6-2018

MODERN CRYPTOGRAPHY

Samuel Lopez

California State University - San Bernardino, samuel222lopez@gmail.com

Modern Cryptography

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Arts

in

Mathematics

---

by

Samuel Lopez

June 2018

Modern Cryptography

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Samuel Lopez

June 2018

Approved by:

Shawnee McMurran, Committee Chair	Date
Joseph Chavez, Committee Member	
Jeremy Aikin, Committee Member	
Charles Stanton, Chair, Department of Mathematics	Corey Dunn Graduate Coordinator, Department of Mathematics

Abstract

We live in an age where we willingly provide our social security number, credit card information, home address and countless other sensitive information over the Internet. Whether you are buying a phone case from Amazon, sending in an on-line job application, or logging into your on-line bank account, you trust that the sensitive data you enter is secure. As our technology and computing power become more sophisticated, so do the tools used by potential hackers to our information. In this paper, the underlying mathematics within ciphers will be looked at to understand the security of modern ciphers.

An extremely important algorithm in today's practice is the Advanced Encryption Standard (AES), which is used by our very own National Security Agency (NSA) for data up to TOP SECRET. Another frequently used cipher is the RSA cryptosystem. Its security is based on the concept of prime factorization, and the fact that it is a hard problem to prime factorize huge numbers, numbers on the scale of 2^{2048} or larger. Cryptanalysis, the study of breaking ciphers, will also be studied in this paper. Understanding effective attacks leads to understanding the construction of these very secure ciphers.

Acknowledgements

I wish to thank Dr. Shawnee McMurran for all her help and support in the development and formation of this paper. I also want to thank Dr. Joseph Chavez and Dr. Jeremy Aikin for their comments on the paper as well. Your assistance was very much appreciated.

Table of Contents

Abstract **iii**

Acknowledgements **iv**

List of Tables **vii**

List of Figures **viii**

1 Introduction **1**

2 Data Encryption Standard **3**
2.1 Motivation . . . 3
2.2 Security . . . 3
2.3 Understanding the Data Encryption Standard . . . 4
2.4 Encryption Method . . . 4
2.5 Key Generation . . . 6
2.6 Decryption . . . 7

3 Advanced Encryption Standard **9**
3.1 Motivation . . . 9
3.2 Security . . . 9
3.3 Understanding the Advanced Encryption Standard . . . 10
3.4 Encryption Method . . . 11
3.5 Key Expansion . . . 12
3.6 Example of Encrypting a 128-bit Message . . . 13
3.7 Decryption . . . 18

4 RSA **20**
4.1 Motivation . . . 20
4.2 Security . . . 20
4.3 Method . . . 21
4.4 A Worked Example . . . 22
4.5 Application . . . 24

5 Discrete Logarithm Problem **25**
5.1 Motivation . . . 25
5.2 Diffie-Hellman Key Exchange . . . 25
5.3 Discrete Logarithm Problem . . . 26
5.4 Small-Step Giant-Step Algorithm . . . 26
5.5 Elliptic Curves . . . 27

6 Hash Functions **31**
6.1 Motivation . . . 31
6.2 Understanding Properties of Secure Hash Functions . . . 33

7 Blockchain Technology **35**
7.1 "Satoshi Nakamoto's" White Paper . . . 35
7.2 Security of Blockchain . . . 35
7.3 Bitcoin . . . 37

8 Conclusion **38**

Bibliography **39**

List of Tables

2.1 Expansion Function bit Mapping 5
2.2 S_1 Table With Decimal Numbers 5
2.3 Permutation Choice 1 7
2.4 Permutation Choice 2 7

3.1 S-box for AES in Hexadecimal 10

4.1 A brute-force attack 23

List of Figures

5.1 The elliptic curve defined by $y^2 = x^3 - 4x + 10$. 29
5.2 Geometric derivation of $P + Q$. 29
5.3 Geometric derivation of $2P$. 30

Chapter 1

Introduction

Many of us realize that mathematics has more applications than we know; growing up we were constantly reminded that math can apply to almost everything. One of those things is keeping our sensitive data secure. One of the earliest uses of cryptography dates back two-thousand years, developed when Julius Caesar realized the need to encode military messages to and from his commanders. It was a simple substitution cipher where every letter would be replaced by a different letter. For Caesar's Cipher the shift was 3 letters; so every "A" in your original message would be replaced by a "D", every "B" by a "E," and so on. To decrypt the message simply shift 3 letters in the opposite direction. Frequency analysis provides a method to easily break substitution ciphers like Ceaser's Cipher, especially with the aid of technology, so cryptographers had to invent new ways to encode information. Symmetric ciphers like the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) encode blocks of information at a time using a private key that all parties trying to communicate must know. Asymmetric ciphers like RSA and Diffie-Helman key exchange use computationally hard mathematical problems to allow one key to be public, so anyone can send an encrypted message, but also incorporate a private key that only selected individuals know. The private key allows one to decrypt the message.

Sometimes we are required to have sensitive data stored on a database, like a password or a pin number to a debit card. These are not messages that you want someone to decode using a key, you just want the data stored to verify it is you every time you log into a website or make a withdrawal from an ATM. For this, hash functions have proven

useful. Recent applications of hash functions have exposed the world of cryptography to a much larger audience. For example, the invention of Blockchain technology, and as a consequence Bitcoin, is shaping the future: from being able to securely send or receive money from someone you may not trust, to establishing a smart contract between two parties that everyone in the network can bear witness to. As we will see, cryptography is a revolutionary application of mathematics.

Chapter 2

Data Encryption Standard

2.1 Motivation

In 1977, the National Security Agency (NSA) announced that there was a need for a secure standardized cipher for commercial use. IBM created the Data Encryption Standard (DES), a block cipher that encodes blocks of bits at a time, as opposed to a stream cipher, which encodes only one bit at a time. Although stream ciphers are impossible to break using a cryptographic secure random number generator, they are too impractical for general applications since keys can not be reused. Block ciphers are more practical because keys are of reasonable length, at most 256 bits, and can be used securely for a whole session of communication.

2.2 Security

In the 1970's, block ciphers were in high demand because of how efficient they are in encoding information. For example, the DES can encrypt 64 bits at a time. The challenge was creating a block cipher both secure from attacks that plagued the block ciphers of the past and, as later discovered, differential cryptanalysis attacks. The inventor of information theory, Claude Shannon, stated that a secure block cipher must have two main attributes: confusion, which is a basic substitution table, and diffusion, which means a single bit should affect the entire message. Although differential cryptanalysis was not public knowledge until 18 years after the publication of DES, the NSA and IBM research teams knew of the attack and created their substitution tables to be resilient to

this attack. Today, DES is no longer secure after being under the microscope of so many cryptographers for over 20 years. But 3DES, which encrypts using DES three times in a row, is still a commonly used block cipher.

2.3 Understanding the Data Encryption Standard

DES is not too complex from an abstract perspective; encryption relies on confusion and diffusion functions. First, DES takes 64 bits of data at a time. The 64 bits will go through 16 rounds that all have the same functionality. First, the 64 bits are split into two groups, each consisting of 32 bits. Let us denote the first set of 32 bits L_0, the last 32 bits by R_0. In any one round of DES, only the first 32 bits are encrypted. For example, round 1 would encrypt L_0 and set the encrypted 32 bits to be R_1. The set R_0 is kept the same and set to L_1. For round 2 of encryption, L_1 is encrypted and set to be R_2, while R_1 is unchanged and set to L_2. This process continues for 16 rounds.

2.4 Encryption Method

Now, with a general understanding of how DES encrypts, we can look at the inner workings of each round and how the encryption actually works. As stated before, 64 bits are encrypted over 16 rounds. All rounds have the same operations, so we need only understand one round. For simplicity, let us look at the first round of DES for 64 bits of data. The 64 bits of data are divided in half, giving us L_0 and R_0 as described above. The block of 32 bits, R_0, has two roles in the first round: R_0 is set to be L_1 ready to be encrypted in the next round, and R_0 along with the round key K_0, are input into a particular function f, which we describe below. We set $R_1 = L_0 \oplus f(R_0, K_0)$, where $\oplus$ denotes the XOR operation.

