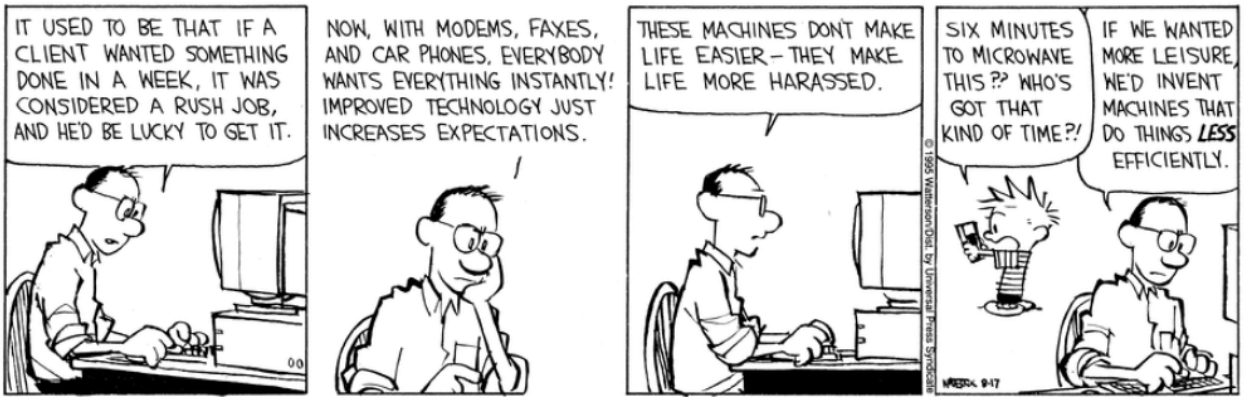
A Crash Course in Cryptology

Securing Electronic Communications

Instructor Information	
Name:	Zachary Porat
Email:	zporat@wesleyan.edu
Office:	ESC 604



Wesleyan University, Middletown, CT
Department of Mathematics and Computer Science

Preface

These lecture notes were originally developed for a mini-course that I gave as a part of SWiM@Wes 2025. One goal of SWiM is to provide mentorship to interested undergraduate students who are considering pursuing graduate studies in mathematics by introducing the workshop-style conference. In many ways, these notes serve as an homage to three of my own undergraduate mentors: Jeffrey Hatley, Kathryn Lesh, and George Todd.

Jeff encouraged me to attend my first math conference. The experience was incredible, giving me insight into what it is like to travel for academic conferences (a topic too often omitted from discussion about academic life). Jeff graciously invited me to tag along to dinner with several other attendees, so that I could experience what being a mathematician outside of the classroom was like. That conference really sold me on pursuing a PhD. I am forever grateful for his (ongoing) support.

Kathryn played a central role in forging my pedagogical approach. She saw promise in me after my first year of college–which was admittedly very rough–and asked if I wanted to be a course assistant in the math department. I agreed, and ended up working with Kathryn frequently. I grew so much as an instructor from seeing how she set up her courses, prepared lesson plans, and managed the classroom. Without her guidance, I would not have learned many of the intricacies required to be a successful educator.

George taught me the majority of the math that is contained in these notes. He introduced many of these concepts to me in a course on mathematical cryptology at Union College (a course that was pioneered by Kathryn). George's energy and enthusiasm for the subject helped inspire me to pursue number theory. He also introduced me to math research. His guidance allowed me to learn how research differed from coursework, while simultaneously fostering novel ideas, which eventually turned into my first ever math publication (in collaboration with George and a few others)!

I would also like to thank Lydia Ahlstrom and Arianna Zikos for organizing the conference with me, and Jacob Tolman and Stefan Hesseling for serving as my project assistants.

Wesleyan University
Middletown, CT

Zachary Porat
March 2025

Contents

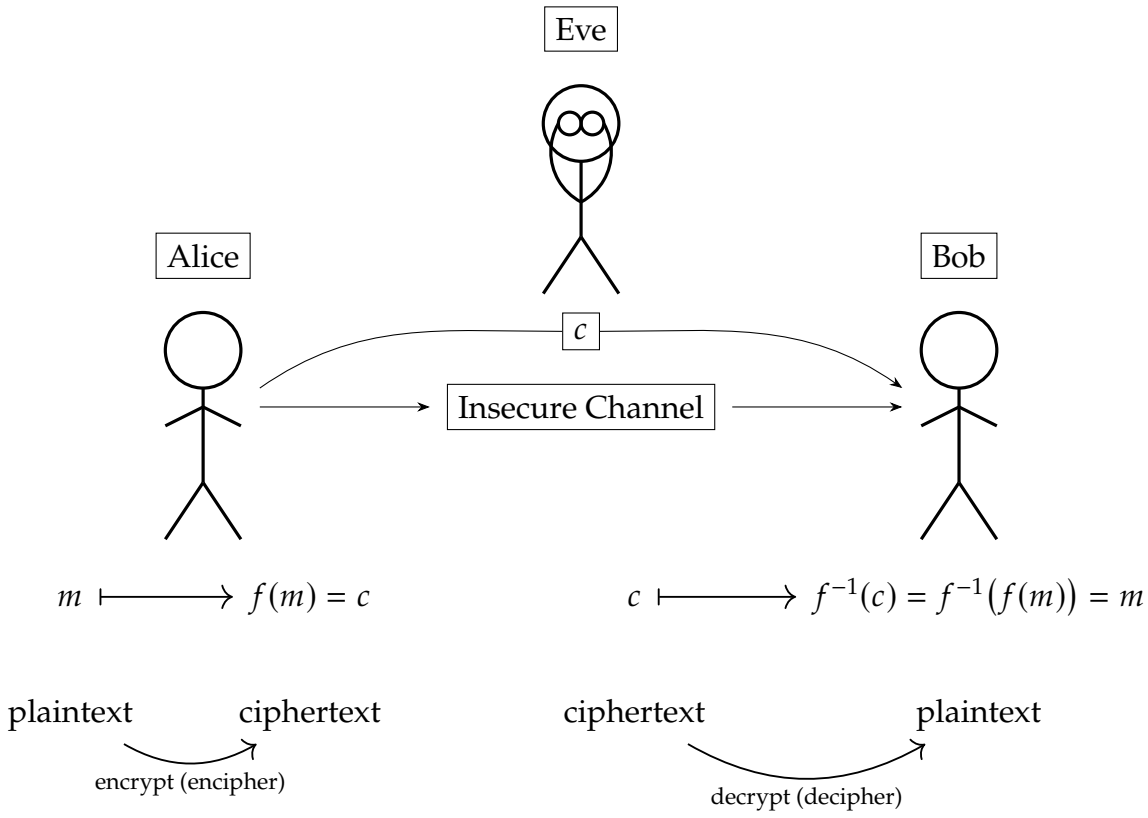
1 Big Picture of Cryptology

1.1 Introduction

Alice and Bob want to communicate on an insecure channel, but a nefarious actor named Eve is watching or listening. So, instead of sending **plaintext**, i.e. data that can be read and understood, Alice decides to **encrypt** (or **encipher**) her message. This process results in **ciphertext**, i.e. content that requires work to understand and is otherwise unintelligible, which she can then send to Bob. Once Bob receives the ciphertext, he goes through the process of **decrypting** (or **deciphering**) the message in order to understand the original plaintext.

