ADVANCED ENCRYPTION STANDARD

Douglas Selent*
Student, M.S. Program in Computer Science, Rivier College

Abstract

Advanced Encryption Standard (AES) is the current standard for secret key encryption. AES was created by two Belgian cryptographers, Vincent Rijmen and Joan Daemen, replacing the old Data Encryption Standard (DES). The Federal Information Processing Standard 197 used a standardized version of the algorithm called Rijndael for the Advanced Encryption Standard. The algorithm uses a combination of Exclusive-OR operations (XOR), octet substitution with an S-box, row and column rotations, and a MixColumn. It was successful because it was easy to implement and could run in a reasonable amount of time on a regular computer.

1 Introduction

On January 2, 1997 the National Institute of Standards and Technology (NIST) held a contest for a new encryption standard. The previous standard, DES, was no longer adequate for security. It had been the standard since November 23, 1976. Computing power had increased a lot since then and the algorithm was no longer considered safe. In 1998 DES was cracked in less than three days by a specially made computer called the DES cracker. The DES cracker was created by the Electronic Frontier Foundation for less than $ 250,000 and won the RSA DES Challenge II-2. [1]

Current alternatives to a new encryption standard were Triple DES (3DES) and International Data Encryption Algorithm (IDEA). The problem was IDEA and 3DES were too slow and IDEA was not free to implement due to patents. NIST wanted a free and easy to implement algorithm that would provide good security. Additionally they wanted the algorithm to be efficient and flexible. [2]

After holding the contest for three years, NIST chose an algorithm created by two Belgian computer scientists, Vincent Rijmen and Joan Daemen. They named their algorithm Rijndael after themselves [2]. Supposedly Rijndael can only be pronounced correctly by people who can speak Dutch and the closest English approximation is “Rhine Dahl”. [3]

On November 26, 2001 the Federal Information Processing Standards Publication 197 announced a standardized form of the Rijndael algorithm as the new standard for encryption. This standard was called Advanced Encryption Standard and is currently still the standard for encryption. [4]

2 Rijndael Block and Key

Before applying the algorithm to the data, the block and key sizes must be determined. AES allows for block sizes of 128, 168, 192, 224, and 256 bits. AES allows key sizes of 128, 192, and 256 bits [2]. The standard encryption uses AES-128 where both the block and key size are 128 bits. The block size is commonly denoted as N_b and the key size is commonly denoted as N_k. N_b refers to the number of columns in the block where each row in the column consists of four cells of 8 bytes each for AES-128 [5].

The following example will show how data is broken up into blocks. Using AES-128 means that each block will consist of 128 bits. N_b can be calculated by dividing 128 by 32. The 32 comes from the

ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

number of bytes in each column. In this case, N_b is 4. The original plain text is stored in bytes in a block. For example, the text “This is a test…” will be stored in a block as shown below in Figure 1.

Block

	0	1	2	3
0	T		a	s
1	h	i		t
2	i	s	t	.
3	s		e	.

Figure 1. AES-128 block example

Each character is stored in a cell of the block. The blank cells shown in the diagram are not really blank as they represent the spaces in the text. Depending on how the algorithm is implemented the characters may be stored as integer values, hexadecimal values, or even binary strings. All three ways represent the same data. Most diagrams show the hexadecimal values, however integer and string manipulation is much easier to do when actually programming AES. Figure 1 shows the values as characters for demonstration purposes to show how the text is stored into the block. The plain text is stored into blocks column by column and block by block until all the data is stored [5]. In the example used above there were exactly 16 characters used for simplicity. In order to use the Rijndael algorithm the data must be a multiple of the block size, since all blocks need to be complete. When the data is not a multiple of the block size some form of padding must be used.

Padding is when extra bits are added to the original data. One forms of padding includes adding the same bytes until the desired size is reached. Another option is padding with all zeros and having the last byte represent the number of zeros. Padding with null characters or random characters are also forms of padding that can be used. [6]

Once a form of padding is chosen the data is represented as some number of complete blocks. The last thing needed before using the algorithm is the key. The key also known as the cipher key is also the same size as the block in this example. Unlike most data and transformations the cipher key can have any values chosen by the designer with no restrictions as long as the key is the correct length. The key is also stored as a block similar to the plain text. [5]

When the plain text data is stored into blocks and the key is chosen the Rijndael encryption algorithm can be applied. Some of the following steps could be done before the encryption process starts, but for simplicity they will be discussed when they are needed.

3 Rijndael Rounds

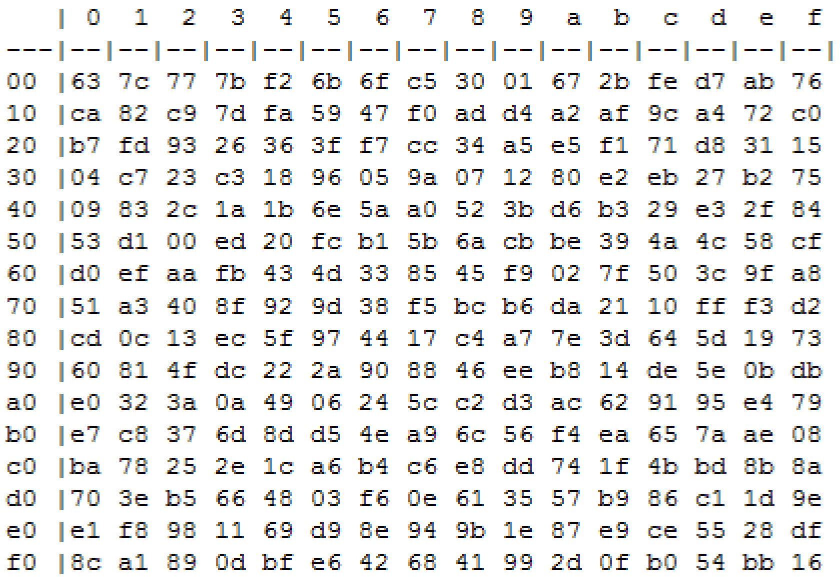
At a basic level the Rijndael algorithm uses a number of rounds to transform the data for each block. The number of rounds used is 6 + the maximum of N_b and N_k. Following from the previous example of AES-128, the number of rounds is 10. This is calculated from 6 plus the maximum of (4,4). Since N_b and N_k are both 4, the number of rounds is 6 + 4 = 10 [2]. The initial block (also known as a state) is added to an expanded key derived from the initial cipher key. Then the round processing occurs consisting of operations of the S-box, shifts, and a MixColumn. The result state is then added to the next expanded key. This is done for all ten rounds, with the exception of the MixColumn operation of the final round. The final result is the encrypted cipher block. [5]