"Exclusive or" (XOR) is a logical operation that outputs true only when inputs differ. For example,

$$\begin{aligned} & 10001110_2 \\ \oplus \quad & 10011000_2 \\ = \quad & 00010110_2. \end{aligned}$$

The function f is where the confusion and diffusion take place in the form of

Table 2.1: Expansion Function bit Mapping

input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
output	31	0	1	2	3	4	3	4	5	6	7	8	7	8	9	10
	11	12	11	12	13	14	15	16	15	16	17	18	19	20	19	20
	21	22	23	24	23	24	25	26	27	28	27	28	29	30	31	0

Table 2.2: S_1 Table With Decimal Numbers

S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

four operations. First, the 32 bits pass through an expansion function E, making the 32 bits into 48 bits. Second, the 48 bits are XOR'd with the 48-bit round sub-key. Third, the 48 bits are separated into eight groups of 6 bits each and are changed via corresponding substitution table S: S_1 for the first group, S_2 for the second group, and so on. Each substitution table inputs 6 bits and outputs 4 bits, leaving us with 32 bits after the substitution. Finally the 32 bits are permuted.

The expansion function E applies diffusion, something essential for block ciphers; E takes 32 bits and outputs 48 bits. This is done by sending half of the 32 bits to two locations. The other half of the 32 bits are mapped to just one bit in the output of E. See Table 2.1 for a complete mapping of bits performed by E.

After the expansion function, the 48 bits are then XOR'd with the 48 bit round key. We then still have 48 bits.

The 48 bits, in order from left to right, are separated into 8 groups, each containing 6 bits of data. Each group of 6 bits have a particular address in a substitution box. There are 8 different substitution boxes, one for each group. Intuitively, the first group would use box S_1, the second group box S_2, and so on. We will illustrate this process with an example. For simplicity we will look at only the first 6 bits of data and calculate the output using S_1.

Reading the S_1 table may seem somewhat counterintuitive at first, but the process is fairly straightforward after seeing it done. For example, if the first 6 bits of data are 011001_2, we would convert the middle four bits 1100_2 to decimal numbers giving us 12. This directs us to look at column 12. We then take the outside two bits 01_2 and convert it to the corresponding decimal number 1. This tells us to look at row 1. Finally, we take the entry in column 12, row 1, and convert back to base 2. Thus, $S_1(011001_2) = 9 = 1001_2$. Another computation, $S_1(110101_2) = 3 = 0011_2$. So, 6 bits in gives us 4 bits out. After all 8 groups have passed through their corresponding S-Box, we are left with 32 bits. This S-box is the element of confusion that DES uses.

Finally, the bits go through a permutation which are then output to encrypt L_i via $R_{i+1} = L_i \oplus f(R_i, K_i)$.

2.5 Key Generation

DES is a block cipher that encrypts data over 16 rounds. For each of those rounds an encryption key is used. DES starts with a 64-bit key that undergoes multiple permutations to generate a unique key for each round. The key generation starts by permutation choice 1 (PC1), Table 2.3. The number in Table 2.3 represents the new location for each bit. For example, the fifty-seventh bit in the original 64-bit key would be the first bit after the permutation, the forty-ninth bit would become the second bit, and so on. Permutation choice 1 is only performed once, on the original 64-bit key. We also note that only 56 bits of the original 64 bits remain after PC1. For this reason DES has a cryptographic strength of a 56-bit cipher, not the strength of a 64-bit cipher. Next, the 56 bits are split into two groups, each group being 28 bits long. The next permutation is a left shift of bits in the two groups. For rounds 1, 2, 9, and 16, the shift is one spot to the left. For all the remaining rounds, the shift is two spots to the left. We observe that the left shift is one spot to the left for four rounds and two spots for the remaining twelve rounds; thus the total number of left shifts is 28 spots. This coincides with the fact that the two groups are each 28 bits long. The final permutation is called permutation choice 2 (PC2) and works in the same way as PC1. See Table 2.4 for the permutation that PC2 performs. The 48 bits that remain after PC2 are used as the encryption key. In general, key generation would go as follows. The original 64-bit key undergoes the first permutation PC1, leaving 56 bits. The 56 bits would be split into two groups each 28

Table 2.3: Permutation Choice 1

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Table 2.4: Permutation Choice 2

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

bits long. Let us denote the first 28 bits c_0 and the second 28 bits d_0. Both c_0 and d_0 are permuted via a left shift of one spot. We will call the first group c_1 and the second group d_1 after the left shift. Next, the 56 bits that make up c_1 and d_1 undergo the second permutation choice PC2. The 48 bits that remain are used as the first key for encryption. For the second round of key generation, c_1 and d_1 undergo a left shift of one spot; after the left shift c_2 and d_2 are permuted via PC2 where the resulting 48 bits are used for the second round of encryption. For the third round of encryption c_2 and d_2 are shifted two spots to the left. The resulting groups c_3 and d_3 are permuted via PC2 where the third key for encryption is made. This process continues until 16 keys are generated for each round of encryption.

2.6 Decryption

After all 16 rounds of encryption, the sender sends the final results L_{16} and R_{16} to the intended receiver. Thus, the receiver will have L_{16} and R_{16} directly from the sender. For symmetric ciphers, it is necessary for all parties to have the encryption key, so it is assumed that the receiver has access to the key and the ability to compute all round sub-keys. Once the receiver has all round keys, they need only do the steps in reverse order. Starting from the last round, $L_{16} = R_{15}$, so the first round of decryption needs

only the calculation of L_{15}. We know from encryption that $R_{16} = L_{15} \oplus f(R_{15}, K_{15})$, thus $R_{16} \oplus f(R_{15}, K_{15}) = L_{15}$. This process continues until L_1 and R_1, the original message, are retrieved.

Chapter 3

Advanced Encryption Standard

3.1 Motivation

After attacks on DES became too efficient, there was a call by the NSA for a new encryption standard. Unlike DES, which was developed in secret and then eventually published a few years later, the need for a new encryption standard was announced as a kind of competition. Many prominent cryptographers of the time formed teams and created ciphers. In addition to the requirement to be very secure, there were other criteria for submissions. Submissions must be 128-bit block ciphers, supporting 128-bit, 196-bit, and 256-bit key lengths. Submissions must also be computationally efficient, enough so for the commercial use the cipher was intended for. After a long evaluation process, Rijndael, the submission by Vincent Rijmen and Joan Daemen, was chosen and became the Advanced Encryption Standard (AES).

3.2 Security

Like DES, the security of AES comes from confusion and diffusion. The major improvements to security in AES were the increased block length to 128 bits and the key length. AES supports three different key lengths, 128, 198, and 256 bits. The different key lengths require a different number of rounds to be considered secure. A 128-bit key needs 10 rounds, a 196-bit key requires 12 rounds, and a 256-bit key calls for 14 rounds. The number of rounds for the various key lengths was determined by the team that developed the cipher.

Table 3.1: S-box for AES in Hexadecimal

S-Box	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
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	A0
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	52	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	EF	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
90	60	B1	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

Understanding Cryptography a Textbook for Students and Practitioners

3.3 Understanding the Advanced Encryption Standard

The AES is a byte cipher in the sense that the operations that apply confusion and diffusion work with bytes; bytes are simply larger bits, 8 bits make a single byte. The AES is also a round cipher, confusion and diffusion take place in multiple rounds. Before the first round, the original key K_0 is XOR'd byte by byte to the original 16 bytes of data. In all the rounds except the last, there are four operations. The first operation is the use of a substitution box. As we know from DES, this is an element of confusion. Unlike DES, there is only one substitution box in AES. After substitution, there is a shift row operation followed by a mix column operation, together applying diffusion among the bytes. Finally, there is a key, which is XOR'd byte for byte. The last round skips the mix column operation. The generation of the S-box for AES requires some general understanding of Galois fields, in particular GF(2^8). This is significant because the S-box has a mathematical generation that can be understood, as opposed to the S-boxes of DES that were made in secret and seemingly random.

