Submission to NIST:

Random Access Counter (RAC)

AES Mode of Operation

Submitter/Author:

Jeff Anderson

5285 Shawnee Rd.

Alexandria, VA 22312

Phone: (301) 379-4069

Email: Jeff.Anderson@jhu.edu

1. Introduction

RAC is a confidentiality mode of operation for block ciphers; it was designed as a variation of Counter mode (CTR) [1], arising from alternative methods for prescribing the generation of counters. More specifically, the generation and operation of counters was designed such that RAC mode can be used to efficiently encrypt and decrypt transactions between a microprocessor and Random Access Memory (RAM) without any degradation in security. As it is a variation of Counter mode, it is based on mechanisms which are supported by a well-understood theoretical foundation and its security follows from reasonable assumptions of the underlying block cipher and Counter mode, in general.

As security of data has moved to the forefront of the public's attention, there exists a compelling need for encryption of data as it leaves a microprocessor's boundary and interfaces with RAM. This requirement has proven troublesome for system designers, as the performance requirements for modern RAM interfaces does not lend itself well to inline encryption. Authenticated encryption modes such as CCM [1] and CBC-MAC [2] do not allow parallelization and add an intolerable amount of latency due to the serial nature of encryption and authentication. GCM [3], while designed to allow parallelization and pipelining, requires some amount of memory for the authentication tag. If the authentication tag is unused, then GCM simplifies to CTR with regard to security and performance; however, the simplicity of the counter is not optimized for the patterns of writes typically seen when a microprocessor interfaces with RAM. The simplicity of the incrementing counter scheme used in both GCM and CTR does not pair well with the natural behavior of a microprocessor interfacing with memory. The behavior of a microprocessor writing to RAM during the course of program execution would require a complex scheme to store and manage counters for various sections of memory; this scheme would use memory that would be better served being made available to the microprocessor for use during execution.

Any AES mode of operation that operates at the boundary of a microprocessor and RAM must be highly parallelizable and add low latency to the system. Additionally, it should be simplistic to implement in hardware and use counters that have been optimized for the behaviors expected at a microprocessor/RAM interface. Using the RAM address is a logical starting point for a counter, as it provides a unique value for each section of RAM. However, the normal execution flow of a microprocessor allows several writes to the same memory location, realizing the need for a traditional counter variable. Additionally, a cache line is equal in length to several 128-bit blocks, making another counter necessary. While the aforementioned counters and address bits allow for unique values during a microprocessor's program execution, they will repeat each time the microprocessor is reset; affirming the need for a nonce generated at startup.

The RAC mode of operation fills the need for encryption of memory. It is a variant of CTR, designed to provide only confidentiality, with counters designed with cache writes and resets in mind. Authentication was not designed into the system, as it adds

unnecessary latency to writes. It is expected that error correcting codes built into RAM modules will allow the microprocessor's memory management unit to identify and possibly correct any bit errors that may occur.

This document is organized as follows. Section 2 contains the complete specification of RAC. Section 3 provides implementation strategies and a discussion of performance. Properties of RAC and rationale is discussed in Section 4. Section 5 covers any restrictions on implementation required to keep data secure with RAC, and a brief discussion of nonce suggestions (as RAM interface behaviors change, the counter may need to evolve as well). Section 6 and section 7 very briefly discuss the security and performance of the RAC mode of operation, respectively, by comparing it to CTR. Section 8 contains intellectual property statements.

2. Definition

RAC is a block cipher mode of operation that provides confidentiality to transactions between a microprocessor and RAM. It was designed for 128-bit block ciphers and 64-byte cache lines, but can be easily extended to other block sizes and cache line lengths. Block lengths shorter than 128 bits are not recommended.

a. Inputs and Outputs

When encrypting a line of data intended for RAM, the sender must provide the following:

- An encryption key *K* suitable for the underlying block cipher.
- A 32-bit nonce *R*. Within the scope of any key *K*, the nonce value *R* shall be unique. Like CTR and CCM, reusing nonces for different messages encrypted with the same key destroys the security properties of this mode. The Birthday Problem then dictates that each nonce/key combination can be used for 2^{46} blocks.
- RAM address *A*, which is defined as 6 bytes long, supporting 2^{48} memory addresses; a 46-bit write counter *X* supporting 2^{46}writes per address and a 2-bit cache
- The message *P*, whose length is 64 bytes (the most common length of a cache line).

