A COMPARISON BETWEEN DIFFIE-HELLMAN AND RSA

RUIZE XU

Abstract. We give a general introduction to the theory of cryptography and introduce some of the key concepts underlying cryptosystems such as Diffie-Hellman and RSA. Then we give an upper bound on the difficulty of third-party decryption.

Contents

1. Cryptography before the computer age and Public Key Cryptography

The history of cryptography dates back to ancient Rome when Julius Caesar commands his troops by his *Caesar cipher*, or *shift cipher*. Another example is *simple substitution cipher*, in which each letter is replaced by another letter. The following example, the cipher wheel, demonstrates the concept. For each letter, we map it to the letter with five offsets, which is a cipher wheel with an offset of five letters. We call these systems *cryptosystems*.

Date: August 27, 2021.

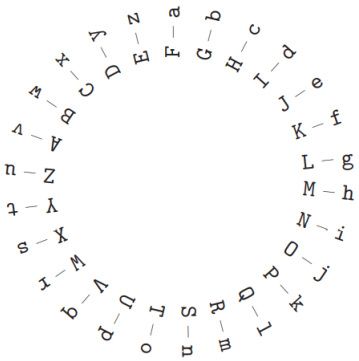

FIGURE 1. Cipher Wheel

Suppose we were to decrypt the message given the *ciphertext*

xmtkojbmvkct dn api

We look up the outer cycle of the cipher wheel. Notice each letter offsets by five letters. Thus, we have decrypted the message, which is

Cryptography is fun

which gives us the original message, or the *plaintext*. This process is called *decryption*.

With the modern development of mathematics and computer science, we rely on cryptography for secure communication. With *encoding schemes* like American Standard Code for Information Interchange (ASCII), we are capable of converting plaintext into integers, and the study of number theory contributes greatly to the study of cryptography.

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

FIGURE 2. ASCII Table

In this paper, we will give two famous cryptosystem examples: the Diffie-Hellman Key exchange cryptosystem and the RSA public-key cryptosystem. Then we introduce the runtime complexity for breaking the two cryptosystems. In this paper, we primarily follow the ideas of the proofs from Hoffstein chapter 1-3, and credits are given at appropriate places.

2. General Definition

In this section, we provide all the relevant mathematical tools needed for cryptography. We assume that readers are most familiar with the basic definition of number theory.

Theorem 2.1. *(Extended Euclidean Algorithm) Let a and b be positive integers. Then the equation*

$$au + bv = gcd(a, b)$$

always has a solution in integers u and v. If (u_0, v_0) is any one solution, then every solution has the form

$$u = u_0 + \frac{b \cdot k}{gcd(a, b)}$$

and

$$v = v_0 - \frac{a \cdot k}{gcd(a, b)}$$

for some $k \in \mathbb{Z}$.

Definition 2.2. We write

$$\mathbb{Z}/m\mathbb{Z} = \{0, 1, 2, \ldots, m-1\}$$

and call $\mathbb{Z}/m\mathbb{Z}$ the ring of integers modulo m. Note that whenever we perform an addition or multiplication in $\mathbb{Z}/m\mathbb{Z}$, we always divide the result by m and take the remainder in order to obtain an element in $\mathbb{Z}/m\mathbb{Z}$. We also write

$$(\mathbb{Z}/m\mathbb{Z})^* = \{a \in \mathbb{Z}/m\mathbb{Z} : \gcd(a, m) = 1\} \quad = \{a \in \mathbb{Z}/m\mathbb{Z} : a \text{ has an inverse modulo } m\}$$

Remark 2.3. For simplicity of notation, we refer to $\mathbb{Z}/p\mathbb{Z}$ as $\mathbb{F}_p$ and $(\mathbb{Z}/p\mathbb{Z})^*$ as $\mathbb{F}_p^*$ for a prime p. For $a, b \in \mathbb{F}_p$, the equality is denoted by $a = b$, while for $a, b \in \mathbb{F}_p^*$, the equality of a, b is $a \equiv b \pmod{p}$.

Proposition 2.1. Let $m \geq 1$ be an integer. Then every nonzero element $a \in \mathbb{Z}/m\mathbb{Z}$ such that $\gcd(m, a) = 1$ has a multiplicative inverse, that is, there is a number b satisfying

$$ab \equiv 1 \pmod{m}$$

We denote this value of b by $a^{-1} \pmod{m}$, or if m has already been specified, then simply by a^{-1}.

Proof. By Extended Euclidean Algorithm, since $\gcd(a, m) = 1$, there exists integers u, v such that

$$au + mv = \gcd(a, m) = 1$$

so we have

$$au - 1 = -mv$$

so $au - 1$ is divisible by mv, hence divisible by m and $au \equiv 1 \pmod{m}$. We have shown there indeed exists the multiplicative inverse $b = u$. □

Remark 2.4. If p is prime, then Proposition 2.1 can be interpreted as

$$\mathbb{F}_p^* = \{1, 2, 3, 4, \ldots, p-1\}\}.$$

In other words, if we remove 0 from $\mathbb{Z}/p\mathbb{Z}$, then the remaining set is closed under multiplication.

Theorem 2.5. *(Fermat's Little Theorem) Let p be a prime number and let a be any integer. Then*

$$a^{p-1} = \begin{cases} 1 \pmod{p} & \text{if } p \nmid a, \\ 0 \pmod{p} & \text{if } p \mid a. \end{cases}$$

Proof. See pg 30 of [2]. □

Definition 2.6. We define the *order of* a modulo p to be the smallest exponent $k \geq 1$ such that

$$a^k \equiv 1 \pmod{p}.$$

We now introduce the Fast Powering Algorithm, which is faster than the usual powering if we are dealing with large numbers.

