Authenticated Encryption Modes of Block Ciphers, Their Security and Implementation Properties

Hequn Chen

2009

Seminararbeit

Ruhr-Universität-Bochum

Chair for Communication Security
Prof. Dr.-Ing. Christof Paar

betreut durch Axel Poschmann

Abstract

In this thesis, four authenticated encryption modes of operation are presented, namely, GCM mode, CCM* mode, OCB mode and CWC mode. These modes can provide confidentiality and authenticity simultaneously. The GCM mode and CCM* mode are in detail introduced. And it shows the comparison of the four modes from different aspects: properties, security, and performance in hardware implementation and software implementation.

Contents

1. Introduction

In this Section first the work is motivated in Section 1.1 and subsequently in Section 1.2 the outline is presented.

1.1 Motivation

Since the block cipher's research began in the 1970th, it has so far a history of more than 30 years, during which time it is developed rapidly. A block cipher is a symmetric key cipher which maps *n*-bit plaintext blocks as input to *n*-bit ciphertext blocks as output. The *n* is called the blocklength and defined as fixed-length. The key, as the second input, is chosen at random. Use of plaintext and ciphertext blocks of equal size avoids data expansion.

While a plaintext may be of any length, the block cipher can only operate a fixed-length message. Therefore, several modes of operation on arbitrary length of message are developed. For example, ECB (Electronic Codebook Mode), CBC (Cipher-block Chaining Mode), CFB (Cipher Feedback Mode) and OFB (Output Feedback Mode), as some of the earliest modes, can only provide confidentiality or authenticity, but are not able to provide both simultaneously. GCM (Galois/Counter Mode) [1] [2] [3], CCM* (variant of the Counter with CBC Mode) [4] [5] [6], OCB (Offset Codebook Mode) [7] [8] and CWC (Carter-Wegman + CTR Mode) [9], as some of the new developed modes, can perform confidentiality and authenticity simultaneously, and thus are called the Authenticated Encryption (AE) mode. In addition to protecting authenticity and confidentiality, authenticated encryption can provide plaintext awareness and security against chosen ciphertext attack. The plaintext awareness is a security notion, which means no one can generate a valid ciphertext without knowing the corresponding plaintext.

The new developed AE modes are built by improving or combining of the well-known modes. Those well-known modes have trusty performance to provide security or other attractive characteristic. For example, the CCM* mode is combination of the Counter mode and the Cipher Block Channing (CBC) mode. And the GCM mode combines the well-known Counter mode with the new Galois mode. The Counter mode can not only provide high security but also can be efficiently implemented. These AE modes extend the advantages of the known modes and improve them by the carefully chosen algorithm to provide confidentiality and authenticity. Furthermore, some AE modes have the additional useful property, which are suitable for different application.

Besides security, implementation as another important factor must be considered for the AE modes in practice. If an AE mode has the satisfying performance in security, it will not make sense, when it can not be efficiently implemented in hardware or software. The efficient implementation is an important factor for obtaining a good performance.

1.2 Outline of this thesis

In this thesis two authenticated encryption modes are presented, namely, the GCM mode and the CCM* mode. Firstly, some important definitions of each mode are given, such as inputs and outputs. Then, their algorithm for operating encryption and authentication is in detail described. And it introduces briefly also the security features of these AE modes and their implementation in hardware and software. Besides, a rough introduction of the OCB mode is given. In last part, the three above called AE modes are compared with each other. It involves the different property of each mode, their performance in security and implementation.

2. GCM Mode

The Galois/Counter Mode (GCM) is a block cipher mode of operation which is used to provide authenticated encryption. As the name suggests, the GCM mode combines the well-known counter mode with the new Galois mode. The counter mode of operation (CTR) has become the mode of choice for high speed applications, because it can be efficiently pipelined in hardware implementations [3]. It is used in the GCM mode to provide the confidentiality, and furthermore the GCM mode provides the authenticity of the confidential data by using a universal hashing over a binary Galois field.

An additional useful properties of GCM are that it can be used as a stand-alone MAC when there is no plaintext inputed. Additionally, GCM can be used as an incremental MAC: if an authentication tag is computed for a message, then part of the message is changed, an authentication tag can be computed for the new message with computational cost proportional to the number of bits that were changed [2]. And GCM also accepts the initialization vector for arbitrary length so that it is easy to generate non-repetitive IVs.
Another important factor is implementation. GCM can perform satisfactorily not only in hardware, but also in software. GCM can provide authenticated encryption at speeds of 10 gigabits per second and above [2]. Because of the counter mode, GCM can be efficiently pipelined and parallelized in hardware implementation for encryption, and the binary field multiplication for authentication can be easily implemented. In software, GCM also can perform well by using table-driven field operations.

2.1 Definition

The GCM mode can be applied to 64-bit block cipher. GCM has two functions, authenticated encryption and authenticated decryption.
The authenticated encryption function has four inputs, they are a bit string:

- A secret key, denoted *K*, which has a length appropriate to the block cipher.
- A plaintext, denoted *P*, which can have any bit length between 0 and $2^{39} – 256$.
- An initialization vector, denoted *IV*, which has any bit length between 1 and 2^{64}.
- Additional authenticated data (AAD), denoted *A*, which has any bit length between 0 and 2^{64}.

The two outputs of the authenticated encryption function are:

- A ciphertext, denoted *C,* which has the same length as that of the plaintext.
- An authentication tag, denoted *T,* which has any bit length between 64 and 128.

The *IV* is applied to a nonce, for each fixed key, each value of the *IV* must be distinct. But as above mentioned, the *IV* can be arbitrary length. GCM protects the authenticity of the plaintext *P* and the Additional authenticated data *A*. The confidentiality of the plaintext is certainly protected, but the AAD must be unencrypted. Because, when GCM is used in a network protocol, the AAD could include addresses, ports, sequence numbers, protocol version numbers, and other fields which explain handle of the plaintext. The bit length of the tag *T,* denoted *t*, is a security parameter which can determine the quality of the authentication of *P*, *IV* and *A*.

The authenticated decryption function has five inputs: *K*, *C*, *IV*, *A* and *T*. The single output of the authenticated decryption function is either the plaintext *P* which corresponds to the input ciphertext

C or a special symbol *FAIL* which means the inputs are not authentic. If the inputs were not generated by the encryption function with the identical *k*, it would return with high probability the special symbol *FAIL*.

2.2 Algorithm

The GCM mode uses a variation of the Counter mode with an incrementing function to ensure the confidentiality. And the authentication is ensured by using a hash function over a binary Galois field, this hash function is called GHASH. Before introducing the functions of GCM, it is necessarily to describe briefly the GHASH function, incrementing function and GCTR function.