Several terms in this subject area sound similar and are often used interchangeably, despite having different definitions. The overarching term for the study of confidential transmissions over insecure channels is **cryptology**. Then, within the topic of cryptology are cryptography and cryptoanalysis. **Cryptography** is the study of keeping messages secure and **cryptoanalysis** is the process of trying to recover plaintext from ciphertext.

Let m denote a plaintext message, f the function used to encipher the message, and $f(m) = c$ the resulting ciphertext. We can place the above definitions into a mathematical schematic as follows:



Instead of sending her plaintext message m over the insecure channel, Alice first encrypts her message by applying f to create ciphertext $f(m) = c$. Alice then sends c over the insecure channel. If Eve intercepts the message, she will have to perform cryptoanalysis on c to try to recover m. However, Eve has no knowledge of what the function f is, which makes analysis difficult. Bob, on the other hand, does know what f is. Therefore, he can simply apply f^{-1} to $c = f(m)$ to recover the original message from Alice. The pair of algorithms needed for encryption and decryption is called a **cipher**.

1.2 Historical Ciphers

Example 1.1. One type of function used to encipher plaintext is called a **shift (Caesar) cipher**. Let $m = \texttt{MATH}$. Further, convert the alphabet to numbers as follows:

$$\begin{aligned} \texttt{A} &\longmapsto 0 \\ \texttt{B} &\longmapsto 1 \\ &\vdots \\ \texttt{Z} &\longmapsto 25 \end{aligned}$$

Translate m into the ordered 4-tuple $(12, 0, 19, 7)$. Define f by taking an ordered n-tuple and adding 3 to each entry. So, Alice takes m and encodes it as

$$c = f(m) = (12 + 3, 0 + 3, 19 + 3, 7 + 3) = (15, 3, 22, 10) = \texttt{PDWK}.$$

Alice sends c to Bob. To decipher, Bob first converts c into a 4-tuple of numbers, then computes $f^{-1}(c)$ to recover m as follows:

$$m = f^{-1}(c) = (15 - 3, 3 - 3, 22 - 3, 10 - 3) = (12, 0, 19, 7).$$

Finally, Bob converts the 4-tuple of numbers he calculated back into letters to recover the intended message.[1]

Remark 1.2. While this cipher is straightforward to apply, it has a few drawbacks. First, f enciphers every letter using the same operation–addition by three–making it very easy to attack. Eve simply needs to check all 26 shifts, until one makes sense! Second,

[1] For those with a background in group theory, you will recognize that in order for this process to work with all letters, we actually must work in the group $\mathbf{Z}/26\mathbf{Z}$ under addition.

Alice and Bob must agree on f before communicating. Note that this was a problem even in our schematic!

One way to attempt to remedy the first problem with the shift cipher is to increase the complexity of our encryption algorithm. Consider the following algorithm based off the work of [Hil29].

Example 1.3. Let $m = (m_1, \ldots, m_n)$. Consider an invertible $n \times n$ matrix k. Encrypt by computing

$$c = f(m) = mk.$$

Then, to decrypt,

$$f^{-1}(c) = ck^{-1} = mkk^{-1} = mI_n = m,$$

where I_n denotes the identity $n \times n$ matrix.

Question 1.4. To illustrate that the algorithm described in Example 1.3 does actually increase security, consider the ciphertext $c = (5, 10, 1, 16)$. Assume that k is a 4×4 matrix with entries in the integers. We leave it to the reader to determine the original message m. (See Exercise [Q1.1] for the solution.)

The reason we can be so blasé about the ciphertext and the algorithm used to encrypt is because the security of the message should not (and does not) rely on either. We have to assume that Eve has access to both of these pieces of data. Therefore, the security of the message must rely exclusively on the strength of the **key** used for encryption and decryption. In Example 1.1, the key is the value by which the letters are shifted, and in Example 1.3, the key is the matrix k. This idea was first stated all the way back in 1883 by Dutch-born cryptographer Auguste Kerckhoffs.

Kerckhoffs's Principle ([Ker83])**.** The security of a cryptosystem must not depend on the secrecy of the algorithm, but should rest entirely on the strength of the keys.

For example, during the Second World War, Nazi Germany used the Enigma machine to encrypt communication. The machine used a predefined algorithm for encryption, but the keys were changed daily. As depicted in the movie *The Imitation Game*, the allies had their own Enigma machine, but they could not decode communications because the keys changed too frequently to be broken.

In the end, the allies did break Enigma by knowing a portion of the original message and the corresponding portion of ciphertext. Exercise [Q1.1] explores how knowing both

m and c in combination can lead to deciphering k, in the context of Example 1.3. We note that all three examples of cryptosystems thus far share the same problem: Alice and Bob (or the Nazi regime) must agree on a key before any ciphertext is sent. This poses a serious dilemma.

Definition 1.5. Alice and Bob must share a key to communicate securely. However, they cannot exchange a key without a secure channel. This is called the **key-sharing problem**.

1.3 Symmetric Key vs. Public Key Cryptosystems

In order to get around the key-sharing problem, the idea of **public key cryptography** was introduced to the public domain by Ralph Merkle in the mid-1970s. [Mer78] explains how a key can be agreed upon over insecure channels in such a way that the integrity of the messages is not compromised.

The main idea is as follows: Alice wants to send something to Bob securely. First, Bob buys a safe with a lock, to which only he has the key. Bob ships the safe, unlocked, to Alice. She puts the item in the safe, locks it, and sends it back to Bob. Bob unlocks it with his key. If Eve steals the safe in transit, she does not have the key to unlock it.