Input: N, g, and A
Output: g^A modulo N
1 initialization;
2 Compute the binary expansion of A as $A = A_0 + A_1 \cdot 2 + A_2 \cdot 2^2 + A_3 \cdot 2^3 + \ldots + A_r \cdot 2^r$ with $A_i \in \{0, 1\}$ for all i and $A_r = 1$.;
3 Set $a_0 \equiv g \pmod{N}$;
4 **for** $i = 1 : r$ **do**
5 $\quad a_i \equiv a_{i-1}^2 \equiv g^{2^i} \pmod{N}$; `*This in total requires r multiplications*`
6 **end**
7 $g^A \equiv \Pi_{j=0}^{r} a_j^{A_j} \pmod{N}$.

Algorithm 1: Fast Powering Algorithm

Theorem 2.7. *(The Fast Powering Algorithm) The Fast Powering algorithm takes at most $2r$ multiplications modulo N to compute g^A. Given $A \geq 2^r$, the computation takes at most $2\log_2(A)$ multiplications modulo N to compute g^A.*

Proof. To compute g^A, we need to first iterate through the for loop from line 4 to line 6, which takes r multiplications in total. Then $\Pi_{j=0}^{r} a_j^{A_j}$ takes r multiplications. Thus, in total g^A takes $2r$ multiplication. Since $A \geq 2^r$, we have

$$\log_2(A) \geq r$$

so

$$2\log_2(A) \geq 2r$$

which means that the total computation takes at most $2\log_2(A)$. □

Remark 2.8. Storing A_i might be undesired had the algorithm been badly implemented. However, this is easily solvable by giving a low-storage variant of the Fast Powering Algorithm.

Input: N,g,A
1 Let $a = g$ and $b = 1$;
2 **while** $A > 0$ **do**
3 **if** $A \equiv 1 \pmod 2$ **then**
4 Let $b = b \cdot a \pmod N$;
5 **end**
6 Let $a = a^2 \pmod N$ and $A = \lfloor A/2 \rfloor$;
7 **end**

Algorithm 2: Low-storage variant of the Fast Powering Algorithm

Essentially, this algorithm differs from the Fast Powering Algorithm in that it combines the for loop in line $4 - 6$ of the Fast Powering Algorithm and the multiplication in line 7 of the Fast Powering Algorithm.

Remark 2.9. With the Fermat's little theorem and the fast powering algorithm, we have an efficient method of computing inverses modulo p, i.e.,

$$a^{-1} \equiv a^{p-2} \pmod p,$$

so if we multiply a^{p-2} by a, by Fermat's Little Theorem $a^{p-2} \cdot a \equiv 1 \pmod p$.

3. Diffie-Hellman Cryptosystem

Having introduced all relevant definitions and concepts, we will now discuss one of the first public-key cryptosystems with examples of how the system will operate. Later in the last section, we provide a way to break the cryptosystem.

Suppose that Alice and Bob want to share a secret message, while Eve has control over the communication and can intercept the messages. Now, Alice and Bob want a secure way to share their key without making it available to Eve. The public key system suffices and we will illustrate the process by examples.

For every cryptosystem, we want to achieve the following:

(1) The encryption is efficient, and the encryption function easy to compute,
(2) The decryption is secure, and the decryption is hard to compute without the key,
(3) With a certain key or secret information, the decryption becomes very easy to compute.

Intuitively, the efficiency here means that any algorithm that runs in less than the age of the universe and seeks to break the encryption would certainly fail.

3.1. Public Key Cryptography.

The public key cryptosystem introduces one-way functions that have a *trapdoor*, a piece of auxiliary information that allows the inverse of the encryption, i.e., the function f easy to compute. Essentially, we are looking for the following function $f : X \to Y$ such that it is is easy to compute $f(X)$ but hard to compute $f^{-1}(Y)$, and with additional information $f^{-1}(Y)$ becomes easy to compute. This gives us the desired functionality as listed above, as we have efficient encryption (the function f), secure decryption, and a key that helps the decryption.

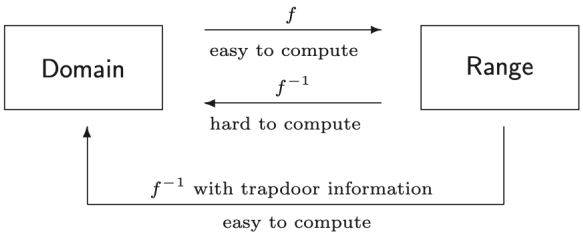

FIGURE 3. Illustration of one-way trapdoor function

3.2. **Discrete Logarithm Problem.** The creation of our cryptosystem is directly related to a hard mathematical problem. We will introduce a computationally difficult number theory problem that serves as the basis for the Diffie-Hellman key exchange cryptosystem. Then we will discuss how the Diffie-Hellman key exchange cryptosystem gets implemented.

The Discrete Logarithm Problem is the first published key construction that fulfills the requirements of the public-key cryptosystem.

Definition 3.1. Let g be a primitive root for $\mathbb{F}_p$ and let h be a nonzero element of $\mathbb{F}_p$. The *Discrete Logarithm Problem* (DLP) is the problem of finding an exponent x such that

$$g^x \equiv h \pmod{p},$$

where x is called the *discrete logarithm of h to the base g* and is denoted by $\log_g(h)$.

One can check that the usual properties of the logarithm holds true in this definition as well.

Remark 3.2. Note that if there is one solution x to the DLP, then there is infinitely many such x because of the Fermat's little theorem. Observe that since

$$g^{p-1} \equiv 1 \pmod{p},$$

if we find $x = x_0$ such that $g^{x_0} \equiv h \pmod{p}$, $x_0 + k(p-1)$ is also a solution for all k, as

$$g^{x_0+k(p-1)} = g^{x_0} \cdot g^{k(p-1)} = g^{x_0} \cdot (g^{p-1})k \equiv h \cdot 1^k \equiv h \pmod{p}.$$

So we say that the solution x is unique up to adding or subtracting multiples of $p-1$.

3.3. **Diffie-Hellman key exchange.** To solve the dilemma we listed at the beginning of the section, the Diffie-Hellman key exchange cryptosystem was born.