3.4 Encryption Method

The key length does not actually change what operations take place in the encryption, although longer keys have more rounds. For simplicity we will look at encryption using a 128-bit key length which calls for 10 rounds. AES will encrypt 128 bits at a time, regardless of key length. AES works with bytes, so the 128 bits are broken up into bytes producing $\frac{128}{8} = 16$ bytes. In this paper, the 16 bytes will be converted to hexadecimal. The 16 bytes are XOR'd with the original key. Then the 16 bytes pass through the substitution box AES employs. The S-box for AES is more intuitive than the S-box of DES. For example, if the byte in question was "C3", we would use "C" to determine which row to use, and "3" to determine which column. Referring to Table 3.1, we see "C3" would have an output of "2E". A byte of "F1" would use row F and column 1, giving output "A1". This is the confusion element of AES. The remaining operations for the round are easier to understand via a matrix. We name the 16 bytes of data after the original key: $A_0, A_1, A_2, ..., A_{15}$. Suppose after the S-box we have $S(A_0) = B_0, S(A_1) = B_1, ..., S(A_{15}) = B_{15}$, still leaving us with 16 bytes. Entering the bytes column-wise into a 4×4 matrix produces,

$$M_0 = \begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \end{bmatrix}.$$

A shift row operation will shift the first row zero spots to the left, the second row one spot to the left, the third row two spots to the left, and the fourth row three spots to the left. Hence, M_0 is mapped to,

$$M_1 = \begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_5 & B_9 & B_{13} & B_1 \\ B_{10} & B_{14} & B_2 & B_6 \\ B_{15} & B_3 & B_7 & B_{11} \end{bmatrix}.$$

A mix column operation is the next step and it involves multiplying M_1 by the constant matrix

$$C = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix},$$

where the matrix multiplication done over $GF(2^8)$. The resulting matrix would then be XOR'd entry-wise with K_1, thus completing round 1.

3.5 Key Expansion

It would be unwise to use the same key for each round of AES encryption. So, an algorithm was developed to take a 128-bit key, and using the original 128 bits, generate unique keys for each of the 10 rounds. This process is called key expansion and goes as follows.

Given an original 128 bit key, 16 bytes, we enter the bytes column-wise into a 4×4 matrix. Let the first byte of the key be k_0, the second byte be k_1, and so on. Then we have the key matrix

$$K_0 = \begin{bmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{bmatrix}.$$

The columns are then stored in what are called *words*, $w_0 = \{k_0, k_1, k_2, k_3\}$, $w_1 = \{k_4, k_5, k_6, k_7\}$, $w_2 = \{k_8, k_9, k_{10}, k_{11}\}$, and $w_3 = \{k_{12}, k_{13}, k_{14}, k_{15}\}$. We need a total of forty-four words for a 128-bit key; four words, the original four, are used before the plain-text enters the first round. The remaining forty words are used four at a time, for each of the ten rounds of encryption. To compute the next four words we would first compute $g(w_3)$ and set $w_4 = w_0 \oplus g(w_3)$, $w_5 = w_4 \oplus w_1$, $w_6 = w_5 \oplus w_2$, and $w_7 = w_6 \oplus w_3$. In general, to calculate w_i, w_{i+1}, w_{i+2}, and w_{i+3}, where i is divisible by 4, we would first compute $w_i = w_{i-4} \oplus g(w_{i-1})$. We then use w_i to calculate $w_{i+1} = w_i \oplus w_{i-3}$, $w_{i+2} = w_{i+1} \oplus w_{i-2}$, and $w_{i+3} = w_{i+2} \oplus w_{i-1}$.

Now, the function g operates in three steps. First it will shift the bytes to the left. So, for example, w_3 would be $\{k_{13}, k_{14}, k_{15}, k_{12}\}$. The bytes would then be substituted

via the S-box as described in the encryption section. Finally, a round constant, c_i, is to be XOR'd. The round constant is generated recursively. To start, $c_1 = 01_2$. Then, the remaining round constants may by calculated by $c_i = 02 \times c_{i-1}$. It is important to note that $\times$ is polynomial multiplication over $GF(2^8)$ reduced by the AES modulo $x^8+x^4+x^3+x+1$. We note that 02 over $GF(2^8)$ is multiplication with the corresponding polynomial c_{i-1} by x.

3.6 Example of Encrypting a 128-bit Message

Alice would like to send the message "THE BOSS IS HERE" to Bob. Alice and Bob have already agreed to use the key "APPLE SAUCE OLI!" in case of an emergency. Throughout the paper all text to hex conversion is done via the American Standard Code for Information Interchange (ASCII). Now, "THE BOSS IS HERE" in hexadecimal is

$$54484520424F53532049532048455245$$

and "APPLE SAUCE OLI!" is

$$4150504C45205341554345204F4C4921.$$

This is the key used before the first round of encryption. Let us compute the key expansion with the given 128-bit starting key:

$$K_0 = \begin{bmatrix} 41 & 45 & 55 & 4F \\ 50 & 20 & 43 & 4C \\ 50 & 53 & 45 & 49 \\ 4C & 41 & 20 & 21 \end{bmatrix}.$$

We have $w_0 = \{41, 50, 50, 4C\}$, $w_1 = \{45, 20, 53, 41\}$, $w_2 = \{55, 43, 45, 20\}$, and $w_3 = \{4F, 4C, 49, 21\}$. We must now compute $g(w_3)$, obtaining $\{4C, 49, 21, 4F\}$ after the left shift of the bytes. Next, using the S-box, Table 3.1, we get $\{29, 3B, FD, 84\}$. Last, we XOR the *word* with the first round constant, $\{01, 00, 00, 00\}$. Consequently we have, $29 = 00101001_2$,

$$\begin{aligned} 29_{16} \oplus 01_2 &= 00101001_2 \oplus 01_2 \\ &= 00101000_2 \\ &= 28_{16}. \end{aligned}$$

The remaining entries of w_3 are unchanged. This yields $g(w_3) = \{28, 3B, FD, 84\}$. We compute w_4 as described above:

$$\begin{aligned} w_4 &= w_0 \oplus g(w_3) \\ &= \{41 \oplus 28, 50 \oplus 3B, 50 \oplus FD, 4C \oplus 84\} \\ &= \{D9, 6B, AD, C8\}. \end{aligned}$$

Performing the remaining XOR operations in the same way gives $w_5 = \{9C, 4B, FE, 89\}$, $w_6 = \{C9, 08, BB, A9\}$, and $w_7 = \{86, 44, F2, 88\}$, giving us our first round key,

$$K_1 = \{D9, 6B, AD, C8, 9C, 4B, FE, 89, C9, 08, BB, A9, 86, 44, F2, 88\}.$$

Now, as before, to compute the next four *words* we start by computing $g(w_7)$:

$$\begin{aligned} \{86, 44, F2, 88\} &\rightarrow \{44, F2, 88, 86\} \\ &\rightarrow \{1B, 89, C4, 44\} \\ &\rightarrow \{19, 89, C4, 44\}. \end{aligned}$$

Multiplication by 2 in binary shifts the number in question to the left one place value and inserts a zero into the one's place, in much the same way as multiplying by 10 with decimal numbers. Thus, computing the round constant is trivial until the 9th round, where the previous round constant was 10000000_2 and multiplication by 02 would push the '1' out of the 8 bits that make the byte. To fix this, the polynomial representation of $10000000 \times 02 = x^7 \cdot x = x^8$, must be reduced back into the field $GF(2^8)$ via the polynomial $x^8 + x^4 + x^3 + x + 1$. So, the polynomial representation of our 9th round constant will be

$$x^8 \mod (x^8 + x^4 + x^3 + x + 1) \equiv x^4 + x^3 + x + 1 \mod (x^8 + x^4 + x^3 + x + 1),$$

which translates to our 9th round constant $c_9 = \{00011011, 00, 00, 00\}$. We compute the 10th round constant by multiplying the 9th round constant by 02, which yields $c_{10} = \{00110110, 00, 00, 00\}$. Following the key expansion scheme as described, we obtain all 10 round keys:

$K_0 = \{41, 50, 50, 4C, 45, 20, 53, 41, 55, 43, 45, 20, 4F, 4C, 49, 21\}$
$K_1 = \{D9, 6B, AD, C8, 9C, 4B, FE, 89, C9, 08, BB, A9, 86, 44, F2, 88\}$

$K_2 = \{C0, E2, 69, 8C, 5C, A9, 97, 05, 95, A1, 2C, AD, 13, E5, DE, 24\}$
$K_3 = \{1D, FF, 5F, F1, 41, 56, C8, F4, D4, F7, E4, 59, C7, 12, 3A, 7D\}$
$K_4 = \{DC, 7F, CD, 37, 9D, 29, 05, C3, 49, DE, E1, 9A, 8E, CC, DB, E7\}$
$K_5 = \{87, C5, 59, 2E, 1A, EC, 5C, ED, 53, 32, BD, 77, DD, FE, 66, 90\}$
$K_6 = \{1C, F6, 39, EF, 06, 1A, 65, 02, 55, 28, D8, 75, 88, D6, BE, E5\}$
$K_7 = \{AA, 58, E0, 2B, AC, 42, 85, 29, F9, 6A, 5D, 5C, 91, BC, E3, B9\}$
$K_8 = \{4F, 59, 39, EF, E3, 0B, BC, C6, 1A, 61, E1, 9A, 8B, DD, 02, 23\}$
$K_9 = \{95, 2E, 1F, D2, 76, 25, A3, 14, 6C, 44, 42, 8E, E7, 99, 40, AD\}$
$K_{10} = \{4D, 27, 8A, 46, 3B, 02, 29, 52, 57, 46, 6B, DC, B0, DF, 2B, 71\}$

Now, we can start the encryption process. Entering our plain-text message into a matrix as described in Section 3.4 we have

$$M_0 = \begin{bmatrix} 54 & 42 & 20 & 48 \\ 48 & 4F & 49 & 45 \\ 45 & 53 & 53 & 52 \\ 20 & 53 & 20 & 45 \end{bmatrix}.$$

We then XOR M_0 entry-wise with the original key K_0, obtaining

$$\begin{bmatrix} 54 & 42 & 20 & 48 \\ 48 & 4F & 49 & 45 \\ 45 & 53 & 53 & 52 \\ 20 & 53 & 20 & 45 \end{bmatrix} \oplus \begin{bmatrix} 41 & 45 & 55 & 4F \\ 50 & 20 & 43 & 4C \\ 50 & 53 & 45 & 49 \\ 4C & 41 & 20 & 21 \end{bmatrix} = \begin{bmatrix} 15 & 07 & 75 & 07 \\ 18 & 6F & 0A & 09 \\ 15 & 00 & 16 & 1B \\ 6C & 12 & 00 & 64 \end{bmatrix}.$$

Next, we substitute each entry of our current matrix via the S-box from Table 3.1, which gives us

$$\begin{bmatrix} 59 & C5 & 9D & C5 \\ AD & A8 & 67 & 01 \\ 59 & 63 & 47 & AF \\ 50 & C9 & 63 & 43 \end{bmatrix}.$$

We then apply the shift row operation to get

$$\begin{bmatrix} 59 & C5 & 9D & C5 \\ A8 & 67 & 01 & AD \\ 47 & AF & 59 & 63 \\ 43 & 50 & C9 & 63 \end{bmatrix}.$$

The next step in Round 1 involves multiplying by the constant matrix

$$C = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

to obtain the matrix

$$\begin{bmatrix} 55 & C7 & B2 & 7D \\ 98 & B1 & BD & 42 \\ BA & 17 & 6E & 0B \\ 82 & 3C & 6D & 5C \end{bmatrix}.$$

Finally, the above matrix is XOR'd entry-wise with K_1, producing

$$M_1 = \begin{bmatrix} 8C & 5B & 7B & FB \\ F3 & FA & B5 & 06 \\ 17 & E9 & D5 & F9 \\ 4A & B5 & C4 & D4 \end{bmatrix}.$$

Round 1 is complete. This process repeats for 9 more rounds. The only exception is Round 10 omits the shift column operation. Even after one round of AES, the 16 bytes

$$8CF3174A5BFAE9B57BB5D5C4FB06F9D4$$

have few recognizable characters, none of which were in the original plain-text message. Continuing after Round 2 we have

$$M_2 = \begin{bmatrix} 34 & 05 & 43 & 18 \\ 91 & 47 & 80 & CE \\ FE & 33 & E3 & C4 \\ 9E & B5 & E0 & 1E \end{bmatrix}.$$

After Round 3 the matrix is

$$M_3 = \begin{bmatrix} 60 & CC & 08 & 3C \\ 92 & 93 & E3 & 19 \\ E9 & 4B & 7C & B3 \\ 36 & 8E & F6 & OA \end{bmatrix}.$$

Let us look at a new example. We will encrypt "THE BOSS iS HERE" and see how the change of a single character, a capital I to lowercase i, affects the matrix after a single round of AES. The key will be the same, so the key expansion is the same. We start as before, with an XOR operation. Our initial matrix this time is

$$M_0^* = \begin{bmatrix} 54 & 42 & 20 & 48 \\ 48 & 4F & 69 & 45 \\ 45 & 53 & 53 & 52 \\ 20 & 53 & 20 & 45 \end{bmatrix}.$$

We compute

$$\begin{bmatrix} 54 & 42 & 20 & 48 \\ 48 & 4F & 69 & 45 \\ 45 & 53 & 53 & 52 \\ 20 & 53 & 20 & 45 \end{bmatrix} \oplus \begin{bmatrix} 41 & 45 & 55 & 4F \\ 50 & 20 & 43 & 4C \\ 50 & 53 & 45 & 49 \\ 4C & 41 & 20 & 21 \end{bmatrix} = \begin{bmatrix} 15 & 07 & 75 & 07 \\ 18 & 6F & 2A & 09 \\ 15 & 00 & 16 & 1B \\ 6C & 12 & 00 & 64 \end{bmatrix}.$$

Again, we substitute via the S-box from Table 3.1 to get,

$$\begin{bmatrix} 59 & C5 & 9D & C5 \\ AD & A8 & E5 & 01 \\ 59 & 63 & 47 & AF \\ 50 & C9 & 63 & 43 \end{bmatrix}.$$

Applying the shift row operation yields the matrix

$$\begin{bmatrix} 59 & C5 & 9D & C5 \\ A8 & E5 & 01 & AD \\ 47 & AF & 59 & 63 \\ 43 & 50 & C9 & 63 \end{bmatrix}.$$

So, unsurprisingly our matrix still has only one byte different from the original example; we have only applied confusion. The next step, mix column, applies the necessary diffusion.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \otimes \begin{bmatrix} 59 & C5 & 9D & C5 \\ A8 & E5 & 01 & AD \\ 47 & AF & 59 & 63 \\ 43 & 50 & C9 & 63 \end{bmatrix} = \begin{bmatrix} 55 & 5A & B2 & 7D \\ 98 & AE & BD & 42 \\ BA & 95 & 6E & 0B \\ 82 & BE & 6D & 5C \end{bmatrix}.$$

Finally, our current matrix is XOR'd with K_1, which yields

$$M_1^* = \begin{bmatrix} 8C & C6 & 7B & FB \\ F3 & E5 & B5 & 06 \\ 17 & 6B & D5 & F9 \\ 4A & 37 & C4 & D4 \end{bmatrix}.$$

Thus, after one round of AES with a single character changed, we have M_1 differing from M_1^* by an entire column. Moreover, in the second round, the shift row step will shift one of these four differing elements into each column. After the mix column operation, the one differing element in each column will affect the entire matrix. Hence, a single byte change has diffused into the entire message after the second round. The matrix after Round 2 becomes

$$M_2^* = \begin{bmatrix} 33 & 04 & 0C & 0A \\ 62 & CA & CF & 6D \\ 0A & BE & 2C & 06 \\ 6A & 39 & 7E & 7F \end{bmatrix},$$

which is a completely different matrix than our original example. This helps illustrate how effectively AES diffuses a single bit flip throughout the entire message.

