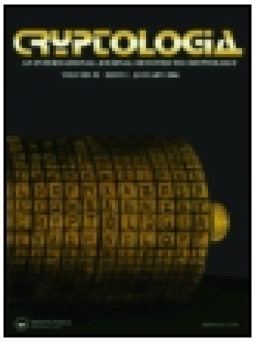
The offset codebook (OCB) block cipher mode of operation for authenticated encryption

Article *in* Cryptologia · February 2018

DOI: 10.1080/01611194.2017.1422048

William Stallings
independent consultant



Cryptologia

ISSN: 0161-1194 (Print) 1558-1586 (Online) Journal homepage: http://www.tandfonline.com/loi/ucry20

The offset codebook (OCB) block cipher mode of operation for authenticated encryption

William Stallings

To cite this article: William Stallings (2018): The offset codebook (OCB) block cipher mode of operation for authenticated encryption, Cryptologia, DOI: 10.1080/01611194.2017.1422048

To link to this article: https://doi.org/10.1080/01611194.2017.1422048

Published online: 01 Feb 2018.

CRYPTOLOGIA
https://doi.org/10.1080/01611194.2017.1422048

Taylor & Francis
Taylor & Francis Group

The offset codebook (OCB) block cipher mode of operation for authenticated encryption

William Stallings

ABSTRACT
This article presents an overview of the concepts of and motivation for the OCB block cipher mode of operation. OCB is well suited for IoT, wireless, and other constrained devices where processing time and energy consumption are design issues. The article describes two versions of the OCB algorithm (OCB1 and OCB3) that have been widely accepted.

KEYWORDS
authenticated encryption; block cipher; modes of operation; OCB; offset codebook; symmetric cipher

Introduction

Authenticated encryption (AE) is a term used to describe encryption systems that simultaneously protect confidentiality and authenticity (integrity) of communications; that is, AE provides both message encryption and message authentication of a plaintext block or message (Stallings 2017). Many applications and protocols require both forms of security, but until relatively recently the two services have been designed separately. AE is implemented using a block cipher mode structure. For example, the National Institute of Standards and Technology (NIST) has standardized several AE modes of operation, including the Counter with Cipher Block Chaining-Message Authentication Code (CCM) and the Galois/Counter (GCM) (Stallings 2010). CCM is defined in NIST SP 800-38C (NIST 2007a), and GCM is defined in NIST SP 800-38D (NIST 2007b).

In this article, we examine a new mode, the Offset Codebook (OCB), which is gaining wide acceptance. Two versions are in use: OCB1 and OCB3. The initial version, OCB1, was designed to provide an extremely efficient algorithm, equal to or more efficient than other AE algorithms and comparable in performance to the most efficient block ciphers that provided only encryption (Rogaway, Bellare, and Black 2003). OCB1 is an NIST proposed block cipher mode of operation (Rogaway et al. 2001). OCB1 is listed as an optional AE technique in the IEEE 802.11 wireless LAN standard as an alternative to CCM (Walker 2005). OCB1 is included in MiniSec, the open source IoT security module (Luk et al. 2007). OCB3 makes some modest improvements in efficiency and adds the ability to handle associated data

CONTACT William Stallings ✉ ws@shore.net Independent Consultant, P. O. Box 2405, Brewster, MA 02631, USA.

Color versions of one or more of the figures in the article can be found online at www.tandfonline.com/ucry.

(Rogaway et al. 2001). Associated data refers to data, such as a message header, that must be authenticated but should not be encrypted. OCB3 is a proposed Internet Standard defined in RFC 7253 (Krovetz and Rogaway 2014).

A mode of operation must be specified when a plaintext source consists of multiple blocks of data to be encrypted with the same encryption key. OCB is provably secure assuming the underlying block cipher is secure. OCB is a one-pass mode of operation, making it highly efficient. Only one block cipher call is necessary for each plaintext block (with an additional one or two calls needed to complete the whole encryption process). OCB is especially well suited for the stringent energy constraints of sensor nodes.

A feature that contributes significantly to the efficiency of OCB is that with one pass through the sequence of plaintext blocks, it produces a ciphertext of equal length and a tag for authentication. To decrypt a ciphertext, the receiver performs the reverse process to recover the plaintext. Then, the receiver ensures that the tag is as expected. If the receiver computes a different tag than the one accompanying the ciphertext, the ciphertext is considered to be invalid. Thus, both message authentication and message confidentiality are achieved with a single, simple algorithm.