4 Rijndael Key Expansion

The original cipher key needs to be expanded from 16 bytes to 16·(r + 1) bytes. In the example, there are ten rounds so r = 10. A round key is needed after each round and before the first round. Each round key needs to be 16 bytes because the block size is 16 bytes. Therefore, the cipher key needs to be expanded from 16 bytes to 16·(r + 1) bytes or 176 bytes. The expanded key is then broken up into round keys. Round keys are added to the current state after each round and before the first round. The details on the key expansion algorithm are complex and will be skipped. [4]

5 Rijndael S-box

The first step to a round is to do a byte by byte substitution with a lookup table called an S-box. An S-box is a one to one mapping for all byte values from 0 to 255. The S-box is used to change the original plain text in bytes to cipher text. The S-box is shown below in Figure 2. All values are represented in hexadecimal notation. This is how the S-box is commonly viewed. [5]



```
   | 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
---|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
00 |63 7c 77 7b f2 6b 6f c5 30 01 67 2b fe d7 ab 76
10 |ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c a4 72 c0
20 |b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 31 15
30 |04 c7 23 c3 18 96 05 9a 07 12 80 e2 eb 27 b2 75
40 |09 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 29 e3 2f 84
50 |53 d1 00 ed 20 fc b1 5b 6a cb be 39 4a 4c 58 cf
60 |d0 ef aa fb 43 4d 33 85 45 f9 02 7f 50 3c 9f a8
70 |51 a3 40 8f 92 9d 38 f5 bc b6 da 21 10 ff f3 d2
80 |cd 0c 13 ec 5f 97 44 17 c4 a7 7e 3d 64 5d 19 73
90 |60 81 4f dc 22 2a 90 88 46 ee b8 14 de 5e 0b db
a0 |e0 32 3a 0a 49 06 24 5c c2 d3 ac 62 91 95 e4 79
b0 |e7 c8 37 6d 8d d5 4e a9 6c 56 f4 ea 65 7a ae 08
c0 |ba 78 25 2e 1c a6 b4 c6 e8 dd 74 1f 4b bd 8b 8a
d0 |70 3e b5 66 48 03 f6 0e 61 35 57 b9 86 c1 1d 9e
e0 |e1 f8 98 11 69 d9 8e 94 9b 1e 87 e9 ce 55 28 df
f0 |8c a1 89 0d bf e6 42 68 41 99 2d 0f b0 54 bb 16
```

Figure 2. S-Box. From http://www.samiam.org/s-box.html

For example if we had the plain text of one character “D” it would translate to the hexadecimal value of 44 by using an ASCII lookup table. When using the S-box the first digit in hexadecimal represents the rows of the table or the values on the left side going down. The second digit represents the column of the S-box, which, in this example, is also 4. Using the S-box we find row number 4 and column number 4 and find that the hexadecimal value in that cell is “1b”. This is how the S-box works. Using this method with all the plain text will generate the new hexadecimal values that are used later in the algorithm.

The obvious question is: where did the S-box come from? This goes into modular arithmetic and a Galois field. A Galois field is a field with a finite number of elements. The Galois field is always a field that is a power of a prime. For each prime number there exists exactly one Galois field. The notation to represent a Galois field is GF(p), where p is the prime number. [8]

For the S-box, the field GF(2^8) was chosen. There are several reasons to why this field was chosen. One obvious reason is that the power of 8 was chosen because there are 8 bits in a byte. The prime 2 was chosen because binary is represented as two possible digit’s a 1 or a 0. In addition arithmetic is simple to do in this field because addition and subtraction are redefined as the exclusion or (XOR) operation. Invertability and resistance to algebraic attacks were also considered when forming the S-box. [5]

To actually generate the S-box from the chosen field requires much more work. Careful thought was put into the transformation for security purposes. Using plain text for the next steps of the algorithm would make it more vulnerable so a byte substitution in the form of the S-box was used. The original bytes are transformed by using the multiplicative inverse and an affinity matrix shown in Figure 3 to make the cipher text resistant to algebraic attacks. [7]

$$\begin{matrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{matrix}$$

Figure 3. Affinity Matrix. From http://www.samiam.org/s-box.html

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	--	01	8d	f6	cb	52	7b	d1	e8	4f	29	c0	b0	e1	e5	c7
10	74	b4	aa	4b	99	2b	60	5f	58	3f	fd	cc	ff	40	ee	b2
20	3a	6e	5a	f1	55	4d	a8	c9	c1	0a	98	15	30	44	a2	c2
30	2c	45	92	6c	f3	39	66	42	f2	35	20	6f	77	bb	59	19
40	1d	fe	37	67	2d	31	f5	69	a7	64	ab	13	54	25	e9	09
50	ed	5c	05	ca	4c	24	87	bf	18	3e	22	f0	51	ec	61	17
60	16	5e	af	d3	49	a6	36	43	f4	47	91	df	33	93	21	3b
70	79	b7	97	85	10	b5	ba	3c	b6	70	d0	06	a1	fa	81	82
80	83	7e	7f	80	96	73	be	56	9b	9e	95	d9	f7	02	b9	a4
90	de	6a	32	6d	d8	8a	84	72	2a	14	9f	88	f9	dc	89	9a
a0	fb	7c	2e	c3	8f	b8	65	48	26	c8	12	4a	ce	e7	d2	62
b0	0c	e0	1f	ef	11	75	78	71	a5	8e	76	3d	bd	bc	86	57
c0	0b	28	2f	a3	da	d4	e4	0f	a9	27	53	04	1b	fc	ac	e6
d0	7a	07	ae	63	c5	db	e2	ea	94	8b	c4	d5	9d	f8	90	6b
e0	b1	0d	d6	eb	c6	0e	cf	ad	08	4e	d7	e3	5d	50	1e	b3
f0	5b	23	38	34	68	46	03	8c	dd	9c	7d	a0	cd	1a	41	1c

Figure 4. Multiplicative Inverse Table GF(2^8). From http://www.samiam.org/galois.html#inverse

To calculate the multiplicative inverse is complicated and too much of a digression; therefore, a given table will be used instead. The field of all the multiplicative inverses is shown in Figure 4 [7].

