

REDNOUR, STEPHANIE D., M.S. An Analysis of a Sparse Linearization Attack on the Advanced Encryption Standard. (2006)
Directed by Dr. Shanmugathan Suthaharan. 106 pp.

Since Rijndael was accepted as the new Advanced Encryption Standard by the NIST, several techniques have been developed to attack it. One of the more controversial techniques is a relatively new mathematically based attack known as Extended Sparse Linearization, or XSL. Estimates for a successful attack on AES using XSL are extremely large (best estimate is 2^{100} encryptions), so no attempt to implement the attack has yet been made.

To show that the attack is viable, a reduced version of AES can be implemented and a modification of the XSL attack can be used on the reduced version. I have implemented the reduced version of AES, referred to as rAES, as well as the attack. In this document it will be shown that the attack fails. Since the attack failed on the reduced version, the result can be extended to show that it cannot be made on the full version either.

AN ANALYSIS OF A SPARSE LINEARIZATION ATTACK
ON THE ADVANCED ENCRYPTION STANDARD

by

Stephanie D. Rednour

A Thesis Submitted to
the Faculty of the Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Greensboro
2006

Approved by

Committee Chair

To my parents Tom and Bonnie Rednour

And

To Tavis Curry

APPROVAL PAGE

This Thesis has been approved by the following committee of the Faculty
of the Graduate School at The University of North Carolina at Greensboro.

Committee Chair _____

Committee Members _____

Date of Acceptance by Committee

Date of Final Oral Examination

ACKNOWLEDGEMENTS

Thanks to my advisor Shanmugathan Suthaharan and to my committee members Lixin Fu and Fereidoon Sadri.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vi
LIST OF FIGURES	vii
CHAPTER	
I. INTRODUCTION	1
The Advanced Encryption Standard	2
The Linearization Techniques	10
II. REDUCED AES	13
III. APPLIED ATTACK	20
The Equations	20
The Attack	23
IV. RESULTS	29
V. CONCLUSIONS	30
General Conclusions	31
Future Work.....	32
BIBLIOGRAPHY	33
APPENDIX A. ADDITIONAL TABLES AND FIGURES	35
APPENDIX B. INPUT DATA	41
APPENDIX C OUTPUT DATA	69

LIST OF TABLES

	Page
TABLE	
1. The Round Constants	7
2. Multiplication in $GF(2^4)$	15
3. The rAES Round Constants	18
4. Matrix Terms - First Half	39
5. Matrix Terms - Second Half	40

LIST OF FIGURES

	Page
FIGURES	
1. The AES SBox.....	3
2. State matrix before and after ShiftRows	4
3. MixColumns Matrix	5
4. State matrix before and after InverseShiftRows.....	8
5. Inverse SBox	9
6. InverseMixColumns Matrix.....	9
7. Matrix Representation of SBox Derivation	17
8. 4-Bit SBox.....	17
9. 4-Bit Inverse SBox	17
10. Modified Gaussian Elimination Version 1.....	24
11. Modified Gaussian Elimination Version 2.....	27
12. All SBox Terms and Mappings.....	35
13. SBox Equations - Set One	36
14. Equations after the SBox	36
15. SBox Equations – Set Two	37
16. ExpandRoundKey Equations	38

CHAPTER I

INTRODUCTION

After investigating several cryptographic algorithms for possible research topics, the Advanced Encryption Standard (AES) was chosen. The AES was selected because it is relatively new and untested, providing the best opportunity in finding a research topic. Once the AES was chosen, the algorithm was examined in detail. The AES was implemented using Java so that the actual running of the algorithm for the encryption and decryption could be observed.

Then others' work into attacking the AES was examined. Several attack techniques have been proposed for the AES including Extended Linearization [4], Extended Sparse Linearization [4], power attacks [2], and a modification of the AES to improve the running time for Extended Sparse Linearization called the Better Encryption Standard (BES) [9]. While investigating the details of these approaches, several authors called for research into applying the XSL technique on a real system [3, 8, 11, 12], to see if it worked in an applied fashion and not just as theory. Therefore it was decided to pursue an attack that would be similar to XSL on an encryption algorithm like the AES to see if the attack works.

The Advanced Encryption Standard

In the late 1990's it became apparent to the National Institute of Standards and Technology (NIST) that the encryption standard, DES, was no longer sufficiently secure. Therefore they issued a request for an algorithm to replace DES. Once agreed upon the new standard would be referred to as the Advanced Encryption Standard. Several algorithms were submitted to become AES, and through a tiered process of elimination ultimately one was chosen. This was the submission from Joan Daemen and Vincent Rijmen and they called their algorithm Rijndael, a combination of both of their last names.

Some modifications were made that restricted the original Rijndael so the AES is not exactly the same as the original submission. For the purposes of this paper we are concerned with the final version in the AES, released by the NIST in November of 2001 [14]. The AES algorithm is a block cipher, which means that the original plaintext message is broken down into blocks of a fixed size, and then the algorithm processes the blocks individually. In the AES the block size is 128 bits. Each block is handled exactly the same way, so we need only concern ourselves with how one block is processed. The AES encryption performs several operations on the block, which are described as four steps. These steps are called SubBytes (Substitute Bytes), ShiftRows, MixColumns, and AddRoundKey. The steps are performed repeatedly on the block. Each repetition is referred to as a round. There is one deviance in the pattern of steps. In the last round the MixColumns step is not performed. Similarly, the AES

decryption performs four steps, three of them are different, called InverseSubBytes, InverseShiftRows, InverseMixColumns, and the last step is the same as the encryption's AddRoundKey.

The first step of the encryption, SubBytes, breaks the 128-bit block into bytes and performs a substitution on each byte. This substitution is made using a look-up table called the SBox. For any byte, the first four bits represent the row in the table and the last four bits represent the column. The intersection of the row and column gives you the output byte. For example, the input byte 00101010 would be split into row 0010 and column 1010. For clarity the figure of the SBox below uses hexadecimal entries so we have row 2 and column A. As you can see in Figure 1, 2A maps to E5 so the output byte is 11100101.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figure 1: The AES SBox

The second step, ShiftRows, takes the bytes in the block and mixes them within their own row. For this step, and the MixColumns step, it is simpler to understand the process if you consider the 128 bits of the block as a matrix, with four bytes per row and four bytes per column. As you can see in Figure 2, ShiftRows leaves the first row of the matrix alone, but the rest of the rows are shuffled. Each entry in the figure represents one byte, so that the top row is composed of bits 0-31 of the block, row two is bits 32-63, row three is bits 64-95, and row four is bits 96-127.

b0	b1	b2	b3
b4	b5	b6	b7
b8	b9	b10	b11
b12	b13	b14	b15

b0	b1	b2	b3
b5	b6	b7	b4
b10	b11	b8	b9
b15	b12	b13	b14

Figure 2: State matrix before and after ShiftRows.

The third step, MixColumns, performs mathematical operations on the columns of the result of the ShiftRows step. This is essentially a matrix multiplication. The input state matrix to MixColumns is multiplied by a given matrix and the result is the output of MixColumns. You can see the matrix used for the multiplication in Figure 3. For instance, the first entry in the output matrix would be calculated as follows:

$$(s_0 \cdot 02) \oplus (s_4 \cdot 03) \oplus s_8 \oplus s_{12}$$

where s_0 is the first byte from the ShiftRows output, s_4 is the fifth byte, and so on.

The addition is done using a bit wise exclusive-or (\oplus). The matrix multiplication is done in the Galois Field $GF(2^8)$.

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

Figure 3: MixColumns Matrix (in hexadecimal)

A field is a set with two binary operations, usually referred to as addition and multiplication, which operate on that set. The two operations must not have results that are outside of the set and there must be a multiplicative inverse for each element in the set [13]. A Galois Field is a finite field, which means that the set of elements of the field is finite. The set for $GF(2^8)$ contains the integers from zero to 255.

One does not need to understand all the properties of a Galois Field in order to perform the multiplication, there are just two rules to use. First, one finds the results of multiplying the number by the powers of two. If the result is greater than 255, then an additional operation must be performed. This operation is to add the result to a given constant, and throw away the bits over eight.

For example,

$$4 \cdot 8 = 00000100 \cdot 00001000 = 00100000 = 32, \text{ but}$$

$$4 \cdot 64 = 00000100 \cdot 00100000 = 100000000$$

so we take that result and add it to the constant 100011011 as defined in the AES. Then one has $100000000 \oplus 100011011 = 00011011 = 27$.

To perform multiplications with numbers that aren't powers of two, one just use a bit-wise exclusive-or with the powers of two that make up that number. For example,

$$\begin{aligned} 4 \cdot 6 &= (00000100 \cdot 00000100) \oplus (00000100 \cdot 00000010) = \\ &00010000 \oplus 00001000 = 00011000 = 24 \end{aligned}$$

which is what one would expect.

The last step is the AddRoundKey, which takes the result of the MixColumns step and performs an exclusive-or with a secret key. The secret key is where all the security of the AES lies. Without the secret key anyone could perform the above operations and get the same results. The secret key needs to be chosen then so that no one can guess it. The use of one secret key in this way makes the AES a private key system. This means that the key must be kept private. In the AES, the secret key is initially either 128, 192, or 256 bits long. An operation called ExpandRoundKey is performed on the key to generate several round keys. One round key is used in each of the AddRoundKey steps since the 4 steps are performed repeatedly. The number of times the four steps are repeated is based on the size of the initial key. Each repetition is referred to

as a round. For a 128-bit key there are 10 rounds, for a 192-bit key there are 12 rounds, and for a 256-bit key there are 14 rounds. The original key is added to the plaintext before the first round, so there are actually 11, 13, and 15 keys used, respectively. Keep in mind, all the round keys are derived mathematically from the original key, so if you have either the round keys or the original key you can calculate the others.

j	1	2	3	4	5	6	7	8	9	10
$Rcon_j$	1	2	4	8	10	20	40	80	1B	36

Table 1: The Round Constants for the 10-round version of AES (in hexadecimal)

The ExpandRoundKey operation is performed only once. The operation works on a row of the original key at a time. A row is also referred to as a word, which is 32-bits long in the 128-bit key case. Therefore the original key can be expressed as four words, which we will call w_0 , w_1 , w_2 , and w_3 . We will index the round keys starting from w_4 for the first word of the first round key (so then the second round key starts at w_8 .) The first three words in each round key are calculated the same way as follows:

$$w_i = w_{i-4} \oplus w_{i-1}$$

where i ranges from 0 to 44 for a 128-bit key. The last word in each round key uses a more complex calculation:

$$w_i = w_{i-4} \oplus \text{SubWord}(\text{RotWord}(w_{i-1})) \oplus Rcon_{i/4}$$

RotWord simply performs a one byte circular left shift on the word. SubWord uses the SBox to substitute each of the bytes for a new byte in the same way as described above. The $Rcon_{i/4}$ is a constant. The constants for the 128-bit key are given in Table 1.

All of the above refers to the encryption of data using the AES, but that is not very useful if you cannot perform a decryption that gives you back the original data. The decryption is very similar to the encryption, but it is not exactly the same. Essentially, all of the steps in the encryption must be performed backwards to get the plaintext again. The input to the decryption is the ciphertext. The last round key is added back to it since the inverse of exclusive-or is just another exclusive-or. Since we are undoing the last round, we do not need to perform the InverseMixColumns step yet, so we perform the InverseShiftRows step. This step just performs the shifts in the opposite direction as can be seen in Figure 4.

b0	b1	b2	b3
b4	b5	b6	b7
b8	b9	b10	b11
b12	b13	b14	b15

b0	b1	b2	b3
b7	b4	b5	b6
b10	b11	b8	b9
b13	b14	b15	b12

Figure 4: State matrix before and after InverseShiftRows.

