CCM-R: Secure Counter Synchronization for IoT Wireless Link

Roy, Upal; Yin, Jiachen; Andersen, Birger

Published in:
Proceedings of the Global Wireless Summit

Publication date:
2016

Document Version
Peer reviewed version

CCM-R: Secure Counter Synchronization for IoT Wireless Link

Upal Roy
DTU Compute
Technical University of Denmark
Copenhagen, Denmark
Email: s127819@student.dtu.dk

Jiachen Yin
DTU Compute
Technical University of Denmark
Copenhagen, Denmark
Email: s127628@student.dtu.dk

Birger Andersen
DTU Diplom
Technical University of Denmark
Copenhagen, Denmark
Email: birad@dtu.dk

***Abstract*—In this paper we propose and evaluate a new version of the CCM mode of operation, CCM-R, which is an extended and alternate version of the original CBC-MAC with Counter Mode(CCM) that was created to address the problem of counter synchronization. While CCM is considered secure when used/implemented correctly, it is still vulnerable to Denial of Service (DoS) attacks where packages in the stream are removed or modified with the purpose of destabilizing the synchronization of the counter states. Another possible way it can be destabilised is through random loss of packets due to noise or weak signals on wireless link. Therefore, in order to solve this problem, we have proposed a solution where we have embedded a secure counter synchronization approach into the original CCM mode of operation. The approach makes it immune to package loss and DoS attacks. The overhead is also kept at a minimum, making it suitable for low-power wireless IoT devices.**

***Keywords*—CCM-R; IoT wireless link; CCM; counter synchronization; Network Security.**

I. Introduction

CCM (CBC-MAC with CTR mode) was created by Niels Ferguson, Doug Whiting and Russ Housely [3]. Concern was the main issues of wireless IoT devices, such as data link issues where messages are usually small (need low overhead) and energy efficiency.

Often the messages are not retransmitted in cases where messages are lost or received with errors in order to save energy and not to use more bandwidth. In our case (with our protocol stack developed with wireless weather stations in mind), the messages lost or received not in order will be retransmitted three more times and then withdrawn in cases messages are still not received correctly.

At this time, many points of contention came up in the sense that while implementing the RFC version of CCM mode would definitely work it would lead to problems in case the recipient loses track of the counter and would not be able to authenticate and decrypt the message. According to these issues, it is very urgent to have efficient counter synchronization and therefore we modified and extended CCM to fit in with our purpose and to actually customize the code so that it would function best with our situation. At the same time, we carefully considered the possible attacks against counter synchronization that could lead to, 1) counter would become unknown, 2) synchronization would fail (DoS attack) and/or, 3) a battery powered device would use up the power during many synchronization attempts (power drain attack). The security concerns of our proposed solution will also be then discussed. Finally, we want to introduce as little overhead as possible by only adding few bits/bytes to messages, which can be important in a battery powered domain.

II. Environment

We are here describing hardware and software used. The most important software that was used to build and run application with protocol stack is MPLABX [11], which was used to code and build the software directly to the hardware. The connection kit we used is the PICkit connector [12], which allowed us to build and execute the program by using MPLABX. We were using MPLABX 3.10 to code and program a 16-bit PIC micro-controller, which is made by microchip complied by using a XC16 complier. For testing purpose we had circuit boards of a weather station and an IoT gateway both with wireless transceivers with PIC microcontrollers. More precisely, our hardware was a RF1 module, which consists of three individual printed circuit boards (PCBs), a 9V battery and a switch, as well as a RF2 module that consists of two 1.5V batteries, a PCB with serial connector and a switch. RF1 was for weather station while RF2 was for IoT gateway.

III. Algorithm

The CCM-R algorithm consists of three essential parts: the authentication algorithm, the encryption algorithm, and the random number generator.