OCB1

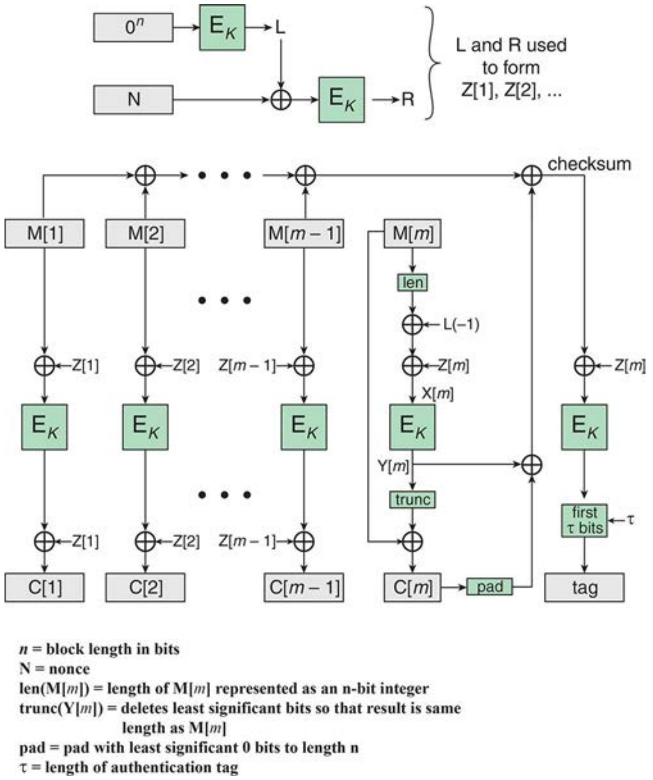
Figure 1 shows the overall structure for OCB encryption and authentication. Typically, AES is used as the encryption algorithm. The message M to be encrypted and authenticated is divided into n-bit blocks, with the exception of the last block, which may be less than n bits. Typically, $n = 128$. Only a single pass through the message is required to generate both the ciphertext and the authentication code. The total number of blocks is $m = \lceil \text{len}(M)/n \rceil$.

Note that the encryption structure for OCB is similar to that of electronic codebook (ECB) mode. Each block is encrypted independently of the other blocks, so that it is possible to perform all m encryptions simultaneously. With ECB, if the same b-bit block of plaintext appears more than once in the message, it always produces the same ciphertext. Because of this, for lengthy messages, the ECB mode may not be secure. OCB eliminates this property by using an offset $\mathrm{Z}[i]$ for each block $\mathrm{M}[i]$, such that each $\mathrm{Z}[i]$ is unique; the offset is XORed with the plaintext and XORed again with the encrypted output. Thus, with encryption key K, we have

$$\mathrm{C}[i] = \mathrm{E}_K(\mathrm{M}[i] \oplus \mathrm{Z}[i]) \oplus \mathrm{Z}[i]$$

where $\mathrm{E}_K(X)$ is the encryption of plaintext X using key K, and $\oplus$ is the bitwise exclusive-OR operation. Because of the use of the offset, two blocks in the same message that are identical will produce two different ciphertexts.

The upper part of Figure 1 indicates the inputs used to generate the $\mathrm{Z}[i]$. An arbitrary n-bit value N called the nonce is chosen; the only requirement



n = block length in bits
N = nonce
len(M[*m*]) = length of M[*m*] represented as an n-bit integer
trunc(Y[*m*]) = deletes least significant bits so that result is same length as M[*m*]
pad = pad with least significant 0 bits to length n
τ = length of authentication tag

Figure 1. OCB1 encryption and authentication.

is that if multiple messages are encrypted with the same key, a different nonce must be used each time such that each nonce is only used once. Each different value of N will produce a different set of Z[i]. Thus, if two different messages have identical blocks in the same position in the message, they will produce different ciphertexts because the Z[i] will be different.

The calculation of the Z[i] is somewhat complex and is summarized in the following equations:

$$
\begin{aligned}
&\mathrm{L}(0) = L = \mathrm{E}_K(0^n) && \text{where } 0^n \text{ is consists of n zero bits.}\\
&\mathrm{R} = \mathrm{E}_K(\mathrm{N} \oplus \mathrm{L}) &&\\
&\mathrm{L}(i) = 2 \cdot \mathrm{L}(i-1) && 1 \leq \mathrm{i} \leq m\\
&\mathrm{Z}[1] = \mathrm{L} \oplus \mathrm{R} &&\\
&\mathrm{Z}[i] = \mathrm{Z}(i-1) \oplus \mathrm{L}(\mathrm{ntz}(i)) && 1 \leq \mathrm{i} \leq m
\end{aligned}
$$

The operator · refers to multiplication over the finite field GF(2^n). The only multiplication that is done is doubling. OCB1 uses GF(2^{128}), defined with the irreducible polynomial $m(x) = x^{128} + x^7 + x^2 + 1$ (see the Appendix for a brief introduction to finite fields).