Building on the work of Merkle, Whitfield Diffie and Martin Hellman devised a scheme for how this idea could be used in practice. Their method, described in [DH76], would become known as the **Diffie-Hellman key exchange**. We will discuss the Diffie-Hellman key exchange in detail in Section 2.5. In 1997, declassified documents revealed that this method of key-sharing was actually devised in 1970 by GCHQ, a British intelligence agency. However, the idea was kept secret from the public.

The major advantage of public key cryptography is that the sender (Alice) only needs to know how to encrypt, not decrypt. Additionally, the encryption and decryption processes have been split, increasing the security of the system by preventing one process being derived from the other. In contrast, in the examples of cryptosystems we have seen thus far, the decryption key could be calculated by knowing the encryption key. For example, in Example 1.3, one can compute the decryption matrix k^{-1} simply by knowing the encryption matrix k.

However, public key cryptography also has downsides. The major drawback is that this system only works in one direction. That is, messages are only secured in one direction, which means Alice and Bob cannot communicate freely over the channel. In this way, public key cryptography is fundamentally different from our previous examples. To emphasize the differences, we make the following definitions:

Definition 1.6. A **symmetric key cryptosystem** is one where the decryption process can be computed from the encryption process in a reasonable amount of time, e.g. Examples 1.1 and 1.3.

Definition 1.7. A **public key cryptosystem** (PKC) is one where it is computationally infeasible to determine the decryption process from the encryption process. This is also called an assymmetric cryptosystem. Examples include the Diffie-Hellman key exchange, RSA, and El Gamal.

In practice, symmetric key cryptosystems are quite fast at passing data, while public key cryptosystems are slow by comparison. However, generating and passing a key for a symmetric key cryptosystem is complicated because of the key-sharing problem. Therefore, many protocols use public key cryptosystems to exchange a key for a symmetric cryptosystem, and then use that system to communicate freely. By doing this, we leverage the benefits of both systems. The following table illustrates the trade-offs between the two systems.

	Speed	**Key Management**
PKC	Slow	Easy
Symmetric	Fast	Hard

2 Public Key Cryptosystems

As discussed at the end of Chapter 1, a primary role of public key cryptosystems is to pass keys for symmetric cryptosystems. Perhaps the most famous of these cryptosystems is the Diffie-Hellman key exhange (DHKE). In an effort to keep within the constraints of the lecture series, these notes will focus on the DHKE and the number theory required to understand it. However, no collection of notes on cryptology would be complete without at least a mention of RSA, another widely-used public key cryptosystem.

In Section 2.4, we will learn that the security of the DHKE relies on the difficulty of the discrete logarithm problem. The security of RSA, on the other hand, is based on the difficulty of factoring the product of two large primes. The method is named after its creators: Ronald Rivest, Adi Shamir and Leonard Adleman. The algorithm was initially described in 1977 and was published in [RSA78]. Similar to Diffie-Hellman, declassified documents released in 1997 revealed that an equivalent cryptosystem was developed by

GCHQ in 1973. See Exercises [Q2.1] and [Q2.2] for additional details on RSA. Further reading can be found in [Was08, Section 6.8] or [HPS14, Chapter 3].

We now return to our main focus of this chapter, the Diffie-Hellman key exchange. We will begin by building the necessary number theoretic background, before describing the algorithm itself.

2.1 Divisibility and Modular Arithmetic

Throughout these notes, we will use the following notation:

- the **integers** are the set $\mathbf{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$;
- the **non-negative integers** are the set $\mathbf{Z}_{\geq 0} = \{0, 1, 2, \ldots\}$;
- the **positive integers** are the set $\mathbf{Z}_{+} = \{1, 2, 3, \ldots\}$.

Definition 2.1. If $a, b \in \mathbf{Z}$ with $a \neq 0$, we say that a **divides** b, denoted $a \mid b$, if there exists $k \in \mathbf{Z}$ such that $b = ka$. If $a \mid b$, we also say that b **is a multiple of** a and a **is a divisor of** b.

Remark 2.2. We note that $a \mid b$ if and only if $\frac{b}{a} \in \mathbf{Z}$.

Example 2.3. Observe the following:

- $7 \mid 28$ because $28 = 4 \cdot 7$. Here, we produced the necessary k-value of 4.
- $7 \nmid 15$ because $2 \cdot 7 < 15 < 3 \cdot 7$.

Definition 2.4. Let $a, b, n \in \mathbf{Z}$ with $n > 0$. We say that a **is congruent to** b **modulo** n, denoted

$$a \equiv b \pmod{n}$$

provided $n \mid (a - b)$. Equivalently, there exists some $k \in \mathbf{Z}$ such that

$$a - b = kn \qquad \text{or} \qquad a = kn + b.$$

The number n is called the **modulus** of the congruence.

Example 2.5. We have the following congruences modulo 4:

$$\begin{aligned} -1 &\equiv 3 \pmod 4 \\ 26 &\equiv 2 \pmod 4 \\ 32 &\equiv 0 \pmod 4. \end{aligned}$$

Remark 2.6. We note that $a \equiv 0 \pmod n$ if and only if $n \mid a$.

Definition 2.7. The **least non-negative residue** of a modulo n is the smallest non-negative integer k such that $a \equiv k \pmod n$. We will abbreviate the least non-negative residue as LNR.

Example 2.8. We note that 2 is the least non-negative residue of 26 modulo 4. Further, we note that 3 is the least non-negative residue of -1 because we have to ensure the value is greater than or equal to zero.

2.2 Multiplicative Order

Let $a, n \in \mathbf{Z}$ with $n > 1$. Then, the **multiplicative order** of a modulo n is the smallest positive integer s such that $a^s \equiv 1 \pmod n$. If such an s exists, then we write $\operatorname{ord}_n(a) = s$.[2]

Example 2.9. Let us compute the multiplicative order of 3 modulo 7, i.e. $\operatorname{ord}_7(3)$. In order to perform this computation, we calculate:

$$\begin{aligned} 3^1 &\equiv 3 \pmod 7 \\ 3^2 &\equiv 2 \pmod 7 \\ 3^3 &\equiv 6 \pmod 7 \\ 3^4 &\equiv 4 \pmod 7 \\ 3^5 &\equiv 5 \pmod 7 \\ 3^6 &\equiv 1 \pmod 7 \end{aligned}$$

Thus, the multiplicative order of 3 modulo 7 is 6, i.e. $\operatorname{ord}_7(3) = 6$.