The GHASH is a keyed hash function but not, on its own, a cryptographic hash function [1]. For an input *X*, where *X* is 128*m* bits string and m is an positive integer, the output of $GHASH_H(X)$ is a 128-bit length string. More exactly, the input *X* is distributed into *m* block, and Y_0 is a 128-bit length "zero block", 0^{128}. For $i=1,..m$, $Y_i=(Y_{i-1} \oplus X_i)*H$, where block *H* is the hash subkey. The "*" operation is multiplication of two blocks and the product is an elements of a certain binary Galois field. The Y_m is the 128-bit length output of $GHASH_H(X)$.

The *s*-bit incrementing function is defined as follows:

$$inc_s(X)=MSB_{len(X)-s}(X) \,||\, [int(LSB_s(X))+1 \bmod 2^s]_s.$$

The right-most *s* bits of *X* is represented as the binary representation of an integer, and incremented, modulo 2^s. The rest of the string is unchanged.

The GCTR function decompositions the string input into 128-bit block and the rightmost block may be a partial block. The 32-bit incrementing function is iterated on the initial counter block input to generate the counter block. The block cipher is applied to the counter block and the results are XOR-ed with the string block, and as output.

The Authenticated Encryption Algorithm:

A plaintext *P* is firstly distributed into blocks. Let *n* and *u* be two positive integers, the number of bits of *P* is equal to (*n*-1)128+*u*, where $1 \le u \le 128$. The plaintext consists of a sequence of *n* bit strings, which is denoted $P_1, P_2, \ldots, P_n^*$ and called data blocks. The last data block has *u* bit length which may not be a complete block, and the bit string of other data blocks is 128. Similarly, the ciphertext is also distributed into blocks and denoted $C_1, C_2, \ldots, C_n^*$, where the last block has *u* bit length. The additional authenticated data A is denoted as $A_1, A_2, \ldots, A_m^*$, where the last block may not be a complete block of length *v*, and, *m* and *v* are two positive integers, the number of bits of *A* is equal to (*m*-1)128+*v*, where $1 \le v \le 128$.
For the inputs *P*, *A*, *K*, *IV*, the authenticated encryption function is specified below [1]:

Steps:

1. Let $H=E(K, 0^{128})$.
2. Define a block J_0, as follows:
   if len(IV) = 96, then let J_0 = IV || 0^{31} ||1.
   If len(IV) ≠ 96, then let *s* = $128\lceil len(IV)/128 \rceil - len(IV)$, and let

J_0=GHASH$_H$(IV||0^s+64|| $\lceil len(IV) \rceil_{64}$).

3. C=GCTR$_k$ (inc32(J_0), P)
4. Let $u = 128\lceil len(C)/128 \rceil - len(C)$ and let $v = 128\lceil len(A)/128 \rceil - len(A)$.
5. Define a block, S, as follows:

$$S=\text{GHASH}_H(A \,||\, 0^u \,||\, C \,||\, 0^v \,||\, [\text{len}(A)]_{64} \,||\, [\text{len}(C)]_{64}).$$

6. Let T = MSB$_t$ (GCTR$_k$(J_0, S)).
7. Return (C,T).

In Step 1, the hash subkey H is generate by using the block cipher to encrypt 128-bit length 0 with the key K. In Step 2, J_0 as the pre-counter block is generated from IV. If IV is a 96-bit length string, the padding string 0^{31}||1 will be attached to the IV. Otherwise, the GHASH function is applied to generating the J_0 from IV, s+64 '0' bit and the 64-bit representation of the length of the IV. In Step 3, the initial counter block is generated by using the 32-bit incrementing function from J_0, it and the plaintext as the inputs of the GCTR function, the ciphertext is output of the GCTR function. In Step 4 and 5, the length of two padding string is computed, which are attached to the A and C to complete the last block. The concatenation of these strings is appended with the 64-bit representation of the length of the AAD and the ciphertext. Then, a block S is generated form these strings by using the GHASH function. In Step 6, the authenticated tag T is the t digits MSB of the result of the GCTR function with J_0 and block S. Finally, it returns ciphertext C and tag T as the outputs in Step 7.

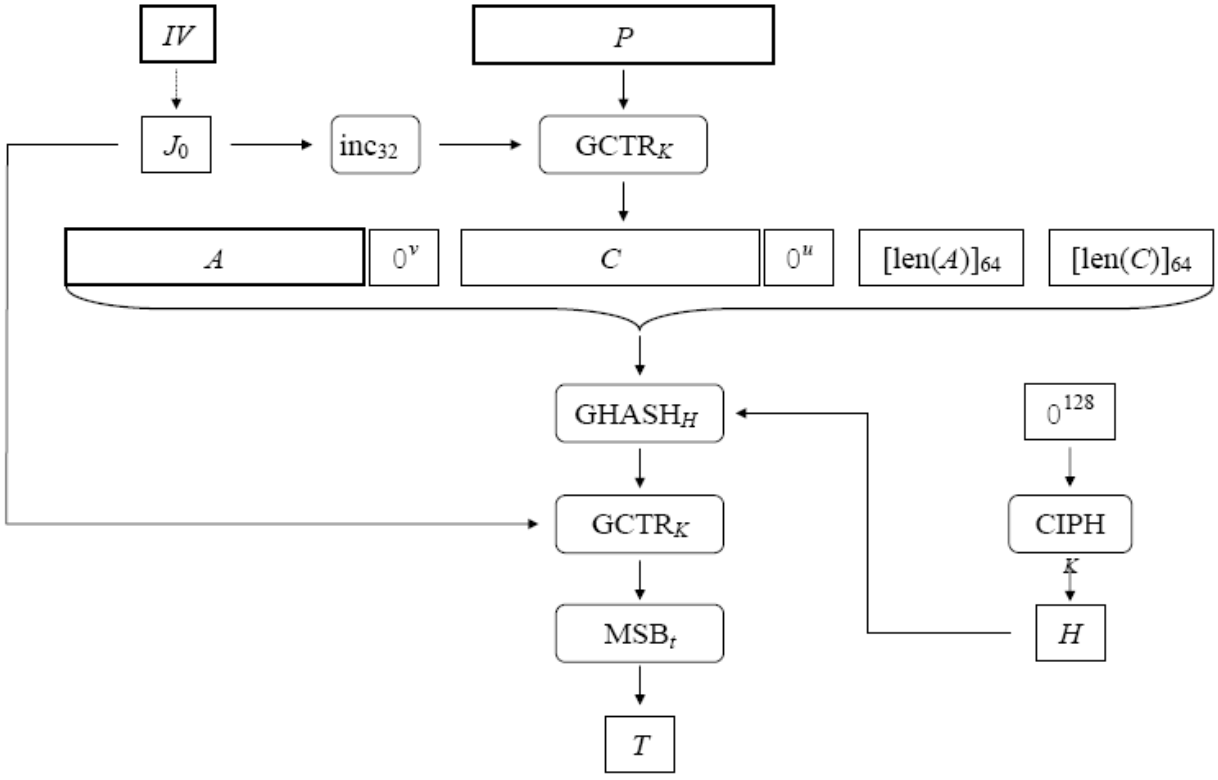

Figure 1: the authenticated encryption algorithm of GCM mode [1].