The operator ntz(i) denotes the number of trailing (least significant) zeros in i. Equivalently, ntz(i) is the largest integer z such that 2^z divides i. The resulting Z[i] are the offsets used in the encryption.

Thus, the values Z[i] are a function of both the nonce and the encryption key. The nonce must not be kept secret and is communicated to the recipient in a manner outside the scope of the specification. For a given K and N, the L[i] can be computed first and saved in a table, so that computing L(ntz(i)) is a simple table lookup.

Note that the Z[i] can all be computed prior to encryption. Indeed, if a system has some occasional idle processing power, multiple sets of Z[i] for different values of the nonce can be precomputed and ready to use as messages become available for AE.

Because the length of M may not be an integer multiple of n, the final block is treated differently, as shown in Figure 1. The length of M[m], represented as an n-bit integer, is used to calculate $\mathrm{X}[m] = \mathrm{len}(\mathrm{M}[m]) \oplus \mathrm{L}(-1) \oplus \mathrm{Z}[m]$. L(−1) is defined as L/2 over the finite field or, equivalently, $\mathrm{L} \cdot 2^{-1}$. Next, $\mathrm{Y}[m] = \mathrm{E}_K(\mathrm{X}[m])$. Then, Y[$m$] is truncated to len(M[$m$]) bits (by deleting the necessary number of least significant bits) and XORed with M[m]. Thus, the final ciphertext C is the same length as the original plaintext M.

A checksum is produced from the message M as follows:

$$\text{checksum} = \mathrm{M}[1] \oplus \mathrm{M}[2] \oplus \ldots \oplus \mathrm{Y}[\mathrm{m}] \oplus \mathrm{C}[\mathrm{m}]0*$$

where C[m]0* consists of C[m] padded with least significant bits to the length n. Finally, an authentication tag of length τ is generated, using the same key as is used for encryption:

$$\text{tag} = \text{first } \tau \text{ bits of } \mathrm{E}_K(\text{checksum} \oplus \mathrm{Z}[m])$$

The bit length τ of the tag varies according to the application. The size of the tag controls the level of authentication. To verify the authentication tag, the decryptor can recompute the checksum, then recompute the tag, and finally check that it is equal to the one that was sent. If the ciphertext passes the test, then OCB produces the plaintext normally.

Figure 2 summarizes the OCB algorithms for encryption and decryption. It is easy to see that decryption is the inverse of encryption. We have

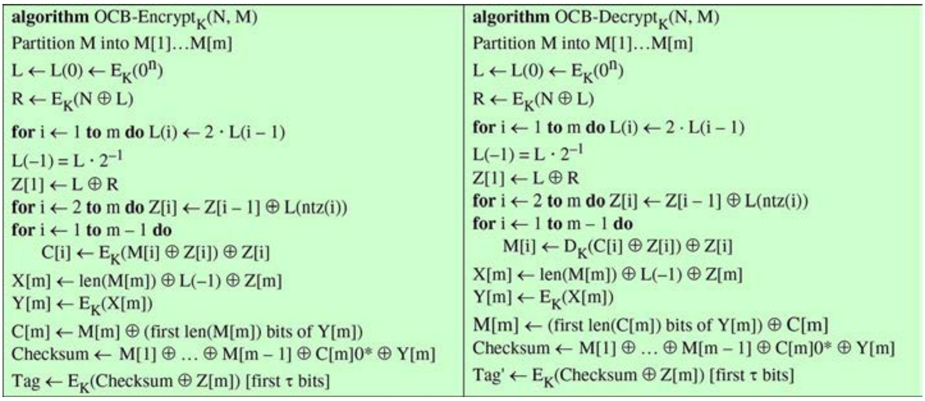
$$\mathrm{E}_K(\mathrm{M}[i] \oplus \mathrm{Z}[i]) \oplus \mathrm{Z}[i] = \mathrm{C}[i]$$

$$\mathrm{E}_K(\mathrm{M}[i] \oplus \mathrm{Z}[i]) = \mathrm{C}[i] \oplus \mathrm{Z}[i]$$