3.7 Decryption

Decryption is required to retrieve the original message. The steps must be reversed and operations inverted. The XOR operation is its own inverse, $X \oplus X = 0$ for all $X \in \mathbb{R}$, so the first operation for decryption would be to XOR the last round key. The first round of decryption would skip inverting the mix column operation as it was not performed in the last round of encryption. For all other rounds, we multiply by the inverse of the constant matrix

$$C^{-1} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}.$$

To invert the shift row operation, one must keep the first row the same, shift the second row one spot to the right, the third row to the right twice, and the fourth row three times

to the right. Finally, to undo the S-box, one could find their entry inside the table and use the corresponding row and column to find the entry's pre-image. For example, if we wished to know what "$6D$" was before the S-box, we would locate "$6D$" in Table 3.1 and notice that it is in row "$B0$" and column "03". Hence, $S^{-1}(6D) = B3$.

Chapter 4

RSA

4.1 Motivation

The advanced encryption standard is a secure and efficient symmetric cipher. The main disadvantage to AES is the need to establish a key that all parties wishing to communicate must know. If the parties cannot physically meet, they must send the key over a channel, which runs the risk of their key being intercepted and, in turn, all their encrypted messages may be easily broken. A solution to this problem is to have one key that encrypts a message, a public key, and one key that decrypts the message, a private key.

4.2 Security

The simplicity of the RSA cryptosystem's security adds to its beauty. It places its bet on the simple fact that it is considered a hard problem to factor numbers. More specifically, numbers composed of two primes. Factoring numbers will always involve some trial and error. A problem like prime factoring 91 can easily be done by hand. Factoring 4168723211, which is the product of two 5-digit primes, is much more difficult to do without the assistance of a computer, even knowing the length of the two primes. However, when the prime factors multiplied together are larger than 2^{2048}, even modern day super computers take countless years of computing time to factor the composite number. On the other hand, multiplication is considered an easy problem because there is a direct way to compute products. It is not too difficult to compute the product

59359×70229 without the help of a computer. This is the core principal of an asymmetric cipher. It is easy to compute one way, multiplication, but very difficult to take the inverse, factoring.

4.3 Method

To set up an RSA system, there are six variables that a user, Alice, would first have to determine. Since RSA is an asymmetric algorithm, a sender, Bob, does not have to compute a key; most of the work is done by Alice. First, Alice would select two primes, normally denoted as p and q. Then, Alice computes $n = p * q$. Next, Alice computes $\Phi(n)$, which normally would not be easy. However, since Euler's Phi Function Φ is multiplicative, along with the fact that $\Phi(u) = u - 1$ if u is prime we have,

$$\begin{aligned} \Phi(n) &= \Phi(p * q) \\ &= \Phi(p) * \Phi(q) \\ &= (p-1) * (q-1). \end{aligned}$$

So, $\Phi(n) = (p-1) * (q-1)$, easy enough for Alice to compute. Alice would then select a value e that is relatively prime to both $p-1$ and $q-1$, making it relatively prime to $\Phi(n)$. Finally, Alice calculates d such that $e * d \equiv 1 \mod \Phi(n)$. Proof of d's existence is a consequence of Bezout's Identity.

Theorem 1 (Bezout's Identity)**.** *Let a and b be integers with greatest common divisor d. Then, there exist integers x and y such that $ax + by = d$. More generally, the integers of the form $ax + by$ are exactly the multiples of d.*

For our application, take $a = e$, $b = \Phi(n)$, and $d = \text{GCD}(e, \Phi(n)) = 1$. Note that the d from the theorem is not the same d that we are trying to show exists. Now, from the theorem we have

$$de + k\Phi(n) = 1.$$

Thus

$$de + k\Phi(n) \mod \Phi(n) = 1 \mod \Phi(n),$$

from which it follows that

$$de \mod \Phi(n) = 1 \mod \Phi(n).$$

In practice, Alice would calculate d via the Extended Euclidean Algorithm. Alice publishes (e, n) for anyone to encrypt with and keeps (d, n) as a personal decryption key. The method, in general, works as follows. Bob. who wants to send plain-text message m, would first compute $c \equiv m^e \mod n$. Bob sends c to Alice. Alice computes $m \equiv c^d \mod n$, thus getting the plain text back. An eavesdropper, Eve, knows e and n, as they are publicly known, as well as c, which is the message Bob sent. Eve is missing the decryption key d, as it is known only by Alice. Why does this work? After all, e and d were inverses with respect to $\mathbb{Z}_{\Phi(n)}$ and the actual cipher text is computed in $\mathbb{Z}_n$. The answer is a consequence of Euler's Theorem.

Theorem 2 (Euler's Theorem)**.** *Let a and n be two coprime positive integers, then* $a^{\Phi(n)} \equiv 1 \mod n$.

Manipulation of Euler's Theorem gives us the following result:

$$\begin{aligned}
& a^{\Phi(n)} \equiv 1 \mod n \\
\Rightarrow \quad & a^{k*\Phi(n)} \equiv 1 \mod n \\
\Rightarrow \quad & a * a^{k*\Phi(n)} \equiv a * 1 \mod n \\
\Rightarrow \quad & a^{k*\Phi(n)+1} \equiv a \mod n.
\end{aligned}$$

Now, since $ed \equiv 1 \mod \Phi(n)$, there is some $k \in \mathbb{Z}$ such that $ed = k * \Phi(n) + 1$. So Alice's decryption is computed as follows:

$$\begin{aligned}
c^d \mod n &\equiv (m^e)^d \mod n \\
&\equiv m^{ed} \mod n \\
&\equiv m^{k\Phi(n)+1} \mod n \\
&\equiv m \mod n.
\end{aligned}$$

Ergo, the plain text message is retrieved.

4.4 A Worked Example

Let Alice be the user, Bob the sender, and Eve an eavesdropper that intercepts everything sent between Alice and Bob. Alice randomly selects $p = 5$ and $q = 11$ and

Table 4.1: A brute-force attack

m	$\frac{m^3-2}{55}$
1	$-\frac{1}{55}$
2	$\frac{6}{55}$
3	$\frac{7}{55}$
⋮	⋮
18	$\frac{5830}{55} = 106$
⋮	⋮
73	7073
⋮	⋮
128	38130
⋮	⋮

computes $n = 55$. Alice also computes $\Phi(55) = 4 * 10 = 40$. She has free choice for e and selects 3 since it is relatively prime to both 4 and 10. Using the Extended Euclidean Algorithm, Alice computes $d = 27$. Alice then publishes $(3, 55)$ for anyone to use. Bob wants to send Alice the secret message $m = 18$. Bob computes

$$\begin{aligned} 18^3 \mod 55 &\equiv 5832 \mod 55 \\ &\equiv 2 \mod 55, \end{aligned}$$

and sends Alice $c = 2$. Now Alice, having the private key $d = 27$, computes

$$\begin{aligned} 2^{27} \mod 55 &\equiv 134217728 \mod 55 \\ &\equiv 18 \mod 55, \end{aligned}$$

retrieving the plain-text message that Bob sent. Eve has ciphertext 2, public modulus 55, and public key 3. Eve knows Bob's encryption scheme: $m^3 \mod 55 \equiv 2$. Therefore, $m^3 = 55k + 2$ for some $k \in \mathbb{N}$. Thus, $\frac{m^3-2}{55} = k \in \mathbb{N}$. In this case the code may be somewhat easily broken via trial and error and using technology. Since $\frac{18^3-2}{55} = 106 \in \mathbb{N}$, we see that $m = 18$ is a possible message. However, there are more possible messages as shown in Table 4.1, so Eve would need to see if any make sense. For the purposes of illustration we had Alice choose very low primes for p and q and a very simple public

key. In practice, the primes would be much larger and the private key less trivial, thereby rendering an encryption that would, with todays technology, be nearly impossible to crack via brute force.