The Authenticated Decryption Algorithm:

For the inputs C, A, T, IV, the authenticated decryption function is specified below [1]:

Steps:

1. If the bit lengths of *IV*, *A* or *C* are not supported, or if len(*T*) ≠ *t*, then return *FAIL*.
2. Let H = $CIPH_K(0^{128})$.
3. Define a block, J_0, as follows:
   If len(*IV*)=96, then J_0 = IV || 0^31 ||1.
   If len(*IV*) ≠ 96, then let $s = 128\lceil len(IV)/128 \rceil - len(IV)$, and
   $J_0 = GHASH_H(IV \| 0^{s+64} \| [len(IV)]_{64})$.
4. Let $P = GCTR_K(inc_{32}(J_0), C)$.
5. let $u = 128\lceil len(C)/128 \rceil - len(C)$ and let $v = 128\lceil len(A)/128 \rceil - len(A)$.
6. Define a block, S, as follows:
   $S = GHASH_H(A \| 0^u \| C \| 0^v \| [len(A)]_{64} \| [len(C)]_{64})$.
7. Let $T' = MSB_t(GCTR_k(J_0, S))$.
8. If $T = T'$, then return *P*; else return *FAIL*.

This Algorithm works like the authenticated encryption algorithm above, the slight different is that the ciphertext *C* replaces the plaintext *P* as the input and the authenticated tag *T* is also as the input. And in Step 1, the lengths of the inputs are verified whether they are supported or not. In last Step, the generated authenticated tag *T′* is compared with the received *T,* to determine the authenticity of the received ciphertext, if they are identical, the plaintext is returned; otherwise, *FAIL* is returned.

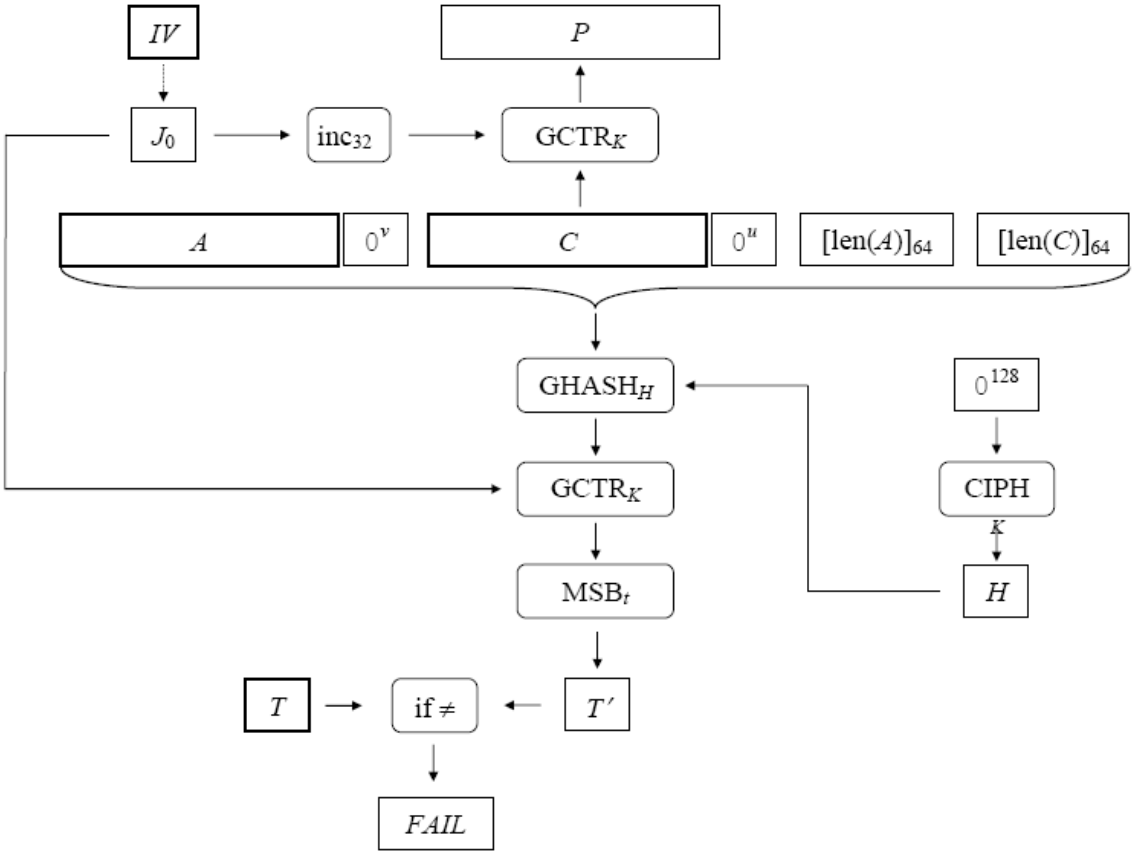

Figure 2: the authenticated decryption algorithm of GCM mode [1].

2.3 Security Features

The encryption and authentication of GCM are secure against the chosen-plaintext attack, and GCM is also secure while using the AES block cipher [3]. The system-security of GCM is considered. In CTR mode, the reusing of a key/IV lead to injure the confidentiality of the plaintext. Because the

GCM mode uses a variation of the Counter mode to ensure the confidentiality, it has also the same problem. In the GCM authenticated encryption function, if the IV is reused, the attacker will be able to solve for the subkey H of GHASH. Then, the attacker can choose IV that will cause colliding counters. Fortunately, the reuse of an IV in the authenticated decryption function dose not cause this problem. Even if the ciphertext are repeatedly decrypted with the same IV, the attacker still cannot gain useful information about H efficiently. Because GCM accepts an arbitrary length IV, it is easily to avoid reusing of the IV.

2.4 Implementation

The implementation of the GCM in hardware and software can be seen as the implementation of the underlying block cipher and the multiplication operation over $GF(2^{128})$. This section presents the efficient way to implement GCM in hardware and software.