First, Alice and Bob need to publicly pick a large prime p (in the section on primality testing, we will discuss how to test if our picked p is indeed a large prime) and a nonzero integer g modulo p. Note that this information is shared by Eve as well. It is recommended g's order in $\mathbb{F}_p^*$ is a large prime, as having a small order can be brute-forced fairly easily (we will also discuss later on how to find such g).

Alice and Bob then pick a secret integer a and b respectively and keep them as secrets to themselves. Alice computes

$$A \equiv g^a \pmod{p},$$

and Bob computes

$$B \equiv g^b \pmod{p}.$$

Then they exchange A, B, i.e., Alice sends A to Bob, and Bob sends B to Alice. Note that this communication channel is insecure and Eve gets to know A, B. Lastly, Alice computes

$$A' \equiv B^a \pmod{p}$$

and

$$B' \equiv A^b \pmod{p}.$$

It is then obvious that $A' \equiv B'$ because

$$A' \equiv B^a \equiv (g^b)^a \equiv (g^a)^b \equiv A^b \equiv B' \pmod{p}.$$

The encryption process can be summarized in the image below:

Public Parameter Creation	
A trusted party chooses and publishes a (large) prime p and an integer g having large prime order in $\mathbb{F}_p^*$.	
Private Computations	
Alice	**Bob**
Choose a secret integer a.	Choose a secret integer b.
Compute $A \equiv g^a \pmod{p}$.	Compute $B \equiv g^b \pmod{p}$.
Public Exchange of Values	
Alice sends A to Bob $\longrightarrow$ A	
B $\longleftarrow$ Bob sends B to Alice	
Further Private Computations	
Alice	**Bob**
Compute the number $B^a \pmod{p}$.	Compute the number $A^b \pmod{p}$.
The shared secret value is $B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \pmod{p}$.	

FIGURE 4. A table summary of Diffie-Hellman key exchange cryptosystem

Remark 3.3. For Eve to intercept the encrypted message, she needs to find a way to efficiently find out a, b. She knows $A \equiv g^a \pmod{p}$ and $B \equiv g^b \pmod{p}$, but it gives her a hard time to find g^{ab}. We can now define the Diffie-Hellman Problem.

Definition 3.4. Let p be a prime and g be an integer. Then the *Diffie-Hellman Problem* (DHP) is the problem of computing the value of $g^{ab} \pmod{p}$ from $g^a \pmod{p}$ and $g^b \pmod{p}$.

It is clear that if Eve can solve DLP, then she can easily solve DHP. But the converse may not be true.

3.4. **Encoding Scheme.**

Definition 3.5. An *encoding scheme* is a method of converting one sort of data into another sort of data while keeping the data secure.

For example, converting text into binary digits(bits) is a typical practice nowadays. This is public to everyone and distinct from encryption. An encryption scheme is designed to hide information from anyone that does not possess the secret key or the trapdoor. Hence, the encoding scheme consists of an invertible function whose computation is fast and efficient, while the inverse function is hard to compute.

3.5. **ElGamal Public key cryptosystem.**

Diffie-Hellman does not achieve the full goal of a public key cryptosystem, because it does not permit an exchange of specific information and only provides a way to exchange a random string of bits. ElGamal public-key encryption algorithm is a public-key cryptosystem based on Diffie-Hellman that achieves the desired function.

Similar to Diffie-Hellman, Alice picks a large prime p of which the DLP is hard to solve and an element g modulo p of large (prime) order. Alice then picks a secret number a as her private key and computes

$$A \equiv g^a \pmod{p}.$$

Then Alice publishes A to the public and keeps a secret.

Suppose now Bob wants to encrypt a message m in the form of integers. Since an encoding scheme can be expressed as an integer by using the ASCII table, the message can be expressed as a string of digits. Then Bob picks a *ephemeral key* k for the sake of encryption and computes

$$c_1 \equiv g^k \pmod{p},$$

and

$$c_2 \equiv mA^k \pmod{p},$$

where A, g, p are all public. Then Bob sends his ciphertext (c_1, c_2) to Alice. To decrypt (c_1, c_2), Alice simply computes

$$x \equiv c_1^a \pmod{p},$$

and computes $x^{-1} \pmod{p}$ by Fast Powering Algorithm and Fermat's Little Theorem as listed in Remark 2.9 . Then we have

$$\begin{aligned} x^{-1} \cdot c_2 &\equiv (c_1^a)^{-1} \cdot (mA^k) \pmod{p} \\ &\equiv (g^{k^a})^{-1} \cdot (mg^a) \pmod{p} \\ &\equiv m \cdot (g^{-a})^k \cdot g^a \pmod{p} \\ &\equiv m \cdot (g^{-a+a})^k \pmod{p} \\ &\equiv m \cdot 1^k \pmod{p} \\ &\equiv m \pmod{p} \end{aligned}$$

Eve, knowing $p, g, A \equiv g^a \pmod{p}$, can find a and decrypt the message if she solves the discrete logarithm problem (we will discuss why this is hard later). Otherwise, it is hard for Eve to find the plaintext.

The image below summarizes the cryptosystem.

Public Parameter Creation	
A trusted party chooses and publishes a large prime p and an element g modulo p of large (prime) order.	
Alice	**Bob**
Key Creation	
Chooses private key $1 \leq a \leq p-1$. Computes $A = g^a \pmod{p}$. Publishes the public key A.	
Encryption	
	Chooses plaintext m. Chooses random ephemeral key k. Uses Alice's public key A to compute $c_1 = g^k \pmod{p}$ and $c_2 = mA^k \pmod{p}$. Sends ciphertext (c_1, c_2) to Alice.
Decryption	
Compute $(c_1^a)^{-1} \cdot c_2 \pmod{p}$. This quantity is equal to m.	