[2]For those who have taken a course in abstract algebra, you will recognize that $\operatorname{ord}_n(a)$ is the order of the element a in the multiplicative group $(\mathbf{Z}/n\mathbf{Z})^\times$.

Example 2.10. Let us try to compute $\text{ord}_{14}(4)$.

$$\begin{aligned} 4^1 &\equiv 4 \pmod{14} \\ 4^2 &\equiv 2 \pmod{14} \\ 4^3 &\equiv 8 \pmod{14} \\ 4^4 &\equiv 4 \pmod{14} \\ 4^5 &\equiv 2 \pmod{14} \\ 4^6 &\equiv 8 \pmod{14} \\ 4^7 &\equiv 4 \pmod{14} \\ &\vdots \end{aligned}$$

This pattern continues and thus, $\text{ord}_{14}(4)$ does not exist.

2.3 Primitive Roots

If p is a prime and $\text{ord}_p(a) = p - 1$, then we say that a is a **primitive root** modulo p. For example, Example 2.9 tells us that 3 is a primitive root modulo 7, since $\text{ord}_7(3) = 6 = 7 - 1$. By Fermat's Little Theorem, if $p \nmid a$, then $a^{p-1} \equiv 1 \pmod{p}$. Therefore, the additional constraint on a primitive root a, that $\text{ord}_p(a) = p - 1$, ensures that a has the maximum possible order.

Theorem 2.11. If $\gcd(a, n) = 1$, then

$$\text{ord}_n(a) \mid k \iff a^k \equiv 1 \pmod{n}.$$

Proof. We must prove both directions of this implication.

($\Rightarrow$) Let $\text{ord}_n(a) \mid k$. Then, there exists $\ell \in \mathbf{Z}$ such that $k = \ell \cdot \text{ord}_n(a)$. Compute

$$a^k \equiv a^{\ell \cdot \text{ord}_n(a)} \equiv (a^{\text{ord}_n(a)})^\ell \equiv 1^\ell \equiv 1 \pmod{n}.$$

($\Leftarrow$) Assume $a^k \equiv 1 \pmod{n}$. By the division algorithm, $k = q(\text{ord}_n(a)) + r$ where $0 \leq r < \text{ord}_n(a)$. By way of contradiction, assume $r > 0$. Compute

$$\begin{aligned} a^k &\equiv a^{q(\text{ord}_n(a))+r} && \pmod{n} \\ &\equiv (a^{\text{ord}_n(a)})^q (a^r) && \pmod{n} \\ &\equiv (1^q)(a^r) && \pmod{n} \end{aligned}$$

$$\equiv a^r \pmod{n}.$$

By assumption, $a^k \equiv 1 \pmod{n}$, thus $a^r \equiv 1 \pmod{n}$. However, $r < \operatorname{ord}_n(a)$, which is a contradiction since by definition, $\operatorname{ord}_n(a)$ is the smallest such exponent. Thus, $r = 0$ and $k = q \cdot \operatorname{ord}_n(a)$, i.e. $\operatorname{ord}_n(a) \mid k$, as desired. □

Theorem 2.12. If a is a primitive root modulo p with $0 < a < p$, then

$$a^i \equiv a^j \pmod{p} \iff i \equiv j \pmod{p-1}.$$

Proof. We must prove both directions of this implication.

($\Rightarrow$) Assume $a^i \equiv a^j \pmod{p}$. By definition, $p \mid (a^i - a^j)$. Without loss of generality, let $i > j$. Then, $a^i - a^j = a^j(a^{i-j} - 1)$. Thus, $p \mid (a^j(a^{i-j} - 1))$, and so by Euclid's lemma, $p \mid a^j$ or $p \mid (a^{i-j} - 1)$. Since $0 < a < p$, $p \nmid a$ and further, $p \nmid a^j$. Therefore, $p \mid (a^{i-j} - 1)$ and so

$$a^{i-j} \equiv 1 \pmod{p}.$$

By Theorem 2.11, $\operatorname{ord}_p(a) \mid (i-j)$. Since a is a primitive root modulo p, $\operatorname{ord}_p(a) = p-1$. Thus, $(p-1) \mid (i-j)$, i.e. $i \equiv j \pmod{p-1}$.

($\Leftarrow$) Assume $i \equiv j \pmod{p-1}$. So, $(p-1) \mid (i-j)$ or $k(p-1) = i - j$ for some $k \in \mathbf{Z}$. We have

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

by Fermat's Little Theorem. So, $a^{i-j} \equiv 1 \pmod{p}$. Multiply both sides by a^j, and we find $a^i \equiv a^j \pmod{p}$, as desired. □

Corollary 2.13. If p is a prime and b is a primitive root modulo p, then

(i) $\{\text{LNR of } b^i : 1 \leq i \leq p-1\} = \{1, \ldots, p-1\}$;

(ii) if $y \in \{1, \ldots, p-1\}$, then

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

has a solution for some $x \in \mathbf{Z}$.

Proof. Exercise [Q2.6]. □

Theorem 2.14. If p is a prime, then p has a primitive root.

Proof. See [Raj13, Theorem 61]. □

Example 2.15. Find all primitive roots for $p = 5$. We check $a \in \{1, 2, 3, 4\}$.

$a = 1$:

$$1^1 \equiv 1 \pmod 5$$

So, $\operatorname{ord}_5(1) = 1 \neq 5 - 1$.

$a = 2$:

$$2^1 \equiv 2 \pmod 5$$
$$2^2 \equiv 4 \pmod 5$$
$$2^3 \equiv 3 \pmod 5$$
$$2^4 \equiv 1 \pmod 5$$

Thus, $\operatorname{ord}_5(2) = 4 = 5 - 1$. Therefore, 2 is a primitive root modulo 5.

$a = 3$:

$$3^1 \equiv 3 \pmod 5$$
$$3^2 \equiv 4 \pmod 5$$
$$3^3 \equiv 2 \pmod 5$$
$$3^4 \equiv 1 \pmod 5$$

Thus, $\operatorname{ord}_5(3) = 4 = 5 - 1$. Therefore, 3 is a primitive root modulo 5.

$a = 4$:

$$4^1 \equiv 4 \pmod 5$$
$$4^2 \equiv 1 \pmod 5$$

So, $\operatorname{ord}_5(4) = 2 \neq 5 - 1$.

In conclusion, 2 and 3 are the primitive roots modulo 5.[3]

[3] For those who have taken a course in abstract algebra, you will notice that the collection of primitive roots modulo n are the generators of the multiplicative group $(\mathbf{Z}/n\mathbf{Z})^\times$.