The next step to perform is InverseSubBytes. This step works the same way as SubBytes but it uses the Inverse SBox to do the substitution. The

Inverse SBox can be seen in Figure 5. If we take the example from before you can see how the Inverse SBox reverses the SBox effect on the bits. So the input byte 11100101 would be split into row 1110 and column 0101. In hexadecimal this is row E column 5. As you can see in Figure 5, E5 maps to 2A, which is what we had input to the SBox before.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Figure 5: Inverse SBox

The third operation is to add the second to last round key to the output of the Inverse SBox. Now we are in the second to last round, so we need to perform the InverseMixColumns. InverseMixColumns works the same way as

MixColumns, but a different matrix is used for the multiplication. This matrix, shown in Figure 6, is the inverse of the MixColumns matrix.

From the second round of the decryption on all of the steps are performed, with an additional AddRoundKey done at the very end which will produce the plaintext message.

0E	0B	0D	09
09	0E	0B	0D
0D	09	0E	0B
0B	0D	09	0E

Figure 6: InverseMixColumns Matrix (in hexadecimal)

The Linearization Techniques

Once Rijndael was submitted to NIST, others began examining it for weaknesses. It was noted in [10] that Rijndael had an unusual structure that allows it to be expressed as equations because it can be broken into distinct parts. This observation led to a technique to attempt to break the encryption using these equations. One of the techniques proposed for an attack on Rijndael was Extended Linearization, or XL [4]. This technique was devised for other encryption algorithms and was applied to the AES in [4].

XL converts the operations performed on the block into linear equations. The derivation of the equations (which is the same for XL and XSL) is detailed in Chapter III. The hope is that this large system of equations can be solved, which

results in obtaining the key. The system of equations is represented as a matrix since it is linear. A technique for diagonalizing the matrix, such as Gaussian Elimination, is applied to see if the system is solvable. If the system is solved then the key has been found and the encryption is cracked. If the system is not solved, an attempt to change the system so that it is solvable is made. This is done by solving the system for one term, then multiplying all of the equations by all the possible second order terms. If any new equations result, they are added to the system. Then the Gaussian Elimination is tried on the updated system. In this fashion the process repeats until the system is solved.

When analyzing the XL technique, the authors estimated that it would be better than brute force, but still outside the realm of feasibility. A second technique was designed by Courtois and Pieprzyk [4] to improve upon XL, which was termed XSL or Extended Sparse Linearization. This technique also attempts to solve a large system of linear equations to obtain the key, but it improves upon the method of increasing the number of equations to speed up the process. In both techniques the system of linear equations is considered sparse. This means that for any given equation in the system only a relatively small number of terms are present. Such a system is shown in Appendix A and you can see it is composed largely of zeroes, meaning those terms have a zero coefficient.

In XSL, once a certain number of linearly independent equations are determined for the system, the two alternating steps are used to solve the system. First, the same as XL, an algorithm to attempt to solve the matrix

representing the equations is used, such as Gaussian Elimination. If the system is unsolvable, because there aren't enough equations compared to the number of terms, the T' Method is used to increase the number of equations to enough to solve all of the terms in the system. The T' Method is where XSL differs from XL. In this method the equations are all solved relative to two terms. This gives two equivalent systems. Next, the terms from the second system, instead of all possible terms, are used to increase the number of equations by multiplying the terms in the second system by the equations in the second system (see [16] for an example of this method). By repeating the Gaussian Elimination and T' Method steps the hope is that eventually a solvable system will be created. For the AES the estimate given by Murphy and Robshaw for creating a solvable system is 2^{100} [9].

CHAPTER II

REDUCED AES

As was stated in the previous chapter, the estimate for the number of executions needed to find the key for the 10-round AES is 2^{100} . This is significantly less than a brute force attack, but it would still require too long to be feasible. To show whether or not there is an attack against the AES that works when applied, the running time of the attack needed to be reduced.

To decrease the running time to something that was feasible there are two choices, to modify the attack or to modify the algorithm the attack is on. For XSL the running time is tied to the size of the system of equations and the number of terms in those equations. Therefore if the number of equations and terms can be reduce then XSL should run more quickly. In order to reduce the number of equations and therefore the running time of XSL, I created a reduced version of AES, that we will call rAES. This version has all the same operations as the full version, but it manipulates four bits in place of bytes. Therefore all of the bit measurements are halved. The block size is 64 bits and the smallest key is 64 bits. The program was implemented to allow the number of rounds to be changed as well. For the 64 bit key anywhere from one to ten rounds can be executed. The hope was that the combination of reducing the block size and the

number of rounds would allow the equations to be solved in a feasible amount of time.

In order to convert AES to four bits, several parts of the algorithm had to be modified. Of course all of the manipulations had to be altered to take four bits instead of a byte, but more complex changes had to be made to the SBox, the Galois Field had to be changed, and the round constants had to be recalculated. The authors of Rijndael were thorough and included their rationale for the choice of SBox and round constants, so the 4-bit version was created to also meet those criteria [13, 14]. Both the encryption and decryption steps for rAES were created. The reduced algorithm was kept as close to the full version at every opportunity in order to ensure it's behavior would be as close to equivalent as possible.

For the AES [14], the SBox was derived following three steps. The first step is to fill the SBox with the values from 00000000 through 11111111 going across the columns and then down the rows, so the first row in hexadecimal is 00, 01, 02, 03, ..., 0F and the second row is 10, 11, 12, 13, ..., 1F. Next, all of the entries are changed to their multiplicative inverse over the Galois Field. Finally an equation is used to manipulate the bits of each byte. The equation exclusive-ors specific bits in each byte and adds them to a constant as follows:

$$y_i = x_i \oplus x_{(i+4) \bmod 8} \oplus x_{(i+5) \bmod 8} \oplus x_{(i+6) \bmod 8} \oplus x_{(i+7) \bmod 8} \oplus c_i$$

where x_i represents the input bits, numbered so that the byte 10010000 would have $x_7 = 1$ and $x_4 = 1$ with the other x_i equal to zero. The value y_i is the output bit that is used as part of the SBox entry for that byte and c_i is a constant, given as

the hexadecimal number 63. The rationale given for this choice was so that the correlation between the input byte and the output byte could not be easily expressed as a mathematical function [13, 14]. The specific choice of which bits to use and what constant to use were chosen to prevent the SBox from having any mappings where $\text{SubBytes}(x) = x$ and no mappings where $\text{SubBytes}(x) = \bar{x}$ (means the 1's in x become 0's in \bar{x} and the 0's in x become 1's in \bar{x} .) Finally, the SBox was designed so that the inverse SBox, used for decryption, would not have a case where $\text{ISBox}(x) = \text{SBox}(x)$.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	2	4	6	8	10	12	14	3	1	7	5	11	9	15	13
3	0	3	6	9	12	15	10	9	11	8	13	14	7	4	1	2
4	0	4	8	12	3	7	11	15	6	2	14	10	5	1	13	9
5	0	5	10	15	7	2	13	8	14	11	4	1	9	12	3	6
6	0	6	12	10	11	13	7	1	5	3	9	15	14	8	2	4
7	0	7	14	9	15	8	1	6	13	10	3	4	2	5	12	11
8	0	8	3	11	6	14	5	13	12	4	15	7	10	2	9	1
9	0	9	1	8	2	11	3	10	4	13	5	12	6	15	7	14
10	0	10	7	13	14	4	9	3	15	5	8	2	1	11	6	12
11	0	11	5	14	10	1	15	4	7	12	2	9	13	6	8	3
12	0	12	11	7	5	9	14	2	10	6	1	13	15	3	4	8
13	0	13	9	4	1	12	8	5	2	15	11	6	3	14	10	7
14	0	14	15	1	13	3	2	12	9	7	6	8	4	10	11	5
15	0	15	13	2	9	6	4	11	1	14	12	3	8	7	5	10

Table 2: Multiplication in $\text{GF}(2^4)$

Several different rAES SBoxes were tried, and through trial and error one was found that meet all of the above criteria. It was found by creating an SBox with a 2-bit index to the row and column and each entry is four bits. This created an SBox with 16 entries instead of 256. The entries were initialized to 0000 through 1111 with the first row being 0000, 0001, 0010, 0011 in binary and the second row being 0100, 0101, 0110, 0111. All of the multiplicative inverses were found using the Galois Field $GF(2^4)$ instead of $GF(2^8)$. For this Galois Field the constant 10011 was chosen to be used if the value exceeded four bits. For example, $4 \cdot 4 = 10000 \oplus 10011 = 0011 = 3$. All of the multiplications in $GF(2^4)$ can be seen in Table 2.

The $GF(2^4)$ containing the set from zero to 16 with the addition and multiplication operations as defined above does create a Galois Field. Since the addition is done as an exclusive-or the addition is automatically in the field. By examining Table 2 it is easy to see that all of the values are in the set. Also, all of the values in the set have a multiplicative inverse (the entries in the table that are equal to one). Therefore this is a Galois Field.

Finally the following equation was applied to get the resulting SBox:

$$y_i = x_i \oplus x_{(i+1) \bmod 4} \oplus x_{(i+3) \bmod 4} \oplus c_i$$

where in this case c_i is five. This equation and constant value were found to be a combination that allowed an SBox that met all the necessary criteria. For instance, the entry at (0, 2) was calculated as follows:

The value was initialized to 0010. The inverse of 0010 (as can be found in Table 2) is 1001. This value is then input to the equation so we have:

$$y_0 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$$

$$y_1 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$$

$$y_2 = 0 \oplus 1 \oplus 0 \oplus 1 = 0$$

$$y_3 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

which gives us the result of 0011 = 3.

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = 3$$

Figure 7: Matrix Representation of SBox Derivation

This example can also be done by representing the equation given above as a matrix, and representing the value as a column as seen in Figure 7. The resulting SBox and Inverse SBox can be seen in Figures 8 and 9.

	0	1	2	3
0	5	14	3	1
1	13	4	7	12
2	10	2	6	0
3	15	11	9	8

Figure 8: 4-Bit SBox

	0	1	2	3
0	11	3	9	2
1	5	0	10	6
2	15	14	8	13
3	7	4	1	12

Figure 9: 4-Bit Inverse SBox

The round constants for rAES were calculated using the same method as those for the AES, but reduced to output four bits. In the AES the round constant $Rcon_j = (RC_j, 0, 0, 0)$ where $1 \leq j \leq 10$ with each member of the list representing a byte. The value $RC_j = 2 \cdot RC_{j-1}$ and $RC_1 = 1$. All of the math is performed over the Galois Field $GF(2^8)$. For rAES the round constant $Rcon_j = (RC_j, 0, 0, 0)$ where each member represents four bits. The value $RC_j = 2 \cdot RC_{j-1}$ and $RC_1 = 1$. All of the math is performed over the Galois Field $GF(2^4)$. For example, the calculation for $RC_2 = 2 \cdot 1 = 2$ so $Rcon_2 = 0010000000000000$. The round constants for rAES can be seen in Table 3.

j	1	2	3	4	5	6	7	8	9	10
$Rcon_j$	1	2	4	8	3	6	12	11	5	10

Table 3: The rAES Round Constants for the 10-round version (in decimal)

The ShiftRows and InverseShiftRows for rAES work exactly the same as in the AES except each entry in the matrix shown in Figures 2 and 4 represent four bits instead of a byte (b0 represents bits 0-3, the top row represents bits 0-15, row two is bits 16-31, row three is bits 32-47, and row four is bits 48-63).

The MixColumns step works the same way in rAES as it does in the AES, but each hex value in the matrix is represented as a 4-bit number instead of as a byte. All of the values in the MixColumns matrix are less than 16 so they can directly be written in four bits. The InverseMixColumns matrix is also just changed to represent all of the values as 4-bits instead of bytes. The

mathematics was checked and the same values work for the InverseMixColumns even though they have been represented as only four bits.