Using an example of the hexadecimal value "31" from the multiplicative inverse table we get the value "45" from the table. This is the value we will use. Our affinity matrix is in binary therefore the number must be in binary as well. 45h is 0100 0101 in binary. An easier way to view the matrix multiplication is as polynomial multiplication. The value 0100 0101 can be represented as the polynomial $0x^7 + 1x^6 + 0x^5 + 0x^4 + 0x^3 + 1x^2 + 0x^1 + 1$. This can be rewritten and simplified as $x^6 + x^2 + 1$. Taking the first row of the affinity matrix we can assign each bit a variable. Therefore, 1 0 0 0 1 1 1 1 can be represented as N_0 N_1 N_2 N_3 N_4 N_5 N_6 N_7. This is generalized to all the rows, so all rows can be represented with the same variables in the same order. To do the multiplication start with the first row and see which variables are a 1 in the affinity matrix. Since the first row is 1 0 0 0 1 1 1 1, we only have to worry about the values N_0 N_4 N_5 N_6 N_7. Next, we look at the polynomial $x^6 + x^2 + 1$. The powers of this polynomial are what correspond to the subscripts of N. Wherever the power in the polynomial is also in the subscript of N we can assign a value of 1. Otherwise we assign the value of 0. This makes N_0 N_4 N_5 N_6 N_7 change to 1 0 0 1 0. A further shortcut can be taken to ignore all 0's completely. This can be done because addition in the field GF(2^8) is just the XOR operation. Therefore, adding an even number of 1's will result in a value of 0 and adding an odd number of 1's will result in a value of 1. Shown below in Figure 5 is the simplified mathematics for the entire affine transformation. [7]

Row 1 = 1 0 0 0 1 1 1 1 =	$N_0 + N_4 + N_5 + N_6 + N_7$	$1 + 0 + 0 + 1 + 0 = 0$
Row 2 = 1 1 0 0 0 1 1 1 =	$N_0 + N_1 + N_5 + N_6 + N_7$	$1 + 0 + 0 + 1 + 0 = 0$
Row 3 = 1 1 1 0 0 0 1 1 =	$N_0 + N_1 + N_2 + N_6 + N_7$	$1 + 0 + 1 + 1 + 0 = 1$
Row 4 = 1 1 1 1 0 0 0 1 =	$N_0 + N_1 + N_2 + N_3 + N_7$	$1 + 0 + 1 + 0 + 0 = 0$
Row 5 = 1 1 1 1 1 0 0 0 =	$N_0 + N_1 + N_2 + N_3 + N_4$	$1 + 0 + 1 + 0 + 0 = 0$
Row 6 = 0 1 1 1 1 1 0 0 =	$N_1 + N_2 + N_3 + N_4 + N_5$	$0 + 1 + 0 + 0 + 0 = 1$
Row 7 = 0 0 1 1 1 1 1 0 =	$N_2 + N_3 + N_4 + N_5 + N_6$	$1 + 0 + 0 + 0 + 1 = 0$
Row 8 = 0 0 0 1 1 1 1 1 =	$N_3 + N_4 + N_5 + N_6 + N_7$	$0 + 0 + 0 + 1 + 0 = 1$

Figure 5. Simplified Multiplication example. Rows are multiplied by the polynomial $x^6 + x^2 + 1$

The result of the multiplication is the vector 0 0 1 0 0 1 0 1. The final step to the transformation is to add the vector 1 1 0 0 0 1 1 0 to our vector. This gives us the final binary string of 1 1 1 0 0 0 1 1. Since we want the last row represented as the first bit we need to reverse this string giving us the new binary string of 1 1 0 0 0 1 1 1. In hexadecimal this value is C7. [7]

We are now finished. To know that we did everything right we look back to where we started. We started with the hexadecimal value of 31 from the original table and after all the work obtained the value C7. Therefore, if we look at the S-box value for 31, we should see C7. All the values in the S-box are calculated the same way we did for our example.

Using the previous example "This is a test…", the following block is a result of the S-box substitution. An ASCII lookup table was also used to convert the character values into hexadecimal values. The resulting block/state has all values shown in hexadecimal notation. Note that key addition was not applied. If this were a real implementation the key addition would be applied to the state before the first round. Therefore, the S-box substitution would be done with the state after the key addition. The block is shown in Figure 6.

	0	1	2	3
0	0x20	0xB7	0xEF	0x8F
1	0x45	0xF9	0xB7	0x92
2	0xF9	0x8F	0x92	0x31
3	0x8F	0xB7	0x4D	0x31