FIGURE 5. A table summary of ElGamal

Proposition 3.1. Fix a prime p and base g for the ElGamal encryption. Suppose that Eve has access to an oracle that decrypts arbitrary ElGamal ciphertexts encrypted using arbitrary ElGamal public keys. Then she can use the oracle to solve the Diffie-Hellman problem.

Proof. Recall that Eve is given

$$A \equiv g^a \pmod{p}$$

and

$$B \equiv g^b \pmod{p}$$

in the Diffie-Hellman problem where p, g, A, B are given. Her goal is to compute $g^{ab} \pmod{p}$. Given that Eve has access to an oracle, if Eve sends $(c_1, c_2), p, g, A$, then the Oracle will return the quantity

$$(c_1^a)^{-1} \cdot c_2 \pmod{p}.$$

Now suppose the oracle is careless. Then Eve can send the ciphertext $(c_1, c_2) = (B, 1)$, then the oracle gives

$$(c_1^a)^{-1} \cdot c_2 \equiv B^{-a} \cdot 1 \equiv g^{-ab} \pmod{p}$$

Then by Remark 2.9 , Eve can compute $g^{ab} \pmod{p}$.

Now suppose the oracle is cautious and never computes ciphertext with $c_2 = 1$. Then Eve simply picks a random c_2 and repeats what she will do if the oracle is careless and obtain

$$m \equiv (c_1^a)^{-1} \cdot c_2 \equiv B^{-a} \cdot c_2 \equiv g^{-ab} \cdot c_2 \pmod{p}.$$

Then again we can compute $m^{-1} \cdot c_2 \equiv g^{ab} \cdot c_2^{-1} \cdot c_2 \equiv g^{ab} \pmod{p}$ for the desired quantity g^{ab}. □

Remark 3.6. By proposition 4.1, we have shown that if one is capable of decrypting arbitrary ciphertexts generated by ElGamal encryption, then he or she is also able to solve the Diffie-Hellman problem. This shows that the ElGamal system is secure against chosen-ciphertext attacks if we assume that the Diffie-Hellman problem is actually hard.

4. RSA public key

In this section, we introduce another widely used cryptosystem: RSA cryptosystem, which is named after its first inventors, Ron Rivest, Adi Shamir. Then in the last section, we provide ways to break the cryptosystem had the cryptosystem been badly implemented.

4.1. Euler's Formula and roots modulo pq.

We begin with a few relevant results from number theory.

Theorem 4.1. *(Euler's Formula for pq) Let p and q be distinct primes and let*

$$g = \gcd(p-1, q-1)$$

Then

$$a^{(p-1)(q-1)/g} \equiv 1 \pmod{pq}$$

for all a such that $\gcd(a, pq) = 1$.

If p, q are odd primes, then

$$a^{(p-1)(q-1)/2} \equiv 1 \pmod{pq}$$

for all a such that $\gcd(a, pq) = 1$.

Proof. Since we assume a to be such that $\gcd(a, pq) = 1$, p and q does not divide a. Since $g = \gcd(p-1, q-1)$, we have that $g \mid q-1$ and $g \mid p-1$, which means that both $\frac{q-1}{g}$ and $\frac{p-1}{g}$ are integers. Hence,

$$a^{(p-1)(q-1)/g} \equiv (a^{p-1})^{(q-1)/g} \pmod{p} \tag{4.2}$$

$$\equiv 1^{(q-1)/g} \pmod{p} \tag{4.3}$$

$$\equiv 1 \pmod{p} \tag{4.4}$$

and

$$a^{(p-1)(q-1)/g} \equiv (a^{q-1})^{(p-1)/g} \pmod{q} \tag{4.5}$$

$$\equiv 1^{(p-1)/g} \pmod{q} \tag{4.6}$$

$$\equiv 1 \pmod{q}, \tag{4.7}$$

where we know (4.3) and (4.6) due to the fact that both $\frac{q-1}{g}$ and $\frac{p-1}{g}$ are integers, (4.4),(4.8) are application of Fermat's Little Theorem. By Chinese remainder theorem, $a^{(p-1)(q-1)/g} \equiv 1 \pmod{pq}$. If p, q are odd prime, then it follows that $2 \mid g$ as both $p-1$ and $q-1$ are even and $a^{(p-1)(q-1)/2} \equiv 1 \pmod{pq}$. □

Diffie-Hellman relies on the fact that

$$a^x \equiv b \pmod{p}$$

is difficult to compute, where a, b, p are known and x is unknown. The RSA public-key cryptosystem will rely on the difficulty of solving

$$x^e \equiv c \pmod{N}$$

where e, c, N are known and x is unknown. We will show that it is easy to compute the eth root modulo N if N is prime.

Proposition 4.1. Let p be a prime and let $e \geq 1$ be an integer satisfying $\gcd(e, p-1) = 1$. Proposition 2.1 tells us that e has an inverse modulo $p-1$, say

$$de \equiv 1 \pmod{p-1}.$$

Then the congruence

$$x^e \equiv c \pmod{p}$$

has the unique solution $x \equiv c^d \pmod{p}$.

Proof. Consider $c \equiv 0 \pmod{p}$. Then simply take $x \equiv 0 \pmod{p}$ and we are done with a unique solution. Suppose $c \not\equiv 0 \pmod{p}$. Since $de \equiv 1 \pmod{p-1}$, we have

$$de = 1 + n(p-1)$$

for some integer n. Then

$$\begin{aligned} x^e &\equiv c^{de} \pmod{p} \\ &\equiv c^{1+n(p-1)} \pmod{p} \\ &\equiv c \cdot c^{n(p-1)} \pmod{p} \\ &\equiv c \cdot (c^{(p-1)})^n \pmod{p} \\ &\equiv c \pmod{p} \end{aligned}$$

by Fermat's Little Theorem. Hence, we see that $x^e \equiv c \pmod{p}$ has a solution $x \equiv c^d \pmod{p}$.