Name	Description	Field Size
K	Block cipher key	Block cipher dependent
R	Nonce	4 bytes
A	RAM address	6 bytes
X	Write counter	46 bits
Y	Cache Block counter	2 bits
P	Line of plaintext	64 bytes

Table 1: Inputs for RAC mode

There is only one output:

- Ciphertext *C* whose length is exactly that of the plaintext *P*.

b. Notation

Notation follows that of the *Recommendation for Block Cipher Modes of Operation* [4]. The main functions used in RAC are block cipher encryption, concatenation and addition modulo-2. The block cipher encryption of the message *P* with the key *K* is denoted as *E(K,P).* The addition of *X* and *Y* is denoted as $X \oplus Y$ and is implemented in this field using the bitwise exclusive-or operation. Concatenation of *X* and *Y* is denoted as $X \,||\, Y$ and is implemented by appending the most significant byte of *Y* to the least significant byte of *X*. Successive counter values are generated using the function incr(*X*,t), which treats the rightmost bits of its argument *X* as a nonnegative integer with the least significant bit on the right, and increments this value modulo 2^t.

c. Encryption

The encryption operation for a 128-bit block cipher is defined as follows:
Let the plaintext *P* be defined as a 64-byte string, or 4 blocks.

$Y_i = \text{incr}(Y_{i-1}, 4)$ for $i = 0,1,2,3$
$X_j = \text{incr}(X_{j-1}, n)$ for $j = 0,\ldots,n-1$
$C_i = P_i \oplus E(K,(R||Y_i||X_j||A))$ for $i = 0,1,2,3$ and $j = 0,\ldots,n-1$

d. Decryption

The decryption operation is identical to the encryption operation, with the operands reversed:

$Y_i = \text{incr}(Y_{i-1})$ for $i = 0,1,2,3$
$X_j = \text{incr}(X_{j-1})$ for $j = 0,\ldots,n-1$
$P_i = C_i \oplus E(K,(R||Y_i||X_j||A))$ for $i = 0,1,2,3$ and $j = 0,\ldots,n-1$

3. Implementation

As RAC is a variant of CTR, with specially generated counters, implementation is as straightforward as CTR. Figure 1 shows a sample implementation of fully parallelized RAC for a 64-byte cache line. Note how the cache line is split into 128-bit blocks due to the underlying block cipher and Y_i is automatically incremented for each block; hardware implementations allow for these lines to be hardwired as they will not change for a fully parallelized implementation. The size of *Y* can be changed to accommodate

cache lines of various sizes by resizing the nonce or other counters. *A* changes automatically as different regions of memory are accessed, and can be directly connected to the address bits of a memory controller in hardware implementations to lessen the memory burden. *R* is a random nonce generated at startup, and can be either stored in its own memory buffer, or wired directly to a random number generator in hardware implementations. *X* for an individual cache line gets incremented once per cache line write. It is recommended that *X* for each cache line is stored in its own memory buffer. For lowest latency, it is recommended that the fully parallelized algorithm from Figure 1 be implemented entirely in hardware.

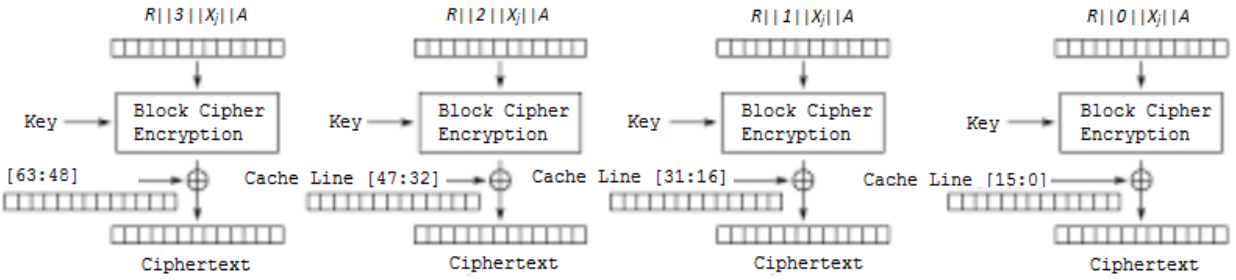

Figure 1: Parallel implementation of RAC encryption

4. Properties and Rationale

The summary of properties for RAC is shown below.