Figure 6. Block / State after S-box substitution

6: Rijndael Shifts

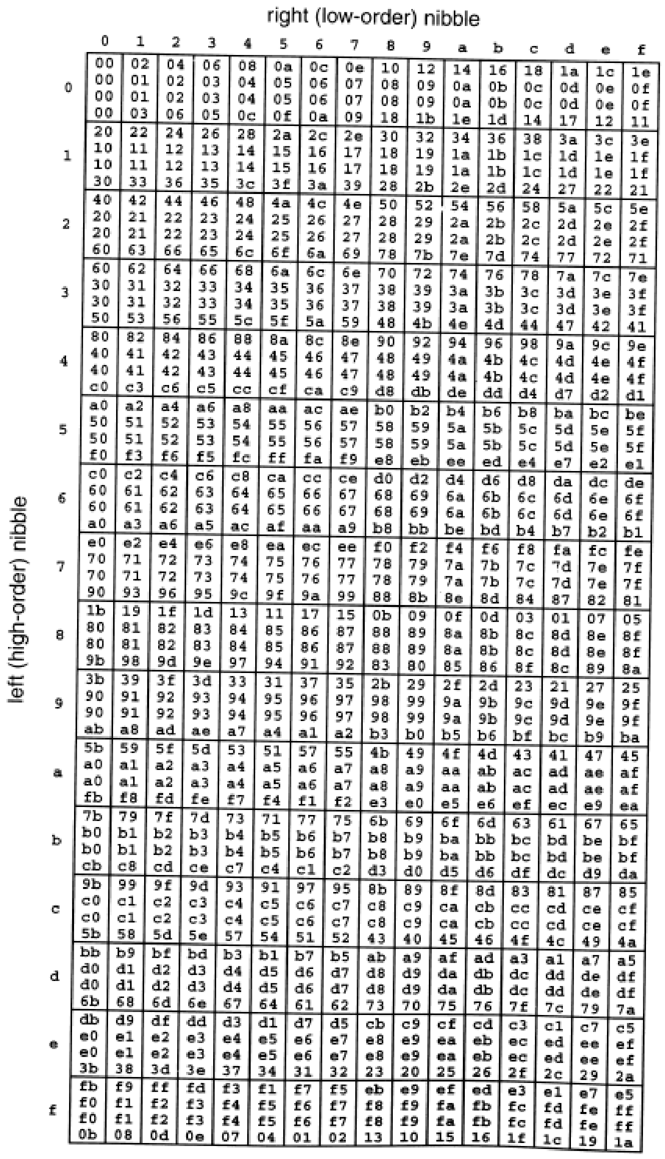
The next step in the round is to shift the rows of the state. The rows are shifted x number of bytes to the left where x is the row number. This means row 0 will not be shifted, row 1 will be shifted 1 byte to the left, row 2 will be shifted 2 bytes to the left, and row 3 will be shifted 3 bytes to the left. The resulting state is shown in Figure 7 by applying the shifts to the previous state. Note that the shifts do not affect the byte substitution so the shifting and S-box operations could be done in a different order. [9]

	0	1	2	3
0	0x20	0xB7	0xEF	0x8F
1	0xF9	0xB7	0x92	0x45
2	0x92	0x31	0xF9	0x8F
3	0x31	0x8F	0xB7	0x4D

Figure 7. Block/State after shifting

7 Rijndael MixColumn

After applying the S-box and shifts to the state the operation of a MixColumn is used. The MixColumn lookup table takes a byte and transforms it into four bytes. The MixColumn table is generated by the following algorithm. Each element in the table consists of four bytes usually represented as hexadecimal values. The second and third bytes are always same as the input byte. So for the value of 54 would result in a second and third value of 54 as well. The first byte is the input byte multiplied by 2. If the result is greater than 0xFF, the result is XORed with the value 0x1B and mod 100. Therefore our the byte 0x54 is (54 XOR 1b) mod 100 = A8. The last byte is the input byte added to the first byte [9]. So, the last byte is A8 XOR 54 = FC. This is a simple way to generate the MixColumn table shown in Figure 8.