To show uniqueness, let x_1 and x_2 be two solutions to the system. Since $c^{de} \equiv c \pmod{p}$,

$$x_1 \equiv x_1^{de} \equiv (x_1^e)^d \equiv c^d \equiv (x_2^e)^d \equiv x_2 \pmod{p}$$

because $x_1^e \equiv x_2^e \equiv c \pmod{p}$ by assumption. Hence, we have shown that $x_1 \equiv x_2 \pmod{p}$, which shows that the solution is unique. □

Proposition 4.2. Let p and q be distinct primes and let $e \geq 1$ satisfy

$$\gcd(e, (p-1)(q-1)) = 1.$$

Proposition 2.1 tells us that e has an inverse modulo $(p-1)(q-1)$,

$$de \equiv 1 \pmod{(p-1)(q-1)}.$$

Then the congruence

$$x^e \equiv c \pmod{pq}$$

has the solution

$$x \equiv c^d \pmod{pq}.$$

Proof. Suppose that c is an integer with $\gcd(c, pq) > 1$. If $pq \mid c$, then $x \equiv 0 \pmod{pq}$ is a solution. If $pq \nmid c$, then either $p \mid c$ or $q \mid c$ by definition of prime. Without loss of generality, suppose $p \mid c$ but $q \nmid c$. Then we have

$$x^e \equiv c^{ed} \equiv 0 \pmod{p}$$

since $p \mid c$. It remains to show that this is also true modulo q. Let

$$de = 1 + n(p-1)(q-1)$$

for some integer n. Observe

$$x^e \equiv c^{ed} \equiv c^{1+n(p-1)(q-1)} \equiv c \cdot (c^{(q-1)})^{n(p-1)} \equiv c \cdot 1^{n(p-1)} \equiv c \pmod{q}$$

by Fermat's Little Theorem.

Suppose that c is an integer with $\gcd(c, pq) = 1$. Given

$$de \equiv 1 \pmod{(p-1)(q-1)},$$

we have

$$de = 1 + n(p-1)(q-1)$$

for some integer n. Then

$$\begin{aligned} x^e &\equiv c^{de} \pmod{pq} \\ &\equiv c^{1+n(p-1)(q-1)} \pmod{pq} \\ &\equiv c \cdot c^{n(p-1)(q-1)} \pmod{pq} \\ &\equiv c \cdot 1^n \pmod{pq} \\ &\equiv c \pmod{pq} \end{aligned}$$

by Euler's Formula, which implies that $c^{(p-1)(q-1)} \equiv c^{(p-1)(q-1)/g} \equiv 1 \pmod{pq}$ where $g = \gcd(p-1, q-1)$. □

Remark 4.8. Proposition 4.2 gives us an algorithm to solve

$$x^e \equiv c \pmod{pq}$$

by computing $de \equiv 1 \pmod{(p-1)(q-1)}$ and then computing $c^d \pmod{pq}$. Let $g = \gcd(p-1, q-1)$. To make things faster, we can instead compute

$$de \equiv 1 \pmod{\frac{(p-1)(q-1)}{g}}.$$

and the remaining follows from Euler's Formula as

$$\begin{aligned} x^e &\equiv c^{de} \pmod{pq} \\ &\equiv c^{1+n(p-1)(q-1)/g} \pmod{pq} \\ &\equiv c \cdot c^{n(p-1)(q-1)/g} \pmod{pq} \\ &\equiv c \cdot 1^n \pmod{pq} \\ &\equiv c \pmod{pq} \end{aligned}$$

where by Euler's Formula we have that $c^{(p-1)(q-1)/g} \equiv c^{(p-1)(q-1)/g} \equiv 1 \pmod{pq}$ and now we find a smaller de.

4.2. **RSA public key crypotsystem.** With all the relevant math details, we now introduce the RSA public-key cryptosystem, which is the first known public-key system.

RSA Key Creation

Bob picks p, q, two large primes. Then Bob computes $N = pq$. Then Bob chooses e to be an integer such that $\gcd(e, (p-1)(q-1)) = 1$. Finally, Bob publishes his public key (N, e).

RSA Encryption

Alice converts her message into an integer m. Using Bob's public key (N, e), Alice computes

$$c \equiv m^e \pmod{N}$$

and sends c to Bob.

RSA Decryption

Bob knows $(p-1)(q-1)$ for he picks p, q. Since Bob knows e, he can solve

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$

for d. Finally, Bob computes

$$c^d \equiv m^{ed} \equiv m \pmod{N}$$

by Proposition 4.2. The image below summarizes the RSA.

Bob	**Alice**
Key Creation	
Choose secret primes p and q. Choose encryption exponent e with $\gcd(e,(p-1)(q-1))=1$. Publish $N=pq$ and e.	
Encryption	
	Choose plaintext m. Use Bob's public key (N,e) to compute $c \equiv m^e \pmod{N}$. Send ciphertext c to Bob.
Decryption	
Compute d satisfying $ed \equiv 1 \pmod{(p-1)(q-1)}$. Compute $m' \equiv c^d \pmod{N}$. Then m' equals the plaintext m.	

4.3. **Primality Testing.** An obvious pitfall is that if the prime that we choose is small, then the hacker could simply brute force by trying every element in the finite field corresponding to that prime. Hence, in this chapter we address the untouched problem: how do we pick a large prime?

Theorem 4.9. *(Fermat's Little Theorem, Version 2) Let p be a prime number. Then*

$$a^p \equiv a \pmod{p}$$

for all integer a.

Proof. It's not hard to observe that if $p \nmid a$, then this follows from Fermat's little theorem. If $p \mid a$, then

$$a^p \equiv a \equiv 0 \pmod{p}.$$

□

This gives us one direction: if p is a prime, then $a^p \equiv a \pmod{p}$. However, we are interested in the other direction, which is to determine if p is a prime given $a^p \equiv a \pmod{p}$.

Definition 4.10. Fix an integer n. Then an integer a is a *witness* for the compositeness of n if

$$a^n \not\equiv a \pmod{n}$$

.