Multiplication in a binary field is the focus in software implementation. For an operation $H \cdot X$, because of the linearity of the bits of X, the most efficient method is the table-driven field operation [2].
Firstly, a filed element is decomposed into a sum of field elements which has only eight nonzero bits. S is denoted as the set of elements in $GF(2^{128})$ that has the leftmost 8 nonzero bits. A is an element in set S, and it is multiplied by H, which is an element in $GF(2^{128})$. The product of $X \cdot H$ can be computed easily, because there are only 2^8=256 elements in S, and a table M is constructed by looping over all 256 cases and computed each result.
The decomposition of X is as follows:

$$X = \bigoplus_{i=0}^{15} x_i \cdot P^{8i}, \quad \text{where} \quad x_i \in S \quad \text{for all } i.$$

Then, the product $X \cdot H$ can be expressed as:

$$X \cdot H = \bigoplus_{i=0}^{15} x_i \cdot H \cdot P^{8i} = \bigoplus_{i=0}^{15} M_i[\text{byte}(X, i)].$$

Where byte (X,i) is the i-th byte of the element X. There is 16 tables $M_0 ... M_{15}$ and each has 256 values, each of the values is 16 bytes long, it is total 63,536 bytes. When the table is used in GCM, X can be decomposed into 32 components with 4 bits each, for a total of 8,192 bytes to save the memory.
There is also a variant of this method, it was described by Shoup. The variant saves the memory, but increases the amount of computation. In the table below shows comparison of these methods [2].

Method	Storage requirement	Throughput (cycles per byte)
Simple, 8-bit tables	65,535 bytes/key	13.1
Simple, 4-bit tables	8,192 bytes/key	17.3
Shoup's, 8-bits tables	1024 bytes + 4096 bytes/key	32.1
Shoup's, 4-bit tables	64 bytes + 256 bytes/key	69.3
No tables	16 bytes/key	119

Table 1: The throughput of GHASH using various different methods for the Galois field multiplication on a Motorola G4 processor.

The GCM mode can be efficiently pipelined and parallelized in hardware implementation for encryption. Figure 3 [2], below shows a pipelined hardware design. There are three inputs: the AAD, the IV and the plaintext at the top ,and two outputs: the tag and the ciphertext at the bottom. The IV streams into the incrementing function to output the counter values, and the values are encrypted in the block cipher E_k. Then, the encrypted counter is sent to encrypt the GHASH output (path 3), and this output is switched so that the other encrypted counters are XOR-ed with the plaintext to generate the ciphertext (path 2). The AAD streams into the GHSAH function (path 1), and the output is switched to the ciphertext into the GHASH function. After all of the data into GHASH has been processed, the output is XOR-ed with the first encrypted counter to generate the authentication tag. In this design, the ciphertext-generating pipeline and the tag-generating pipeline are independent. These two pipelines can be made completely independent by adding another AES engine dedicated to the encryption of the GHASH output [2].
The binary Galois field multiplication can be also efficiently implemented in hardware by some methods, such as Bit Serial, Digital Serial or Super Serial [2].

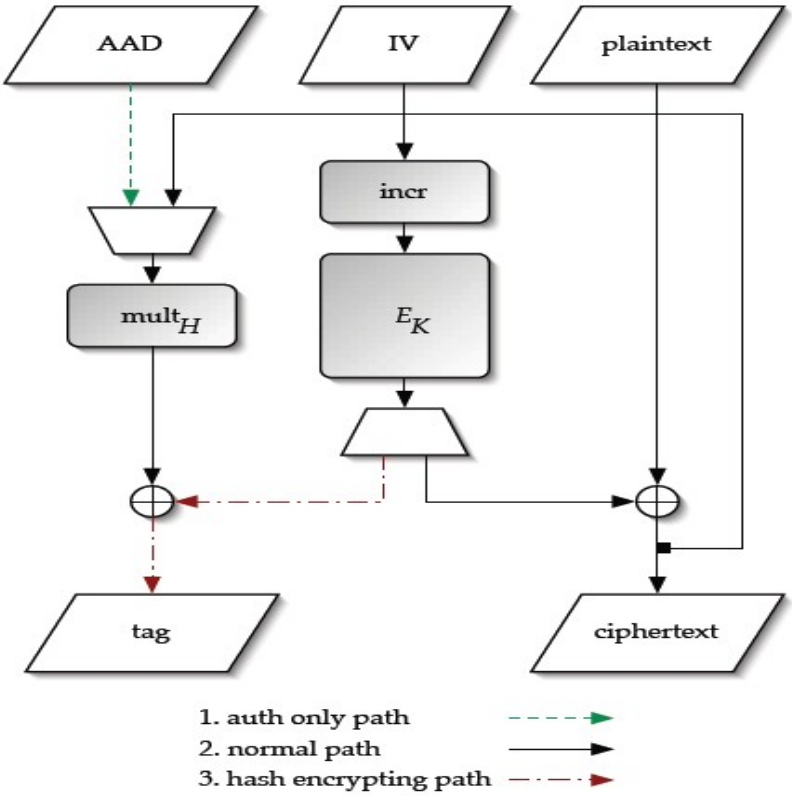

Figure 3: A hardware pipelined implementation of GCM

3. CCM* Mode

The CCM* mode is an another block cipher mode of operation which is used to provide authenticated encryption. This mode is a variant of the CCM mode, it extend the advantage and improves the disadvantage of the CCM mode. It is necessary to describe first the CCM mode roughly.
The CCM mode [5] is a combination of the Counter (CTR) mode and the Cipher Block Channing (CBC) mode, where the Counter mode and the CBC mode are applied respectively to provide confidentiality and authenticity with using a single key. The CCM mode is only defined to support

the 128-bit block cipher algorithm, such as AES-128. The CCM mode accepts a variable-length authentication tags (from 32-bits to 128-bits), thus allowing varying degrees of protection against unauthorized modifications [4]. Because it only needs to implement the encryption transformation of the underlying block cipher and uses a single key. However, the CCM mode has also some disadvantages as followings [4]:

- The CCM mode dose not provide for confidentiality-only services. For some cases, it uses data authenticity mechanism from external.
- The CCM mode can be vulnerable to specific attacks, if used with variable-length authenticated tag. Although CCM can be securely used with the same key in settings with fixed-length authentication tag. Support for variable-length is a very useful property in some implementation environments.

The CCM* mode extends the original CCM mode and takes away the disadvantages of CCM mode, such that providing for confidentiality-only services and to be securely using with variable-length authentication tags. On the whole, CCM* mode is an improved CCM mode.

3.1 Definition

The CCM* mode has two functions, namely encryption-authentication function and decryption-verification function. There are some prerequisites for the operation of the generic CCM* mode: A block cipher function E with a 128-bit block size, the length L of the message length field and the length M of the authentication field. The length L and M are in octets, and valid values for L are the integers from 2 to 8 and for M are the integers 0, 4, 6, 8, 10, 12, 14 and 16. The encryption-authentication function has 4 inputs as followings:

- A secret key, which has a length appropriate to the block cipher.
- A message m, which has a length $l(m)$ octets between 0 and 2^{8l}.
- A authentication field a, which has a length $l(a)$ octets between 0 and 2^{64}.
- A nonce N, which has a length 15-L octets.