Security Function	Encryption
Error propagation	None
Synchronization	Same nonce used by sender and recipient
Parallelizability	Encryption can be parallelized
Keying Material Requirements	One key
Counter/Nonce/IV Requirements	Counter and Nonce are part of the counter block
Memory Requirements	Same requirements as CTR mode for underlying block cipher. Counter requires memory equal to Sizeof(X_i) * n, where n is equal to the number of cache lines.
Pre-processing Capability	Encryption key stream can be precomputed
Message Length Requirements	64 bytes (one cache line)
Ciphertext Expansion	None

Table 2: A summary of the properties of RAC

CTR, with its ability to be fully parallelized and outputs preprocessed, was deemed the most appropriate mode to meet the high performance requirements of memory encryption. Due to limitations associated with the simplicity of the CTR counter scheme, it was deemed unable to fulfill the requirements of a memory encryptor. RAC was

developed as a CTR variant using alternative methods for prescribing the generation of counters.

The nonce and counters were chosen based on the expected behavior of a microprocessor accessing RAM during program execution. The expected behaviors during program execution are cache writes, reads and rewrites and system resets. Assigning address *A* as a portion of the counter is natural for RAM encryption as it ensures that each physical address of RAM will have a unique counter value. However, this ensures security for only the first memory write for each physical address, as rewrites would reuse the counter value if the address was the only component. To address this, the write counter *X* was added for each cache line. For each write, this counter is incremented, ensuring that each physical address of memory has a unique counter value. A counter comprised of a memory address and write counter was still deemed insufficient, however, because an attacker with the ability to reset the system could compromise security of the mode by resetting the system; the write counters reset to 0, which effectively reuses counters as writes are made to RAM. A 32-bit nonce *R* was added to address this concern. 32 bits was chosen as the nonce size because it is a common output of random number generators; this size is not necessary for collision-free operation due to the relative infrequency of resets. A counter comprised of the aforementioned elements would be sufficient based on the behavior of a memory controller. However, the common cache line size is 64 bytes. This is equal to four blocks of a 128-bit block cipher. Since an entire cache line will be written per physical memory address, a block counter *Y* was added to ensure uniqueness among blocks in a cache line.

As technology changes, cache line size and block cipher size is expected to evolve; and RAC can be easily modified when such changes occur. Due to the use of common cache line size of 64 bytes and a 128-bit underlying block cipher, only 2 bits are needed for the block counter. Increases in cache line size would necessitate an increase in the size of the block counter. This would need to be offset by decreasing the size of the nonce, write counter or address. The address word needs to be large enough to encompass the entirety of RAM, which is system dependent, and therefore may not be available to be changed. The write counter is 48 bits, which allows up to 2^{48} writes to take place. As mentioned before, the nonce is 32 bits for convenience, but does not need to be that size for security purposes. The author recommends decreasing the size of the nonce or address (if extra bits are available) to offset an increase in the block counter, prior to decreasing the size of the write counter, as many assumptions need to be made on system properties (system availability and number of writes per second to name a few) that are out of user control.

5. Restrictions

As with other block cipher modes of operation, security degrades as more data is processed with a single key. Determination of the number of encryption operations

available without degradation of security is dependent on the number of bits in the write counter *X*. While there are several other elements that comprise the counter, the nonce and block counter can be treated as static. Additionally, the address should also be treated as static, as the memory controller dictates what physical address will be written to; the scenario where only one address is written to continuously, while uncommon, cannot be ignored.

While a 32-bit nonce *R* was chosen for convenience, it is not required to maintain security of the mode. However, the sender shall ensure that the nonce shall never be less than 26 bits in length, as this will degrade the security of RAC.