$$\mathrm{D}_K(\mathrm{E}_K(\mathrm{M}[i] \oplus \mathrm{Z}[i])) = \mathrm{D}_K(\mathrm{C}[i] \oplus \mathrm{Z}[i])$$

$$\mathrm{M}[i] \oplus \mathrm{Z}[i] = \mathrm{D}_K(\mathrm{C}[i] \oplus \mathrm{Z}[i])$$

$$\mathrm{M}[i] = \mathrm{D}_K(\mathrm{C}[i] \oplus \mathrm{Z}[i]) \oplus \mathrm{Z}[i]$$



```
algorithm OCB-Encrypt$_K$(N, M)
Partition M into M[1]...M[m]
$L \leftarrow L(0) \leftarrow E_K(0^n)$
$R \leftarrow E_K(N \oplus L)$
for i ← 1 to m do $L(i) \leftarrow 2 \cdot L(i-1)$
$L(-1) = L \cdot 2^{-1}$
$Z[1] \leftarrow L \oplus R$
for i ← 2 to m do $Z[i] \leftarrow Z[i-1] \oplus L(ntz(i))$
for i ← 1 to m – 1 do
    $C[i] \leftarrow E_K(M[i] \oplus Z[i]) \oplus Z[i]$
$X[m] \leftarrow len(M[m]) \oplus L(-1) \oplus Z[m]$
$Y[m] \leftarrow E_K(X[m])$
$C[m] \leftarrow M[m] \oplus$ (first len(M[m]) bits of Y[m])
$Checksum \leftarrow M[1] \oplus \ldots \oplus M[m-1] \oplus C[m]0^* \oplus Y[m]$
$Tag \leftarrow E_K(Checksum \oplus Z[m])$ [first τ bits]
```

```
algorithm OCB-Decrypt$_K$(N, M)
Partition M into M[1]...M[m]
$L \leftarrow L(0) \leftarrow E_K(0^n)$
$R \leftarrow E_K(N \oplus L)$
for i ← 1 to m do $L(i) \leftarrow 2 \cdot L(i-1)$
$L(-1) = L \cdot 2^{-1}$
$Z[1] \leftarrow L \oplus R$
for i ← 2 to m do $Z[i] \leftarrow Z[i-1] \oplus L(ntz(i))$
for i ← 1 to m – 1 do
    $M[i] \leftarrow D_K(C[i] \oplus Z[i]) \oplus Z[i]$
$X[m] \leftarrow len(M[m]) \oplus L(-1) \oplus Z[m]$
$Y[m] \leftarrow E_K(X[m])$
$M[m] \leftarrow$ (first len(C[m]) bits of Y[m]) $\oplus C[m]$
$Checksum \leftarrow M[1] \oplus \ldots \oplus M[m-1] \oplus C[m]0^* \oplus Y[m]$
$Tag' \leftarrow E_K(Checksum \oplus Z[m])$ [first τ bits]
```

Figure 2. OCB1 algorithms.

OCB3

The are several changes in going from OCB1 to OCB3, which can be summarized as follows:

1. OCB1 uses $B + 2$ block cipher calls to encrypt a message of $B = \lceil |M|/128 \rceil$ blocks. As shown subsequently, the average number of block cipher calls for OCB3 encryption is $B + 1.016$, yielding a slight savings.
2. OCB3 has the potential to remove a pipeline stall prior to the calculation of the checksum that is present in OCB1.
3. OCB3 includes the ability to incorporate associated data, which are authenticated but not encrypted, with each message.

The upper part of Figure 3 indicates the inputs used to generate the offset values (labeled $\Delta[i]$). The nonce N is restricted to a maximum of 96 bits in length, and is padded on the left with a sufficient number of zero bits and a 1 bit to make a 128-bit final nonce. As with OCB1, the nonce is changed with each new message. For OCB3, N is a counter, incremented by one for each new message. The calculation of the $\Delta[i]$ can be defined as follows:

$\mathrm{Nonce} \leftarrow 0^{127-|N|} \| 1 \| \mathrm{N}$

$\mathrm{Top} \leftarrow \mathrm{Nonce} \oplus 1^{122}0^6$

$\mathrm{Ktop} = \mathrm{E}_K(\mathrm{Top})$

$\mathrm{Bottom} \leftarrow \mathrm{Nonce} \oplus 0^{122}1^6$

$\mathrm{Stretch} \leftarrow \mathrm{Ktop} \,\|\, (\mathrm{Ktop} \oplus (\mathrm{Ktop} \ll 8))$ a 256-bit value