Because the CCM* mode uses single key, the key does not allow to be accessed without evidence. The value of the nonce should be unique for each key in encryption.

The CCM* mode has a specific formatting of the strings and a specific representation of integers as octet strings, therefore, the input strings must be transformed before used by encryption-authentication function. The transformation is as followings [4]:

1. Form the octet string representation $L(a)$ of the length l(a) of the octet string a, as follows:
   - If $l(a)=0$, then $L(a)$ is the empty string.
   - If $0 < l(a) < 2^{16}-2^8$, then $L(a)$ is the 2-octets encoding of $l(a)$.
   - If $2^{16}-2^8 \leq l(a) < 2^{32}$, then $L(a)$ is the right-concatenation of the octet 0xff, the octet 0xfe, and the 4-octets encoding of $l(a)$.
   - d. If $2^{32} \leq l(a) < 2^{64}$, then $L(a)$ is the right-concatenation of the octet 0xff, the octet 0xff, and the 8-octets encoding of $l(a)$.
2. Right-concatenate the octet string $L(a)$ with the octet string a itself. Note that the resulting string contains $l(a)$ and a encoded in a reversible manner.
3. Form the padded message *AddAuthData* by right-concatenating the resulting string with the smallest nonnegative number of all-zero octets such that the octet string *AddAuthData* has length divisible by 16.

4. Form the padded message *PlaintextData* by right-concatenating the octet string m with the smallest nonnegative number of all-zero octets such that the octet string *PlaintextData* has length divisible by 16.
5. Form the message *AuthData* consisting of the octet strings *AddAuthData* and *PlaintextData*:

$$AuthData = AddAuthData \,||\, PlaintextData.$$

The output of this function is the right-concatenation of the encrypted message *Ciphertext* and the encrypted authentication tag *U*.

The decryption-verification function has also four inputs, but the slight difference is that the octet string of ciphertext replaces the plaintext as the input. The ciphertext octet string has the length of $l(c)$ octets, where $0 \leq l(c) < 2^{8L}$. If the authentication data has failed by verifying, output 'invalid' and reject the ciphertext; otherwise output the plaintext.

3.2 Algorithm

The CCM* mode uses the CBC mode to ensure authenticity and the CTR mode to ensure confidentiality. The encryption-authentication function as well as the decryption-verification function is separated into two parts, each provides authenticity and confidentiality respectively.

For the encryption-authentication function, to establish the authentication tag *T* [4], the 1-octet *Flags* are generated firstly which consists of the 1-bit *Reserved* field, the 1-bit *Adata* field, and the 3-bit representations of the integers *M* and *L*:

$$Flags = Reserved \,||\, Adata \,||\, M \,||\, L.$$

The *Reserved* field is reserved for future expansions and has the value '0'. The 1-bit *Adata* filed depends on the value of $l(a)$. If $l(a)=0$, it will be set to '0', and set to '1' if $l(a)>0$. The *M* field is the 3-bit representation of the integer $(M-2)/2$ if $M>0$ and of the integer 0 if $M=0$, in most-significant-bit-first order. The *L* field is the 3-bit representation of the integer $L-1$, in most-significant-bit-first order.
Then, the 16-octet B_0 field is generated which consists of the 1-octet *Flags* field above, the $15-L$ octet nonce field *N*, and the *L*-octet representation of the length field $l(m)$:

$$B_0 = Flags \,||\, N \,||\, l(m).$$

The message *AuthData* defined above, is decomposed into 16-octet string blocks, such as:
$B_1 \,||\, B_2 \,||\, \ldots \,||\, B_t$.

Next, X_{i+1} as value of the CBC-MAC is generated by:

$$X_0 := 0^{128},\ X_{i+1} = E(Key,\ X_i \oplus B_i) \text{ for } i=0, \ldots, t.$$

Finally, value of the authentication tag *T* is the leftmost *M* octets of the CBC-MAC value X_{i+1}.

To encrypt a plaintext, the 1-octet *Flags* are also generated firstly which consists of two 1-bit *Reserved* field, and the 3-bit representations of the integers 0 and *L*:

$Flags = Reserved \,||\, Reserved \,||\, 0 \,||\, L$.

The two 1-bit *Reserved* fields are reserved for future expansions and have the value '0'. The 0 field is the 3-bit representation of the integer 0, in most-significant-bit-first order. The *L* field is the 3-bit representation of the integer *L*-1, in most-significant-bit-first order.
Then, the 16-octet A_i field is generated which consists of the 1-octet *Flags* field above, the 15-*L* octet nonce field *N*, and the *L*-octet representation of the integer *i*:

$A_0 = Flags \,||\, N \,||\, Counter\ i$, for i=0,1,2, ...

The message *PlaintextData* is decomposed into 16-octet string blocks, such as:
$M_1 \,||\, M_2 \,||\, ... \,||\, M_t$.

Next, the ciphertext blocks C_1, ... , C_t are generated by:

$C_i := E(Key, A_i) \oplus M_i$) for i=0, ... , t.

And the ciphertext string is the leftmost $l(m)$ octets of the string $C_1 \,||\, ... \,||\, C_t$.
The 16-octet encryption block S_0 is defined by $S_0 := E(Key, A_0)$, whose leftmost *M* octets are XOR-ed with the authentication tag *T*. The result is the encrypted authentication tag *U* and it is attached to right of the ciphertext string, to be *c*.

For the decryption-verification function, to decrypt the received $c=C||M$, it is checked first, whether the right-most string *U* is an *M*-octet string or not. If not, output 'invalid' and stop. Then, the *CiphertextData* is padded by right-concatenating the string *C* with the smallest non-negative number of all-zero octets such that the octet string *CiphertextData* has length divisible by 16. Next, the function runs as same as the encryption function with the *CiphertextData* an the tag *U* as the input. Finally, the output string is $m||T$, where the right-most string *T* is *M*-octet string, and it is the authentication tag *T*, and the rest is the plaintext.

To verify the authentication data, the function uses the output of the decryption function and generates like the authentication function the authentication tag *T*. It compares both tags to verify.

3.3 Security Features

The CCM* mode as a variant of the original CCM mode inherits also some security features of the CCM mode.
The confidentiality of CCM depends on the CTR mode, because of action of the nonce in CTR mode, the input blocks are fresh, and the output is very close to at random. The “birthday” attack seems the only successful method for the adversary, since all input blocks of the chosen block cipher are distinct, the probability of appearing collision among the CTR output is negligible [6]. The authenticity is in the same situation as the confidentiality, it is hard to distinguish the output at random.