So, we can combine the Fermat's Little Theorem version 2 with many witness to test if a number is composite. But this is still inefficient because there are composite numbers having no witnesses (they are called *Carmichael numbers*). For example, the number $561 = 3 \cdot 11 \cdot 17$ yet $a^{561} \equiv a \pmod{561}$ by simply verifying all $a = 0, 1, 2, \ldots, 560$.

Proposition 4.3. Let p be an odd prime and write

$$p - 1 = 2^k q$$

with q being odd. Let a be such that $p \not| a$. Then one of the following is true:

(1) $a^q \equiv 1 \pmod{p}$
(2) $a^{2^i q} \equiv -1 \pmod{p}$ for some $0 \leq i \leq k - 1$.

Proof. By Fermat's little theorem,

$$a^{p-1} \equiv a^{2^k q} \equiv 1 \pmod{p}.$$

And observe that for all i, $a^{2^i q}$ is the square of $a^{2^{i-1} q}$. Hence, one of the following must be true:

(1) The base case, or the first of the list, $a^q \equiv 1 \pmod{p}$
(2) One of the $a^{2^i q} \not\equiv 1 \pmod{p}$ but $a^{2^{(i+1)} q} \equiv 1 \pmod{p}$.

But for the second case to be true, the only number b satisfying

$$b \not\equiv 1 \pmod{p} \text{ and } b^2 \equiv 1 \pmod{p}$$

is clearly -1, which shows that one of the list of the number is congruent to -1 modulo p. □

So we introduce the following definition.

Definition 4.11. Let n be odd and write $n-1 = 2^k q$ with q being odd. An integer a satisfying $\gcd(a, n) = 1$ is called a *Miller-Rabin* witness for (the compositeness of) n if both of the following conditions are met:

(1) $a^q \not\equiv 1 \pmod{n}$.
(2) $a^{2^i q} \not\equiv -1 \pmod{n}$ for all $i = 0, 1, 2, \ldots, k - 1$.

It then follows that for an odd number n, if there does exist a *Miller-Rabin* witness for n, then n is composite. Thus, we have the algorithm listed in next page. If we try every potential witness of n and still gets Test Fail, then n is prime.

Input: An integer n to be tested. An integer a as the potential witness
initialization;
if n *is even* **then**
  return Composite;
end
Let $n-1=2^k q$ with q odd;
Let $a \equiv a^q \pmod{n}$;
if $a \equiv 1 \pmod{n}$ **then**
  return Test Fail;
end
for $i = 0 : k-1$ **do**
  if $a \equiv -1 \pmod{n}$ **then**
    return Test Fail;
  end
  $a \equiv a^2 \pmod{n}$;
end
Return Composite;

Algorithm 3: Miller-Rabin Test

Proposition 4.4. Let n be an odd composite number. Then at least 75% of the numbers between 1 and $n-1$ are *Miller-Rabin* witness for n.

Proof. See pg 121, theorem 10.6 [1] □

Now it becomes a lot easier for us to verify for compositeness: we simply take a few hundred numbers between 1 and $n-1$, and the probability of none of them being a Miller-Rabin witness while n is not a prime is $(25\%)^{100} \approx 10^{-6}$.

4.4. **Distribution of Prime numbers.** If one wants to be absolutely sure that the integer chosen is prime, we also have ways to show it.

Definition 4.12. For any number X, let

$$\pi(X) = \{\#\text{ of primes p satisfying } 2 \le p \le X\}$$

Theorem 4.13. *The Prime Number Theorem*

$$\lim_{x\to\infty} \frac{\pi(X)}{X/\ln(X)} = 1.$$

So we make the following observation:

Remark 4.14. A randomly chosen number N has a probability of $1/ln(N)$ of being prime, since

$$\lim_{x\to\infty} \frac{\pi(X)}{X/\ln(X)} = 1$$

gives us that

$$\lim_{x\to\infty} \pi(X) = X/\ln(X)$$

which implies that for every positive integer smaller than X, it has the probability $P = \frac{X/\ln(X)}{X} = \frac{1}{X}$ of being prime

Now if Bob is satisfied with the probability, then he can simply test a few 100 numbers between 1 and $n-1$ to test if n is prime or composite. However, if Bob wants a solid answer rather than relying on probability, then he needs better answers. The Riemann Hypothesis, which gives a closed-form expression for $\pi(X)$, is the following:

$$\pi(X) = \int_2^X \frac{dt}{\ln t} + \mathcal{O}(\sqrt{X} \cdot \ln(X))$$

And if Riemann Hypothesis were proven to be true, then we would have

Proposition 4.5. If a generalized version of the Riemann Hypothesis is true, then every composite number n has a Miller-Rabin witness a for its compositeness satisfying

$$a \leq 2(\ln n)^2.$$

Proof. [4] shows that every composite number n has a witness satisfying $a = O((\ln n)^2)$, and [5] and [6] shows that $a \leq 2(\ln n)^2$. □

And finally,

Theorem 4.15. *(AKS Primality Test) For every $\epsilon > 0$, there is an algorithm that conclusively determines whether a given number n is prime in no more than $\mathcal{O}((\ln)^{6+\epsilon})$.*

Proof. See pg 132, theorem 3.25 [1] □

5. Upper bounds for third party decryption

In this section, we will compare the runtime complexity of the two public-key cryptosystems and analyze the key factors that make them computationally hard to solve.

5.1. Computational Complexity for breaking DLP.

In this section, we will give an upper bound for breaking the DLP. Before delving into the chapter we give a few definitions that measure how fast or slow a function grows. We use the Big-Oh and little-oh notation:

Definition 5.1. Let $f(x)$ and $g(x)$ be functions of x whose values are positive. We say that a function $f(x) = o(g(x))$ if

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$$

Definition 5.2. Let $f(x)$ and $g(x)$ be functions of x whose values are positive. We say that a function $f(x) = \mathcal{O}(g(x))$ if there are constants c and C such that

$$f(x) \leq cg(x)$$

for all $x \geq C$. Similarly, we say a function $f(x) = \Omega(g(x))$ if there are constants c and C such that

$$f(x) \geq cg(x)$$

for all $x \geq C$. If $f(x) = \mathcal{O}(g(x)) = \Omega(g(x))$, then $f(x) = \Theta(g(x))$.