$\Delta[0] \leftarrow (\mathrm{Stretch} \ll \mathrm{Bottom})[1..128]$

$\mathrm{L}_* \leftarrow \mathrm{E}_K(0^{128})$

$\mathrm{L}_\$ \leftarrow 2 \cdot \mathrm{L}_*$

$\mathrm{L}[0] \leftarrow 2 \cdot \mathrm{L}_\$$

$\mathrm{L}[i] = 2 \cdot \mathrm{L}[i-1]$ $1 \leq \mathrm{i} \leq m$

$\Delta[i] = \Delta[i-1] \oplus \mathrm{L}[\mathrm{ntz}(i)]$ $1 \leq \mathrm{i} \leq m$

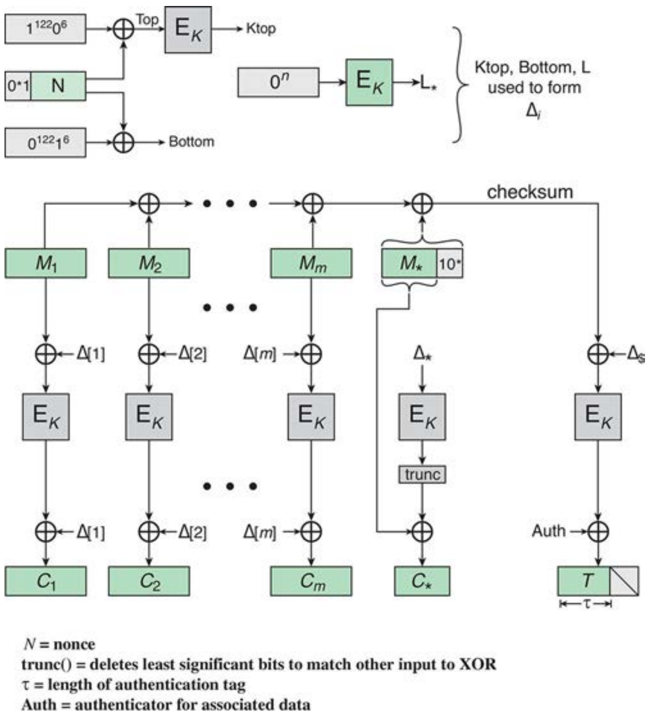

1^122 0^6
Top
E_K
Ktop
0*1
N
0^n
E_K
L*
Ktop, Bottom, L used to form Δ_i
0^122 1^6
Bottom
checksum
M_1
M_2
M_m
M*
10*
Δ[1]
Δ[2]
Δ[m]
Δ*
Δ$
E_K
trunc
Δ[1]
Δ[2]
Δ[m]
Auth
C_1
C_2
C_m
C*
T
τ
N = nonce
trunc() = deletes least significant bits to match other input to XOR
τ = length of authentication tag
Auth = authenticator for associated data

Figure 3. OCB3 encryption and authentication.

$\Delta_* \leftarrow \Delta[m] \oplus \mathrm{L}_*$
$\Delta_\$ \leftarrow \Delta[m] \oplus \mathrm{L}_\$$

where $\|$ denotes concatenation and where $(x \ll a)$ is a left shift of x by a bits, with the a most significant bits lost and the value filled in on the right with a zeros.

The initialization routine creates the offsets $\Delta[i]$. The initial value of $\Delta[0]$ is in effect a hash function of the nonce, created as follows. The 6 low-order bits are mapped into a 128-bit string Bottom with the low-order bits padded to the left by zeros. Similarly, the 122 high-order bits are mapped to a 128-bit string padded to the right with zeros. This latter string is then XORed with Nonce and encrypted to create Ktop. The result is stretched to 256 bits and then 128 bits are extracted. This method produces a strong hash for the offset. The remaining $\Delta[i]$ are computed with the same formula as is used for the $\mathrm{Z}[i]$ offsets of OCB1. Note that it appears that two encryptions are required to compute all the $\Delta[i]$ values, as is the case with the OCB1 $\mathrm{Z}[i]$ values. However, if N is a counter that is incremented by 1 for each new message, then Top and therefore Ktop changes only once for every 64 messages. Top captures the upper 90 bits of N, while Bottom captures the lower 6 bits. In effect, Bottom is a 6-bit counter. To assure that the $\Delta[i]$ values are different for each message, Bottom and Ktop are combined to compute $\Delta[0]$. Putting it all together, 63 out of 64 times, all that will be needed to compute $\Delta[0]$ is a logical operation to extract the lowest six bits of N and then a logical shift of the string Stretch by this many bits, which requires only a few processor cycles. Only one time out of 64 ($1/64 \approx 0.016$) requires a block cipher call. To form the value $\Delta[0]$, Stretch is shifted left by Bottom bits, and the leftmost (most significant) half is used. Rogaway and Krovetz (2011) provided an analysis of the validity of computing $\Delta[0]$ in this fashion.