2.4 Discrete Log Problem

Let $b, x \in \mathbf{Z}$ and $n \in \mathbf{Z}_+$. Let y be the least positive residue of b^x modulo n. That is,

$$b^x \equiv y \pmod{n}, \qquad 0 \leq y < n.$$

The **discrete log problem** asks whether x can be determined knowing b, y, and n.

The name stems from the following scenario. Consider $b^x = y$ for $x \in \mathbb{R}$ instead of $x \in \mathbf{Z}$. Then, $x = \log_b(y)$. If b and y are known, then x is easily calculable. For example, let us solve $10^x = 102$. Then, we know $x = \log(102)$. Since 10^x is an increasing function, we note

$$\begin{aligned} 10^0 &= 1 \\ 10^1 &= 10 \\ 10^2 &= 100 \\ 10^3 &= 1000. \end{aligned}$$

Since $100 < 102 < 1000$, we know that $2 < \log(102) < 3$. We note that $\log(102)$ is much closer to 2 than 3, so we compute

$$10^{2.1} = 125.8893.$$

Thus, $2 < \log(102) < 2.1$. Since again $\log(102)$ is closer to 2 than 2.1, we compute

$$10^{2.05} = 112.202.$$

So, $2 < \log(102) < 2.05$. Since 2.05 is still too big, we compute

$$10^{2.025} = 105.995.$$

Therefore, $2 < \log(102) < 2.025$. We can continue in this way to estimate $\log(102)$ as precisely as we would like.

Over $\mathbb{R}$, solving $b^x = y$ is as simple as calculating logarithms. In the discrete case (i.e. using modular arithmetic), consider solving the same style equation:

$$10^x \equiv 102 \pmod{115}.$$

Unlike in the real case, 10^x is not an increasing function modulo 115, as seen from the lack

of a pattern below.

$$\begin{aligned} 10^0 &\equiv 1 \pmod{115} & 10^5 &\equiv 65 \pmod{115} \\ 10^1 &\equiv 10 \pmod{115} & 10^6 &\equiv 75 \pmod{115} \\ 10^2 &\equiv 100 \pmod{115} & 10^7 &\equiv 60 \pmod{115} \\ 10^3 &\equiv 80 \pmod{115} & 10^8 &\equiv 25 \pmod{115} \\ 10^4 &\equiv 110 \pmod{115} & 10^9 &\equiv 20 \pmod{115} \end{aligned}$$

This is an example of the discrete log problem, and this problem is thought to be hard. Here, the term "hard" refers to the notion of **computational hardness** in the computer science sense. That is, the discrete log problem relies on the **computational hardness assumption**, which is the hypothesis that a particular problem cannot be solved efficiently, i.e. in polynomial time. We will continue to use the term "hard" in this manner.

2.5 Diffie-Hellman Key Exchange

We now describe the Diffie-Hellman key exchange, described initially in [DH76]. In order to understand a public-key cryptosystem, we need to detail the following four attributes:

1. Key Setup
2. Encryption Algorithm
3. Decryption Algorithm
4. Basis for Security

Idea: Alice and Bob are communicating over an insecure channel and need to agree on a key for a symmetric cryptosystem.

Description:

1. Alice (or Bob) picks a prime p and a primitive root modulo p, call it b.
2. Alice chooses an x with $1 \leq x < p$. She sends the least positive residue of b^x modulo p to Bob.
3. Bob picks a y with $1 \leq y < p$. He sends the least positive residue of b^y modulo p to Alice.
4. Everyone knows p, b^x, b^y, b. But only Alice knows x and only Bob knows y.

5. Alice computes $(b^y)^x \equiv b^{xy} \pmod{p}$.
6. Bob computes $(b^x)^y \equiv b^{xy} \pmod{p}$.
7. Since both Alice and Bob know b^{xy}, they choose their key to be the least positive residue of b^{xy} modulo p.

Security: Eve knows b, p, b^x, b^y. She can break the Diffie-Hellman key exchange if:

Option 1: She can compute discrete logs. That is, if given $b^x \equiv z \pmod{p}$, she can determine x. Then, she could compute $(b^y)^x \equiv k \pmod{p}$.

Option 2: Given b^x and b^y, she can compute b^{xy} perhaps without computing discrete logs. This is called the Diffie-Hellman problem, and thought to be hard.

Remark 2.16. The discrete log problem and the Diffie-Hellman problem are different problems, and it is not known if they are equivalent.

Example 2.17. Alice chooses $p = 71$ and $b = 56$. (This data is publicly available.)

1. Alices chooses $x = 22$ and computes
$$A = b^x = 56^{22} \equiv 29 \pmod{71}.$$
She sends $A = 29$ to Bob.
2. Bob chooses $y = 61$ and computes
$$B = b^y = 56^{61} \equiv 13 \pmod{71}.$$
He sends $B = 13$ to Alice.
3. Alice computes
$$B^x = (b^y)^x = 13^{22} \equiv 8 \pmod{71}.$$
4. Bob computes
$$A^y = (b^x)^y = 29^{61} \equiv 8 \pmod{71}.$$
5. Alice and Bob use $k = 8$ as their shared key.

Strictly speaking, b does not have to be a primitive root modulo p for the Diffie-Hellman key exchange to work. However, consider the following. Let $b \in \mathbf{Z}$, and p be prime. If $\gcd(b, p) = 1$, then $b^{p-1} \equiv 1 \pmod{p}$ by Fermat's Little Theorem. So, $\mathrm{ord}_p(b) \leq p - 1$. If b is such that $\mathrm{ord}_p(b) < p - 1$, then $\{\text{LNR of } b^i : 1 \leq i < p\}$ is smaller than $\{1, \ldots, p - 1\}$. So, there are fewer possibilities for k. If b is a primitive root modulo p, then the set $\{\text{LNR of } b^i : 1 \leq i < p\}$ is as large as possible.

> **Warning:** There are some **bad** choices for p. If $p - 1$ factors as a product of small primes, for example, then the Pohlig-Hellman algorithm [PH78] can solve the discrete log problem very efficiently.

We conclude by noting that the Diffie-Hellman key exchange is still in use today in some TLS protocols. In practice, the primes chosen as the modulus in the algorithm must be over 600 digits long to ensure safe communication! While the DHKE is still secure, the majority of implementations have been replaced by the Elliptic Curve Diffie-Hellman key exchange (ECDH). The algorithm for ECDH is essentially the same as that for the DHKE, with the problems now posed in the context of points on elliptic curves, instead of integers. To learn more, see [HPS14, Section 5.4.1].