However, the CCM* mode restricts further value of the nonce, one can uniquely determine from the nonce the value of the authentication field. But as above mentioned, the CCM* mode provides the confidentiality-only services, it means that the authentication field could be an empty string namely M=0. In this case, there are no additional restrictions on the nonce, and the CCM* mode has the same security features as the CCM mode.

For the fixed-length authentication tags, the confidentiality over the input string *m* and data authenticity over the input string *a* and *m*, relative to the length of the authentication tag [4]. Because of $B_0 = Flags \parallel N \parallel l(m)$, the B_0 field can uniquely determine the value of the nonce *N*, and because of *AuthData = AddAuthData || PlaintextData,* the authentication data can uniquely determine the input string *a* and *m*, and the authentication tag is generated from the authentication data. In this case, the CCM* mode also has the same security features as the CCM mode.

For the variable-length authentication tags, the CCM mode can be vulnerable to specific attacks, since the decryption function does not depend on the length of the authentication tag itself [4]. In this case, the CCM* mode generates the A_i field from the nonce and the counter to provide the security.

3.4 Implementation

The CCM* mode uses CBC-MAC mode to provide the authenticity, but CBC-MAC mode can not be pipelined or parallelized. And the CCM* mode is highly complex. Therefore, there is not more efficient method for the CCM* mode to implement in hardware or software. CCM* can be simplify implemented by using the same block cipher key for the encryption and authentication functions.

In hardware implementation [10], CCM* can be generally separated into three modules, namely encryption module, authentication module and a control unit module. Figure 4 [10] below shows a general architecture of CCM* mode.

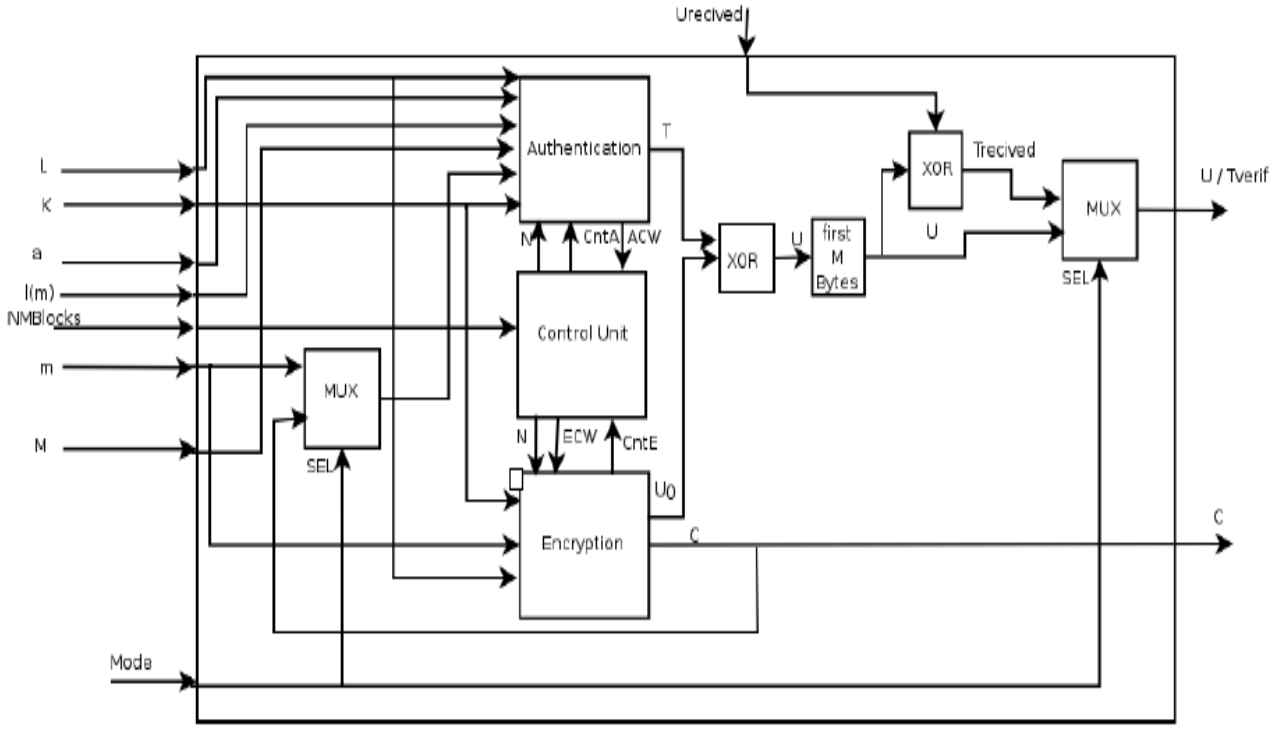

Figure 4: A general architecture of CCM* mode.

The authentication module consists an Authentication Block Generator, a CBC-MAC module and a Control Unit. The encryption module consists of an Encryption Block Generator, a CTR block and a Control Unit. General Control Unit controls the Authentication and Encryption Processes. It synchronizes the information flow in order to parallelize the entire process and to achieve a good performance [10].

4. Other Authenticated Encryption Modes

The GCM mode and CCM* mode are two representative authenticated encryption modes, which are in detail described above. In this part, some other authenticated encryptions modes are roughly introduced, namely the OCB mode and the CWC mode. Here gives only the general description about their definitions and properties.

4.1 OCB Mode

The Offset Code Book (OCB) mode [7] is another AE mode, which provides the confidentiality and the authenticity simultaneously. Before appearing of the AE modes, the confidentiality and the authenticity are provided separately by two systems: block cipher for the confidentiality and MAC for the authenticity. However, the OCB mode combines appropriately block cipher and MAC, and the computational cost is lower as the two separate systems.

The OCB mode accepts the block cipher, which have the size of 128, 192 and 256. The length t of the authentication tag T is an integer between 0 and n. The adversary is able to forge a valid ciphertext with probability 2^{-t}, a suggested default of t is 64. This mode takes the key, plaintext and nonce as the inputs of the encryption algorithm [7] and outputs are the nonce, ciphertext and authentication tag. The decryption algorithm [7] works similar to the encryption algorithm, according to the outputs of the the encryption algorithm to compute the tag and plaintext, after comparing both tags, returns the plaintext or *INVALID*.

Because of using MAC to provide the authenticity, the OCB mode is plaintext-aware. Furthermore, this mode is also IND-CPA secure [7]. IND-CPA means semantic security against chosen-plaintext attack, namely, a ciphertext dose not leak any significant information about the plaintext And together with the authenticity of ciphertexts, this mode could be secure against stronger attack, namely, semantic security under chosen-ciphertext attack. Therefore, this mode can provide satisfactory security guarantees. Because the OCB mode is fully parallelizable, the computing the different ciphertext blocks can be done at the same time, it can be efficiently implemented in hardware and software.