Comparing Figures 1–3, we see that for OCB1 the checksum cannot be computed until the final ciphertext block is computed. Thus, even if a fully parallel operation is possible, there will be a pipeline delay of one encryption before the checksum is computed. OCB3 removes this restriction, allowing the checksum to be computed in parallel with all the ciphertext block computations.

There is one minor point of confusion. This article follows the notation used in the formal specifications for OCB1 and OCB3. Unfortunately, OCB1 numbers blocks from 1 through m, with the mth block being a partial block, while OCB3 numbers the full blocks as 1 through m, plus an additional partial block. Thus OCB1 has m blocks, and OCB3 has $m + 1$ blocks.

Additional data

The additional data require authentication, without developing a ciphertext for confidentiality. Figure 4 illustrates this operation. This is a keyed hash function, also referred to as a message authentication code (MAC). A hash code, or authenticator, for the additional data (AD) is formed that is a function of the data plus the encryption key. The structure of this MAC is similar to the Cipher-based Message Authentication Code (CMAC) defined in NIST SP 800-38B (NIST 2016). In both cases, the AD is broken up into blocks, with each block XORed with a value and then encrypted. The significant difference is that with CMAC, the second XOR argument is the output of the encryption of the previous block (with the exception of the first block), whereas with the OCB3 MAC, the second XOR argument is an offset value. All the offset values δ[i] can be precomputed, so that the individual AD blocks can be encrypted in parallel.

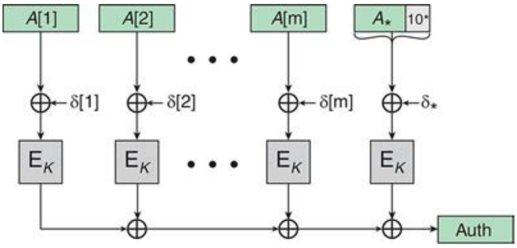
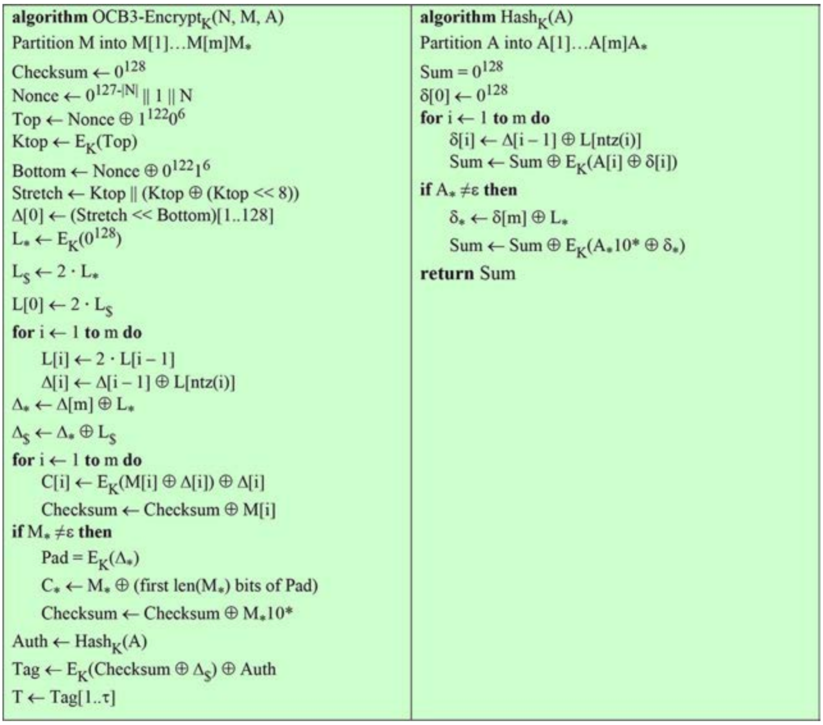