right (low-order) nibble

left (high-order) nibble	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	00 00 00 00	02 01 01 03	04 02 02 06	06 03 03 05	08 04 04 0c	0a 05 05 0f	0c 06 06 0a	0e 07 07 09	10 08 08 18	12 09 09 1b	14 0a 0a 1e	16 0b 0b 1d	18 0c 0c 14	1a 0d 0d 17	1c 0e 0e 12	1e 0f 0f 11
1	20 10 10 30	22 11 11 33	24 12 12 36	26 13 13 35	28 14 14 3c	2a 15 15 3f	2c 16 16 3a	2e 17 17 39	30 18 18 28	32 19 19 2b	34 1a 1a 2e	36 1b 1b 2d	38 1c 1c 24	3a 1d 1d 27	3c 1e 1e 22	3e 1f 1f 21
2	40 20 20 60	42 21 21 63	44 22 22 66	46 23 23 65	48 24 24 6c	4a 25 25 6f	4c 26 26 6a	4e 27 27 69	50 28 28 78	52 29 29 7b	54 2a 2a 7e	56 2b 2b 7d	58 2c 2c 74	5a 2d 2d 77	5c 2e 2e 72	5e 2f 2f 71
3	60 30 30 50	62 31 31 53	64 32 32 56	66 33 33 55	68 34 34 5c	6a 35 35 5f	6c 36 36 5a	6e 37 37 59	70 38 38 48	72 39 39 4b	74 3a 3a 4e	76 3b 3b 4d	78 3c 3c 44	7a 3d 3d 47	7c 3e 3e 42	7e 3f 3f 41
4	80 40 40 c0	82 41 41 c3	84 42 42 c6	86 43 43 c5	88 44 44 cc	8a 45 45 cf	8c 46 46 ca	8e 47 47 c9	90 48 48 d8	92 49 49 db	94 4a 4a de	96 4b 4b dd	98 4c 4c d4	9a 4d 4d d7	9c 4e 4e d2	9e 4f 4f d1
5	a0 50 50 f0	a2 51 51 f3	a4 52 52 f6	a6 53 53 f5	a8 54 54 fc	aa 55 55 ff	ac 56 56 fa	ae 57 57 f9	b0 58 58 e8	b2 59 59 eb	b4 5a 5a ee	b6 5b 5b ed	b8 5c 5c e4	ba 5d 5d e7	bc 5e 5e e2	be 5f 5f e1
6	c0 60 60 a0	c2 61 61 a3	c4 62 62 a6	c6 63 63 a5	c8 64 64 ac	ca 65 65 af	cc 66 66 aa	ce 67 67 a9	d0 68 68 b8	d2 69 69 bb	d4 6a 6a be	d6 6b 6b bd	d8 6c 6c b4	da 6d 6d b7	dc 6e 6e b2	de 6f 6f b1
7	e0 70 70 90	e2 71 71 93	e4 72 72 96	e6 73 73 95	e8 74 74 9c	ea 75 75 9f	ec 76 76 9a	ee 77 77 99	f0 78 78 88	f2 79 79 8b	f4 7a 7a 8e	f6 7b 7b 8d	f8 7c 7c 84	fa 7d 7d 87	fc 7e 7e 82	fe 7f 7f 81
8	1b 80 80 9b	19 81 81 98	1f 82 82 9d	1d 83 83 9e	13 84 84 97	11 85 85 94	17 86 86 91	15 87 87 92	0b 88 88 83	09 89 89 80	0f 8a 8a 85	0d 8b 8b 86	03 8c 8c 8f	01 8d 8d 8c	07 8e 8e 89	05 8f 8f 8a
9	3b 90 90 ab	39 91 91 a8	3f 92 92 ad	3d 93 93 ae	33 94 94 a7	31 95 95 a4	37 96 96 a1	35 97 97 a2	2b 98 98 b3	29 99 99 b0	2f 9a 9a b5	2d 9b 9b b6	23 9c 9c bf	21 9d 9d bc	27 9e 9e b9	25 9f 9f ba
a	5b a0 a0 fb	59 a1 a1 f8	5f a2 a2 fd	5d a3 a3 fe	53 a4 a4 f7	51 a5 a5 f4	57 a6 a6 f1	55 a7 a7 f2	4b a8 a8 e3	49 a9 a9 e0	4f aa aa e5	4d ab ab e6	43 ac ac ef	41 ad ad ec	47 ae ae e9	45 af af ea
b	7b b0 b0 cb	79 b1 b1 c8	7f b2 b2 cd	7d b3 b3 ce	73 b4 b4 c7	71 b5 b5 c4	77 b6 b6 c1	75 b7 b7 c2	6b b8 b8 d3	69 b9 b9 d0	6f ba ba d5	6d bb bb d6	63 bc bc df	61 bd bd dc	67 be be d9	65 bf bf da
c	9b c0 c0 5b	99 c1 c1 58	9f c2 c2 5d	9d c3 c3 5e	93 c4 c4 57	91 c5 c5 54	97 c6 c6 51	95 c7 c7 52	8b c8 c8 43	89 c9 c9 40	8f ca ca 45	8d cb cb 46	83 cc cc 4f	81 cd cd 4c	87 ce ce 49	85 cf cf 4a
d	bb d0 d0 6b	b9 d1 d1 68	bf d2 d2 6d	bd d3 d3 6e	b3 d4 d4 67	b1 d5 d5 64	b7 d6 d6 61	b5 d7 d7 62	ab d8 d8 73	a9 d9 d9 70	af da da 75	ad db db 76	a3 dc dc 7f	a1 dd dd 7c	a7 de de 79	a5 df df 7a
e	db e0 e0 3b	d9 e1 e1 38	df e2 e2 3d	dd e3 e3 3e	d3 e4 e4 37	d1 e5 e5 34	d7 e6 e6 31	d5 e7 e7 32	cb e8 e8 23	c9 e9 e9 20	cf ea ea 25	cd eb eb 26	c3 ec ec 2f	c1 ed ed 2c	c7 ee ee 29	c5 ef ef 2a
f	fb f0 f0 0b	f9 f1 f1 08	ff f2 f2 0d	fd f3 f3 0e	f3 f4 f4 07	f1 f5 f5 04	f7 f6 f6 01	f5 f7 f7 02	eb f8 f8 13	e9 f9 f9 10	ef fa fa 15	ed fb fb 16	e3 fc fc 1f	e1 fd fd 1c	e7 fe fe 19	e5 ff ff 1a

Figure 8. MixColumn Table. Image from *Network Security Private Communication in a Public World*, page 86

What is really happening is a matrix multiplication of the input with the matrix in Figure 9. For our purposes the short cut is more practical at generating the table. It also avoids the multiplication operation which is somewhat complicated in the field GF(2^8).

$$\begin{matrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{matrix}$$

Figure 9. Matrix for Generating the MixColumn Table. From http://www.samiam.org/mix-column.html

The next question is how to use the MixColumn table. For our example we need to split up out state into columns. Each column will have its own MixColumn operation applied to it. Using the lookup table for all the values in the first column of our state we get the following.

```
0x20
    40
    20
    20
    60
0xF9
    E9
    F9
    F9
    10
0x92
    3F
    92
    92
    AD
0x31
    62
    31
    31
    53
```

Figure 10. First column transformed with the MixColumn table

The MixColumn step is best shown as a diagram. The diagram is shown in Figure 11.

```
                    40 + B3 = F3
               E9 + 20  + 9C = 55
          3F + F9  + 20  + 53 = B5
     62 + 92 + F9  + 60         =  69
     31 + 92 + 10 = B3
     31 + AD = 9C
     53
```

Figure 11. MixColumn operation

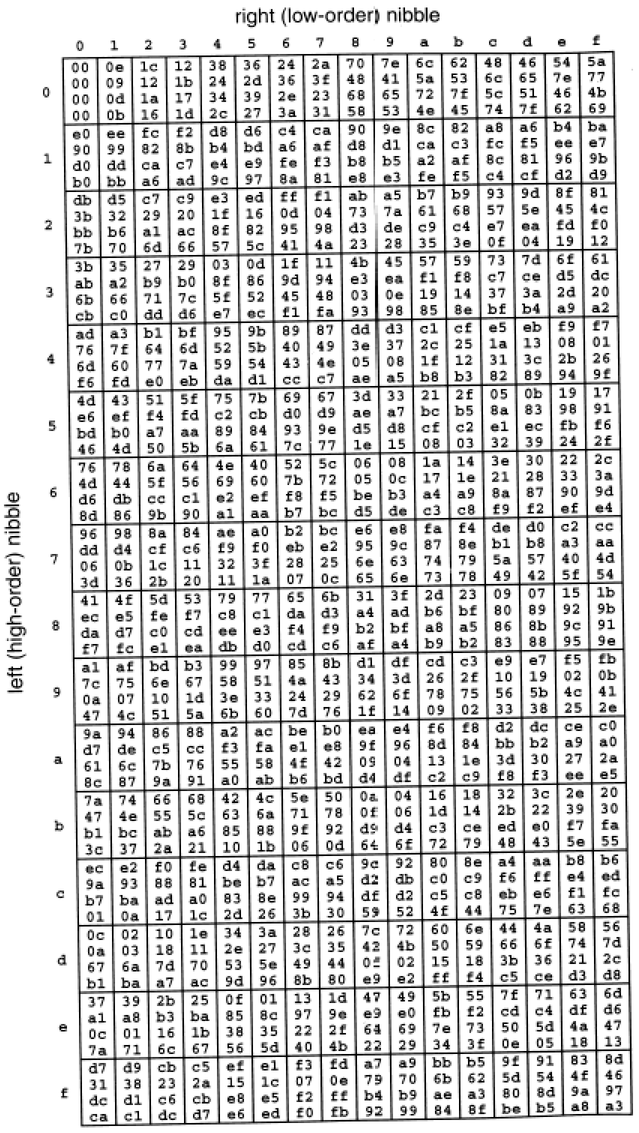
The result of using the MixColumn operation on the first column is the bytes 0x49, 0x34, 0xA7, and 0xBD. The blue values in the diagram indicate that result is used in the final addition. Similarly the other columns in the state go through the same process. The result is a state with the new values after the MixColumn process.