The number of rounds was allowed to be variable in rAES. The minimum allowed is one round and the maximum is 10 for a 64-bit key, 12 for a 96-bit key, and 14 for a 128-bit key. For the one round option, the MixColumns is not performed, but for all other number of rounds the algorithm operates as normal, with the last round not having MixColumns but all the other rounds perform all of the steps. The decryption is done using the inverse steps as described above.

The rAES was designed to improve the running time of a possible attack. The design of rAES was made to follow the AES as closely as possible as can be seen in the description given above. Now an attempt to attack rAES could be made to see if there are any weaknesses. The rAES is inherently weaker than the AES, so if an attack for rAES cannot be found then a similar attack on AES would be even less effective. With the reduced version complete, the equations representing it can be found. The equations used in the attack on rAES are described in Chapter III.

CHAPTER III

APPLIED ATTACK

The Equations

In order to use an XSL-style attack on rAES, the equations that represent the steps of the algorithm had to be derived for the 4-bit version. This derivation was similar to that described in [4]. The equations from the encryption and not the decryption were found since the equations that result from the MixColumns step are simpler than those that result from the InverseMixColumns. This is because performing the multiplication in $GF(2^4)$ on the values in the MixColumns matrix requires fewer calculations than performing them with the InverseMixColumns matrix. The manipulations performed on the bits of the plaintext can be grouped into two types, the diffusion done by the AddRoundKey, ShiftRows, and MixColumns steps, and the nonlinear step done by SubBytes. The equations for the diffusion steps are fairly straight forward to derive, but the SubBytes step was designed to try to prevent a simple representation of the manipulations on the bits being described as linear equations [13, 14].

Due to the criteria used in the design of the SBox for the AES, it was found in [4, 9, 10] that equations that are true for all the different mappings of the SBox can be found. It was found that using first order and second order terms from the SBox mappings were sufficient to describe the SBox [4]. The terms

came from the input and output bits of the SBox. Similarly, such equations were found to exist for rAES. Once the SBox for rAES was created (see Fig. 7 on page 17) an algorithm to find all of the true equations for this SBox was devised. The terms for the rAES SBox are just the bits of the four bits input and four bits output, and all the possible second order combinations of the input and output bits. This creates 24 terms. Since there are 16 entries in the SBox, there are 16 possible true values for all of the terms. The terms and all possible values for the mappings can be seen in Figure 12 in Appendix A. The algorithm to find the true equations used a nested loop structure to sum under GF(2) all combinations of the terms and returned the combinations where the result was 0 or 1 for all equations. This means that no matter what the input to the SBox was, the equations would have the same result. A total of 2039 such equations were found to be true for the rAES SBox, with the equations varying in length from only five terms all the way up to 20 terms. Since there are only 24 SBox terms, we decided to use 24 of the 2039 SBox equations in the implementation. The specific equations chosen can be seen in Figure 13 in Appendix A.

Since the SBox is performed on four bits at a time and there are 64 bits in a block, it was necessary to have a different set of 24 SBox equations for each of the 16 times the SBox gets used in a single round. This gives a total of 384 SBox equations and 384 terms. Of these 384 terms, 128 are first order and the remaining 256 are second order terms.

The equations from the other three steps were more directly determined since they are already linear in nature. The equations for the first AddRoundKey step were from the bits of the input plaintext, the bits of the key, and the bits that result from using the exclusive-or operation on them. There are 64 such equations with 128 terms (the plaintext bits are known, the key and result bits are unknown). The equations are as follows:

$$r_i \oplus k_i = p_i$$

where r_i is a bit of the result of the AddRoundKey operation, k_i is a bit of the key, and p_i is a bit of the plaintext, with $0 \leq i < 64$.

The operations on the bits done by the ShiftRows, MixColumns, and second AddRoundKey steps were all combined into a single set of equations. These terms in these equations were from the bits of the output of the SBox, the bits of the second round key, and the resulting bits from after the round key was added. The locations of the SBox output bits were tracked through the ShiftRows and MixColumns steps so that the bit from the SBox that was added to a specific bit of the round key was known. This resulted in the 64 equations. To see if the technique would work even without the MixColumns step, which is what occurs if only one round is used, the equations ignoring the MixColumns manipulations on the bits were also devised. These can be seen in Figure 14 in Appendix A. In the one round case there are 64 equations with 128 unknown terms since the resulting bits from the final AddRoundKey step are the known ciphertext bits.

This gives us a total of 512 equations and 512 terms. All of the equations found above are unique, but the terms for the output bits of the first AddRoundKey are the same as the input bits to the SBox, and the output bits from the SBox are the same as the input bits to the second AddRoundKey. It was considered to attempt to solve subsets of these equations as separate systems, but the ratio of the number of terms to the number of equations for any subset is not large enough. For instance, if only the equations from the first AddRoundKey are considered we have 128 terms but only 64 equations. When solving linear systems of equations at least an equal number of equations to terms is necessary in order to uniquely solve the system using conventional methods such as Gaussian Elimination. Once the equations were determined it remained to attempt to solve the system of equations found.

The Attack

The system of equations described above was input to a modified version of Gaussian Elimination with Backwards Substitution. The algorithm was modified as shown in Figure 10, to optimize it for operations in $GF(2)$. The normal version solves systems with decimal coefficients and solutions; here all the coefficients and solutions will be binary.

The modification to the Gaussian Elimination algorithm occurred to the steps shown in bold in Figure 10. Originally the multiplication, subtraction, and replacement of row j was performed in every execution of the loops over i and j . When m is zero, these steps just end up replacing row j with the same values it

already had. Therefore, if m is zero, all three of these steps can be skipped. The value m can only be zero or one, and if it is one then the steps are executed.

```

Modified Gaussian Elimination
Input: Matrix representation of the system of equations
Output: Array containing the solutions for each term
 $m \leftarrow$  matrix column count - 2
solutions[m+1]
for ( $i \leftarrow 0$ ;  $i < m$ ;  $i++$ )
     $p \leftarrow n$ 
    for ( $k \leftarrow 0$ ;  $k < m$ ;  $k++$ )
        if matrix location ( $k,i$ ) is one
             $p \leftarrow k$ 
    end loop over  $k$ 

    if  $p \geq m$ 
        return false
    if  $p \neq i$ 
        switch row  $i$  of the matrix with row  $p$  of the matrix
    for ( $j \leftarrow i+1$ ;  $j \leq m$ ;  $j++$ )
         $m \leftarrow 0$  //  $m$  is the matrix location ( $j,i$ ) divided by location ( $i,i$ )
        if matrix location ( $j,i$ ) is one and matrix location ( $i,i$ ) is one
             $m \leftarrow 1$ 
            multiply row  $i$  of the matrix with  $m$ 
            subtract the above from row  $j$  of the matrix
            replace row  $j$  of the matrix with the above

    if matrix location ( $m,m$ ) = 0
        return false
    if matrix location ( $m-1,m$ ) is one and matrix location ( $m-1,m-1$ ) is one
        solutions[m]  $\leftarrow 1$ 
    else solutions[m]  $\leftarrow 0$ 
    for ( $i \leftarrow m-1$ ;  $i \geq 0$ ;  $i--$ )
        sum  $\leftarrow 0$ 
        for ( $j \leftarrow i+1$ ;  $j \leq m$ ;  $j++$ )
            sum  $\leftarrow$  sum + matrix location ( $i,j$ ) + solutions[j]
        sum  $\leftarrow$  sum mod 2 // math is over GF(2)
        sum  $\leftarrow$  matrix location ( $i, m+1$ ) - sum
        sum  $\leftarrow$  sum / matrix location ( $i,i$ )
        if sum = -1
            sum  $\leftarrow 1$  // math in GF(2)
        solutions[i]  $\leftarrow$  sum
    return true

```

Figure 10: Modified Gaussian Elimination Version 1

The value m is a result of dividing two entries in the matrix, so m is only one if both entries are one. Therefore on average m is only one 25% of the time. This means that the three steps are skipped 75% of the time with this modification. Therefore the running time of the Gaussian Elimination has been optimized, though the asymptotic running time is the same. Additional modifications were made to the italicized steps because all of the math needs to be done in binary.

It turned out that the system I found to represent rAES did not work well with the T' Method. The T' Method relies upon each term appearing a relatively large number of times in the entire system. For our system of 512 terms, any given term was found to appear in less than 4% of the equations. This can be verified using Appendix A. This means the number of executions of the T' Method for the rAES equations is beyond the realm of feasibility. Even though I had a system with an equal number of terms and equations, it was found that the system was not solvable in its initial state. In the light of this observation, a different approach was taken to attempt to solve the system of equations.

For only one round of rAES, the number of equations that could be determined was much larger than the number of terms needed to solve the system, since typically an equal number of terms and equations are required. Therefore, the number of equations was increased to over 512.

The Gaussian Elimination algorithm was modified to allow for a non-square matrix as can be seen in Figure 11. The constant n was added to the algorithm to represent the number of equations in the system. Before, the

number was the same as the number of terms so only one constant, m , was required. In the situations where the number of equations and not the number of terms was needed, n has been substituted for m . Essentially this means that all of the equations will be checked to try to find a subset of them that will allow all 512 terms to be solved.

This technique essentially has the same end effect as the T' Method, but achieves it in a different manner. In the T' Method the number of total equations is increased each time the T' Method is executed, using our technique the number of equations starts out larger than the number of terms. Then all that was required was to run the second modified version of Gaussian Elimination on the matrix.

The additional equations came from two sources, the SBox Equations found previously, and new equations from the ExpandRoundKey. There were 2039 SBox equations found and so far only 24 of them have been used in the attack. That leaves 2015 left to try. Initially, the shortest 24 equations were used, now the longest 24 equations will be added to increase the total to 48 SBox equations over 24 SBox terms. The second set of SBox equations can be seen in Figure 15 in Appendix A. The ExpandRoundKey step, as described on page 7, uses exclusive-or, circular shifts, and the SBox to manipulate the bits. The first set of equations used for the SubBytes step was used for the SBox part of the ExpandRoundKey equations. Only one out of every four words is passed through the SBox, the other three words per round key just use the exclusive-or

operation. The equations for the ExpandRoundKey are combinations of the SBox equations using the ExpandRoundKey terms, and the equations shown in Figure 16 in Appendix A.

```

Modified Gaussian Elimination
Input: Matrix representation of the system of equations
Output: Array containing the solutions for each term
n ← matrix row count – 1
m ← matrix column count – 2
solutions[m+1]
for (i ← 0; i < m; i++)
    p ← n
    for (k ← 0; k < n; k++)
        if matrix location (k,i) is one
            p ← k
    end loop over k

    if p ≥ n
        return false
    if p ≠ i
        switch row i of the matrix with row p of the matrix
    for (j ← i+1; j ≤ n; j++)
        m ← 0 // m is the matrix location (j,i) divided by location (i,i)
        if matrix location (j,i) is one and matrix location (i,i) is one
            m ← 1
            multiply row i of the matrix with m
            subtract the above from row j of the matrix
            replace row j of the matrix with the above
if matrix location (m,m) = 0
    return false
if matrix location (m-1,m) is one and matrix location (m-1,m-1) is one
    solutions[m] ← 1
else solutions[m] ← 0
for (i ← m-1; i ≥ 0; i--)
    sum ← 0
    for (j ← i+1; j ≤ n; j++)
        sum ← sum + matrix location (i,j) + solutions[j]
    sum ← sum mod 2 // math is over GF(2)
    sum ← matrix location (i, m+1) – sum
    sum ← sum / matrix location (i,i)
    if sum = -1
        sum ← 1 // math in GF(2)
    solutions[i] ← sum
return true

```

Figure 11: Modified Gaussian Elimination Version 2

The additional 24 SBox equations were added per each four bits. This resulted in an addition of another 384 equations, bringing the total to 896 equations with the terms remaining at 512. There are 64 equations from the creation of the 64-bit round key. There are 96 additional SBox equations from the round key expansion. This gives a total of 1056 equations. The RotWord and SubBytes parts of the ExpandRoundKey step produce some additional terms, 16 to be exact. This increases the number of terms to 528. There are also a separate set of second order SBox terms for the ExpandRoundKey which increases the total number of terms to 592. Therefore the system used in the attack had 1056 equations describing 592 terms.