3 Cryptoanalysis

The goal of this chapter is to discuss the baby-step giant-step algorithm introduced in [Sha71]. Developed by Daniel Shanks and published in 1971, the algorithm provides a more efficient method for solving the discrete log problem and thereby, an effective strategy for breaking the Diffie-Hellman key exchange.

However, since we brought up RSA at the start of Chapter 2, we would be remiss not to mention a technique that could be used to attack it, as well. So, before discussing baby-step giant-step, we make a small digression. Recall, the security of RSA is based on the difficulty of factoring the product of two large prime numbers. One method for effective integer factorization was developed by John M. Pollard in 1974 and published in [Pol74].

The method, called Pollard's $p - 1$ algorithm, is not useful for all large numbers; however, it is extremely efficient for certain numbers. We note that the existence of such an algorithm suggests that, similar to the Diffie-Hellman key exchange, there are also some bad choices for moduli in RSA. The fact that such an algorithm was published prior to the introduction of RSA indicates that we must be cognizant of potential attacks when

preparing a cryptosystem. For an in-depth treatment of Pollard's $p-1$ algorithm, see [HPS14, Section 3.5].

3.1 Logarithms and Modular Arithmetic

Before introducing the baby-step giant-step algorithm, we first need to understand how logarithms behave with respect to modular arithmetic. We start by making the following definition.

Definition 3.1. If p is a prime, $x, y \in \mathbf{Z}$, and b is a primitive root modulo p, then we write:

$$x \equiv \log_b(y) \pmod{p-1} \iff b^x \equiv y \pmod{p}.$$

Remark 3.2. Note that the moduli are different on each side of the implication.

Recall from Theorem 2.12 that

$$a^i \equiv a^j \pmod{p} \iff i \equiv j \pmod{p-1},$$

so Definition 3.1 makes sense. Moreover, we notice that the two operations are inverses.

Theorem 3.3 (Properties of Logarithms)**.** Suppose p is a prime and b is a primitive root modulo p, then for $x, y, z \in \mathbf{Z}$ with $x, y, z \not\equiv 0 \pmod{p}$

(a) $\log_b(b) \equiv 1 \pmod{p-1}$ and $\log_b(1) \equiv 0 \pmod{p-1}$;

(b) $\log_b(xy) \equiv \log_b(x) + \log_b(y) \pmod{p-1}$;

(c) if $k \in \mathbf{Z}$, then $\log_b(x^k) \equiv k \log_b(x) \pmod{p-1}$.

Proof. Exercise [Q3.1] □

Theorem 3.4. Let p be a prime and b a primitive root modulo p. Suppose $y \equiv b^x \pmod{p}$. Then, for any $m > 0$ such that $m^2 > p-1$, there exists i, j with $0 \le i, j \le m-1$ such that

$$yb^{-i} \equiv b^{mj} \pmod{p}.$$

Proof. Exercise [Q3.2] □

Recall, our goal is to find x such that $b^x \equiv y \pmod{p}$, given p, b, y. The idea of the baby-step giant-step algorithm is as follows. By Theorem 3.4, there exists $m > 0$ such that $m^2 > p - 1$. Moreover, there exists i, j such that $0 \leq i, j \leq m - 1$ and $yb^{-i} \equiv b^{mj} \pmod{p}$. Compute two lists:

- Baby steps: $yb^{-i} \equiv y(b^{-1})^i \pmod{p}$ for $0 \leq i \leq m - 1$.
- Giant steps: $b^{mj} \equiv (b^m)^j \pmod{p}$ for $0 \leq j \leq m - 1$.

There are m steps in each list. If we find a common element, i.e. the lists "meet in the middle", then $yb^{-i} \equiv b^{mj} \pmod{p}$ or $y \equiv b^{mj+1} \pmod{p}$. So, $x \equiv mj + i \pmod{p-1}$.

3.2 Baby-Step Giant-Step

We now present the rigorous steps of the baby-step giant-step algorithm. Due to its setup of finding a common element, baby-step giant-step is an example of a **meet-in-the-middle** (MITM) algorithm. We will see later how the complexity of the algorithm comapres to a naive approach.

Baby-step Giant-step Algorithm [Sha71].

1. Choose $m > 0$ such that $m^2 > p - 1$.
2. Compute b^{-1} and the baby steps (m of them)

$$y, yb^{-1}, yb^{-2}, \ldots, yb^{-(m-1)} \pmod{p}$$

   by multiplying by b^{-1} at each subsequent step.

3. Compute b^m and the giant steps (m of them)

$$1, b^m, b^{2m}, \ldots, b^{(m-1)m} \pmod{p}$$

   by multiplying by b^m at each subsequent step.

4. Find a common element, as guaranteed by Theorem 3.4.
5. Since $yb^{-i} \equiv b^{mj} \pmod{p}$ for some $0 \leq i, j \leq m - 1$, we have $y \equiv b^{mj+i} \pmod{p}$, and so by Definition 3.1, $x \equiv mj + i \pmod{p-1}$.

Example 3.5. Let $p = 29$ and $y = 19$. We note that $b = 3$ is a primitive root modulo 29. Our goal is to find x such that $3^x \equiv 19 \pmod{29}$.

1. We need an $m > 0$ such that $m^2 > p - 1 = 28$. Choose $m = 6$.

2. Compute b^{-1}, i.e. we want to find a z such that $3z \equiv 1 \pmod{29}$. We note that $z = 10$ works since $3(10) = 30 \equiv 1 \pmod{29}$. (If a solution is not immediate by inspection, use the Euclidean algorithm.) Thus, $3^{-1} \equiv 10 \pmod{29}$. Now, we compute the baby steps yb^{-i} for $y = 19$:

$$\begin{aligned} y &\equiv 19 \pmod{29} \\ yb^{-1} &\equiv 19(10) \equiv 190 \equiv 16 \pmod{29} \\ yb^{-2} &\equiv 16(10) \equiv 160 \equiv 15 \pmod{29} \\ yb^{-3} &\equiv 15(10) \equiv 150 \equiv 5 \pmod{29} \\ yb^{-4} &\equiv 5(10) \equiv 50 \equiv 21 \pmod{29} \\ yb^{-5} &\equiv 21(10) \equiv 210 \equiv 7 \pmod{29} \end{aligned}$$

3. Compute giant steps for $b = 3$:

$$\begin{aligned} b^0 &\equiv 1 \pmod{29} \\ b^m &\equiv 3^6 \equiv 27^2 \equiv (-2)^2 \equiv 4 \pmod{29} \\ b^{2m} &\equiv 4^2 \equiv 16 \pmod{29} \end{aligned}$$

   We can stop here because we have found 16 in both lists!

