCM, which both have a stream cipher encryption mode (CTR mode). Both consequently leak the XOR of the two plaintexts, which normally presents a serious threat to security. The CCFB+H mode behaves a bit more robust: Although the XOR of the first plaintext segments leaks, subsequent segments suffer the same fate only if all previous plaintext segments had been identical. OCB mode has the most favorable properties in the event of recurring IVs. Only if the length of the two plaintexts is identical, then the XOR of the last plaintext blocks leaks. For all other blocks, identical plaintext blocks will just result in identical ciphertext blocks.

Although OCB mode and CCFB+H mode are more robust than EAX mode and GCM in the face of repeating IVs, this situation should be strictly avoided in any of the four examined modes. Whenever the counter value is about to repeat, the key should be changed with the help of a key update protocol. Additionally, the size of the counter could be extended (e.g., by adding extra bytes to the packet header or by "borrowing" a few unused bits from other header fields) so that more packets can be sent without IV repetition.

5. Practical Results

We implemented the four described modes of operation from scratch in C and some core parts in assembly. Compilation was done with *avr-gcc 3.2* with *-O2* optimization. We have obtained our performance figures with simulation in *AVR Studio 4.13*.

We concentrated on the most expensive operations for optimization and we consider our implementations to be at an equal level of optimization. Whenever possible, we strived to reuse as much code as possible in order to preserve program memory.

An AES-128 implementation optimized for AVR microcontrollers has been used. It has an on-the-fly key expansion in order to limit the amount of RAM required for holding the key schedule.

5.1. Packet Structure and Size

We benchmarked the four AEAD modes with test data which is similar in structure and size to the packet format used in TinySec [14]. In general, these packets consist of an 8 byte header which is only authenticated[9], a message of up to 29 bytes[10] which is both authenticated and encrypted and a resulting 32-bit MAC tag. The packet format is depicted in Figure 2. Three different scenarios for the message size (header size is always 8 byte) were examined:

6. In fact, the authors mention in [14] that AES is likely to be a good replacement for Skipjack.

7. Skipjack encryption performance is reported at about 3,000 cycles per 64-bit block (about 380 cycles/byte), while our AES implementation requires about 4,000 cycles per 128-bit block (about 250 cycles/byte).

8. IVs reoccur only if the counter value repeats and all other header fields of the packet (source address, destination address, active messaging field, packet length are also identical.

9. Note that in TinySec, the header is also used as IV.

10. The current limit of TinyOS for the packet payload is 29 bytes.

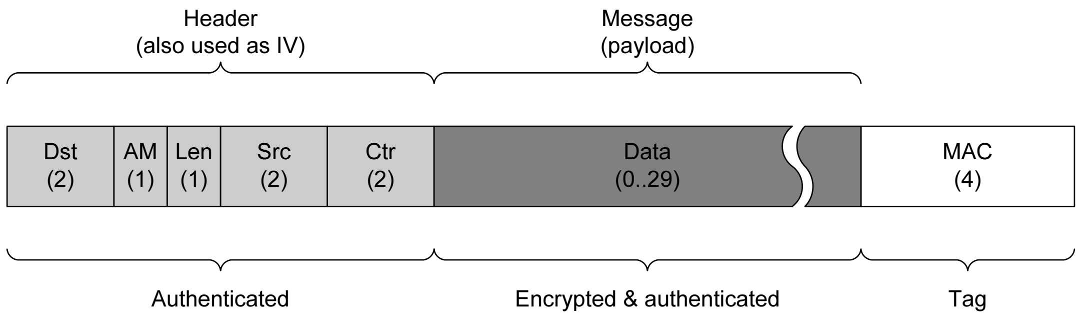

Figure 2. TinySec packet structure (according to [14]).

- 1 byte message: Short packets which convey very simple information (e.g., yes/no response).
- 17 byte message: Medium-size packets.
- 29 byte message: Full message size used for bulk communication.

In general, we used an 8-byte nonce as IV and generated a MAC tag of 4 bytes. For GCM and CCFB, we used an IV of 12 bytes[11]. GCM provides a considerable shortcut for nonces of this length. CCFB requires fewer encryption calls for longer IVs, due to its internal structure.

5.2. Performance Comparison

Table 2 gives the performance figures for our implementations on the MICAz motes. *Net mode overhead* stands for the cost of the particular mode minus the cycles for the block cipher operations[12]. *Setup costs* are the number of cycles needed for precomputations and caching of the results, which only occurs once per key (see Section 6.2).

CCFB+H performs best for short and a medium-sized messages. Additionally, it ties with OCB for long messages. EAX is behind both modes by at least 5,000 cycles. GCM does not perform well on our architecture.