CHAPTER IV

RESULTS

Initially, the code was run on the 512 term, 512 equation system. It was found that this system was not sufficient to get solutions for all the terms. When run using the first modified version of the Gaussian Elimination algorithm it was able to diagonalize the matrix down to term 133 as shown in Appendix A. The code was run on several keys using the same plaintext (the word “plaintext” was used since it completely filled one block). For any key the matrix could only be diagonalized to the same term. Term 133, as can be seen in Table 4 of Appendix A, is j_2 which is the third bit of the round key. Since it was apparent that more equations related to the round keys were needed, those equations were added to the code next.

With the round key equations added in, the system was then 592 terms and 672 equations. When the code was run with these changes, using the second modified version of the Gaussian Elimination algorithm, the matrix was diagonalized down to term 267, r_2y_3 , the second order term of the third bit of the input to the SBox and the fourth bit of the output from the SBox. Again, this situation occurred for any key used. The point where the diagonalization stops is shown in an example execution of the attack on page 81.

Now, it seemed that more SBox equations were needed to create a solvable system, so the second set of SBox equations was added. This increased the number of equations to 1056, leaving the number of terms at 592. Unexpectedly, doubling the number of SBox equations had no effect on the results of the Gaussian Elimination. It was still only able to diagonalize the matrix to term 267. Despite having nearly twice as many equations as terms, the system was found to be unsolvable. Increasing the number of equations from 672 to 1056 had no effect on the solvability of the system. Considering these results it was decided that attempting to include more of the SBox equations would not help the situation.

In order to see if there was a pattern in the terms that could not be diagonalized by the Gaussian Elimination, the code was modified to not break when a term was found that could not be diagonalized. Instead that term was added to the matrix in the correct location and just set to zero. A statement noting that a term could not be diagonalized was output each time this occurred. When this was done it was found that the same terms in the SBox equations were not diagonalizable. These terms were $x_2 y_3$, $x_3 y_0$, $x_3 y_1$, $x_3 y_2$, and $x_3 y_3$ through term 319, then in the next set of SBox equations it increased to be $x_2 y_1$, $x_2 y_2$, $x_2 y_3$, $x_3 y_0$, $x_3 y_1$, $x_3 y_2$, and $x_3 y_3$ and continued in that pattern until term 383. From term 384 on the pattern was $x_2 y_0$, $x_2 y_1$, $x_2 y_2$, $x_2 y_3$, $x_3 y_0$, $x_3 y_1$, $x_3 y_2$, and $x_3 y_3$. There were also later terms that were found to be undiagonalizable.

CHAPTER V

CONCLUSIONS

General Conclusions

The attack on rAES failed. Despite having a sparse system of linear equations with nearly twice as many equations as terms the system was unsolvable, so the key could not be retrieved. This result is good news for the AES. As noted previously, if the attack on rAES fails, an attack on the AES would be even less likely to be successful.

In the process of attempting to attack rAES, some interesting side results were found. Based on the experimental results, the solution of the linear system of equations representing the steps in rAES is not dependent on the plaintext choice or the key choice. Logically, this makes sense, because the solution for an equation equal to one gives you no more information about the terms than an equation equal to zero. This is important because it means that the security of rAES is the same no matter how cleverly the plaintext is chosen. It follows that the strength of AES against this type of attack is also independent of the plaintext choice or key choice.

The improvements to the Gaussian Elimination to optimize it for equations in $GF(2)$ improve the running time, but do not reduce the number of executions of

any of the loops. Therefore it would not affect the general execution estimates given by others in [4, 9].

Future Work

The fact that specific terms in the SBox equations were the ones that were unsolvable indicates that some rules for selecting the SBox equations could be devised. If such a rule or set of rules could be found, then the entire system of equations could be solvable.

Using the all available equations instead of the T' Method reduces the running time to only one execution of the Gaussian Elimination algorithm instead of approximately 2^{100} or more. Since the key and plaintext appear to have no effect on the solvability of the system, if a set of equations could be found that was solvable, then only one execution of the Gaussian Elimination would be needed. The running time would then be $O(n^2)$, where n is the number of equations in the system. The results indicate that finding a set of equations to solve even one round was not possible. The advantage is if a system could be found that was solvable, it would only need to be found once. Then the only work required is to set the specific plaintext and ciphertext bits and run the Gaussian Elimination on the matrix using the predetermined equations.

BIBLIOGRAPHY

1. Cheon, Jung Hee and Dong Hoon Lee. "Resistance of S-boxes against Algebraic Attacks." 2004. Online. Internet. 6 Oct. 2005. Available http://www.math.snu.ac.kr/~jhcheon/Published/2004_FSE/FSE04_CL.pdf
2. Courtois, Nicolas T. and Louis Goubin. "An Algebraic Masking Method to Protect AES Against Power Attacks." 2005. Online. Internet. 6 Oct. 2005. Available <http://eprint.iacr.org/2005/204.pdf>
3. Courtois, Nicolas T. and Louis Goubin. "Open Problems in Multivariate Cryptanalysis –Stork Submission--." Online. Internet. 6 Oct. 2005. Available <http://eprint.iacr.org/2002/044.pdf>
4. Courtois, Nicolas T. and Josef Pieprzyk. "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations." Mar. 2002. Online. Internet. 3 Jan. 2006. Available <http://eprint.iacr.org/2002/044.pdf>
5. Daemen, Joan and Vincent Rijmen. "Answer to "New Observations on Rijndael." 11 Aug. 2000. Online. Internet. Aug. 2005. Available <http://www.iaik.tugraz.at/research/krypto/AES/old/~rijmen/rijndael/answer.pdf>
6. Ferguson, Niels, et al. "Improved Cryptanalysis of Rijndael." 2000. Online. Internet. 6 Oct. 2005. Available <http://www.schneier.com/paper-rijndael.pdf>
7. "Gaussian Elimination." *Wikipedia*. 7 Feb. 2006. Online. Internet. 15 Feb. 2006. Available http://en.wikipedia.org/wiki/Gaussian_elimination
8. Murphy, Sean and Matt Robshaw. "Comments on the Security of the AES and the XSL Technique." 20 Sep. 2002. Online. Internet. Dec. 2005. Available http://www.cosic.esat.kuleuven.ac.be/nessie/reports/phase2/Xslbes8_Ness.pdf

9. Murphy, Sean and Matt Robshaw. "Essential Algebraic Structure Within the AES." Aug. 2002. Online. Internet. Aug. 2005. Available <http://www.isg.rhul.ac.uk/~sean/crypto.pdf>
10. Murphy, Sean and Matt Robshaw. "New Observations on Rijndael." 7 Aug. 2000. Online. Internet. Aug 2005. Available http://www.isg.rhul.ac.uk/~sean/rijn_newobs.pdf
11. Schneier, Bruce. "AES News." *Crypto-Gram Newsletter*. 15 Sep. 2002. Online. Internet. 12 Oct. 2005. Available <http://www.schneier.com/crypto-gram-209.html#1>
12. Schneier, Bruce. "More on AES Cryptanalysis." *Crypto-Gram Newsletter*. 15 Oct. 2002. Online. Internet. 6 Oct. 2005. Available <http://www.schneier.com/crypto-gram-210.html#2>
13. Stallings, William. *Cryptography and Network Security: Principles and Practices*. 3rd ed. Upper Saddle River, New Jersey: Prentice Hall, 2003.
14. United States. National Institute of Standards and Technology. *Announcing the Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197. 26 Nov. 2001. Online. Internet. 7 Dec. 2005. Available <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
15. Wilson, J.C. "Principal Ideal Domain." *Wikipedia*. 10 Jan. 2006. Online. Internet. 15 Feb. 2006. Available http://en.wikipedia.org/wiki/Principal_ideal_domain
16. Yacoumis, Paul. *On the Security of the Advanced Encryption Standard*. Nov. 2005. Online. Internet. Dec. 2005. Available http://ntcourtois.free.fr/yacoumis_aes.pdf

APPENDIX A

ADDITIONAL TABLES AND FIGURES

		Mapping:															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Terms:	0 x_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	1 x_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	2 x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	3 x_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	4 y_0	1	0	1	1	1	0	1	0	0	0	0	0	1	1	1	0
	5 y_1	0	1	1	0	0	0	1	0	1	1	1	0	1	1	0	0
	6 y_2	1	1	0	0	1	1	1	1	0	0	1	0	1	0	0	0
	7 y_3	0	1	0	0	1	0	0	1	1	0	0	0	1	1	1	1
	8 x_0y_0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
	9 x_0y_1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0
	10 x_0y_2	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
	11 x_0y_3	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1
	12 x_1y_0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	0
	13 x_1y_1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0
	14 x_1y_2	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0
	15 x_1y_3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
	16 x_2y_0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	0
	17 x_2y_1	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0
	18 x_2y_2	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0
	19 x_2y_3	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1	1
	20 x_3y_0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
	21 x_3y_1	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0
	22 x_3y_2	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0
	23 x_3y_3	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1

Figure 12: All SBox Terms and Mappings

$x_0y_0 + x_1y_0 + x_1y_1 + x_3y_0 + x_3y_2 = 0$	$x_0 + x_3 + y_1 + x_0y_1 + x_0y_2 + x_1y_0 = 0$
$x_0y_2 + x_0y_3 + x_1y_3 + x_2y_0 + x_2y_2 = 0$	$x_1 + x_3 + x_0y_1 + x_0y_3 + x_1y_0 + x_3y_3 = 0$
$x_1y_3 + x_2y_0 + x_2y_1 + x_2y_3 + x_3y_0 = 0$	$x_0 + x_1 + x_2 + x_0y_1 + x_1y_1 + x_1y_3 + x_2y_0 = 0$
$y_1 + x_0y_0 + x_0y_3 + x_1y_0 + x_1y_3 + x_3y_1 = 0$	$x_0 + x_2 + x_3 + y_3 + x_0y_0 + x_1y_2 + x_3y_0 = 0$
$y_3 + x_0y_1 + x_2y_3 + x_3y_1 + x_3y_2 = 0$	$x_0 + y_0 + y_1 + x_0y_1 + x_1y_0 + x_3y_0 = 1$
$y_3 + x_0y_2 + x_1y_2 + x_2y_2 + x_3y_2 + x_3y_3 = 0$	$x_1 + y_2 + x_1y_2 + x_3y_1 + x_3y_2 = 1$
$y_0 + y_2 + x_0y_2 + x_1y_0 + x_2y_1 + x_3y_2 = 0$	$x_1 + y_2 + y_3 + x_0y_1 + x_1y_2 + x_2y_3 = 1$
$x_2 + x_1y_2 + x_1y_3 + x_2y_1 + x_2y_2 + x_3y_2 = 0$	$x_3 + y_0 + x_0y_2 + x_3y_0 = 1$
$x_2 + y_1 + x_0y_1 + x_1y_1 + x_2y_2 + x_3y_3 = 0$	$x_3 + y_0 + x_0y_3 + x_2y_1 + x_2y_2 + x_2y_3 = 1$
$x_2 + y_3 + x_0y_2 + x_1y_3 + x_2y_1 + x_3y_3 = 0$	$x_3 + y_2 + x_0y_0 + x_1y_1 + x_2y_1 = 1$
$x_0 + x_1 + y_1 + x_0y_2 + x_0y_3 + x_3y_3 = 0$	$x_3 + y_2 + x_1y_0 + x_2y_1 + x_3y_0 + x_3y_2 = 1$
$x_0 + x_3 + x_0y_0 + x_0y_1 + x_2y_0 + x_2y_2 + x_3y_1 = 0$	$x_2 + x_3 + y_0 + y_3 + x_2y_0 + x_2y_3 + x_3y_3 = 1$

Figure 13: SBox Equations - Set One

$y_i \oplus j_i = c_i$	for $0 \leq i < 16$
$y_m \oplus j_i = c_i$	for $16 \leq i < 28$, and $20 \leq m < 31$
$y_m \oplus j_i = c_i$	for $28 \leq i < 32$, and $16 \leq m < 20$
$y_m \oplus j_i = c_i$	for $32 \leq i < 40$, and $40 \leq m < 48$
$y_m \oplus j_i = c_i$	for $40 \leq i < 48$, and $32 \leq m < 40$
$y_m \oplus j_i = c_i$	for $48 \leq i < 52$, and $60 \leq m < 64$
$y_m \oplus j_i = c_i$	for $52 \leq i < 64$, and $48 \leq m < 60$

Figure 14: Equations after the SBox

Where y_i and y_m are the bits output from the SBox adjusted for the ShiftRows, j_i are the bits of the second round key, and c_i are the bits of the ciphertext.