4. Thus,
$$16 \equiv yb^{-1} \equiv b^{2m} \equiv 16 \pmod{29}.$$

5. Therefore,
$$y \equiv b^{2m+1} \equiv b^{13} \pmod{29},$$
   so $x = 13$. We can verify by solving
$$3^{13} \equiv 3^{12}(3) \equiv (27)^4(3) \equiv (-2)^4(3) \equiv (16)(3) \equiv 48 \equiv 19 \pmod{29}.$$

Remark 3.6. In order to solve $b^x \equiv y \pmod{p}$ naively, we would need to check roughly p exponents. For the baby-step giant-step algorithm, though, we only need $2m$ modular arithmetic calculations. Since $m^2 \approx p-1$, we have that $m \approx \sqrt{p}$. So, we only need around $2\sqrt{p}$ calculations, which is a significant improvement.

4 Exercises

Chapter 1

[Q1.1] The solution to Question 1.4 is $m = (3, 20, 2, 10) =$ `duck`. Working with vectors and matrices defined over the integers, there are actually infinitely many possible matrices k that can be used for encryption. However, in practice, we often work over a finite field of prime order.

[Eis98, Section 3.1] describes a method for encrypting m as a matrix defined over $\mathbf{Z}/29\mathbf{Z}$, by extending the alphabet with three additional characters. Using the algorithm discussed in [Eis98, Section 3.3], one can calculate the original key used for encryption knowing both m and c. Determine the unique matrix k, defined over $\mathbf{Z}/29\mathbf{Z}$, used to encode the original message. Then, using that key, decode the following second piece of ciphertext $c = (27, 28, 17, 20)$.

Transposition ciphers are a type of cryptosystem that have been used throughout history. This method of encryption scrambles the positions of characters without actually changing the characters themselves.

[Q1.2] An example of a transposition cipher is the **route cipher**, which was used by the Union army during the Civil war. In a route cipher, plaintext is arranged in a grid of given dimension. Then, the key is a phrase that describes the motion used to navigate the grid to rearrange the letters. For example, consider the message `WE NEED HELP RIGHT NOW`. We start by arranging in a 3×6 grid:

```
W  E  H  P  G  N
E  E  E  R  H  O
N  D  L  I  T  W
```

Then, consider the key phrase, "Snake from the bottom right, start by moving left." Such a key would result in the ciphertext `WTI LDN EEE RHO NGP HEW`, with spacing for readability.

(a) Using the same key as above, encode the following message. Be sure to use an appropriately-sized grid:

$$m = \texttt{ENCRYPTION ROCKS}$$

(b) A route cipher was used to encode the following ciphertext, see if you can decipher it! Again, spacing is used for readability.

$$c = \texttt{CED PYR OIT URN SEL}$$

(Hint 1: The subject is relevant to your task at hand.)

(Hint 2: Use a 3×5 grid.)

(Hint 3: The key phrase is, "Only up, starting from the bottom left.")

[Q1.3] [Chr15] describes **columnar transposition**, another type of transposition cipher used throughout the 19th and 20th centuries.

(a) Encode the message `SWIM AT WESLEYAN` using columnar transposition with the keyword "fried."

(b) A message was encrypted using columnar transposition. Determine the original message by using the frequency analysis techniques discussed in [Chr15, pp. 3-7] and the resulting ciphertext below:

$$c = \texttt{WILWE LEILF XATEA NOM}$$

If required, the digraphic frequencies table from [Sin09, Appendix A] can be found here. Alternatively, a spreadsheet version is available here, to aid in computations.

Blackout ciphers are yet another popular type of cryptosystem. They are frequently depicted in film, as they translate well onto the screen. A blackout cipher works by starting with some source text, and then using a key to highlight certain words or letters from that source text (and thus blacking out the rest). The Bible is often used as the source text because of its ubiquity, as well as its standard chapter and verse markings. These types of cryptosystems are also called **book ciphers**, since a book is often used as the source text.

Consider the following source text, using the key, "Odd prime words only."

`I suppose we would need medication to deal with the flea problem.`

Blacking out all words that are not odd primes, we find:

Therefore, the message being communicated is, "We need to flee!" Note the use of the homophone "flea" to further obscure the true meaning.

[Q1.4] Decode the secret message being passed using the key phrase, "Fibonacci numbers from eight," and the source text:

`The monthly department meeting will be on Thursday at 11am. First up, we have to approve John's request for medical leave. Then, we can discuss upcoming curriculum changes. Lunch will be provided from noon until 2pm.`

[Q1.5] The key for a blackout cipher has been encrypted using one of the cryptosystems discussed in Chapter 1 or the Chapter 1 Exercises. Below is the ciphertext that resulted from that encryption:

$$c = \texttt{UJWKJHY XVZFWJX}$$

(a) Determine the cryptosystem used and decode the key for the blackout cipher.

(b) Decode the message using the key from part (a) and the following source text:

`I made a request to FedEx for their delivery trucks to enter the loading dock immediately, so that we can unload more quickly.`

[Q1.6] Encode a message for the instructor and teaching assistants using one of the methods from Chapter 1 or the Chapter 1 Exercises. Present the ciphertext, key or key phrase, and cryptosystem used; time permitting, they will decode the message.

Chapter 2

The **RSA cryptosystem** has the following attributes:

Key Setup: The recipient of the message, Bob, creates the keys as follows:

1. Choose two primes p and q.
2. Compute $n = pq$, and Euler's totient function $\varphi(n) = (p-1)(q-1)$.
3. Choose an e such that $\gcd(e, \varphi(n)) = 1$.
4. Compute d such that $ed \equiv 1 \pmod{\varphi(n)}$.
5. Then, the keys are as follows:

- Encryption/Public Key: The pair (n, e).
- Decryption/Private Key: The triple (d, p, q). Do NOT reveal d, p, q or $\varphi(n)$!

Encryption:

1. Alice wants to send Bob a message.
2. She retrieves Bob's public key, the pair (n, e).
3. She encodes her messages as M with $0 \leq M < n$.
4. Alice computes the least positive residue of $M^e \pmod{n}$, and calls it C.
5. She sends C to Bob.