Definition 5.3. Suppose now we are solving a mathematical problem. Suppose that there is a constant $A \geq 0$, independent of the size of the input, such that if the input is $\mathcal{O}(k)$ bits long, then it takes $\mathcal{O}(k^A)$ steps to solve the problem. This problem is said to be solvable in *polynomial time*, i.e., there exists an algorithm whose runtime complexity is at least as good as $\mathcal{O}(k^A)$, which is considered to be fast.

Suppose there is a constant $c > 0$ such that for inputs of size $\mathcal{O}(k)$ bits, there is an algorithm to solve the problem in $\mathcal{O}(e^{ck})$ steps, then the problem is said to be solvable in *exponential time*, which is considered to be slow.

A *subexponential-time algorithm* has the property that for every $\epsilon > 0$, the algorithm can solve the problem in $\mathcal{O}_\epsilon(e^{\epsilon k})$, where the subscript $\mathcal{O}_\epsilon$ indicates that the constant c and C of the definition of the Big-Oh notation can depend on ϵ.

Proposition 5.1. (A Naive Upper Bounds for DLP) The Discrete Logarithm Problem in the field $\mathbb{F}_p^*$ can be solved in $\mathcal{O}(2^{\log_2 p})$ steps.

Proof. For the original discrete logarithm problem $g^x = h$ in the field $\mathbb{F}_p^*$ for some p, g, h, suppose p is chosen between 2^k and 2^{k+1} bits, then $g, h \in \mathbb{F}_p^*$ and p all can be expressed in $\mathcal{O}(k)$ bits. Observe that $\mathcal{O}(k)$ here is equivalent to $\mathcal{O}(\log_2 p)$ because of the binary expression. An obvious brute-force way to break DLP is by the trial-and-error method, which will take $\mathcal{O}(p)$. If $p \mid g$, then we are done. Otherwise, since the order of g is at most $p-1$ ($g^{p-1} \equiv 1 \pmod{p}$ by Fermat's little Theorem), we make a list of $x = 0, 1, \ldots, p-2$ for $g^x \equiv h \pmod{p}$ and one of the x will give us h. Hence, the problem is solvable in

$$\mathcal{O}(p) = \mathcal{O}(2^k) = \mathcal{O}(2^{\log_2 p})$$

which means that the problem is solvable in exponential time. □

Remark 5.4. Of course, this is the least desirable solution. We will give a few algorithms that drastically improves the runtime efficiency

Now we introduce a better upper bound, which solves the discrete logarithm problem in $\mathcal{O}(\sqrt{N} \cdot \log N)$ steps.

Proposition 5.2. (Shank's Babystep-Giantstep Algorithm) Let G be a group and let $g \in G$ be an element of order $N \geq 2$. The following algorithm solves the discrete logarithm problem $g^x = h$ in $\mathcal{O}(\sqrt{N} \cdot \log N)$ steps.

initialization;
Let $n = 1 + \lfloor \sqrt{N} \rfloor$, so $n > \sqrt{N}$;
Create two lists:
$(1) e, g, g^2, g^3, \ldots, g^n$;
$(2) h, h \cdot g^{-n}, h \cdot g^{-2n}, h \cdot g^{-3n}, \ldots, h \cdot g^{-n^2}$;
Find a match such that $g^i = hg^{-jn}$;
return $x = i + jn$;

Proof. To show that the match indeed exists, we need to show that there exists the pair i, j such that $x = i + jn$. Suppose

$$x = qn + r$$

for $0 \leq r < n$. To show j indeed exists, we need to show that we can find $h \cdot g^{-qn}$ in the list $h, h \cdot g^{-n}, h \cdot g^{-2n}, h \cdot g^{-3n}, \ldots, h \cdot g^{-n^2}$. By definition $n > \sqrt{N}$. Observe also that $N > x - r$ because $x < N$ by order of x (if $x = N$ then DLP is trivial). Therefore, we have

$$n > \frac{N}{n} > \frac{x - r}{n} = q$$

Hence, we can write the expression

$$g^x \equiv h \pmod{p}$$

into

$$\begin{aligned} g^{qn+r} &\equiv h \pmod{p} \\ g^r &\equiv h \cdot g^{-qn} \pmod{p}. \end{aligned}$$

Since $0 \leq r < n$ and $0 \geq q < n$, $g^r, h \cdot g^{-qn}$ must be in the two list by the construction of the list. Thus, the algorithm does return the desired solution.

The two lists each take n computation, so in total the creation of two lists takes $2n$. For finding the match, we have the upper bound (for simplicity we ignored the implementation details) of the searching algorithm, which takes $O(n \log n)$. Together, we have that the total running time of the algorithm is

$$\mathcal{O}(n \log n) = \mathcal{O}(\sqrt{N} \log \sqrt{N}) = \mathcal{O}(\sqrt{N} \log N)$$