Figure 5 [7] below shows the authenticated encryption algorithm of OCB mode. For the given nonce N and plaintext M, the output of the algorithm is the ciphertext and authentication tag $C \parallel T$.

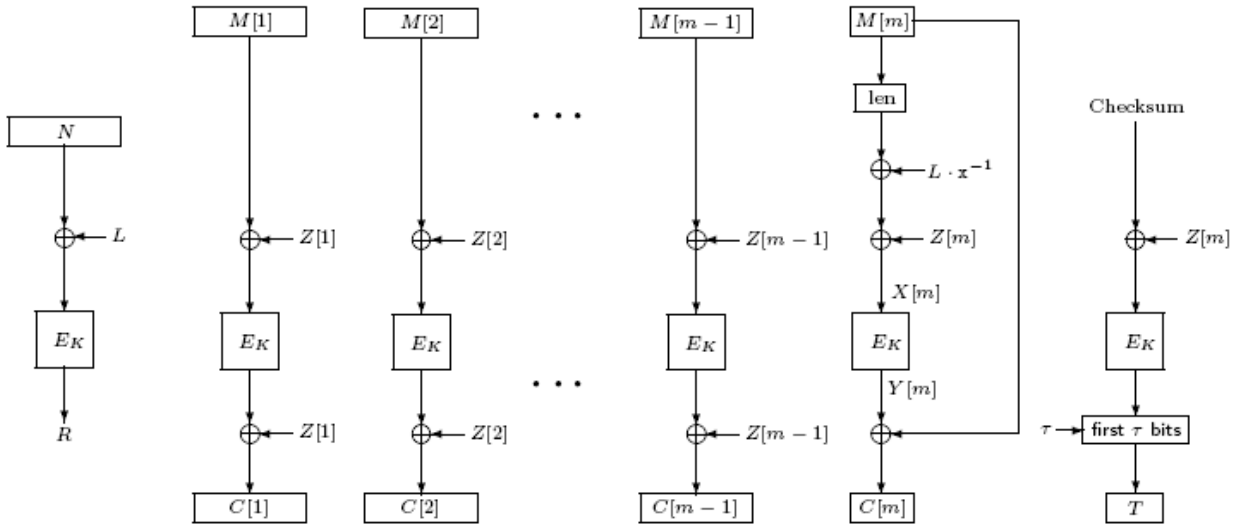

Figure 5: the authenticated encryption algorithm of OCB mode.

4.2 CWC Mode

The Cater-Wegman Counter (CWC) mode [9] is an AE mode, which uses the Counter mode and the Carter-Wegman universal hash function to provide confidentiality and authenticity respectively. The outstanding point is that this mode has five important properties: provable security, parallelizable, high performance in hardware, high performance in software, and free from intellectual property concerns.

The CWC mode has three mostly algorithms and accepts the block cipher, which have the size of 128, 192 and 256. For the given nonce and plaintext, the CWC-CTR algorithm encrypts the plaintext with the key and returns the ciphertext. For this ciphertext together with the nonce and additional authenticated data, the CWC-MAC algorithm generates the authentication tag with key, and uses the CWC-HASH algorithms as the underlying universal hash function [9]. The outputs are ciphertext and authentication tag. The decryption algorithm works similar to the encryption algorithm, the CWC-MAC algorithm generates the authentication tag again and compares it with the received authentication tag, if both are equally, using the CWC-CTR algorithm to decrypt and return the plaintext or else return *INVALID*.

CWC is a provably-secure AEAD mode assuming that the underlying block cipher (e.g., AES) is a secure pseudorandom function or permutation [9]. CWC can be efficiently implemented in hardware and software, because of using CTR mode to provide confidentiality, which is parallelizable. Since polynomial evaluation can be parallelized, Carter-Wegman universal hash function also can be efficiently implemented [9].

5. Comparison

In this part, all the four above named authenticated encryption modes are compared with each other from different aspects. There are some general properties for all the modes. Besides, each mode has some unique properties. These properties make them to be useful or suitable for different implementation environment. And security is naturally the key point for a mode, some of them is convincing, because they can provide provable security. The last aspect is performance, by the comparison shows which one has the best performance in hardware or software.

The table 2 above summarize the properties of four modes. It shows, all the four modes are authenticated encryption and use one block cipher key, this mechanism request for the memory resources to save the key. Besides the CCM* mode, the other three modes are parallelizable. This property is an important for obtaining good performance, if the blocks can be computed in parallel.
The authenticity of the ciphertext depends on the IV or the nonce in these modes, therefore it must be kept identical in encryption and decryption.
For Message Length Requirements, the OCB mode allows any bit string, it can be satisfied different applications. And the CCM* mode requests the message length must be a multiple of 16, it is certainly to restrict the range of application.
The OCB mode and the CWC mode are suitable for 128, 196 and 256-bit cipher block, it considers more further. These modes can keep on using for the new block ciphers with a large block size.

	GCM	**CCM***	**OCB**	**CWC**
Security Function	Authenticated encryption	Authenticated encryption	Authenticated encryption	Authenticated encryption

Synchronization	Same IV used by sender and recipient	Same nonce used by sender and recipient	Same nonce used by sender and recipient	Same nonce used by sender and recipient
Parallelizability	Encryption block level Authentication bit level	None	Fully parallelizable	Fully parallelizable
Keying Material Requirements	One block cipher key	One block cipher key	One block cipher key	One block cipher key
Message Length Requirements	Arbitrary message up to 2^{39}-256 bits Arbitrary additional authenticated data up to 2^{64} bits	Arbitrary message up to 2^{8L} bits, where L=2,...,8 Arbitrary additional authenticated data up to 2^{64} bits	Any bit string allowed	Arbitrary message up to 2^{39}-256 bits Arbitrary additional authenticated data up to 2^{39}-256 bits
Underlying Cipher Block Size Requirements	64, 128	Only 128	128, 192, 256	128, 192, 256

Table 2: Summary of the general properties of the four modes.

The table 3 below shows some unique properties of modes. The GCM mode can be used as stand-alone MAC and an incremental MAC, is unique among all of the modes. It is practical for the case, that only the authenticity is required. And the GCM mode is also on-line, it means the length of the plaintext and the additional authenticated data must not be known before on can be processed. There are many cases in practice, it requires to process the data without knowing it in advance.
The CCM* mode provides confidentiality-only services, it is useful for the case, that the data provided by an external mechanism. The allowing for variable-length authentication tags of the CCM mode cases some security risks, if the schemes are not carefully implemented. However, the CCM* mode solve well the problem, it can provide securely this service.