A. Authentication

The authentication part is one of the most important parts because it ensures that the message has not been modified by any means and that the message is from expected source. The first step is to compute the authentication tag T. This acts as the MAC tag. This is done by using CBC-MAC. First a sequence of blocks has to be defined, B_0, B_1, ... , B_n and then the CBC encryption operation is run on these. CBC-

MAC uses an initialization vector(IV) of 0. The first plain-text block is XOR'ed with this IV and this result is then encrypted with the key to form the cipher-text for the first block. This continues till the result of the second to last block i.e. the cipher text of it is XOR'ed with the last block plain-text and then this is encrypted with the Key to form the cipher-text of the last block. The most significant bytes or MSB of the cipher-text of the last block is taken. Assuming blocks B_0, B_1, ... , B_n are defined, then the first ciphertext block, X_1 can be represented as:

$$X_1 := E_{K,\ B_0} \oplus IV \tag{1}$$

Based on that the final cipher text block can be represented as:

$$X_n := EK,\ B_i \oplus X_{i-1} \tag{2}$$

Thus the authentication field value or the MAC tag is defined as:

$$T := MSB(X_n) \tag{3}$$

B. Encryption

For encryption we are using counter mode. Counter mode uses a nonce and counter as input generally where there is a fixed nonce and the counter is updated using the nonce as a base. But in our case we use CTR_0 = nonce and as the counter updates by 1, CTR_1 just becomes the "nonce +1" and CTR2 as "nonce + 2". The counter is generally a fixed value throughout an entire session. For our implementation we used a variable and secret 8-byte counter which changes for each packet being sent. The counter is generated using a true random number generator. This would increase the security since using a fixed nonce with counter generates sequences which with enough time and computation power can be broken. Another advantage of generating our own randomized counter is that since we are sending it along with the packet there will never be any counter synchronization problems. If packages are lost the counter would become unsynchronized and cause incorrect authentication and decryption of the packet. This is the major advantage of our implementation as CCM-R in the sense that it completely takes out the need for maintaining synchronization between the counters on the receiving and sending ends using additional synchronization messages. Due to the entire counter being sent the only thing that needs to be taken into account is that the counter generating sequence is the same, i.e. $counter_{0+1}$, $counter_{0+2}$ and so on is the same on both sides which is programmatically very simple. The first CTR or CTR_0 is generated using a true random number generator function. This counter is then encrypted with the secret key and the result is then XOR'ed with the plain-text first block. The next keystream is block is generated by encrypting successive values of the counter. So for the final block to be encrypted the final counter which is $CTR_{0 + \text{number of blocks encrypted}}$ is encrypted with the key and the result is then XOR'ed with the plaintext.

For the first block being encrypted, it is represented by:

$$X_1 := E_{K,\ CTR_0} \oplus B_0 \tag{4}$$

And therefore the final block to be encrypted is:

$$X_i = E_{K,\ CTR_i} \oplus B_i \tag{5}$$

C. Output

The output which is to be sent is then the encrypted message from X_1 to X_i in counter mode which we dub as "m", the MAC tag and the CTR0. While the cipher text is secure enough to be sent as is, the MAC tag and the counter is then encrypted with AES-128 [2] to provide added security since this is impossible to crack as of now. To note is that the MAC tag and the CTR_0 are each 8 bytes in length to fit in with CCM mode restrictions.

D. Decryption

For decryption the exact opposite needs to be taken place. Using AES-128 the CTR0 and MAC tag are decrypted. The CTR_0 along with the key is passed through the encryption cipher and the result is XOR'ed with the first cipher text block and the plaintext of the first block is the result. Same is repeated till the last block, where the final counter which is basically counter + number of blocks is passed through the encryption cipher and the result XOR'ed with the last cipher text block to produce the plaintext.

E. Frame Format

Although not being part of CCM-R, we have present its implementation at the link layer in details. Some parts of frame header was already defined by the protocol in previous version where AES-CBC was used but with only encrypted CRC fields for protecting integrity.

Figure 1 shows the wireless link frame format while packages are transmitting. It starts with one-byte value identifying the packet length, and 4-byte value for the address from the destination, a 2-byte CRC16 value, and then, the AES-CTR mode is used for encrypt the set of 4-byte source address, 1-byte command, status value along with the data that needs to be transferred. The initial counter value as well as the CBC-MAC Tag value are combined into one block of 16-byte encrypted by AES-ECB mode transferred along with the entire frame.