8 Rijndael Decryption

Decryption is simple after understanding the encryption process. It is basically just the inverse. The algorithm was designed for all the steps to be invertible so decryption is basically like doing everything backwards. Therefore, for decryption starts at the last round and the last round key. When processing each round do the process backwards. So, the round key is added first to the last round. Addition is its own inverse, which is nice. Then the MixColumn step is applied. The MixColumn step is applied to all rounds except the last one. Also the inverse MixColumn table is used [5]. This table is generated with another matrix similar to the way the MixColumn table was generated. The difference is that there are no short cuts to generate the table. Therefore, the matrix multiplication needs to be performed in the field GF(2^8). The matrix is shown in Figure 12 and the inverse table is shown in Figure 13. [10]

$$\begin{matrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{matrix}$$

Figure 12. Matrix for Generating the Inverse MixColumn Table. This is the integer representation of the hexadecimal data in the table from http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, Figure 5.10.



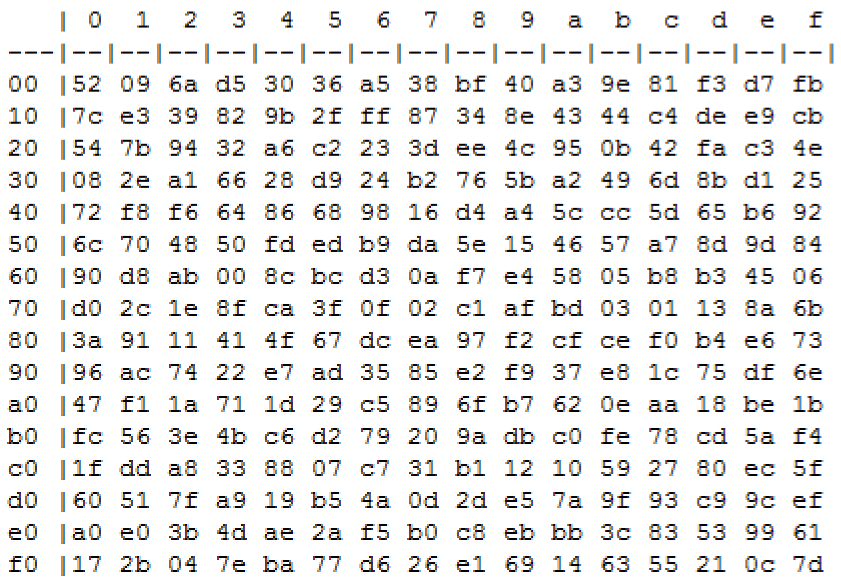
right (low-order) nibble

left (high-order) nibble

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	00 00 00 00	0e 09 0d 0b	1c 12 1a 16	12 1b 17 1d	38 24 34 2c	36 2d 39 27	24 36 2e 3a	2a 3f 23 31	70 48 68 58	7e 41 65 53	6c 5a 72 4e	62 53 7f 45	48 6c 5c 74	46 65 51 7f	54 7e 46 62	5a 77 4b 69
1	e0 90 d0 b0	ee 99 dd bb	fc 82 ca a6	f2 8b c7 ad	d8 b4 e4 9c	d6 bd e9 97	c4 a6 fe 8a	ca af f3 81	90 d8 b8 e8	9e d1 b5 e3	8c ca a2 fe	82 c3 af f5	a8 fc 8c c4	a6 f5 81 cf	b4 ee 96 d2	ba e7 9b d9
2	db 3b bb 7b	d5 32 b6 70	c7 29 a1 6d	c9 20 ac 66	e3 1f 8f 57	ed 16 82 5c	ff 0d 95 41	f1 04 98 4a	ab 73 d3 23	a5 7a de 28	b7 61 c9 35	b9 68 c4 3e	93 57 e7 0f	9d 5e ea 04	8f 45 fd 19	81 4c f0 12
3	3b ab 6b cb	35 a2 66 c0	27 b9 71 dd	29 b0 7c d6	03 8f 5f e7	0d 86 52 ec	1f 9d 45 f1	11 94 48 fa	4b e3 03 93	45 ea 0e 98	57 f1 19 85	59 f8 14 8e	73 c7 37 bf	7d ce 3a b4	6f d5 2d a9	61 dc 20 a2
4	ad 76 6d f6	a3 7f 60 fd	b1 64 77 e0	bf 6d 7a eb	95 52 59 da	9b 5b 54 d1	89 40 43 cc	87 49 4e c7	dd 3e 05 ae	d3 37 08 a5	c1 2c 1f b8	cf 25 12 b3	e5 1a 31 82	eb 13 3c 89	f9 08 2b 94	f7 01 26 9f
5	4d e6 bd 46	43 ef b0 4d	51 f4 a7 50	5f fd aa 5b	75 c2 89 6a	7b cb 84 61	69 d0 93 7c	67 d9 9e 77	3d ae d5 1e	33 a7 d8 15	21 bc cf 08	2f b5 c2 03	05 8a e1 32	0b 83 ec 39	19 98 fb 24	17 91 f6 2f
6	76 4d d6 8d	78 44 db 86	6a 5f cc 9b	64 56 c1 90	4e 69 e2 a1	40 60 ef aa	52 7b f8 b7	5c 72 f5 bc	06 05 be d5	08 0c b3 de	1a 17 a4 c3	14 1e a9 c8	3e 21 8a f9	30 28 87 f2	22 33 90 ef	2c 3a 9d e4
7	96 dd 06 3d	98 d4 0b 36	8a cf 1c 2b	84 c6 11 20	ae f9 32 11	a0 f0 3f 1a	b2 eb 28 07	bc e2 25 0c	e6 95 6e 65	e8 9c 63 6e	fa 87 74 73	f4 8e 79 78	de b1 5a 49	d0 b8 57 42	c2 a3 40 5f	cc aa 4d 54
8	41 ec da f7	4f e5 d7 fc	5d fe c0 e1	53 f7 cd ea	79 c8 ee db	77 c1 e3 d0	65 da f4 cd	6b d3 f9 c6	31 a4 b2 af	3f ad bf a4	2d b6 a8 b9	23 bf a5 b2	09 80 86 83	07 89 8b 88	15 92 9c 95	1b 9b 91 9e
9	a1 7c 0a 47	af 75 07 4c	bd 6e 10 51	b3 67 1d 5a	99 58 3e 6b	97 51 33 60	85 4a 24 7d	8b 43 29 76	d1 34 62 1f	df 3d 6f 14	cd 26 78 09	c3 2f 75 02	e9 10 56 33	e7 19 5b 38	f5 02 4c 25	fb 0b 41 2e
a	9a d7 61 8c	94 de 6c 87	86 c5 7b 9a	88 cc 76 91	a2 f3 55 a0	ac fa 58 ab	be e1 4f b6	b0 e8 42 bd	ea 9f 09 d4	e4 96 04 df	f6 8d 13 c2	f8 84 1e c9	d2 bb 3d f8	dc b2 30 f3	ce a9 27 ee	c0 a0 2a e5