One may immediately think, $\Phi(n)$ is very close to n since $n = p * q$ and $\Phi(n) = (p-1)(q-1)$, and that a descending approach from n to find $\Phi(n)$ could be a possible attack. Of course, if $\Phi(n)$ was discovered by an attacker they need not worry about factoring n, as they can just compute d via the Extended Euclidean Algorithm. The problem with this strategy is that in practice, $p \geqslant 2^{1024}$ and $q \geqslant 2^{1024}$, making $n \geqslant 2^{2048}$, and $\Phi(n) \geqslant 2^{2048}$ which, unfortunately for any attacker, yields 2^{2047} integers with bit length 2048. Let us put this in perspective by converting 2^{2047} bits to something more tangible, decimal digits:

$$\begin{aligned} & 2^{2047} = 10^u \\ \Rightarrow \quad & \log 2^{2047} = \log 10^u \\ \Rightarrow \quad & 616 \approx u. \end{aligned}$$

Thus, 2047 bits is approximately 616 decimal digits. The most powerful super computers in the world (as of this writing) can compute 20 decimal digits per second. Thus, we find it would take approximately 10^{596} seconds or greater than 3×10^{589} years to find $\Phi(n)$ with a backward approach starting at n. Needless to say, users following the RSA protocol are safe from the naïve attacker.

4.5 Application

In practice, RSA encryption involves extremely large numbers and it is not always practical to encrypt a whole message using a RSA protocol. A potential application of RSA would be the establishment of a session key, a key used for a particular session of sending encrypted information. This is called a key exchange protocol. It could work many ways, one way would be to encrypt a message using AES. We know AES is an extremely efficient encryption algorithm; the main downside is that the two parties communicating need to know the key. One could encrypt the AES key with RSA and send the encrypted key and the encrypted message over an insecure network. The receiver would be able to calculate the AES key using their RSA private key. Then using the decrypted AES key, decrypt the message.

Chapter 5

Discrete Logarithm Problem

5.1 Motivation

The demand for an asymmetric cipher was still high even after RSA. There were already very secure and computationally efficient symmetric algorithms in DES and later AES. Their main weakness was that the establishment of keys could be difficult if the two parties could not physically meet before communication began. The Diffie-Hellman key exchange was invented to satisfy the demand.

5.2 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange uses cyclic groups, a class of groups with generators. Let us say that $\mathbb{G}$ is a cyclic group and that $a \in \mathbb{G}$ is a generator of $\mathbb{G}$. So, for all $b \in \mathbb{G}$ there exists some $\alpha \in \mathbb{N}$ such that $a^{\alpha} = b \mod \mathbb{G}$. Now, say Alice and Bob would like to exchange a key but do not have a secure chain of communication. They could publicly select a cyclic group, $\mathbb{G}$, and generator, a. They then each select a group element and keep that element secret. Suppose Alice picks r and Bob selects s. Alice would compute $A = a^r \mod \mathbb{G}$ and send the result to Bob. Bob computes $A^s \mod \mathbb{G}$. Bob now has the private key he and Alice will use for encryption. The process is repeated for Alice to have the private key: Bob computes $B = a^s \mod \mathbb{G}$ and sends the result to Alice. Alice then computes $B^r \mod \mathbb{G}$ and they now have the same key. An eavesdropper, Eve, knows the group $\mathbb{G}$, the generator a, and the public messages between Alice and Bob, A and B. Eve can formulate the equation $a^r = A \mod \mathbb{G}$. The

only missing variable for Eve is r. If Eve could solve for r, then the system has been broken as $B^r \mod \mathbb{G}$ is easy to compute. The way to solve for an exponent is by using logarithms; unfortunately for Eve, the typical logarithm approach learned in algebra class is useless here. The approach to solve for the missing exponential in the Diffie-Hellman key exchange leads to the discrete logarithm problem; the security of this key exchange relies on the difficulty of the problem.

5.3 Discrete Logarithm Problem

The discrete logarithm problem is formulated as follows: Given prime number p, an element $\beta \in \mathbb{Z}_p^*$, and primitive element α, find x such that $\alpha^x \equiv \beta \mod p$. If a group has integer elements, as does $\mathbb{Z}_p^*$, there are powerful algorithms that can solve this problem if p is not too large. To have the same security as AES with a 128-bit key, p must be 3072 bits long. Furthermore, to have the same level of security as AES with a 256-bit key, p must be 15360 bits long. With numbers this large, trivial operations run slow. What we will see is a cyclic group whose elements are points on an elliptic curve. Using this type of group, a level of security comparative to AES with a 256-bit key requires p to be 512 bits. This gives a huge boost in efficiency when compared to the former discrete logarithm problem.

5.4 Small-Step Giant-Step Algorithm

A very powerful algorithm to solve the discrete logarithm problem, and in turn a powerful attack against the Diffie-Hellman key exchange, is the small-step giant-step algorithm. Consider the discrete logarithm problem. The small step is to select some integer k, compute $a^1, a^2, a^3, ..., a^{k-1}$, and also compute $ba^{-k}, ba^{-2k}, ba^{-3k}, ..., ba^{-rk}$ where $rk > N$; all computations are done in $\mathbb{Z}_p^*$. Now, if we have a collision, i.e., if $a^n \equiv ba^{-mk} \mod \mathbb{Z}_p^*$ for any n and m, we have essentially solved the problem. All that is required is to multiply both sides of the congruence by a^{mk} and we will have $a^{n+mk} \equiv b \mod \mathbb{Z}_p^*$. The solution to the discrete logarithm problem is $n + mk$.

5.5 Elliptic Curves

The elliptic curve E over $\mathbb{Z}_p$ is defined as follows.

Definition. *Suppose $p > 3$ and $a,b \in \mathbb{Z}_p$. Further, suppose $4a^3 + 27b^2 \neq 0 \mod p$. Then the elliptic curve E over $\mathbb{Z}_p$ is defined to be the set of all pairs $(x, y) \in \mathbb{Z}_p$ such that $y^2 \equiv x^3 + ax + b \mod p$, together with an imaginary point at infinity $\mathcal{O}$.*

For example, consider the elliptic curve $y^2 = x^3 - 4x + 10$. There is a very nice geometric approach to point addition and point doubling inside of the elliptic curve. To add two points P and Q where $P \neq Q$ in the elliptic curve, we construct the line connecting the two points, which leads to a third point of intersection. That point is reflected across the x-axis. The resulting point is defined to be $P + Q$. If $P = Q$, we construct the tangent to the curve at P which leads to a second point of intersection with the elliptic curve. That point is reflected across the x-axis, and the resultant point is defined to be $2P$.

There is, of course, an analytic method for determining the points of intersection. There are two cases. The first case is when $P \neq Q$, so there is a unique line connecting the two points. To find $P + Q$, one solves the system of equations defined by the linear equation and the elliptic curve equation to find the third solution. If $P = Q$ then one must find the equation of the line tangent to the elliptic curve at point P, then solve the corresponding system of equations. We demonstrate the general solution.
Given $P(x_1, y_1)$ and $Q(x_2, y_2)$ we calculate $[P + Q](x_3, y_3)$ by

$$x_3 = s^2 - x_1 - x_2 \mod p,$$
$$y_3 = s(x_1 - x_3) - y_1 \mod p,$$

where

$$s = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \mod p & \text{if } P \neq Q, \\ \dfrac{3x_1^2 + a}{2y_1} \mod p & \text{if } P = Q. \end{cases}$$

Using these equations it can be shown that some elliptic curves contain cyclic subgroups. In practice, standardized elliptic curve equations with calculated group orders readily available. The identity element of the group is the point at infinity, $\mathcal{O}$.