$x_0 + x_1 + y_0 + y_2 + y_3 + x_0y_1 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_1 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_2 + x_3y_3 = 0$	$x_0 + x_1 + x_3 + y_0 + y_1 + x_0y_0 + x_0y_1 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_1 + x_1y_2 + x_1y_3 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_2 = 1$
$x_0 + x_1 + y_0 + y_1 + y_2 + y_3 + x_0y_0 + x_0y_1 + x_0y_2 + x_1y_1 + x_1y_3 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_2 + x_3y_3 = 0$	$x_0 + x_1 + x_3 + y_0 + y_1 + y_3 + x_0y_0 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_1 + x_1y_2 + x_1y_3 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 = 1$
$x_0 + x_2 + y_1 + y_2 + y_3 + x_0y_0 + x_0y_1 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_1 + x_1y_3 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_3 = 1$	$x_0 + x_2 + x_3 + y_0 + y_2 + y_3 + x_0y_0 + x_0y_1 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_1 + x_1y_3 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_2 + x_3y_3 = 0$
$x_0 + x_3 + y_0 + y_1 + y_2 + y_3 + x_0y_0 + x_0y_1 + x_0y_2 + x_1y_0 + x_1y_1 + x_1y_2 + x_1y_3 + x_2y_0 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_2 + x_3y_3 = 0$	$x_0 + x_2 + x_3 + y_0 + y_1 + y_2 + y_3 + x_0y_0 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_1 + x_1y_2 + x_2y_0 + x_2y_3 + x_3y_0 + x_3y_1 = 0$
$x_0 + x_3 + y_0 + y_1 + y_2 + y_3 + x_0y_0 + x_0y_1 + x_0y_3 + x_1y_0 + x_1y_1 + x_1y_2 + x_1y_3 + x_2y_0 + x_2y_1 + x_3y_0 + x_3y_2 + x_3y_3 = 0$	$x_1 + x_2 + x_3 + y_0 + y_2 + x_0y_0 + x_0y_1 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_1 + x_1y_2 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_2 + x_3y_3 = 0$
$x_0 + x_3 + y_0 + y_1 + y_2 + y_3 + x_0y_0 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_1 + x_1y_3 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_2 = 0$	$x_1 + x_2 + x_3 + y_0 + y_1 + y_2 + x_0y_1 + x_0y_2 + x_1y_1 + x_1y_2 + x_1y_3 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_1 + x_3y_2 + x_3y_3 = 0$
$x_1 + x_3 + y_0 + y_2 + y_3 + x_0y_0 + x_1y_0 + x_1y_1 + x_1y_3 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_2 + x_3y_3 = 0$	$x_1 + x_2 + x_3 + y_0 + y_1 + y_2 + x_0y_0 + x_0y_1 + x_0y_2 + x_1y_0 + x_1y_2 + x_1y_3 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_3 = 0$
$x_0 + x_1 + x_2 + y_1 + y_3 + x_0y_0 + x_0y_1 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_1 + x_1y_2 + x_2y_0 + x_2y_2 + x_3y_1 + x_3y_2 + x_3y_3 = 0$	$x_1 + x_2 + x_3 + y_0 + y_2 + y_3 + x_0y_0 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_1 + x_1y_2 + x_1y_3 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_3 = 0$
$x_0 + x_1 + x_2 + y_1 + y_3 + x_0y_0 + x_0y_1 + x_1y_0 + x_1y_1 + x_1y_2 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_2 + x_3y_3 = 0$	$x_1 + x_2 + x_3 + y_0 + y_1 + y_2 + y_3 + x_0y_0 + x_0y_1 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_1 + x_1y_2 + x_1y_3 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_2 = 0$
$x_0 + x_1 + x_2 + y_0 + y_1 + y_2 + y_3 + x_0y_0 + x_0y_1 + x_0y_2 + x_0y_3 + x_1y_1 + x_1y_2 + x_1y_3 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_2 + x_3y_3 = 0$	$x_1 + x_2 + x_3 + y_0 + y_1 + y_2 + y_3 + x_0y_0 + x_0y_2 + x_1y_0 + x_1y_2 + x_1y_3 + x_2y_0 + x_2y_1 + x_2y_2 + x_3y_0 + x_3y_2 + x_3y_3 = 0$
$x_0 + x_1 + x_2 + y_0 + y_1 + y_2 + y_3 + x_0y_0 + x_0y_1 + x_0y_2 + x_0y_3 + x_1y_1 + x_1y_2 + x_1y_3 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_2 + x_3y_3 = 0$	$x_0 + x_1 + x_2 + x_3 + y_0 + x_0y_0 + x_0y_2 + x_0y_3 + x_1y_0 + x_1y_2 + x_1y_3 + x_2y_0 + x_2y_1 + x_3y_0 + x_3y_1 + x_3y_2 + x_3y_3 = 1$
$x_0 + x_1 + x_2 + y_0 + y_1 + y_2 + y_3 + x_0y_1 + x_0y_3 + x_1y_0 + x_1y_2 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_2 + x_3y_3 = 0$	$x_0 + x_1 + x_2 + x_3 + y_2 + x_0y_2 + x_1y_0 + x_1y_1 + x_1y_2 + x_1y_3 + x_2y_0 + x_2y_1 + x_2y_2 + x_2y_3 + x_3y_0 + x_3y_1 + x_3y_2 + x_3y_3 = 1$

Figure 15: SBox Equations – Set Two

$$\begin{aligned}
0 &= k_0 + z_0 + j_0 \\
0 &= k_1 + z_1 + j_1 \\
0 &= k_2 + z_2 + j_2 \\
1 &= k_3 + z_3 + j_3 \\
0 &= k_t + z_t + j_t && \text{for } 4 \leq t < 15 \\
0 &= j_u + k_u + j_v && \text{for } 16 \leq u < 64 \text{ and } 0 \leq v < 48
\end{aligned}$$

Figure 16: ExpandRoundKey Equations

Where j_i with $0 \leq i < 64$ are the bits of the first round key and z_m with $0 \leq m < 16$ are the bits of the output of the SubBytes operation

Terms from Before SBox	r0	k0	r1	k1	r2	k2	r3	k3	r4	k4	r5	k5	r6	k6	r7	k7
Location in the Matrix	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Terms from Before SBox, cont...	r8	k8	r9	k9	r10	k10	r11	k11	r12	k12	r13	k13	r14	k14	r15	k15
Location in the Matrix	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Terms from Before SBox, cont...	r16	k16	r17	k17	r18	k18	r19	k19	r20	k20	r21	k21	r22	k22	r23	k23
Location in the Matrix	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Terms from Before SBox, cont...	r24	k24	r25	k25	r26	k26	r27	k27	r28	k28	r29	k29	r30	k30	r31	k31
Location in the Matrix	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Terms from Before SBox, cont...	r32	k32	r33	k33	r34	k34	r35	k35	r36	k36	r37	k37	r38	k38	r39	k39
Location in the Matrix	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
Terms from Before SBox, cont...	r40	k40	r41	k41	r42	k42	r43	k43	r44	k44	r45	k45	r46	k46	r47	k47
Location in the Matrix	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
Terms from Before SBox, cont...	r48	k48	r49	k49	r50	k50	r51	k51	r52	k52	r53	k53	r54	k54	r55	k55
Location in the Matrix	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
Terms from Before SBox, cont...	r56	k56	r57	k57	r58	k58	r59	k59	r60	k60	r61	k61	r62	k62	r63	k63
Location in the Matrix	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
Terms from After SBox	y0	j0	y1	j1	y2	j2	y3	j3	y4	j4	y5	j5	y6	j6	y7	j7
Location in the Matrix	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
Terms from After SBox, cont...	y8	j8	y9	j9	y10	j10	y11	j11	y12	j12	y13	j13	y14	j14	y15	j15
Location in the Matrix	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
Terms from After SBox, cont...	y16	j16	y17	j17	y18	j18	y19	j19	y20	j20	y21	j21	y22	j22	y23	j23
Location in the Matrix	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
Terms from After SBox, cont...	y24	j24	y25	j25	y26	j26	y27	j27	y28	j28	y29	j29	y30	j30	y31	j31
Location in the Matrix	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
Terms from After SBox, cont...	y32	j32	y33	j33	y34	j34	y35	j35	y36	j36	y37	j37	y38	j38	y39	j39
Location in the Matrix	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
Terms from After SBox, cont...	y40	j40	y41	j41	y42	j42	y43	j43	y44	j44	y45	j45	y46	j46	y47	j47
Location in the Matrix	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
Terms from After SBox, cont...	y48	j48	y49	j49	y50	j50	y51	j51	y52	j52	y53	j53	y54	j54	y55	j55
Location in the Matrix	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
Terms from After SBox, cont...	y56	j56	y57	j57	y58	j58	y59	j59	y60	j60	y61	j61	y62	j62	y63	j63
Location in the Matrix	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Table 4: Matrix Terms - First Half

Above are the terms used in the system of equations, they are numbered in the order they are stored in the matrix shown in the sample data.