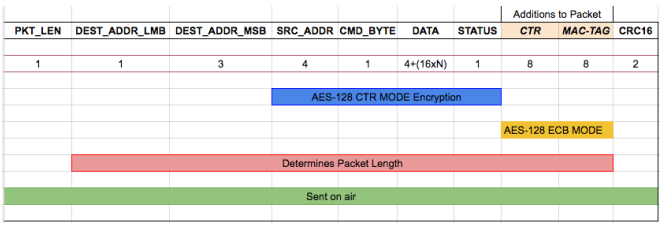

Fig. 1. Wireless Link Frame Format

IV. Evaluation

Our implementation of CCM-R version deviates from the RFC version considerably. Also concerning of the issues of what a wireless link would occur as well as the importance to solve the counter synchronization problem to maintain the security of the system, a sequence of packets lost and receiver does not know about the sequence number of the counter and which counter will be the next one, and also an attack could remove or change a long sequence of packets, e.g., DoS attack. Both of the cases discussed above could have terminated the network communication and cause threats to the system for instance. In such a situation, one counter synchronization algorithm would not help a lot as for instance the attacker could just simply remove or change a long sequence of packets once again. Therefore, in order to maintain the security level of the RFC CCM version and also improve it by solving the counter synchronization problem, CCM-R comes to the place. Figure 2 shows the basic overview

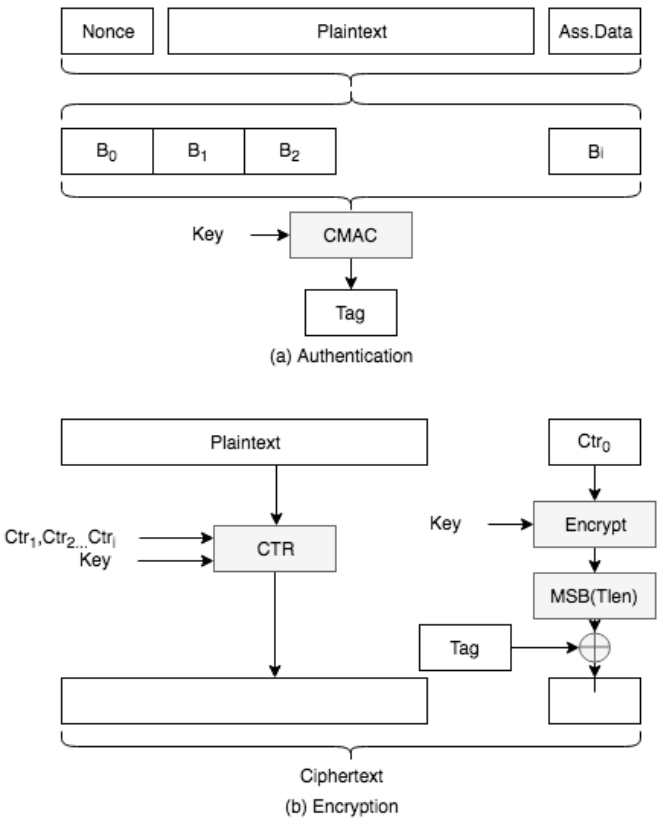

Fig. 2. CCM Structure and Encryption/Decryption

of the RFC CCM version. The diagram also describes the entire encryption and authentication happening and also the results, which are being sent. Figure 2 shows CCM-R, where the packet structure can be easily seen. We put the 8-byte initial counter, generated by using a true random number generator, for the AES-CTR mode and 8-byte CBC-MAC tag value as one block encrypted and decrypted by AES-ECB when transfer the packet. We are using the tag value to check if the packet is sent correctly and send the initial