The most interesting result is the good performance of CCFB+H compared to the one-pass scheme OCB. It manages to save one block cipher encryption for messages of up to 24 bytes because of its internal structure: The block cipher is used for encryption and authentication at the same time, reducing the second pass to an XOR operation on several blocks with the size of the resulting tag. For a tag size of 4 bytes, up to a maximum of 12 bytes of an AES-128 output can be used as a key stream to encrypt the message. This increases CCFB+H performance for short messages. For longer messages, the performance deteriorates.

OCB gets more competitive for long packets. EAX needs more block cipher calls in order to authenticate header and payload and thus is behind CCFB+H and OCB. EAX also needs several XOR operations on block-size data, which is a further penalty.

GCM does not perform well on the target platform. The high setup costs are due to the precomputation of a lookup table, as described in Section 6.1. Since the MAC function GHASH does not use the block cipher encryption, it needs the least block cipher calls of all four modes. However, the polynomial multiplication in GF(2^{128}) is very expensive on an 8-bit architecture and it cannot be optimized well due to memory constraints.

We did not measure decryption costs separately, since they are the same for all modes except OCB. OCB is the only mode using the block cipher decryption (for OCB mode decryption). This makes OCB less attractive, as it requires more program memory to realize the block cipher decryption and also renders OCB decryption slower whenever the block cipher decryption is slower than encryption[13].

5.3. Memory Requirements

Table 3 contains a comparison of the working memory required to cache precomputed data. All values are given in bytes. The block size of the assumed block cipher is 16 bytes, which affects the required memory.

As can be seen in Table 3, CCFB+H needs the least memory to store pre-computed values. OCB uses one additional 16-byte buffer. EAX comes in third but could be improved by leaving out the precomputation of seldomly used values, which can save 48 bytes in total (see 6.2 for details). GCM needs the most memory, which is due to the lookup table used (as described in Section 6.1).

The size of the programs themselves is between 5.4 KB and 5.8 KB. This is approximately 4.3% of the total program memory available on the MICAz motes.

6. Optimization Measures

We made several approaches in order to optimize the performance of the benchmarked modes. These approaches

11. Padding with a fixed value was used for modes which require a longer IV.

12. The cost per AES-128 block cipher invocation was about 4,000 clock cycles.

13. The measurements of our AES implementation showed that decryption is about 50% slower than encryption, averaging at roughly 6,000 cycles compared to the 4,000 cycles of encryption.

	OCB	CCFB+H	EAX	GCM
Setup costs	8,076	**4,636**	26,679	504,486
1 byte message, 8 byte header				
Total cycles	18,991	**14,450**	20,061	162,190
Block cipher calls	4	3	4	2
Net mode overhead	2,803	2,309	3,873	154,096
17 byte message, 8 byte header				
Total cycles	23,430	**18,919**	28,528	196,567
Block cipher calls	5	4	6	3
Net mode overhead	3,195	2,731	4,246	184,426
29 byte message, 8 byte header				
Total cycles	**23,810**	**23,356**	28,672	196,711
Block cipher calls	5	5	6	3
Net mode overhead	3,575	3,121	4,390	184,570

Table 2. Performance comparison of the four modes of operation on MICAz. Note that the performance figures for CCFB+H for 17-byte messages have been erroneous in the original publication and are corrected in this table.)

	OCB	CCFB+H	EAX	GCM
Memory usage	96	80	176	352

Table 3. Memory requirements (in bytes) for precomputed data.

can be divided into several categories:

- Use of lookup tables
- Precomputation of key-dependent results
- Memory optimizations
- Assembler-level optimizations

In this section, we describe some of the measures we have taken in order to speed up our implementations.

6.1. Lookup Tables in GCM

GCM is the only mode that relies on a polynomial multiplication for the MAC tag generation. This multiplication is dependent on the so-called hash subkey, whose value in turn only depends on the used key. It is possible to exploit this property by creating a precomputed table that stores some of the multiplication results.

Due to memory restrictions, we implemented only 4-bit tables. With these tables, the required GF(2^{128}) multiplications can be done considering 4-bit chunks (instead of single bits), thus leading to an increase in performance. With 4-bit tables, the additional required memory is a fixed buffer of 32 bytes plus a key-dependent table of 256 byte in total.

Nevertheless, this optimization did not result in competitive performance for GCM. We did not implement bigger tables due to the added storage costs, which we considered unsuitable for our target platform.

6.2. Precomputation of Key-Dependent Results

If the key is valid for a number of packets, some key-dependent operations can be precomputed to increase performance. We applied this method wherever sensible, and every mode can gain at least some increase in performance. In addition to an initial encryption of a fixed-value string, which all modes need, we removed additional duplicate calculations in the MAC functions PMAC, OMAC and OMAC^t.