□

5.2. **Computational Complexity for RSA.**

5.3. **Pollard's $p-1$ factorization algorithm.** We introduce a method to break the RSA if RSA is not securely implemented.
Given N, we need to determine its prime factors p and q. Consider now that by luck we discover an integer L such that $p - 1 \mid L$ and $q - 1 \nmid L$. Then there are integer i, j, k where $k \neq 0$ and

$$L = i(p - 1)$$

and

$$L = j(q - 1) + k$$

Then if we pick any integer a, by Fermat's Little Theorem we have

$$a^L = a^{i(p-1)} = (a^{p-1})^i \equiv 1^i \equiv 1 \pmod{p}$$

and

$$a^L = a^{j(q-1)+k} = a^{j(q-1)} \cdot a^k \equiv a^k \pmod{q}.$$

Given $k \neq 0$, it would be unlikely that $a^k \equiv 1 \pmod{q}$, so with high probability, we find that $p \mid a^L - 1$ and $q \nmid a^L - 1$. This gives us

$$p = \gcd(a^L - 1, N).$$

To find an exponent L that is divisible by $p-1$ but not by $q-1$, Pollard's observation is that if $p-1$ happens to be a product of many small primes, then it will divide $n!$ for some relatively small value of n. Therefore, for $n = 2, 3, 4, \ldots$, we choose value of a and compute

$$g = \gcd(a^{n!} - 1, N).$$

If $g = 1$, then we try next n. If $g = N$ for every n, then we have been unlucky with a and retry the argument with another a. If $1 < g < N$, then we have found the nontrivial factor of N.

Remark 5.5. One might ask that computing $a^{n!} - 1$ would not be feasible with large n. We are only interested in $\gcd(a^{n!} - 1, N)$, so it suffices to compute

$$a^{n!} - 1 \pmod{N}$$

And we don't even need to compute $n!$ as

$$a^{(n+1)!} \equiv (a^{n!})^{n+1} \pmod{N}$$

if $a^{n!}$ has been computed, which simplifies the computation.

The remark is true by the following proposition

Proposition 5.3. Let $g = \gcd(a, N)$ and $k = \gcd(a \mod N, N)$. We want to show that $g = k$.

Proof. If we take

$$x = a \mod N$$

then $a = x + qN$ for some integer q. Since $k = \gcd(x, N)$, $k \mid x$ and

$$x = kq_x$$

for some integer q_x, and

$$N = kq_N$$

for some integer q_N. Therefore, we have that

$$a = x + qN = k \cdot q_x + k \cdot q \cdot q_N = k(q_x + qq_N)$$

which shows that $k \mid a$.

Suppose $g < k$. Then $k \mid a$ and $k \mid N$, yet $g = \gcd(a, N)$ and $g < k$, so this is a contradiction. Hence, by contradiction, $g \geq k$.

Suppose $g > k$. Then $g \mid a$ and $g \mid N$. Then let us assume

$$N = pg$$

for some integer p. Now, since

$$a = x + qN = x + qpg$$

and $g \mid a$, it follows that $g \mid x$ as well. However, this means that $g \mid x$, $g \mid N$, and $g > k$, yet $k = \gcd(x, N)$, so this is a contradiction. Hence, by contradiction, $g \leq k$

Since $g \leq k$ and $g \geq k$, it follows then $g = k$. □

So finally we implement Pollard's $p - 1$:

Proposition 5.6. *(Upper Bound for Pollard's algorithm) Pollard's $p-1$ algorithm can be computed in $\mathcal{O}(n(\log n)^2)$ steps.*

Proof. Observe that the algorithm is essentially computing

$$a^{n!} \mod N$$

along with $\gcd(a^i - 1 \mod N, N)$ for each ith iteration . We can compute $a^{n!} \mod N$ in $\mathcal{O}(n \log n)$ steps, because by the Fast Powering Algorithm, we can compute $a^k \mod N$ in at most $2 \log_2 k$ steps. Let $a_n \sim b_n$ denotes that $\lim_{n\to\infty} \frac{a_n}{b_n} = 1$. By Stirling's Formula,

$$n! \sim \sqrt{2\pi n}(\frac{n}{e})^n.$$

Input: Integer N to be factored
initialization;
Set $a = 2$ (or other convenient value);
Let n denote a desired bound **for** $i = 2 : n$ **do**
    Let $a = a^i \pmod N$;
    $d = \gcd(a - 1, N)$;
    if $1 < d < N$ **then**
        return d;
    end
end

Algorithm 4: Pollard's $p - 1$ factorization algorithm

Therefore, computing $a^{n!} \mod N$ takes at most $2\log_2(\sqrt{2\pi n}(\frac{n}{e})^n)$, and

$$2\log_2(\sqrt{2\pi n}(\frac{n}{e})^n) = 2\log_2(\sqrt{2\pi n}) + 2n\log_2(n) - 2n\log_2(e) = \mathcal{O}(n\log n).$$

Now, at each iteration, we are computing $\gcd(a^i - 1, N)$, and by Euclidean Algorithm, we can compute it in $\mathcal{O}(\log n)$ steps. Hence, in total they would take $\mathcal{O}(n(\log n)^2)$ □

Lemma 1. *The Euclidean Algorithm can be computed in $\mathcal{O}(\log n)$ steps.*

Proof. See runtime section of [3] □

6. Acknowledgement

It is my pleasure to thank my mentor, Samanda Hu, who contributed greatly to my understanding of the proof and algorithm, found resources, and edited this paper. This paper would not have been possible without her guidance and support. I would also like to thank Professor Daniil Rudenko and Professor Peter May for organizing an instructive program.

References

[1] V. Shoup. A Computational Introduction to Number Theory and Algebra. Cambridge University Press, 2005. http://shoup.net/ntb/ntb-b5.pdf
[2] Hoffstein, Pipher, and Silverman, An Introduction to Mathematical Cryptography, Springer, 2008
[3] https://www.csuohio.edu/sites/default/files/85-%202015.pdf
[4] G. L. Miller. Riemann's hypothesis and tests for primality. J. Comput. System Sci., 13(3):300–317, 1976. Working papers presented at the ACM-SIGACT Symposium on the Theory of Computing (Albuquerque, N.M., 1975).
[5] E. Bach. Explicit bounds for primality testing and related problems. Math. Comp., 55(191):355–380, 1990.
[6] E. Bach and J. Shallit. Algorithmic Number Theory. Vol. 1. Foundations of Computing Series. MIT Press, Cambridge, MA, 1996. Efficient algorithms.