b	7a 47 b1 3c	74 4e bc 37	66 55 ab 2a	68 5c a6 21	42 63 85 10	4c 6a 88 1b	5e 71 9f 06	50 78 92 0d	0a 0f d9 64	04 06 d4 6f	16 1d c3 72	18 14 ce 79	32 2b ed 48	3c 22 e0 43	2e 39 f7 5e	20 30 fa 55
c	ec 9a b7 01	e2 93 ba 0a	f0 88 ad 17	fe 81 a0 1c	d4 be 83 2d	da b7 8e 26	c8 ac 99 3b	c6 a5 94 30	9c d2 df 59	92 db d2 52	80 c0 c5 4f	8e c9 c8 44	a4 f6 eb 75	aa ff e6 7e	b8 e4 f1 63	b6 ed fc 68
d	0c 0a 67 b1	02 03 6a ba	10 18 7d a7	1e 11 70 ac	34 2e 53 9d	3a 27 5e 96	28 3c 49 8b	26 35 44 80	7c 42 05 e9	72 4b 02 e2	60 50 15 ff	6e 59 18 f4	44 66 3b c5	4a 6f 36 ce	58 74 21 d3	56 7d 2c d8
e	37 a1 0c 7a	39 a8 01 71	2b b3 16 6c	25 ba 1b 67	0f 85 38 56	01 8c 35 5d	13 97 22 40	1d 9e 2f 4b	47 e9 64 22	49 e0 69 29	5b fb 7e 34	55 f2 73 3f	7f cd 50 0e	71 c4 5d 05	63 df 4a 18	6d d6 47 13
f	d7 31 dc ca	d9 38 d1 c1	cb 23 c6 dc	c5 2a cb d7	ef 15 e8 e6	e1 1c e5 ed	f3 07 f2 f0	fd 0e ff fb	a7 79 b4 92	a9 70 b9 99	bb 6b ae 84	b5 62 a3 8f	9f 5d 80 be	91 54 8d b5	83 4f 9a a8	8d 46 97 a3

Figure 13. Inverse MixColumn Table. From *Network Security Private Communication in a Public World*, p. 88

All the shifts are done backwards as well. So, instead of shifting left, we shift right. Lastly, the S-box is applied using the inverse S-box table. The inverse S-box table can easily be generated by taking the S-box value at some row and column index and assigning the row and column to the inverse S-box value at the inverse S-box index defined by the S-box value. For example, the S-box has a value of 0x00 at index 5,2. This translates to the inverse S-box having the value 0x52 at index 0,0. The inverse S-box table is shown in Figure 14 [2]. After all the rounds have been completed in opposite order the final state will contain the original plain text.



```
   | 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
---|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
00 |52 09 6a d5 30 36 a5 38 bf 40 a3 9e 81 f3 d7 fb
10 |7c e3 39 82 9b 2f ff 87 34 8e 43 44 c4 de e9 cb
20 |54 7b 94 32 a6 c2 23 3d ee 4c 95 0b 42 fa c3 4e
30 |08 2e a1 66 28 d9 24 b2 76 5b a2 49 6d 8b d1 25
40 |72 f8 f6 64 86 68 98 16 d4 a4 5c cc 5d 65 b6 92
50 |6c 70 48 50 fd ed b9 da 5e 15 46 57 a7 8d 9d 84
60 |90 d8 ab 00 8c bc d3 0a f7 e4 58 05 b8 b3 45 06
70 |d0 2c 1e 8f ca 3f 0f 02 c1 af bd 03 01 13 8a 6b
80 |3a 91 11 41 4f 67 dc ea 97 f2 cf ce f0 b4 e6 73
90 |96 ac 74 22 e7 ad 35 85 e2 f9 37 e8 1c 75 df 6e
a0 |47 f1 1a 71 1d 29 c5 89 6f b7 62 0e aa 18 be 1b
b0 |fc 56 3e 4b c6 d2 79 20 9a db c0 fe 78 cd 5a f4
c0 |1f dd a8 33 88 07 c7 31 b1 12 10 59 27 80 ec 5f
d0 |60 51 7f a9 19 b5 4a 0d 2d e5 7a 9f 93 c9 9c ef
e0 |a0 e0 3b 4d ae 2a f5 b0 c8 eb bb 3c 83 53 99 61
f0 |17 2b 04 7e ba 77 d6 26 e1 69 14 63 55 21 0c 7d
```

Figure 14. Inverse S-box. From http://www.samiam.org/s-box.html

9 Conclusion

The Rijndael algorithm was chosen as the new Advanced Encryption Standard (AES) for several reasons. The purpose was to create an algorithm that was resistant against known attacks, simple, and quick to code. Choosing to use field GF(2)^8 was a very good decision. The inverse of the addition operation was itself, making much of the algorithm easy to do. In fact, every operation is invertible by design. In addition, the block size and key size can vary making the algorithm versatile. AES was originally designed for non-classified U.S. government information, but, due to its success, AES-256 is usable for top secret government information [11]. As of July 2009, no practical attacks have been successful on AES [12].