Second Order SBox Terms	r0y0	r0y1	r0y2	r0y3	r1y0	r1y1	r1y2	r1y3	r2y0	r2y1	r2y2	r2y3	r3y0	r3y1	r3y2	r3y3
Location in the Matrix	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271
Second Order SBox Terms, cont...	r4y4	r4y5	r4y6	r4y7	r5y4	r5y5	r5y6	r5y7	r6y4	r6y5	r6y6	r6y7	r7y4	r7y5	r7y6	r7y7
Location in the Matrix	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287
Second Order SBox Terms, cont...	r8y8	r8y9	r8y10	r8y11	r9y8	r9y9	r9y10	r9y11	r10y8	r10y9	r10y10	r10y11	r11y8	r11y9	r11y10	r11y11
Location in the Matrix	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303
Second Order SBox Terms, cont...	r12y12	r12y13	r12y14	r12y15	r13y12	r13y13	r13y14	r13y15	r14y12	r14y13	r14y14	r14y15	r15y12	r15y13	r15y14	r15y15
Location in the Matrix	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319
Second Order SBox Terms, cont...	r16y16	r16y17	r16y18	r16y19	r17y16	r17y17	r17y18	r17y19	r18y16	r18y17	r18y18	r18y19	r19y16	r19y17	r19y18	r19y19
Location in the Matrix	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335
Second Order SBox Terms, cont...	r20y20	r20y21	r20y22	r20y23	r21y20	r21y21	r21y22	r21y23	r22y20	r22y21	r22y22	r22y23	r23y20	r23y21	r23y22	r23y23
Location in the Matrix	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351
Second Order SBox Terms, cont...	r24y24	r24y25	r24y26	r24y27	r25y24	r25y25	r25y26	r25y27	r26y24	r26y25	r26y26	r26y27	r27y24	r27y25	r27y26	r27y27
Location in the Matrix	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367
Second Order SBox Terms, cont...	r28y28	r28y29	r28y30	r28y31	r29y28	r29y29	r29y30	r29y31	r30y28	r30y29	r30y30	r30y31	r31y28	r31y29	r31y30	r31y31
Location in the Matrix	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383
Second Order SBox Terms, cont...	r32y32	r32y33	r32y34	r32y35	r33y32	r33y33	r33y34	r33y35	r34y32	r34y33	r34y34	r34y35	r35y32	r35y33	r35y34	r35y35
Location in the Matrix	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399
Second Order SBox Terms, cont...	r36y36	r36y37	r36y38	r36y39	r37y36	r37y37	r37y38	r37y39	r38y36	r38y37	r38y38	r38y39	r39y36	r39y37	r39y38	r39y39
Location in the Matrix	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415
Second Order SBox Terms, cont...	r40y40	r40y41	r40y42	r40y43	r41y40	r41y41	r41y42	r41y43	r42y40	r42y41	r42y42	r42y43	r43y40	r43y41	r43y42	r43y43
Location in the Matrix	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431
Second Order SBox Terms, cont...	r44y44	r44y45	r44y46	r44y47	r45y44	r45y45	r45y46	r45y47	r46y44	r46y45	r46y46	r46y47	r47y44	r47y45	r47y46	r47y47
Location in the Matrix	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447
Second Order SBox Terms, cont...	r48y48	r48y49	r48y50	r48y51	r49y48	r49y49	r49y50	r49y51	r50y48	r50y49	r50y50	r50y51	r51y48	r51y49	r51y50	r51y51
Location in the Matrix	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463
Second Order SBox Terms, cont...	r52y52	r52y53	r52y54	r52y55	r53y52	r53y53	r53y54	r53y55	r54y52	r54y53	r54y54	r54y55	r55y52	r55y53	r55y54	r55y55
Location in the Matrix	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479
Second Order SBox Terms, cont...	r56y56	r56y57	r56y58	r56y59	r57y56	r57y57	r57y58	r57y59	r58y56	r58y57	r58y58	r58y59	r59y56	r59y57	r59y58	r59y59
Location in the Matrix	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495
Second Order SBox Terms, cont...	r60y60	r60y61	r60y62	r60y63	r61y60	r61y61	r61y62	r61y63	r62y60	r62y61	r62y62	r62y63	r63y60	r63y61	r63y62	r63y63
Location in the Matrix	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511
Output of the SubBytes Operation in the ExpandRoundkey Step	z0	z1	z2	z3	z4	z5	z6	z7	z8	z9	z10	z11	z12	z13	z14	z15
Location in the Matrix	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527
Second Order SBox Terms for the ExpandRoundkey Step	k5z20	k5z21	k5z22	k5z23	k5z20	k5z21	k5z22	k5z23	k5z20	k5z21	k5z22	k5z23	k5z20	k5z21	k5z22	k5z23
Location in the Matrix	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543
Second Order SBox Terms for the ExpandRoundkey Step, cont...	k56z4	k56z5	k56z6	k56z7	k57z4	k57z5	k57z6	k57z7	k58z4	k58z5	k58z6	k58z7	k59z4	k59z5	k59z6	k59z7
Location in the Matrix	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559
Second Order SBox Terms for the ExpandRoundkey Step, cont...	k60z8	k60z9	k60z10	k60z11	k61z8	k61z9	k61z10	k61z11	k62z8	k62z9	k62z10	k62z11	k63z8	k63z9	k63z10	k63z11
Location in the Matrix	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575
Second Order SBox Terms for the ExpandRoundkey Step, cont...	k48z12	k48z13	k48z14	k48z15	k49z12	k49z13	k49z14	k49z15	k50z12	k50z13	k50z14	k50z15	k51z12	k51z13	k51z14	k51z15
Location in the Matrix	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591

Table 5: Matrix Terms - Second Half

Above are the terms used in the system of equations, they are numbered in the order they are stored in the matrix shown in the sample data.

APPENDIX B

INPUT DATA

The following 27 pages contain an example of the matrix input to the Gaussian Elimination. There are ten bits per each column of the table. If all the bits in a cell were zero, only one zero is shown instead of ten.

This example had the following data from the execution of rAES:

The plaintext is:

0111000001101100011000010110100101101110011101000110010101111000

The ciphertext is:

1011100010110111110110000111100011100101100101011011100110100010

The key is:

1000111110100100011111110000101100110100100101011110100011000001

43

44

45

48

49

	5-9	10-19	20-29	30-39	40-49	50-59	60-69	70-79	80-89	90-99	100-109	110-119	120-129	130-139	140-149	150-159	160-169	170-179	180-189	190-199
1040	0	0	0	0	0	0	0	0	0	0000000101	0100000000	0	0	0	0	0	0	0	0	0
1041	0	0	0	0	0	0	0	0	0	0000000101	0100000000	0	0	0	0	0	0	0	0	0
1042	0	0	0	0	0	0	0	0	0	0000000101	0001000000	0	0	0	0	0	0	0	0	0
1043	0	0	0	0	0	0	0	0	0	0000000001	0101000000	0	0	0	0	0	0	0	0	0
1044	0	0	0	0	0	0	0	0	0	0000000001	0101000000	0	0	0	0	0	0	0	0	0
1045	0	0	0	0	0	0	0	0	0	0000000001	0101000000	0	0	0	0	0	0	0	0	0
1046	0	0	0	0	0	0	0	0	0	0000000001	0101000000	0	0	0	0	0	0	0	0	0
1047	0	0	0	0	0	0	0	0	0	0000000001	0101000000	0	0	0	0	0	0	0	0	0
1048	0	0	0	0	0	0	0	0	0	0000000001	0100000000	0	0	0	0	0	0	0	0	0
1049	0	0	0	0	0	0	0	0	0	0000000001	0001000000	0	0	0	0	0	0	0	0	0
1050	0	0	0	0	0	0	0	0	0	0000000101	0100000000	0	0	0	0	0	0	0	0	0
1051	0	0	0	0	0	0	0	0	0	0000000101	0100000000	0	0	0	0	0	0	0	0	0
1052	0	0	0	0	0	0	0	0	0	0000000100	0101000000	0	0	0	0	0	0	0	0	0
1053	0	0	0	0	0	0	0	0	0	0000000101	0101000000	0	0	0	0	0	0	0	0	0
1054	0	0	0	0	0	0	0	0	0	0000000101	0	0	0	0	0	0	0	0	0	0

[illegible]

[illegible]

55

56

58

	200-209	210-219	220-229	230-239	240-249	250-259	260-269	270-279	280-289	290-299	300-309	310-319	320-329	330-339	340-349	350-359	360-369	370-379	380-389	390-399
1040	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1041	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1042	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1043	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1044	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1045	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1046	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1047	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1048	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1049	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1050	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1051	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1052	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1053	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1054	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

61

64

65

[illegible]

67

	400-409	410-419	420-429	430-439	440-449	450-459	460-469	470-479	480-489	490-499	500-509	510-519	520-529	530-539	540-549	550-559	560-569	570-579	580-589	590-599
1040	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1041	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1042	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1043	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1044	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1045	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1046	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1047	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1048	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1049	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1050	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1051	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1052	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1053	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1054	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

APPENDIX C

OUTPUT DATA

The following 28 pages contain an example of the matrix output from the Gaussian Elimination. There are ten bits per each column of the table. If all the bits in a cell were zero, only one zero is shown instead of ten.

This example had the following data from the execution of rAES:

The plaintext is:

0111000001101100011000010110100101101110011101000110010101111000

The ciphertext is:

1011100010110111110110000111100011100101100101011011100110100010

The key is:

1000111110100100011111110000101100110100100101011110100011000001

	0-9	10-19	20-29	30-39	40-49	50-59	60-69	70-79	80-89	90-99	100-109	110-119	120-129	130-139	140-149	150-159	160-169	170-179	180-189
0	1500000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0110100000	0	0	0	0	0	0	0	0	0	0	0	0	1000100000	0	0	0	0	0
2	0011000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0001101000	0	0	0	0	0	0	0	0	0	0	0	0000000010	0010000000	0	0	0	0	0
4	0000110000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0000010000	0	0	0	0	0	0	0	0	0	0	0	0000000010	0010000000	0	0	0	0	0
6	0000001100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0000000100	0	0	0	0	0	0	0	0	0	0	0	0	0010100000	0	0	0	0	0
8	0000000011	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0000000001	1010000000	0	0	0	0	0	0	0	0	0	0	0	0	0010000000	0	0	0	0
10	0	1100000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0110100000	0	0	0	0	0	0	0	0	0	0	0	0000001000	1000000000	0	0	0	0
12	0	0011000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0001000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0000100000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0000010000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0000001100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0000000100	1000000000	0	0	0	0	0	0	0	0	0	0	0	0000001000	1000000000	0	0	0
18	0	0000000011	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0000000001	1010000000	0	0	0	0	0	0	0	0	0	0	0	0000100010	0	0	0	0
20	0	0	1100000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0100000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0011000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0001000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0000100000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0000010000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0000001000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0000000100	1000000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0000000011	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0000000001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	1100000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0100000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0011000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0001010000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0000100000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0000010000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0000001000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0000000100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0000000011	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0000000001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	1100000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0110100000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
42	0	0	0	0	0011000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
43	0	0	0	0	0001010000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
44	0	0	0	0	0000100000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
45	0	0	0	0	0000010000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
46	0	0	0	0	0000001000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
47	0	0	0	0	0000000100	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0000000011	0	0	0	0	0	0	0	0	0	0	0	0	0	0
49	0	0	0	0	0000000001	1010000000	0	0	0	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	1100000000	0	0	0	0	0	0	0	0	0	0	0	0	0
51	0	0	0	0	0	0110100000	0	0	0	0	0	0	0	0	0	0	0	0	0
52	0	0	0	0	0	0011000000	0	0	0	0	0	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0001000000	0	0	0	0	0	0	0	0	0	0	0	0	0
54	0	0	0	0	0	0000100000	0	0	0	0	0	0	0	0	0	0	0	0	0
55	0	0	0	0	0	0000010000	0	0	0	0	0	0	0	0	0	0	0	0	0
56	0	0	0	0	0	0000001000	0	0	0	0	0	0	0	0	0	0	0	0	0
57	0	0	0	0	0	0000000100	1000000000	0	0	0	0	0	0	0	0	0	0	0	0
58	0	0	0	0	0	0000000011	0	0	0	0	0	0	0	0	0	0	0	0	0
59	0	0	0	0	0	0000000001	1010000000	0	0	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	1100000000	0	0	0	0	0	0	0	0	0	0	0	0
61	0	0	0	0	0	0	0100000000	0	0	0	0	0	0	0	0	0	0	0	0
62	0	0	0	0	0	0	0011000000	0	0	0	0	0	0	0	0	0	0	0	0
63	0	0	0	0	0	0	0001000000	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0000100000	0	0	0	0	0	0	0	0	0	0	0	0
65	0	0	0	0	0	0	0000010000	0	0	0	0	0	0	0	0	0	0	0	0
66	0	0	0	0	0	0	0000001000	0	0	0	0	0	0	0	0	0	0	0	0
67	0	0	0	0	0	0	0000000100	1000000000	0	0	0	0	0	0	0	0	0	0	0
68	0	0	0	0	0	0	0000000011	0	0	0	0	0	0	0	0	0	0	0	0
69	0	0	0	0	0	0	0000000001	0	0	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	1100000000	0	0	0	0	0	0	0	0	0	0	0
71	0	0	0	0	0	0	0110100000	0	0	0	0	0	0	0	0	0	0	0	0
72	0	0	0	0	0	0	0011000000	0	0	0	0	0	0	0	0	0	0	0	0
73	0	0	0	0	0	0	0001000000	0	0	0	0	0	0	0	0	0	0	0	0
74	0	0	0	0	0	0	0000100000	0	0	0	0	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0000010000	0	0	0	0	0	0	0	0	0	0	0	0
76	0	0	0	0	0	0	0000001000	0	0	0	0	0	0	0	0	0	0	0	0
77	0	0	0	0	0	0	0000000100	1000000000	0	0	0	0	0	0	0	0	0	0	0
78	0	0	0	0	0	0	0000000011	0	0	0	0	0	0	0	0	0	0	0	0
79	0	0	0	0	0	0	0000000001	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	1100000000	0	0	0	0	0	0	0	0	0	0	0
81	0	0	0	0	0	0	0	0110100000	0	0	0	0	0	0	0	0	0	0	0
82	0	0	0	0	0	0	0	0011000000	0	0	0	0	0	0	0	0	0	0	0
83	0	0	0	0	0	0	0	0001000000	0	0	0	0	0	0	0	0	0	0	0
84	0	0	0	0	0	0	0	0000100000	0	0	0	0	0	0	0	0	0	0	0
85	0	0	0	0	0	0	0	0000010000	0	0	0	0	0	0	0	0	0	0	0
86	0	0	0	0	0	0	0	0000001000	0	0	0	0	0	0	0	0	0	0	0
87	0	0	0	0	0	0	0	0000000100	1000000000	0	0	0	0	0	0	0	0	0	0
88	0	0	0	0	0	0	0	0000000011	0</										