The author does not recommend that this mode be defined for block sizes less than 128 bits. A 56-bit block cipher does not allow for sufficient degrees of freedom for counters. For instance, a 26-bit nonce leaves only 30 bits for address and counters. 3 bits will be needed for block counters, leaving only 27 bits to share between address and write counters. For large memories, in the GB range, this leaves less than 10 bits for a write counter, forcing the microprocessor to rekey every thousand memory writes. This level of performance is unacceptable for a high speed component such as a memory encryptor.

6. Security

As RAC is a variant of CTR mode with a specialized counter, its security proof follows the security proof for CTR. CTR mode, developed by Diffie and Hellman [5], was proven secure in the concrete model by Bellare et al [6]. As RAC does not deviate from CTR with regard to rules of interfacing counters to the underlying block cipher, it follows that the CTR security proof would be valid for RAC as well.

7. Performance Estimates

As with other modes of encryption, performance depends on the speed of the implementation of the underlying block cipher.

Encrypting a 32-bit message requires one block cipher encryption operation. Encrypting a 64-byte cache line or a 128-byte cache line requires four block cipher encryption operations, or eight clock cipher encryptions, respectively. As with CTR mode, RAC mode is fully parallelizable, so an implementation that takes advantage of this can encrypt a full cache line, regardless of size, in the time it takes to execute one block cipher encryption operation.

Additionally, as this mode is a variation of Counter mode, it requires the same hardware as CTR mode and executes at exactly the same speed as CTR mode with an arbitrary counter.

8. Intellectual Property Statements

The Author hereby explicitly releases any intellectual property rights to RAC to the public domain. Further, the author is not aware of any patent or patent application that covers RAC mode. It is my belief that RAC is a simple combination of well-established techniques, and that RAC is obvious to any person with ordinary skill in the art.

References

[1] H. Lipmaa, P. Rogaway, and D. Wagner. Comments to NIST Concerning AES Modes of Operations: CTR-Mode Encryption. *Block Cipher Modes Workshop 1,* October, 2000. Available online at http://csrc.nist.gov/groups/ST/toolkit/BCM/workshops.html.

[2] D. Whiting, N. Ferguson, and R. Housley. Counter with CBCMAC(CCM). *Submission to NIST,* 2002. Available online at http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/.

[3] DES Modes of Operation. *Federal Information Processing Standards Publication 81,* December, 1980.

[4] D. McGrew, and J. Viega. The Galois/Counter Mode of Operation(GCM). *Submission to NIST,* 2005. Available online at http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/.

[5] M. Dworkin, Recommendation for Block Cipher Modes of Operation: Methods and Techniques, *NIST Special Publication 800-38A.*

[6] W. Diffie, and M. Hellman. Privacy and Authentication: An Introduction to Cryptography. *Proceedings of the IEEE,* Volume 67, Number 3, March, 1979.

[7] M. Bellare, A. Desai, E. Jokipii and P. Rogaway. A concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation. *Proceedings of the 38th Annual Symposium on Foundations of Computer Science, IEEE,* 1997.

AES Test Vectors

The following are test vectors for AES-RAC. The test vectors were produced in a software simulation of hardware designed to perform AES-RAC with 128 or 256-bit keys on 64-byte cache lines. For more test vectors, see AES CTR test vectors in [4]. The test vectors are presented in the following format:

Plaintext	64-byte hexadecimal string (MSB…..LSB)
Ciphertext	64-byte hexadecimal string (MSB…LSB)
Key length	integer
Key	hexadecimal string (128 or 256 bits)
Nonce	32-bit hexadecimal string
Address	6-byte hexadecimal string
Write Counter	46-byte hexadecimal string
Output Block 3 (MSB)	16-byte hexadecimal string
Output Block 2	16-byte hexadecimal string
Output Block	16-byte hexadecimal string
Output Block 0 (LSB)	16-byte hexadecimal string