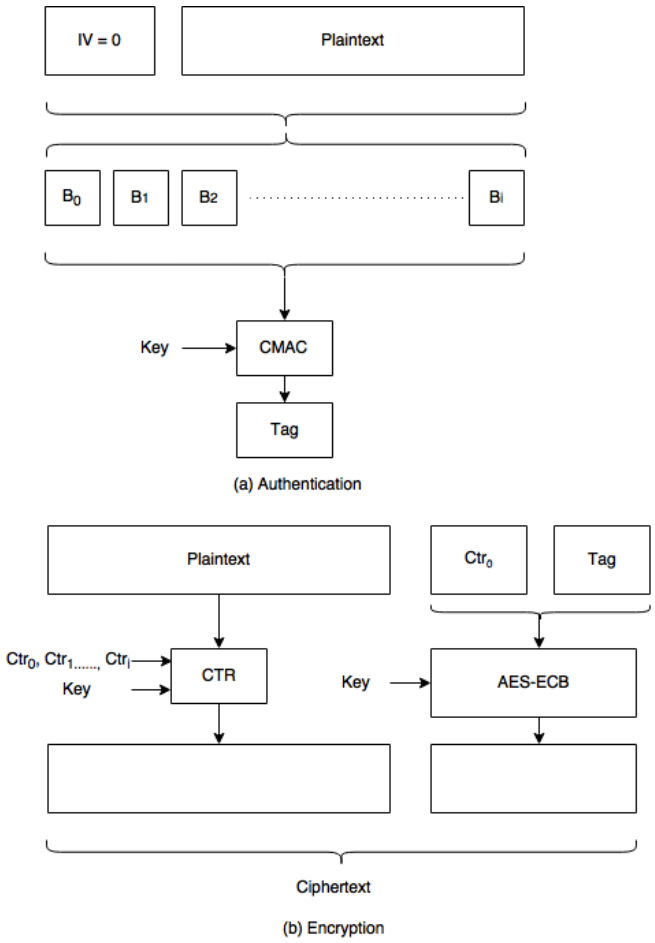

Fig. 3. CCM-R Structure and Encryption/Decryption

counter for decrypt the packet in AES-CTR mode to tell the packet sequence number. As we can see here, the difference we make here from the RFC CCM version is that we take out the initial counter and send it along with the CBC-MAC tag value as one 16 bytes block by using AES-ECB to encrypt, which can also help CCM-R to solve the counter synchronization problem that exists in the RFC CCM version, where if the packet is not completely transferred, the counter synchronization will be messed up. Since using this strategy we solved this problem by sending the encrypted initial counter. There are several test cases are made in order to make sure there are no such security holes in the system that can be exploited by attackers to potentially steal the data or cause irreparable changes to it, and also how is it better than the current version. The tests are all on CCM-R by comparing results on before encrypt and after decrypted, also changing some parts of the data to see how CCM-R deals with it and by testing for a weaker signal that leads to more packet loss.

Concerning the possible attacks against CCM-R, such as DoS and analyzing messages. A DoS attack, which is cyber-attack that the attackers is attempting to make the users not able to access the information or service is immune by CCM-

R since the message itself is self-contained. Also to be noted, the counter value is randomized with an 8-byte value, for instance the counter might be used for a second time or maybe even third time. CCM-R is also very secure even with this issue, because the counter is secret, which is encrypted sent along with the Tag by using AES-ECB. It is very rare to happen that both of the Tag and counter will be the same again, and we believe that CCM-R is enough secure at the current level of the network and at least at the same security level as the original CCM. Therefore, CCM-R should provide the following security services:

- Confidentiality, which ensures only authorized parties can access the information.
- Authentication, which provides proof of authority of the user.
- Integrity, which ensures changes can be detected.
- Access control in relations with layer management.

Theoretically, the meet-in-the-middle attacks can be used to limit the key size of $2^{k/2}$ operations, where k is the size of the key in bits. Due to CCM-R still remains the same key size as CCM, it is secure enough against the attacks to 2^{64} steps of operations.

V. Experiments

As mentioned in the previous section, there are several tests being made. In order to prove that CCM-R works properly, we therefore executed the program with the CCM-R cipher on the software simulators along with two RF modules to compare the inputs and outputs, which lead to great success after several attempts. For the security manner of CCM-R, we decided to test a bit more on the algorithm and see how it reacted to protect the data packet. Therefore, we decided first to make the signal of the packet transfer between the gateway and the simulator for the packet sender weaker by using attenuator, with the purpose to take into account of reality that there may be some jamming of networks leading weaker signal or lead to more packet loss.

By testing this, the same equipment was then being used with an attenuator to change the strength of the signal. The CBC-MAC tags for authentication of the messages from both ends keep match each other until the weakest point of the attenuator can be changed to. When the signal is very weak, the packet cannot be received, and therefore we decided to drop the packet that cannot be received after three times attempts, and for testing purpose the interval of the attempts was set to 3 seconds, but it can be changed to have longer time interval, for instance 20 seconds for reality. There is another test for CCM-R, which is to deliberately change of the packet structure and the initial randomized counter, on the purpose of where the hackers tries to attack and modify the packet or the generated random counter. By doing the tests, we could make sure that by solving counter synchronization problem by CCM-R, it is also being more secure. As has been explored, we believe that CCM-R is secure enough and also authenticated.