72

[illegible]

75

77

	0-9	10-19	20-29	30-39	40-49	50-59	60-69	70-79	80-89	90-99	100-109	110-119	120-129	130-139	140-149	150-159	160-169	170-179	180-189
1032	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1033	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1034	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1035	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1036	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1037	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1038	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1039	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1040	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1041	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1042	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1043	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1044	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1045	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1046	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1047	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1048	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1049	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1050	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1051	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1052	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1053	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1054	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

86

	190-199	200-209	210-219	220-229	230-239	240-249	250-259	260-269	270-279	280-289	290-299	300-309	310-319	320-329	330-339	340-349	350-359	360-369	370-379
1032	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1033	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1034	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1035	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1036	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1037	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1038	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1039	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1040	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1041	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1042	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1043	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1044	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1045	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1046	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1047	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1048	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1049	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1050	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1051	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1052	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1053	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1054	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[illegible]

92

93

94

95

	380-389	390-399	400-409	410-419	420-429	430-439	440-449	450-459	460-469	470-479	480-489	490-499	500-509	510-519	520-529	530-539	540-549	550-559	560-569
1032	0	0	0	0	0	0	0	0000000001	0001101110	0010000000	0	0	0	0	0	0000011110	0	0	0
1033	0	0	0	0	0	0	0	0111101110	0011000000	0	0	0	0	0	0	0000011110	0	0	0
1034	0	0	0	0	0	0	0	0000000001	0101000010	0000000000	0	0	0	0	0	0000011000	0	0	0
1035	0	0	0	0	0	0	0	0000000011	1000110111	1110000000	0	0	0	0	0	0000010110	0	0	0
1036	0	0	0	0	0	0	0	0000000010	1101010101	1100000000	0	0	0	0	0	0000011100	0	0	0
1037	0	0	0	0	0	0	0	0000000010	1101010101	1100000000	0	0	0	0	0	0000010110	0	0	0
1038	0	0	0	0	0	0	0	0111101110	0011000000	0	0	0	0	0	0	0000011110	0	0	0
1039	0	0	0	0	0	0	0	0111101110	0011000000	0	0	0	0	0	0	0000011110	0	0	0
1040	0	0	0	0	0	0	0	0000000010	1010111011	1111000000	0	0	0	0	0	0000011110	0	0	0
1041	0	0	0	0	0	0	0	0000000010	1010111011	1111000000	0	0	0	0	0	0000011110	0	0	0
1042	0	0	0	0	0	0	0	0000000001	0101000010	0000000000	0	0	0	0	0	0000011010	0	0	0
1043	0	0	0	0	0	0	0	0000000010	1101010101	1100000000	0	0	0	0	0	0000010100	0	0	0
1044	0	0	0	0	0	0	0	0000000010	1101010101	1100000000	0	0	0	0	0	0000011100	0	0	0
1045	0	0	0	0	0	0	0	0000000010	1101010101	1100000000	0	0	0	0	0	0000011110	0	0	0
1046	0	0	0	0	0	0	0	0000000010	1101010101	1100000000	0	0	0	0	0	0000011110	0	0	0
1047	0	0	0	0	0	0	0	0000000010	1100101111	0101000000	0	0	0	0	0	0000000100	0	0	0
1048	0	0	0	0	0	0	0	0000000011	1000010111	0100000000	0	0	0	0	0	0000001110	0	0	0
1049	0	0	0	0	0	0	0	0000000001	0110000000	0001000000	0	0	0	0	0	0000010110	0	0	0
1050	0	0	0	0	0	0	0	0000000010	1010111011	1111000000	0	0	0	0	0	0000000100	0	0	0
1051	0	0	0	0	0	0	0	0000000010	1010111011	1111000000	0	0	0	0	0	0000000100	0	0	0
1052	0	0	0	0	0	0	0	0000000011	1100111011	1110000000	0	0	0	0	0	0000011110	0	0	0
1053	0	0	0	0	0	0	0	0000000010	1100101111	0101000000	0	0	0	0	0	0000010000	0	0	0
1054	0	0	0	0	0	0	0	0000000001	0001101110	0010000000	0	0	0	0	0	0000010110	0	0	0

	570-579	580-589	590-592	570-579	580-589	590-592	570-579	580-589	590-592	570-579	580-589	590-592	570-579	580-589	590-592	570-579	580-589	590-592	
0	0	0	000	129	0	0	001	258	0	0	000	387	0	0	000	516	0	0	000
1	0	0	000	130	0	0	000	259	0	0	000	388	0	0	000	517	0	0	000
2	0	0	001	131	0	0	000	260	0	0	000	389	0	0	000	518	0	0	001
3	0	0	001	132	0	0	000	261	0	0	001	390	0	0	001	519	0	0	000
4	0	0	001	133	0	0	001	262	0	0	000	391	0	0	000	520	0	0	000
5	0	0	001	134	0	0	000	263	0	0	000	392	0	0	000	521	0	0	000
6	0	0	001	135	0	0	001	264	0	0	000	393	0	0	000	522	0	0	000
7	0	0	000	136	0	0	000	265	0	0	001	394	0	0	000	523	0	0	000
8	0	0	000	137	0	0	001	266	0	0	001	395	0	0	000	524	0	0	000
9	0	0	000	138	0	0	000	267	0	0	000	396	0	0	000	525	0	0	000
10	0	0	000	139	0	0	000	268	0	0	000	397	0	0	000	526	0	0	000
11	0	0	000	140	0	0	000	269	0	0	000	398	0	0	000	527	0	0	000
12	0	0	000	141	0	0	000	270	0	0	000	399	0	0	000	528	0	0	000
13	0	0	000	142	0	0	000	271	0	0	000	400	0	0	000	529	0	0	000
14	0	0	000	143	0	0	000	272	0	0	000	401	0	0	000	530	0	0	000
15	0	0	001	144	0	0	001	273	0	0	000	402	0	0	000	531	0	0	000
16	0	0	000	145	0	0	001	274	0	0	000	403	0	0	000	532	0	0	000
17	0	0	000	146	0	0	000	275	0	0	000	404	0	0	000	533	0	0	000
18	0	0	001	147	0	0	000	276	0	0	000	405	0	0	000	534	0	0	001
19	0	0	001	148	0	0	000	277	0	0	000	406	0	0	000	535	0	0	000
20	0	0	001	149	0	0	001	278	0	0	000	407	0	0	000	536	0	0	000
21	0	0	001	150	0	0	000	279	0	0	000	408	0	0	000	537	0	0	000
22	0	0	000	151	0	0	001	280	0	0	000	409	0	0	000	538	0	0	000
23	0	0	001	152	0	0	000	281	0	0	000	410	0	0	000	539	0	0	000
24	0	0	001	153	0	0	000	282	0	0	000	411	0	0	000	540	0	0	000
25	0	0	001	154	0	0	000	283	0	0	000	412	0	0	000	541	0	0	000
26	0	0	001	155	0	0	001	284	0	0	000	413	0	0	000	542	0	0	000
27	0	0	000	156	0	0	000	285	0	0	000	414	0	0	000	543	0	0	000
28	0	0	000	157	0	0	001	286	0	0	000	415	0	0	000	544	0	0	000
29	0	0	000	158	0	0	000	287	0	0	000	416	0	0	000	545	0	0	000
30	0	0	000	159	0	0	001	288	0	0	000	417	0	0	000	546	0	0	000
31	0	0	001	160	0	0	000	289	0	0	000	418	0	0	000	547	0	0	000
32	0	0	000	161	0	0	001	290	0	0	000	419	0	0	000	548	0	0	000
33	0	0	000	162	0	0	001	291	0	0	000	420	0	0	000	549	0	0	000
34	0	0	001	163	0	0	001	292	0	0	000	421	0	0	000	550	0	0	000
35	0	0	001	164	0	0	000	293	0	0	000	422	0	0	000	551	0	0	000
36	0	0	001	165	0	0	000	294	0	0	001	423	0	0	000	552	0	0	000
37	0	0	001	166	0	0	000	295	0	0	000	424	0	0	000	553	0	0	000
38	0	0	000	167	0	0	001	296	0	0	000	425	0	0	000	554	0	0	000
39	0	0	001	168	0	0	000	297	0	0	000	426	0	0	000	555	0	0	000
40	0	0	000	169	0	0	001	298	0	0	000	427	0	0	000	556	0	0	000
41	0	0	000	170	0	0	000	299	0	0	000	428	0	0	000	557	0	0	000
42	0	0	000	171	0	0	000	300	0	0	000	429	0	0	000	558	0	0	000
43	0	0	000	172	0	0	000	301	0	0	000	430	0	0	000	559	0	0	000
44	0	0	000	173	0	0	000	302	0	0	001	431	0	0	000	560	0	0	000
45	0	0	000	174	0	0	000	303	0	0	000	432	0	0	000	561	0	0	000
46	0	0	001	175	0	0	000	304	0	0	000	433	0	0	000	562	0	0	000
47	0	0	000	176	0	0	000	305	0	0	000	434	0	0	000	563	0	0	000
48	0	0	000	177	0	0	000	306	0	0	000	435	0	0	000	564	0	0	000
49	0	0	000	178	0	0	000	307	0	0	000	436	0	0	000	565	0	0	000
50	0	0	001	179	0	0	001	308	0	0	000	437	0	0	000	566	0	0	000
51	0	0	001	180	0	0	000	309	0	0	000	438	0	0	001	567	0	0	000
52	0	0	001	181	0	0	001	310	0	0	000	439	0	0	000	568	0	0	000
53	0	0	000	182	0	0	000	311	0	0	000	440	0	0	000	569	0	0	000
54	0	0	000	183	0	0	001	312	0	0	000	441	0	0	000	570	0	0	000
55	0	0	001	184	0	0	000	313	0	0	000	442	0	0	000	571	0	0	000
56	0	0	001	185	0	0	001	314	0	0	000	443	0	0	000	572	0	0	000
57	0	0	000	186	0	0	000	315	0	0	000	444	0	0	000	573	0	0	000
58	0	0	000	187	0	0	000	316	0	0	000	445	0	0	000	574	0	0	000
59	0	0	000	188	0	0	000	317	0	0	000	446	0	0	000	575	0	0	000
60	0	0	000	189	0	0	000	318	0	0	000	447	0	0	000	576	0	0	000
61	0	0	000	190	0	0	000	319	0	0	000	448	0	0	000	577	0	0	000
62	0	0	001	191	0	0	000	320	0	0	000	449	0	0	000	578	0	0	000
63	0	0	000	192	0	0	001	321	0	0	000	450	0	0	000	579	0	0	000
64	0	0	000	193	0	0	001	322	0	0	000	451	0	0	000	580	0	0	000
65	0	0	000	194	0	0	000	323	0	0	000	452	0	0	000	581	0	0	000
66	0	0	001	195	0	0	000	324	0	0	000	453	0	0	001	582	0	0	001
67	0	0	001	196	0	0	000	325	0	0	000	454	0	0	000	583	0	0	000
68	0	0	001	197	0	0	001	326	0	0	000	455	0	0	000	584	0	0	000
69	0	0	001	198	0	0	000	327	0	0	000	456	0	0	000	585	0	0	000
70	0	0	000	199	0	0	000	328	0	0	000	457	0	0	000	586	0	0	000
71	0	0	001	200	0	0	000	329	0	0	000	458	0	0	000	587	0	0	000
72	0	0	001	201	0	0	000	330	0	0	000	459	0	0	000	588	0	0	000
73	0	0	001	202	0	0	000	331	0	0	000	460	0	0	000	589	0	0	000
74	0	0	001	203	0	0	000	332	0	0	000	461	0	0	000	590	0	0	000
75	0	0	001	204	0	0	000	333	0	0	000	462	0	0	000	591	0	0	000
76	0	0	001	205	0	0	000	334	0	0	000	463	0	0	000	592	0	0	000
77	0	0	001	206	0	0	000	335	0	0	000	464	0	0	000	593	0	0	000
78	0	0	000	207	0	0	000	336	0	0	000	465	0	0	000	594	0	0	000
79	0	0	001	208	0	0	000	337	0	0	000	466	0	0	000	595	0	0	000
80	0	0	000	209	0	0	001	338	0	0	000	467	0	0	000	596	0	0	000
81	0	0	000	210	0	0	000	339	0	0	000	468	0	0	000	597	0	0	000
82	0	0	001	211	0	0	000	340	0	0	000	469	0	0	000	598	0	0	000
83	0	0	001	212	0	0	000	341	0	0	000	470	0	0	000	599	0	0	000
84	0	0	001	213	0														