Figure 4. Authenticator for additional data.



```
algorithm OCB3-Encrypt$_K$(N, M, A)
Partition M into $M[1] \ldots M[m]M_*$
Checksum $\leftarrow 0^{128}$
Nonce $\leftarrow 0^{127-|N|} \| 1 \| N$
Top $\leftarrow$ Nonce $\oplus 1^{122}0^6$
Ktop $\leftarrow E_K$(Top)
Bottom $\leftarrow$ Nonce $\oplus 0^{122}1^6$
Stretch $\leftarrow$ Ktop $\|$ (Ktop $\oplus$ (Ktop << 8))
$\Delta[0] \leftarrow$ (Stretch << Bottom)[1..128]
$L_* \leftarrow E_K(0^{128})$
$L_{\$} \leftarrow 2 \cdot L_*$
$L[0] \leftarrow 2 \cdot L_{\$}$
for $i \leftarrow 1$ to m do
    $L[i] \leftarrow 2 \cdot L[i-1]$
    $\Delta[i] \leftarrow \Delta[i-1] \oplus L[ntz(i)]$
$\Delta_* \leftarrow \Delta[m] \oplus L_*$
$\Delta_{\$} \leftarrow \Delta_* \oplus L_{\$}$
for $i \leftarrow 1$ to m do
    $C[i] \leftarrow E_K(M[i] \oplus \Delta[i]) \oplus \Delta[i]$
    Checksum $\leftarrow$ Checksum $\oplus M[i]$
if $M_* \neq \varepsilon$ then
    Pad $= E_K(\Delta_*)$
    $C_* \leftarrow M_* \oplus$ (first len($M_*$) bits of Pad)
    Checksum $\leftarrow$ Checksum $\oplus M_*10^*$
Auth $\leftarrow$ Hash$_K$(A)
Tag $\leftarrow E_K$(Checksum $\oplus \Delta_{\$}$) $\oplus$ Auth
T $\leftarrow$ Tag[1..$\tau$]
```

```
algorithm Hash$_K$(A)
Partition A into $A[1] \ldots A[m]A_*$
Sum $= 0^{128}$
$\delta[0] \leftarrow 0^{128}$
for $i \leftarrow 1$ to m do
    $\delta[i] \leftarrow \Delta[i-1] \oplus L[ntz(i)]$
    Sum $\leftarrow$ Sum $\oplus E_K(A[i] \oplus \delta[i])$
if $A_* \neq \varepsilon$ then
    $\delta_* \leftarrow \delta[m] \oplus L_*$
    Sum $\leftarrow$ Sum $\oplus E_K(A_*10^* \oplus \delta_*)$
return Sum
```

Figure 5. OCB3 encryption and authentication of additional data.

Figure 5 shows the algorithms for OCB3 encryption and the MAC.

Summary

Because of its streamlined design, OCB is well suited for IoT devices, wireless sensors, and other constrained devices where processing power and energy consumption are concerns. As well, for larger multicore devices that have the ability to perform parallel processing, OCB excels at speed of execution. Thus, OCB is a versatile AE technique for a wide range of applications.

About the author

William Stallings holds a PhD from MIT in Computer Science. He is an independent consultant and author of numerous textbooks on security, computer networking, and computer architecture. He has 12 times received the award for the Best Computer Science and Engineering Textbook of the Year from the Textbook and Academic Authors Association. His most recent book is *Practical Cybersecurity: Guide to Best Practices and Standards* (Pearson, 2017). He is also author of *Cryptography and Network Security, Principles and Practice, Seventh Edition* (Pearson, 2017), and co-author, with Lawrie Brown, of *Computer Security, Principles and Practice, Third Edition* (Pearson, 2015). He created and maintains the Computer Science Student Resource Site at ComputerScienceStudent.com. This site provides documents and links on a variety of subjects of general interest to computer science students (and professionals). Dr. Stallings is on the editorial board of *Cryptologia*.

References

Krovetz, T., and P. Rogaway. 2014. The OCB Authenticated-Encryption Algorithm, RFC 7253, May.

Luk, M., G. Mezzour, A. Perrig, and V. Gligor. 2007. MiniSec: A secure sensor network communication architecture. *International Conference on Information Processing in Sensor Networks*.

National Institute of Standards and Technology. 2007a. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. NIST Special Publication 800–38C.

National Institute of Standards and Technology. 2007b. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800–38D.

National Institute of Standards and Technology. 2016. Recommendation for block cipher modes of operation: The CMAC mode for authentication. NIST Special Publication 800–38B.

Rogaway, P., and T. Krovetz. 2011. The Software performance of authenticated encryption modes. Fast Software Encryption—FSE 2011.