VI. Conclusion

In this paper we have presented a simpler solution to the counter synchronization issue faced with the use of CTR mode. We have discussed the implementation as well as the tests we have performed to ensure the working and security of the algorithm. CCM in the original RFC version matches well with the CIA (Confidentiality, Integrity and Availability) criteria and was a very useful algorithm to use. Regarding to the pivotal issue of synchronization of counters as well as the possible attacks against CCM-R such as DoS and analyzing messages, the security level of the CCM-R has been improved by solving the counter synchronization problem as well as the solutions against those possible attacks. In spite of this we believe due to the simplicity of solving the counter synchronization issue and also maintain security level it could be used in other real life scenarios practically to complement the original CCM.

VII. Related Works

CCM was originally designed by D. Whiting, R. Housely, and N. Ferguson [3]. CCM is considered secure when used correctly but does not provide a solution to the counter synchronization problem. In case an unknown number of packages have been lost due to link layer failures or DoS attack, recipient will fail to decrypt because the counter values are no more in sequence. Solutions where counters are displayed in clear-text like in WiFi are in general unsafe or at least risky.

One approach to counter synchronization is taken in the SPINS protocol [7]. Here counters are agreed upon and synchronized by means of plaintext messages, meaning that the counter value is not secret and therefore the counter (assumed never reused) only guarantees freshness and prevents against reply attacks. However, it was suggested to send the counter encrypted with each message in order to protect against counter synchronization DoS attacks, but this approach was not further developed.

Zigbee [8] has been based on the SPINS approach but does not include encrypted counter values. IP ESP [9] takes a different approach and uses a window where a limited number of succeeding counters are tried at recipient side until successful authentication (including integrity validation). Counters are never transmitted as plaintext. Start value is always zero. IP ESP will fail in case the resynchronization fails. Therefore, a DoS attach against a number of succeeding packages can efficiently interrupt an IP ESP session. Also power drain might be consequence.

MiniSec [10] uses the time as replacement for the counter and messages are accepted at recipient whenever their timestamps (included with the packages) are within the allowed window. This approach is exactly opposite to SPINS which

is not transmitting the counter value. MiniSec implements weak freshness as two messages with same timestamp will be accepted if they arrive within the window. A message with older timestamp than the previous will not be accepted. SPINS and IP ESP provides strong freshness (another message with same counter will never get accepted).

ACKNOWLEDGMENT

This research was partly funded by the FP7 EU FRAMEWORK PROGRAMME under grant agreement No. 604659(project CLAFIS) [1].

REFERENCES

[1] CLAFIS-Project, *Crop, Livestock and Forests Integrated System for Intelligent Automation*, http://www.clafis-project.eu/ , Website

[2] Joan Daemen and Vincent Rijmen, *The Design of Rijndael: AES-The Advanced Encryption Standard*, Dated: 2002, Springer, Journal Submission

[3] D. Whiting, R. Housley, N. Ferguson, *Counter with CBC-MAC*, RFC, https://tools.ietf.org/html/rfc3610, RFC/Website

[4] Jakob Jonsson, *On the Security of CTR+CBC-MAC*, RSA Laboratory Stockholm.

[5] Pierre-Alain Fouque, Gwenalle Martinet, Frederic Valette and Sebastien Zimmer , *On the Security of the CCM Encryption Mode and of a Slight Variant*, Ecole Normale Superieure/DCSSI Crypto Lab/CELAR, http://www.di.ens.fr/ fouque/pub/acns08.pdf, Journal/Paper/Website

[6] William Stallings, 'Network Security Essentials Applications and Standards, Fourth Edition', 2011, Textbook

[7] Adrian Perrig, Robert Szewczyk, J.D. Tygar, Victor Wen and David E. Culler,*SPINS: Security Protocols for Sensor Networks*,Wireless Networks 8, 521?534, Kluwer Academic Publishers 2002.

[8] Paolo Baronti, Prashant Pillai, Vince W.C. Chook, Stefano Chessa, Alberto Gotta, Y. Fun Hu, *Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards*,Computer Communications 30 (2007) 1655?1695, Elsevier 2007.

[9] Stephen Kent, RFC 4303 IP Encapsulating Security Payload (ESP), IETF 2005.

[10] Mark Luk, Ghita Mezzour, Adrian Perrig, Virgil Gligor, *MiniSec: A Secure Sensor Network Communication Architecture*,IPSN?07, April 25-27, 2007, Cambridge, Massachusetts, USA, ACM 2007.

[11] MPLAB-X IDE, http://www.microchip.com/mplab/mplab-x-ide

[12] PIC KIT, www.microchip.com/pickit3