Plaintext	76777475F1F2F3F4F8F9E6E77770717276777475F1F2F3F4F8F9E6E77770717276777475F1F2F3F4F8F9E6E777707172 76777475F1F2F3F4F8F9E6E777707172
Ciphertext	4c2061fc8b6f13cd25957c79757efffa31e232f9e85af6bb16bd5095efb5fd219 7b7cc9ba67b527590c7ba83723fbc493 d526e534612a2a5358a3355377b56dc0
Key length	128
Key	E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FA
Nonce	014BAF22
Address	80103643E99A
Write Counter	000000000000
Output Block 3 (MSB)	3a5715897a9de039dd6c9a9e020e8e88
Output Block 2	368545beb745d9845932cefb98c2fa36b
Output Block	d0bbdcf9647d4adf4824ed0548bb5e1
Output Block 0 (LSB)	a351914190d8d9a7a05ad3b400c51cb2

Plaintext	76777475F1F2F3F4F8F9E6E77770717276777475F1F2F3F4F8F9E6E77770717276777475F1F2F3F4F8F9E6E777707172 76777475F1F2F3F4F8F9E6E777707172
Ciphertext	c70f0fa6e68621a60e21042a265eeea37c3195b4938162b88e456e4c2dddf0fac062f79a0689f36b9376f138fbbd94c3 99802395c49be7b49ec7a097d90c6e15
Key length	128
Key	E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FA
Nonce	014BAF22
Address	7596840598AB
Write Counter	000000000000
Output Block 3 (MSB)	b1787bd31774d252f6d8e2cd512e9fd1
Output Block 2	0a46e1c16273914c76bc88ab5aad8188
Output Block	b61583eff77b009f6b8f17df8ccde5b1
Output Block 0 (LSB)	eff757e035691440663e4670ae7c1f67

Plaintext	76777475F1F2F3F4F8F9E6E77770717276777475F1F2F3F4F8F9E6E77770717276777475F1F2F3F4F8F9E6E777707172 76777475F1F2F3F4F8F9E6E777707172
Ciphertext	6511c6ba1816d3b921967d52e8409b917e21cd3017fba749f872bcfc5dfc503137bc7cd024ed15dc7b87612dbacaa4f31 8c354c14c1c309d921edabdf73c7ece5
Key length	128
Key	E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FA
Nonce	014BAF22
Address	7596840598AB
Write Counter	000000000001
Output Block 3 (MSB)	1366b2cfe9e4204dd96f9bb59f30eae3
Output Block 2	856b945e60954bd008b5a1b2a8c2143
Output Block	30db0b977bf23ae33408ff43cdbda3e43

Output Block 0 (LSB)	fa4238613031fa2dd9144d3804b79d97

Plaintext	76777475F1F2F3F4F8F9E6E77770717276777475F1F2F3F4F8F9E6E77770717276777475F1F2F3F4F8F9E6E777707172 76777475F1F2F3F4F8F9E6E777707172
Ciphertext	9db6ab30e78e7d6728828864c4b71e6a57abe5ec4cbaaf2345e94fc6fba3181871a3f299fb61d8bcffa18c3bd260d8737466fc7e512734592ab719f735b16519
Key length	128
Key	E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FA
Nonce	014BAF22
Address	7596840598AB
Write Counter	000000000002
Output Block 3 (MSB)	ebc1df45167c8e93d07b6e83b3c76f18
Output Block 2	21dc9199bd485cd7bd10a9218cd3696a
Output Block	07d486ec0a932b4807586adca510a901
Output Block 0 (LSB)	0211880ba0d5c7add24eff1042c1146b

Plaintext	76777475F1F2F3F4F8F9E6E77770717276777475F1F2F3F4F8F9E6E77770717276777475F1F2F3F4F8F9E6E777707172 76777475F1F2F3F4F8F9E6E777707172
Ciphertext	2de8b02d390389dc1e5c3e6c17f45ea6b37bc98ea2f6fcef0ceacf39e7312190550ffde863262a071fe660fa3bdc08ffcd689bdff30add03e485e5fb18970693
Key length	128
Key	E8E9EAEBEDEEEFF0F2F3F4F5F7F8F9FA
Nonce	014BAF22
Address	7596840598AB
Write Counter	000000000003
Output Block 3 (MSB)	5b9fc458c8f17a28e6a5d88b60842fd4
Output Block 2	c50cbdfb53040f1bf41329de904150e2
Output Block	2378899d92d4d9f3e71f861d4cac798d
Output Block 0 (LSB)	bb1fefaa02f82ef71c7c031c6fe777e1