Rogaway, P., M. Bellare, and J. Black. 2003. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security*, August.

Rogaway, P., M. Bellare, J. Black, and T. Krovetz. 2001. OCB: A block-cipher mode of operation for efficient authenticated encryption. NIST proposed block cipher mode, August 2001. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ocb/ocb-spec.pdf.

Stallings, W. 2010. NIST block cipher modes of operation for authentication and combined confidentiality and authentication. *Cryptologia* 34 (3):225–35.

Stallings, W. 2017. *Cryptography and network security: Principles and practice, seventh edition*. Upper Saddle River, NJ: Pearson.

Walker, J. 2005. 802.11 Security Series. Part III: AES-based Encapsulations of 802.11 Data. Platform Networking Group, Intel Corporation, 2005.

Appendix: Brief introduction to finite fields

In essence, a finite field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set. Division is defined with the following rule: $a/b = a(b^{-1})$. There is a way of defining a finite field containing 2^n elements; such a field is referred to as GF(2^n). GF stands for Galois field, in honor of the mathematician who first studied finite fields. Consider the set, *S*, of all polynomials of degree $n - 1$ or less with binary coefficients.

Thus, each polynomial has the form

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

where each a_i takes on the value 0 or 1. There are a total of 2^n different polynomials in S. For $n = 3$, the $2^3 = 8$ polynomials in the set are

0	1	x	x + 1	x^2	$x^2 + 1$	$x^2 + x$	$x^2 + x + 1$
000	001	010	011	100	101	110	111

The second row shows the values of $a_2 a_1 a_0$.

With the appropriate definition of arithmetic operations, each such set S is a finite field. The definition consists of the following elements:

1. Arithmetic follows the ordinary rules of polynomial arithmetic using the basic rules of algebra, with the following two refinements.
2. Arithmetic on the coefficients is performed modulo 2. This is the same as the XOR operation.
3. If multiplication results in a polynomial of degree greater than $n - 1$, then the polynomial is reduced modulo some irreducible polynomial $m(x)$ of degree n. That is, we divide by $m(x)$ and keep the remainder. For a polynomial $f(x)$, the remainder is expressed as $r(x) = f(x) \bmod m(x)$. A polynomial $m(x)$ is called irreducible if and only if $m(x)$ cannot be expressed as a product of two polynomials, both of degree lower than that of $m(x)$.

A polynomial in GF(2^n) can be uniquely represented by its n binary coefficients ($a_{n-1}a_{n-2} \ldots a_0$). Therefore, every polynomial in GF(2^n) can be represented by an n-bit number. Addition is performed by taking the bitwise XOR of the two n-bit elements. There is no simple XOR operation that will accomplish multiplication in GF(2^n). However, a reasonably straightforward, easily implemented technique is available. In essence, it can be shown that multiplication of a number in GF(2^n) by 2 consists of a left shift followed by a conditional XOR with a constant. Multiplication by larger numbers can be achieved by repeated application of this rule.

For example, to construct the finite field GF(2^3), we must choose an irreducible polynomial of degree 3. There are only two such polynomials: ($x^3 + x^2 + 1$) and ($x^3 + x + 1$). Using the latter, Table 1 shows the multiplication table for GF(2^3). Addition is equivalent to taking the XOR of like terms. Thus, $(x + 1) + x = 1$.

OCB1 and OCB3 use GF(2^{128}), defined with the irreducible polynomial $m(x) = x^{128} + x^7 + x^2 + 1$. Over this field, the representation of the value 2 is

Table 1. Polynomial multiplication modulo $(x^3 + x + 1)$.

	000	001	010	011	100	101	110	111
000	0	0	0	0	0	0	0	0
001	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
010	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
011	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
100	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
101	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
110	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
111	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	$x+1$

binary $0^{126}10$. It can be shown that multiplication of a value x by 2 for this finite field is achieved by the following definition:

$$2 \cdot x = \begin{cases} x \ll 1 & \text{if } \mathrm{firstbit}(x) = 0 \\ (x \ll 1) \oplus 10^{120}1000011 & \text{if } \mathrm{firstbit}(x) = 1 \end{cases}$$

Division of x by 2, expressed as $x \cdot 2^{-1}$, is achieved by the following definition:

$$x \cdot 2^{-1} = \begin{cases} x \gg 1 & \text{if } \mathrm{lastbit}(x) = 0 \\ (x \gg 1) \oplus 10^{120}1000011 & \text{if } \mathrm{lastbit}(x) = 1 \end{cases}$$