10 Code Samples

The code samples (below) have been developed by the author. The following Java-code module creates the Inverse S-box from the original S-box. Both "boxes" store the values of integers.

```
public void createInverseSBox()
{
    for(int i=0; i<16; i++)
    {
        for(int j=0; j<16; j++)
        {
            inverseSBox[i][j] = findInverseLocation(i*16 + j);
        }

    }
 }
```

This program finds the inverse location for creating the inverse S-box:

```
public int findInverseLocation(int num)
{
    int spotI = -1;
    int spotJ = -1;

    for(int i=0; i<16; i++)
    {
        for(int j=0; j<16; j++)
        {
            if(sBox[i][j] == num)
            {
                spotI = i;
                spotJ = j;
            }
        }
    }

    return spotI * 16 + spotJ;
}
```

This program generates the MixColumn table:

```
public void createMixColumn()
{
    for(int i=0; i<16; i++)
    {
        for(int j=0; j<16; j++)
        {
            mixColumn[i][j][0] = (i*16 + j)*2;

            if(mixColumn[i][j][0] >= 128)
            {
                mixColumn[i][j][0] = (mixColumn[i][j][0] ^ 27) % 256;
            }

            mixColumn[i][j][1] = i*16 + j;
            mixColumn[i][j][2] = i*16 + j;

            mixColumn[i][j][3] = (mixColumn[i][j][0] ^ mixColumn[i][j][1]);
        }
    }
}
```

This module performs the S-box substitution with a byte represented as an integer:

```
public int sboxSubstitution(int plainText)
{
    int tableSpotI = getIValue(plainText);
    int tableSpotJ = plainText % 16;

    return sBox[tableSpotI][tableSpotJ];
}
```

This program creates the blocks out of the plain text represented as a character array:

```
public int[][][] createStates(char[] textArray, int numStates)
{
    int states[][][] = new int[numStates][4][4];
    int z = 0;
    //state,row,column

    for(int i=0; i<numStates; i++)
    {
        for(int j=0; j<4; j++)
        {
            for(int k=0; k<4; k++)
            {
                states[i][k][j] = (int)textArray[z];
                z++;
            }
        }
    }

    return states;
}
```

This module performs a Rijndael round, but it is not completely finished yet:

```
public int[][][] round(int sNum, int roundNum, int states[][][], /*expanded key*/)
{
    for(int i=0; i<4; i++)
    {
        for(int j=0; j<4; j++)
        {
            states[sNum][i][j] = sboxSubstitution(states[sNum][i][j]);
        }
    }

    for(int i=1; i<4; i++)
    {
        for(int j=0; j<i; j++)
        {
            int temp = states[sNum][i][0];

            states[sNum][i][0] = states[sNum][i][1];
            states[sNum][i][1] = states[sNum][i][2];
            states[sNum][i][2] = states[sNum][i][3];
            states[sNum][i][3] = temp;
        }
    }

    if(roundNum != 10)
    {
        //mix column
    }
    //add expanded key

    return states;
}
```

References

1. *DES Encryption*. Tropical Software. 2010. http://www.tropsoft.com/strongenc/des.htm (accessed March, 15, 2010).
2. Kaufman, C., Perlman, R., and Speciner, M. *Network Security: Private Communication in a Public World*. 2nd ed. Upper Saddle River, N.J.: Prentice Hall PTR, 2002.
3. *Rijndael*. Knowledgerush. 2009. http://www.easybib.com/cite/form/website (accessed March, 15, 2010).
4. *Advanced Encryption Standard (AES)*. FIPS. November 23, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (accessed March, 15, 2010).
5. Daemen, J., and Rijmen, V. *AES Proposal: Rijndael*. September 3, 1999. http://www.comms.scitech.sussex.ac.uk/fft/crypto/rijndael.pdf (accessed March, 15, 2010).
6. *Using Padding in Encryption*. DI Management. January 3, 2010. http://www.di-mgt.com.au/cryptopad.html (accessed March, 15, 2010).
7. Trenholme, S. "S-box." *AES*. 2010. http://www.samiam.org/s-box.html (accessed March, 15, 2010).
8. "Finite Field — from Wolfram MathWorld". *Wolfram MathWorld: The Web's Most Extensive Mathematics Resource*. March 3, 2010. http://mathworld.wolfram.com/FiniteField.html (accessed March, 15, 2010).
9. Trenholme, S. "Rijndael's MixColumn Stage." *AES*. 2010. http://www.samiam.org/mix-column.html (accessed March, 15, 2010).
10. *The Advanced Encryption Standard (Rijndael)*. 2010. http://www.quadibloc.com/crypto/co040401.htm (accessed March, 15, 2010).
11. *CNSS Policy No. 15, Fact Sheet No. 1*. June 2003. http://www.cnss.gov/Assets/pdf/cnssp_15_fs.pdf (accessed March, 15, 2010).
12. Schneier, B. "Another New AES Attack." *Schneier on Security*. July 30, 2009. http://www.schneier.com/blog/archives/2009/07/another_new_aes.html (accessed March, 15, 2010).

* **DOUGLAS SELENT** received his B.S. in Computer Science at Merrimack College in 2009. He is currently in the M.S. program for Computer Science at Rivier College and plans to go for his Ph.D. in the near future. Doug's favorite subject in Computer Science is algorithms. His favorite thing to do is to create his own algorithms to solve unique problems. In his free time he likes to casually play video games and watch the Patriots win on Sunday.