The OCB mode needs fewer block-ciphers calls while operating, this property is important when messages are shorts, it can happen in many cases, and it can certainly improve the efficiency.

	The Unique Properties
GCM	Can be used as stand-alone MAC Can used as an incremental MAC On-line Minimal circuit depth
CCM*	Provides confidentiality-only services Allows for variable-length authentication tags (from 32-bits to 128-bits)
OCB	Fewer block-ciphers calls
CWC	None

Table 3: Summary of some unique properties of the four modes.

The next aspect is security. The security of GCM mode stands on a single cryptographic conjecture:

the block cipher is assumed to be a secure pseudorandom permutation (PRP) [3]. The confidentiality and authenticity of GCM mode is secure against the chosen-plaintext attack [3]. The CCM* mode can archive the computational security.

The OCB mode and the CWC mode are the best, they both can provide provable security[8] [9]. Especially, the OCB mode is semantic security under chosen-plaintext attack. Because of using the nonce to provide the authenticity, the mode can be the security notion: non-malleability. It implies the mode is also secure against the stronger chosen-ciphertext attack.

The last aspect is performance. Because the CCM* mode can not efficiently implemented in hardware, it is not considered in comparison of hardware implementation.
The performance of the modes of operation of the AES-128 block cipher are compared. These modes are constructed to be pipelined. A simple model of network crypto module is chosen in order to analyze the performance of different modes under realistic conditions and the result is shown in the table 4 [3] below:

Bytes	**16**	**20**	**40**	**44**	**64**	**128**	**256**	**552**	**576**	**1024**	**1500**	**8192**	**IPI**
GCM	64.0	71.1	91.4	93.9	102	114	120	124	124	126	127	128	77.7
OCB	5.82	7.19	13.6	14.8	20.5	35.3	55.4	79.6	80.8	96.4	105	123	22.8
CWC	10.7	13.1	23.7	25.6	34.1	53.9	75.9	97.0	98.0	109	115	125	35.3

Table 4: Hardware performance in bits per clock cycle, with three significant digits, for a variety of packet sizes and the Internet Performance Index (IPI).

This table shows that the GCM mode performs best among them. IPI means the performance of throughput on Internet data. On the IPI, GCM is about twice better than CWC and about three times better than OCB. For short packet size, GCM has the obvious advantage, and the almost same performance for the large packet size. The OCB mode is more efficient than the CWC mode.

For the software implementation, the AES-128 is also chosen as the block cipher. Because the GCM mode is implemented by table-driven method, the four variants of this method is are listed, which ask for the different storage. The result shows in the table 5 [2] below:

Mode	**Message size (bytes)**				
	16	64	256	1024	8192
GCM, 64Kb storage	60.8	44.8	36.1	36.6	38.1
GCM, 8Kb storage	89.9	51.9	42.9	43.0	40.1
GCM, 4Kb storage	118	69.1	46.5	54.1	53.5
GCM, 256b storage	179	108	89.5	85.4	84.6
CCM	159	75.6	54.5	49.2	47.6
OCB	89.4	43.3	31.4	29.3	29.0
CWC	227	102	72.7	63.3	61.2

Table 5: Software performance for various different AES modes of operation, expressed in clock cycles per byte.

This table shows that all modes need more clock cycles per byte for short message size than for

large message size. The GCM with 64Kb storage is the most efficient one among all the GCM modes. The OCB is the greatest one, whatever for short messages or large messages. The GCM mode performs also greater with 64Kb storage. And for large messages, the GCM mode 8Kb storage is only 4/3 time slower than the OCB mode. Furthermore, the CCM and CWC mode are slower, especially for short messages.

6. Conclusion

In this thesis, the four authenticated encryption modes GCM, CCM*, OCB and CWC are presented. All of the modes can provide confidentiality and authenticity simultaneously by combining of two systems. The GCM mode uses Counter mode and universal hashing over a binary Galois filed. The CCM* mode is combination of two well-known modes: Counter mode and CBC mode. The OCB mode integrates a MAC into a block cipher. The CWC mode uses also Counter mode together with Carter-Wegman Counter mode. But the CCM* mode works still separately, the other modes mix the two systems in an algorithm.

The Counter mode seems to be a popular choice for proving confidentiality, because three modes above have chosen it. However, the Counter mode can be efficiently pipelined in hardware implementation, it is an important point for obtaining a good performance. Another important is parallelizability. CCM* uses CBC mode to to provide authenticity, but it can not be pipelined or parallelizable so that the CCM* mode can be efficiently implemented.

The efficiency is a very important factor for the mode in practice. As in the tables above shown, the GCM mode and the OCB mode are the most efficient modes, they both have a good performance not only in hardware implementation, but also in software implementation. Furthermore, the OCB mode can provide the strong security property, and the GCM mode can provide some additional properties, which can be suitable for different application. The two modes seem to be the good choice amongst these modes.

References

[1] Morris Dworkin, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D, November, 2007.

[2] David A. McGrew, John Viega, The Galois/Counter Mode of Operation (GCM), Submission to NIST Modes of Operation Process, May 31, 2005.

[3] David A. McGrew, John Viega, The Security and Performance of the Galois/Counter Mode (GCM) of Operation, Progress in Cryptology-INDOCRYPT, Springer-Verlag, 2004.

[4] René Struik, Formal Specification of the CCM* Mode of Operation, IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs), September 9, 2005.

[5] Morris Dworkin, Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, NIST Special Publication 800-38C, July 20, 2007.

[6] Jakob Jonsson, On the Security of CTR + CBC-MAC, NIST Modes of Operation - Additional CCM Documentation, Selected Areas of Cryptography (SAC) 2002, Springer-Verlag, 2003..

[7] Phillip Rogaway, OCB Mode: Proposal to NIST for a block-cipher mode of operation which simultaneously provides privacy and authenticity, 2nd NIST Modes of Operation Workshop, April 2001.

[8] Phillip Rogaway, OCB Mode: Parallelizable Authenticated Encryption, Comments to NIST concerning AES Modes of Operation, Draft October 16 2000.

[9] Tadayoshi Kohno, John Viegay, Doug Whitingz, CWC: A high-performance conventional authenticated encryption mode, Proceedings of FSE 2004, LNCS 3017, pp. 408-426, Springer-Verlag, 2004.

[10] Emmanuel Lopez-Trejo, Francisco Rodriguez-Henriquez, Arturo Diaz-PerezAn, Efficient FPGA implementation of CCM mode using AES, International Conference on Information Security and Cryptology, volume 3935 of Lecture Notes in Computer Science, pages 208–215, Seoul, Korea, Springer-Verlag, December 2005..