We will illustrate how the points of an elliptic curve form a cyclic subgroup using the elliptic curve

$$y^2 \equiv x^3 + 2x + 2 \mod 17.$$

Using the point $P = (3, 1)$ we generate the elements of the subgroup:

$P = (3, 1)$	$8P = (16, 4)$	$15P = (10, 6)$
$2P = (13, 7)$	$9P = (6, 14)$	$16P = (0, 6)$
$3P = (0, 11)$	$10P = (6, 3)$	$17P = (13, 10)$
$4P = (10, 11)$	$11P = (16, 13)$	$18P = (3, 16)$
$5P = (5, 1)$	$12P = (7, 11)$	$19P = \mathcal{O}$
$6P = (9, 16)$	$13P = (9, 1)$	
$7P = (7, 6)$	$14P = (5, 16)$	

Since $19P = \mathcal{O}$, we know $20P = (3, 1)$ because $\mathcal{O}$ is the identity of the group. This also implies that $21P = (13, 7)$, and so on. Thus the point P generates a cyclic subgroup of the elliptic curve.

Now that we have a cyclic group, we can create a discrete log problem as follows. Say Alice and Bob want to set up a key for AES and all of their messages are intercepted by Eve. Let the elliptic curve equation be $E : y^2 \equiv x^3 + 2x + 2 \mod 17$. The equation E is publicly known. Next, select a generator of a cyclic subgroup of E, in this case we will use the point $(3, 1)$. The generator is also publicly known. Now, Alice and Bob each privately select an integer modulo 17. Say Alice selects 11 and Bob selects 15. Alice would then calculate $11P$ and send the resulting point $(16, 13)$ to Bob. Likewise Bob would calculate $15P$ and send the resulting point $(10, 6)$ to Alice. Alice would take $15P$, which she received from Bob, and use her private key 11 to calculate $11 * 15P$. Bob would take $11P$, which he received from Alice, and use his private key 15 to calculate $15 * 11P$. They now have the same point and can use either the x or y value of the generated point as a key for AES. Eve knows only the point $(16, 13)$, which was Alice's message to Bob, and the point $(10, 6)$, which was Bob's message to Alice. Assuming Alice and Bob each selected their private integer randomly, the quickest way of solving for their private integers will take at least $\sqrt{p}$ steps. The lower bound $\sqrt{p}$ is computed in [1].

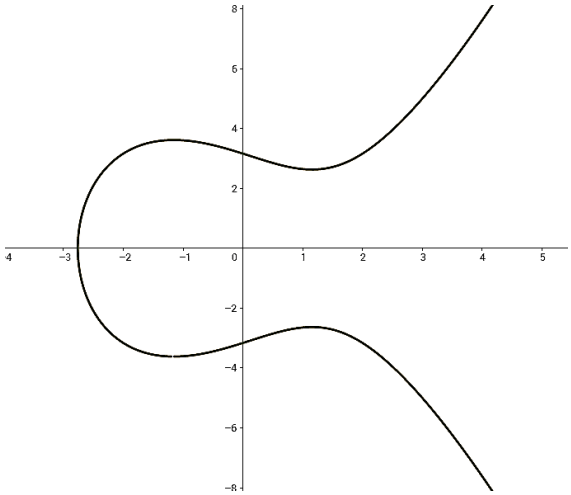

Figure 5.1: The elliptic curve defined by $y^2 = x^3 - 4x + 10$.

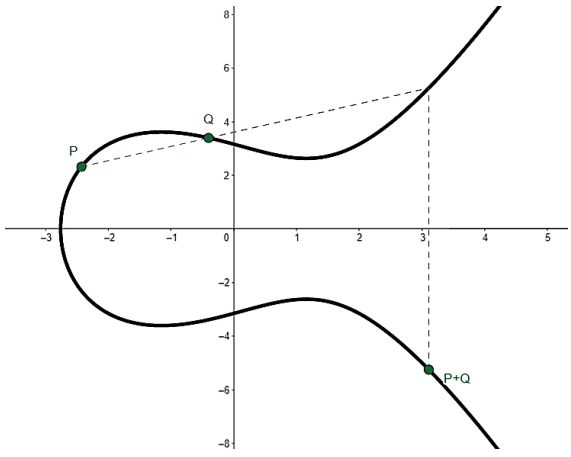

Figure 5.2: Geometric derivation of $P + Q$.

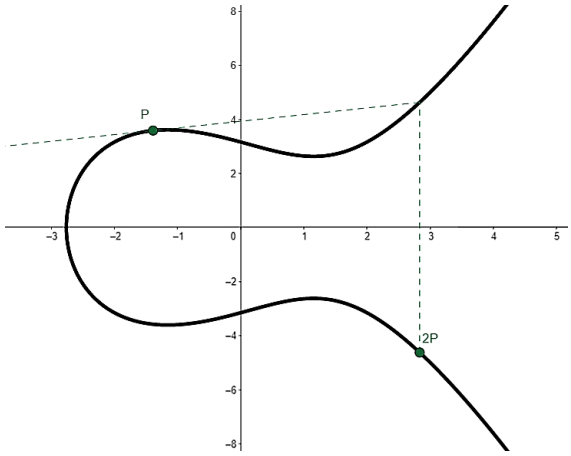

Figure 5.3: Geometric derivation of $2P$.

Chapter 6

Hash Functions

6.1 Motivation

Hash functions are a different type of cryptographic tool. The purpose of a hash function is not to encode information, rather it serves as a one-way function to verify the integrity of a message. An example is Alice and Bob playing a game of rock-paper-scissors over the Internet. How might we prevent one person from changing their answer in order to win? This is where a hash function comes in. Like any function, the same input yields the same output. For example, suppose Alice selects scissors and Bob selects rock. They each will hash their choice and send the result to one another. Bob reveals to Alice that he selected rock, if Alice were to respond with a lie and claim she selected paper, Bob can hash "paper" and compare the result with the hash originally sent by Alice. Bob will realize that the hash does not match and know Alice was lying. Another typical use of a hash function is storing pin numbers in a database. When you first create a pin number it is hashed and the result is stored in a database. The purpose of this is to ensure that even if an attacker were to access the database, all they would be able to retrieve are the hash results, not the actual pin. When you use your debit card and enter your pin, the pin is hashed and compared to the hash recorded on the database; if the two hashes match, it is assumed that the entered pin was correct and the ATM will give you the requested money.

A hash function h is considered secure if it has the following attributes. First, a hash function must be able to take an arbitrary number of characters and output a

fixed number of characters. If "Hello" is hashed through Sha-256, a very popular hashing algorithm, the result is the 64 character output

$$66a045b452102c59d840ec097d59d9467e13a3f34f6494e539ffd32c1bb35f18.$$

If "Hello my name is Samuel, I am a math major at California State University San Bernardino" is hashed through Sha-256, the result is

$$b813f6fe19598833d21e21b14fff4eda378f90ace49dfe84da3301a9b7ea7e073$$

which has 64 characters, the same number of characters as the previous hash.

The second property of a secure hash function is that it must be second pre-image resistant. A *collision* occurs when two different messages have the same hash. Collision is unavoidable as a consequence of the first property since there are fewer possible outputs than inputs. Second pre-image resistance is described as follows. Given a message x_1 and hash $y_1 = h(x_1)$, it should be infeasible to come up with an input $x_2 \neq x_1$ such that $y_1 = h(x_2)$.

A third property is collision resistance. It should be infeasible to find two distinct messages x_1 and x_2 such that $h(x_1) = h(x_2)$.

A fourth property ties in with the idea of collision resistance. Given a hash y_1, it should be impossible to create a plain-text message x_1 where $h(x_1) = y_1$.

A fifth property is similar to what we have seen in previous chapters, diffusion. A small change in plain-text results in a completely different hash. Compare the hash of "Hello" given earlier as

$$66a045b452102c59d840ec097d59d9467e13a3f34f6494e539ffd32c1bb35f18.$$

to the hash of "hello" which is

$$5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f6be03.$$

Notice the apparent complete lack of similarity between the two hashes.

The final property is that hash functions should be one-way functions. Given a hash, there should not be a way to invert the hash to retrieve the plain text. A hash is secure only if it has all the attributes described above.

6.2 Understanding Properties of Secure Hash Functions

We will now look at the purpose of the various properties for secure hash algorithms by considering potential attacks. The first property is primarily for efficiency. In practice it would be extremely inefficient to manually hash blocks of a message and then for the receiver to again hash blocks of the message and check every single block to verify that the message was not changed. By allowing any length messages, a hash must only be computed and checked once.