In both OMAC and PMAC, we did a pre-calculation of the key-dependent buffers, which included the expensive encryption as well as several shifting operations. For the tweaked OMAC version OMAC^t in EAX, up to three encryption calls as well as some shift operations could be saved due to its special structure. This optimization pre-calculates the first CBC message block which is used in OMAC. Because of padding problems regarding empty headers or messages, this optimization calculates three additional buffers of block length. If an application does not use empty data or does not use it often, this additional optimization can be skipped, resulting in a reduced memory usage.

6.3. Assembler-Level Optimization

We implemented several core functions in assembly. Optimized block-shift operations offered the best gains. This optimization is very effective for GCM, where the shifting operation is heavily used. OCB, EAX and CCFB+H receive a smaller, but still noticeable performance boost.

6.4. Additional Optimizations

There are additional optimization measures, which we did not incorporate. McGrew et al. describe in [6] a way to calculate the lookup table of GCM more efficiently. Since the performance of GCM is too far behind its contestants, we did not pursue this optimization.

Furthermore, we did not make heavy use of assembler-level code, but concentrated only on some core routines. Thus, rewriting parts of the implementation (or all of it) in assembly is likely to improve the performance of the modes. However, we do not expect the ranking to change if all modes are optimized equally.

7. Recommendations

Our measurements show that CCFB+H performs best for the assumed packet format. It has little overhead and makes clever use of the block cipher, thus reducing the number of block cipher invocations. Similarly to the other modes, it provides minimum overhead (which is a crucial factor on the target platform). It is rather easy to implement and there are no pending patents. On the downside, up to now there seem to be no free implementations of CCFB+H and there is little support for this mode (e.g., no available test vectors). Still, we would recommend this mode due to its good performance (especially for short packets) and its small memory requirements.

OCB performs well with long packets, but it is patented. Furthermore, OCB decryption has some drawbacks. While EAX is free, it generally requires more block cipher calls than its contestants.

In contrast to CCFB+H, OCB, and EAX, we do not deem GCM suitable for the target platform. A decent optimization cannot be achieved due to the limited memory. With the possible optimizations, the overhead is too large compared to the other modes.

Acknowledgment

We would like to thank Christoph Herbst for providing an optimized AES implementation and Peter Holzer for support regarding TinyOS.

The research described in this paper has been supported by the European Commission under grant number FP6-IST-033563 (Project SMEPP). The information in this document reflects only the authors' views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

[1] M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm," in *Proceedings of ASIACRYPT 2000*, Springer, December 2000, pp. 531–545.

[2] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of CCS 2002*, ACM Press, November 2002, pp. 98–107.

[3] T. Krovetz and P. Rogaway, "The OCB Authenticated-Encryption Algorithm," Available online at http://www.cs.ucdavis.edu/~rogaway/papers/draft-krovetz-ocb-00.txt, March 2005.

[4] M. Bellare, P. Rogaway, and D. Wagner, "The EAX Mode of Operation," in *Proceedings of FSE 2004*, Springer, February 2004, pp. 389–407.

[5] S. Lucks, "Two-Pass Authenticated Encryption Faster Than Generic Composition," in *Proceedings of FSE 2005*, Springer, February 2005, pp. 284–298.

[6] D. A. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM)," Available online at http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-revised-spec.pdf, May 2005, revised Submission to NIST Modes of Operation Process.

[7] Crossbow Technology, Inc., "MICAz Wireless Measurement System," Available online at http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf.

[8] TinyOS Community, "The TinyOS Website," http://www.tinyos.net.

[9] D. Chakraborty and P. Sarkar, "A General Construction of Tweakable Block Ciphers and Different Modes of Operations," in *Information Security and Cryptology*, Springer, 2006, pp. 88–102.

[10] National Institute of Standards and Technology (NIST), "Special Publication 800-38A 2001 ED, Recommendation for Block Cipher Modes of Operation - Methods and Techniques," December 2001, available online at http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

[11] P. Rogaway, M. Bellare, J. Black, , and T. Krovetz, "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption," in *Proceedings of CCS 2001)*, ACM Press, 2001, pp. 196–205.

[12] National Institute of Standards and Technology (NIST), "Special Publication 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," Available online at http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf, November 2007.

[13] T. Kohno, J. Viega, and D. Whiting, "CWC: A high-performance conventional authenticated encryption mode," in *Proceedings of FSE 2004*, Springer, February 2004, pp. 408–426.

[14] C. Karlof, N. Sastry, and D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks," in *Proceedings of SenSys 2004*. ACM Press, November 2004, pp. 162–175.