570-579	580-589	590-592	570-579	580-589	590-592	570-579	580-589	590-592	570-579	580-589	590-592	
645	0	0 000	774	0	0 001	903	0	0 001	1032	0000001110	0101111111	111
646	0	0 000	775	0	0 000	904	0	0 000	1033	0000001011	1101111111	100
647	0	0 000	776	0	0 000	905	0	0 000	1034	0000001111	1111110111	101
648	0	0 000	777	0	0 000	906	0	0 000	1035	0000001111	1001110111	111
649	0	0 000	778	0	0 000	907	0	0 000	1036	0000001110	1011111111	010
650	0	0 000	779	0	0 000	908	0	0 000	1037	0000001011	1111001111	010
651	0	0 000	780	0	0 000	909	0	0 001	1038	0000001110	1111011000	110
652	0	0 000	781	0	0 000	910	0	0 001	1039	0000001101	1111110010	110
653	0	0 000	782	0	0 000	911	0	0 000	1040	0000001101	0111001111	010
654	0	0 001	783	0	0 000	912	0	0 000	1041	0000000101	1010111011	110
655	0	0 000	784	0	0 000	913	0	0 000	1042	0000001011	1111111110	001
656	0	0 000	785	0	0 000	914	0	0 001	1043	0000001111	1110111100	110
657	0	0 000	786	0	0 000	915	0	0 000	1044	0000000110	0111111101	110
658	0	0 000	787	0	0 000	916	0	0 001	1045	0000001111	1000011111	100
659	0	0 000	788	0	0 000	917	0	0 001	1046	0000001010	1011111010	110
660	0	0 000	789	0	0 000	918	0	0 000	1047	0000000110	1111111111	110
661	0	0 000	790	0	0 000	919	0	0 000	1048	0000001111	1101011110	011
662	0	0 000	791	0	0 000	920	0	0 000	1049	0000001000	1101111111	111
663	0	0 000	792	0	0 000	921	0	0 001	1050	0000001111	1110101001	110
664	0	0 000	793	0	0 000	922	0	0 001	1051	0000001100	1110110111	110
665	0	0 000	794	0	0 000	923	0	0 001	1052	0000001011	1110100111	001
666	0	0 000	795	0	0 000	924	0	0 001	1053	0000001011	1011110011	111
667	0	0 000	796	0	0 000	925	0	0 000	1054	0000000111	1100111110	111
668	0	0 000	797	0	0 000	926	0	0 001				
669	0	0 000	798	0	0 000	927	0	0 001				
670	0	0 000	799	0	0 000	928	0	0 001				
671	0	0 000	800	0	0 000	929	0	0 000				
672	0	0 000	801	0	0 000	930	0	0 000				
673	0	0 000	802	0	0 000	931	0	0 000				
674	0	0 000	803	0	0 000	932	0	0 001				
675	0	0 000	804	0	0 000	933	0	0 001				
676	0	0 000	805	0	0 000	934	0	0 001				
677	0	0 000	806	0	0 001	935	0	0 000				
678	0	0 001	807	0	0 000	936	0	0 000				
679	0	0 000	808	0	0 000	937	0	0 001				
680	0	0 000	809	0	0 000	938	0	0 000				
681	0	0 000	810	0	0 000	939	0	0 000				
682	0	0 000	811	0	0 000	940	0	0 000				
683	0	0 000	812	0	0 000	941	0	0 001				
684	0	0 000	813	0	0 000	942	0	0 000				
685	0	0 000	814	0	0 000	943	0	0 000				
686	0	0 000	815	0	0 000	944	0	0 001				
687	0	0 000	816	0	0 000	945	0	0 001				
688	0	0 000	817	0	0 000	946	0	0 001				
689	0	0 000	818	0	0 000	947	0	0 000				
690	0	0 000	819	0	0 000	948	0	0 000				
691	0	0 000	820	0	0 000	949	0	0 001				
692	0	0 000	821	0	0 000	950	0	0 000				
693	0	0 000	822	0	0 001	951	0	0 000				
694	0	0 000	823	0	0 000	952	0	0 001				
695	0	0 000	824	0	0 000	953	0	0 000				
696	0	0 000	825	0	0 000	954	0	0 000				
697	0	0 000	826	0	0 000	955	0	0 001				
698	0	0 000	827	0	0 000	956	0	0 000				
699	0	0 000	828	0	0 000	957	0	0 000				
700	0	0 000	829	0	0 000	958	0	0 001				
701	0	0 000	830	0	0 000	959	0	0 001				
702	0	0 000	831	0	0 000	960	0	0 001				
703	0	0 000	832	0	0 000	961	0	0 001				
704	0	0 000	833	0	0 000	962	0	0 000				
705	0	0 000	834	0	0 000	963	0	0 001				
706	0	0 000	835	0	0 000	964	0	0 000				
707	0	0 000	836	0	0 000	965	0	0 000				
708	0	0 000	837	0	0 000	966	0	0 001				
709	0	0 000	838	0	0 000	967	0	0 001				
710	0	0 001	839	0	0 000	968	0	0 001				
711	0	0 000	840	0	0 000	969	0	0 001				
712	0	0 000	841	0	0 000	970	0	0 000				
713	0	0 000	842	0	0 000	971	0	0 000				
714	0	0 000	843	0	0 000	972	0	0 000				
715	0	0 000	844	0	0 000	973	0	0 000				
716	0	0 000	845	0	0 000	974	0	0 000				
717	0	0 000	846	0	0 000	975	0	0 000				
718	0	0 000	847	0	0 000	976	0	0 000				
719	0	0 000	848	0	0 000	977	0	0 000				
720	0	0 000	849	0	0 000	978	0	0 001				
721	0	0 000	850	0	0 000	979	0	0 001				
722	0	0 000	851	0	0 000	980	0	0 001				
723	0	0 000	852	0	0 000	981	0	0 001				
724	0	0 000	853	0	0 000	982	0	0 001				
725	0	0 000	854	0	0 000	983	0	0 000				
726	0	0 001	855	0	0 000	984	0	0 001				
727	0	0 000	856	0	0 000	985	0	0 001				
728	0	0 000	857	0	0 000	986	0	0 000				
729	0	0 000	858	0	0 000	987	0	0 000				
730	0	0 000	859	0	0 000	988	0	0 001				
731	0	0 000	860	0	0 000	989	0	0 001				
732	0	0 000	861	0	0 000	990	0	0 001				
733	0	0 000	862	0	0 000	991	0	0 001				
734	0	0 000	863	0	0 000	992	0	0 000				
735	0	0 000	864	0	0 000	993	0	0 000				
736	0	0 000	865	0	0 000	994	0	0 000				
737	0	0 000	866	0	0 000	995	0	0 001				
738	0	0 000	867	0	0 000	996	0	0 001				
739	0	0 000	868	0	0 000	997	0	0 001				
740	0	0 000	869	0	0 000	998	0	0 001				
741	0	0 000	870	0	0 001	999	0	0 001				
742	0	0 000	871	0	0 000	1000	0	0 001				
743	0	0 000	872	0	0 000	1001	0	0 000				
744	0	0 000	873	0	0 000	1002	0	0 000				
745	0	0 000	874	0	0 000	1003	0	0 000				
746	0	0 000	875	0	0 000	1004	0	0 000				
747	0	0 000	876	0	0 000	1005	0	0 000				
748	0	0 000	877	0	0 000	1006	0	0 001				
749	0	0 000	878	0	0 000	1007	1111110000	0 001				
750	0	0 000	879	0	0 000	1008	1111110000	0 001				
751	0	0 000	880	0	0 000	1009	1111100000	0 001				
752	0	0 000	881	0	0 000	1010	1011100000	0 001				
753	0	0 000	882	0	0 000	1011	0111110000	0 001				
754	0	0 000	883	0	0 000	1012	1111100000	0 000				
755	0	0 000	884	0	0 000	1013	1111010000	0 000				
756	0	0 000	885	0	0 000	1014	1100110000	0 001				
757	0	0 000	886	0	0 000	1015	0010110000	0 001				
758	0	0 000	887	0	0 000	1016	1111100000	0 001				
759	0	0 000	888	0	0 000	1017	1011100000	0 001				
760	0	0 000	889	0	0 000	1018	1110000000	0 001				
761	0	0 000	890	0	0 000	1019	1100110000	0 000				
762	0	0 000	891	0	0 000	1020	1101110000	0 000				
763	0	0 000	892	0	0 000	1021	1111100000	0 000				
764	0	0 000	893	0	0 000	1022	1010110000	0 000				
765	0	0 000	894	0	0 000	1023	1111110000	0 001				
766	0	0 000	895	0	0 000	1024						