The rest of the properties are for security purposes. Second pre-image resistance aims to thwart the following attack. Imagine Alice sending $10 to Eve, an active attacker who wants to change the dollar amount Alice is sending. If Eve were to change the $10 to $20, the hash would not match and the bank would not release the funds. It is extremely unlikely that the hashes would match for any random amount that Eve might choose. Eve must find a dollar amount x_1 such that $h(x_1) = h(10)$. If the hash function is not secure, Eve will be able to calculate such a value for x_1 and replace $10 with the new amount. Since the hash results would match, the bank would release the funds to Eve. This attack becomes easier to defend against by having a sufficiently large pool of hash results. For example, SHA-256 has a 256-bit hash output resulting in 2^{256} possible results. Along with the other properties, the only way to find a particular hash is by brute force. Every dollar amount hashed has a probability of $\frac{1}{2^{256}}$ to have the same hash as $10.

Collision attacks are statistically harder to defend against. Imagine Eve finds two dollar amounts that have the same hash, $h(x_1) = h(x_2)$. Eve could list an item for sale at dollar amount x_1, then when someone is buying the item, switch x_1 with x_2, which would execute because the hashes match. This attack is very powerful because of the cascading effect that occurs while checking hash results. It goes as follows. Assume Eve is trying to find a collision in SHA-256 and can choose any dollar amount x_1 to start with. Then Eve can choose any $x_2 \neq x_1$ and check if $h(x_2) = h(x_1)$. The probability of a match is $\frac{1}{2^{256}}$, which implies the probability of the results not colliding is $1 - \frac{1}{2^{256}}$. If x_1 and x_2 do not collide, Eve chooses x_3 and computes $h(x_3)$. The major danger is that now Eve can check to see if $h(x_3) = h(x_1)$ or $h(x_3) = h(x_2)$. The probability of a collision is $\frac{2}{2^{256}}$ which implies the probability of not having a collision is $1 - \frac{2}{2^{256}}$. The probability of there not being a collision after 3 dollar amounts are hashed and checked is $(1-\frac{1}{2^{256}})(1-\frac{2}{2^{256}})$. If a collision still has not occurred, Eve can then select x_4 and compute

$h(x_4)$ to check if $h(x_4) = h(x_1)$ or $h(x_4) = h(x_2)$ or $h(x_4) = h(x_3)$. The probability of a collision on this step is $\frac{3}{2^{256}}$, thus the probability of no collisions after four dollar amounts are hashed and checked is $(1 - \frac{1}{2^{256}})(1 - \frac{2}{2^{256}})(1 - \frac{3}{2^{256}})$. The probability that at least t checks are required before achieving a collision is $\Pi_{i=1}^{t-1}(1 - \frac{i}{2^{256}})$. The formula may be used to estimate t, hence $t \approx 2^{\frac{n+1}{2}}\sqrt{\ln \frac{1}{1-\lambda}}$ where n is the number of bits and λ is the probability for at least one collision. As mentioned before SHA-256 has 2^{256} different possible hash results or 256 bits. How many checks would be required to provide a 50% chance of collision? Substituting $n = 256$ and $\lambda = .5$ into the formula we would have

$$\begin{aligned} t &\approx 2^{\frac{256+1}{2}}\sqrt{\ln\left(\frac{1}{1-.5}\right)} \\ &= \sqrt{2^{257} \cdot \ln(2)} \\ &< \sqrt{2^{257} \cdot 2} \\ &= 2^{128} \end{aligned}$$

We can see that the number of steps required for achieving a collision attack with 50% probability is about the square root of n. This indicates that SHA-256 is still secure from this type of attack because of its colossal pool of possible hash results. It is still infeasible to execute 2^{128} steps with today's technology, which is why SHA-256 is still used.

The remaining properties for hash functions are to ensure that the only viable attacks are brute force attacks.

Chapter 7

Blockchain Technology

7.1 "Satoshi Nakamoto's" White Paper

In 2008, a new type of cryptosystem protocol was published in the form of a white paper under the pseudonym "Satoshi Nakamoto." It is unknown who "Satoshi" is, but since the publication of his white paper, thousands of crypto assets have been developed. The technology went unseen by the majority of the population for many years. More recently, the boom in value towards various crypto assets caught the attention of mainstream media and investors hoping to ride Bitcoin or other altcoins to a Lamborghini. The term *altcoin* is used to refer to all crypto assets that are not Bitcoin.

7.2 Security of Blockchain

The core cryptographic property used by blockchains are hash functions; the Bitcoin blockchain uses SHA-256. The white paper starts by claiming there is a problem with double spending money. The current way to avoid the double-spending problem is by including a mutually trusted third party, the bank for example. Satoshi proposed an alternate solution:"We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work" (Nakamoto 1). There are two main principles being introduced here. The first is that all transactions to be timestamped enter a block. That block is hashed along with the hash of the previous block. By chaining the blocks,

using the hash of the previous block in the hash of the current block, an ongoing chain of timestamps is created in which each validates the timestamps of the previous block. Proof-of-work is the key security feature that makes the blockchain secure. The "work" for Bitcoin is to find a solution to a computationally hard mathematical problem. The problem is to find a value x_1 such that $h(x_1)$ starts with a given number of zero bits. As we know from hash functions, the best way to find a particular hash is by brute force. The idea is that all transactions and items enter a block; if there is no current block then a block is created. For the block to be completed a solution to the problem described above must be found. All CPUs on the network work together to find a solution for the block. Once the solution is found, it is published for other CPUs to verify. It is much easier to verify a solution than to find a solution as verification requires only hashing the proposed solution and checking to see if it is indeed a true solution. If the solution is correct, the block is verified and the CPUs work on creating a the next block. The hash of the previous block is included in the current block, forming a chain of blocks. Visit [4] to see the Bitcoin blockchain in action.

The second principle introduced in the quote is that the record of transactions cannot be changed without redoing the proof-of-work. A major part of the security is that for a typical attacker, the amount of CPU power and money required to attack the network is not profitable. If someone wants to attack the network, for example reverse a previous transaction, the attacker must resolve the proof-of-work problem for the block where the transaction is stored, as well as all blocks that were chained after. In order for the attack to be successful, the attacker must force the network to accept the chain that contains the attacked block. The network accepts the longest chain as the true chain, so the attacker must chain more blocks on the attacked block than the original chain has. This type of attack is called the 51% attack because if an attacker possessed at least 51% of all computing power on the network, they can solve blocks faster than all of the rest of the network combined. Thus, if the attackers change a transaction they can chain blocks faster than the rest of the network and force the whole network to accept the chain with manipulated blocks.

7.3 Bitcoin

The first recorded use of Bitcoin as a currency was its use in an order of two pizzas from Papa John's. Reportedly the two parties agreed to trade 10,000 Bitcoin for the pizzas. On January 9, 2018, the value of a single Bitcoin was listed at $14,770. This puts the price of the two pizzas at $147,700,00 USD.

Chapter 8

Conclusion

Mathematics has played a crucial role in cryptography throughout the centuries, particularly in the last century. As computers continue to get faster and, perhaps more troublesome, as algorithms become more efficient, encryption based on computationally hard mathematical problems may become less useful. Problems like factoring and the discrete logarithm problem, where security relies in having a pool too large for computers to attack encryption via brute force, will be easily broken with quantum computers and quantum computing algorithms. When quantum computing becomes a reality, the cryptography world will have to move to post quantum cryptography. But for now, computationally hard mathematical problems are the key to effective encryption methods.

Bibliography

[Ire18] David Ireland. RSA Algorithm, 2002-18. DI Management Services. `https://www.di-mgt.com.au/rsa_alg.html`.

[Kak18] Avi Kak. *Lecture Notes on "Computer and Network Security",* Lecture 8: AES: The Advanced Encryption Standard. Purdue University, 2018.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. March 2009.

[PP14] Christof Paar and Jan Pelzl. *Understanding Cryptography a Textbook for Students and Practitioners.* Springer, 2014.