Decryption:

1. Bob receives C.
2. Bob computes the least positive residue of $C^d \pmod{n}$, and calls it D.
3. Bob claims $M = D$.

Theorem 4.1. Let $n = pq$, where $p \neq q$ primes. Suppose $e, d \in \mathbf{Z}$ such that $ed \equiv 1 \pmod{\varphi(n)}$. If $C \equiv M^e \pmod{n}$, then $M \equiv C^d \pmod{n}$.

Proof. Exercise [Q2.2] □

Security of RSA: Eve knows n, e, and C. She wants to find d so that she can compute $M \equiv C^d \pmod{n}$. How could she do it?

Option 1: Eve could factor n into its product of primes, i.e. find p and q. Having p and q, she could easily compute $\varphi(pq) = \varphi(n)$. She knows e already, and could use Euclid's algorithm to compute d.

Thankfully, factoring is hard! Naively, we have to check every x with $2 \leq x < \sqrt{n}$. If n has 600 decimal digits, then there are roughly $10^{300} = \sqrt{10^{600}}$ numbers to try. That is, we would have to divide n by 10^{300} numbers. That is *a lot* of calculations.

To put this in perspective, the fastest supercomputers on Earth compute roughly 10^{18} instructions per second. If you harnessed the power of 100 supercomputers, you could do 100×10^{18} instructions per second. At this rate, this calculation would take:

$$\frac{10^{300} \text{ instructions}}{10^{20} \text{ instructions/second}} \approx 10^{280} \text{ seconds} \approx 10^{272} \text{ years.}$$

The universe is only 10^{10} years old, so these calculations would take a *very* long time.

Option 2: Eve could find $\varphi(n)$ without knowing p or q directly. If Eve had $\varphi(n)$, then knowing e, she could again use Euclid's algorithm to compute d. However, it turns out that computing $\varphi(n)$ is no easier than factoring n!

Option 3: Eve could guess! Since she has e, n, and C, and $M^e \equiv C \pmod{n}$, Eve could just try to guess M. Over the real numbers, this problem is straightforward. For example, consider $x^2 = 105$. We note

$$10^2 = 100 < 105 < 121 = 11^2.$$

Therefore, $10 < x < 11$. However, in the world of modular arithmetic, the same data does not yield nearly as useful results:

$$10^2 \equiv 100 \pmod{115} \qquad \text{and} \qquad 11^2 \equiv 6 \pmod{115}.$$

[Q2.1] The following exercise will walk through an example of both encrypting and decrypting via RSA.

(a) Bob's public key is $(1007, 89)$. Playing the role of Alice, encode the message $M = 538$.

(b) Bob's private key is $(19, 53, 305)$. Playing the role of Bob, decode the ciphertext $C = 243$.

[Q2.2] Prove Theorem 4.1.

[Q2.3] Compute $\text{ord}_{10}(2)$.

[Q2.4] Let p be an odd prime. Let r be the multiplicative order of x modulo p for x in the multiplicative group of the finite field $\mathbb{F}_p$, i.e. $x \in \mathbb{F}_p^\times$. Prove that r exists and that it divides $p - 1$.

[Q2.5] Find all primes p with $p < 18$ such that 2 is a primitive root modulo p. (Hint: Computers are very good at computing powers of 2 due to their architecture.)

[Q2.6] Prove Corollary 2.13. (Hint: Use Theorems 2.11 and 2.12.)

[Q2.7] Alice has the public key $p = 941$ and $b = 627$.

(a) Bob wants to correspond with Alice. So, Bob chooses $y = 781$. Playing the role of Bob, calculate the number B that you send to Alice.

(b) Alice, having received Bob's correspondence request, sends Bob the number $A = 390$ in response. As Bob, determine your shared symmetric key with Alice.

(c) Verify the key in part (b) by now playing the role of Alice. To start, Alice chose $x = 347$ after receiving Bob's initial number B from part (a).

[Q2.8] Consider the public key $p = 29$ and $b = 8$. Your goal is to communicate with a second group of undergraduate students using the symmetric cryptosystem described in [Eis98, Section 3].

The two groups will need to pass four keys $k_1, \ldots, k_4$ to one another using the Diffie-Hellman key exchange in order to build a 2×2 key matrix

$$K = \begin{pmatrix} k_1 & k_2 \\ k_3 & k_4 \end{pmatrix}.$$

Once you have agreed on K, begin encrypting messages and sending them to the other group. Be careful, some evil graduate students are eager to intercept and decode your messages!

Chapter 3

[Q3.1] Prove Theorem 3.3.

[Q3.2] Prove Theorem 3.4. (Hint: Use the division algorithm.)

[Q3.3] Solve $6^x \equiv 10 \pmod{13}$ using the baby-step giant-step algorithm. (Note: This exercise is small enough to compute by hand.)

[Q3.4] Using any programming language of your choosing, write a script that performs the baby-step giant-step algorithm. Then, solve $521^x \equiv 1234 \pmod{300023}$.

[Q3.5] The computer algebra system SageMath has a built-in function for solving discrete logs. Call the function by running the following code in order to solve for x in the discrete log $b^x \equiv y \pmod{p}$:

```
discrete_log(Mod(y, p), Mod(b, p), p-1)
```

You can run SageMath in your browser here. Solve the discrete log problem from the previous exercise using the SageMath function. Using the `time` command,

compare the speed of the built-in implementation to the speed of your baby-step giant-step algorithm. Are there any ways you can increase the efficiency of your algorithm?

References

[Chr15] C. Christensen, *Columnar transposition*, 2015. (Link)

[DH76] W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), no. 6, 644–654. (Link)

[Eis98] M. Eisenberg, *Hill ciphers and modular linear algebra*, 1998. (Link)

[Hil29] L. S. Hill, *Cryptography in an algebraic alphabet*, The American Mathematical Monthly **36** (1929), no. 6, 306–312. (Link)

[HPS14] J. Hoffstein, J. Pipher, and J. H. Silverman, *An introduction to mathematical cryptography*, 2nd ed., Undergraduate Texts in Mathematics, Springer, 2014. (MR3289167)

[Ker83] A. Kerckhoffs, *La cryptographie militaire*, Journal des Sciences Militaires (1883), 161–191. (Link)

[Mer78] R. C. Merkle, *Secure communications over insecure channels*, Commun. ACM **21** (April 1978), no. 4, 294–299. (Link)

[PH78] S. Pohlig and M. Hellman, *An improved algorithm for computing logarithms overgf(p)and its cryptographic significance (corresp.)*, IEEE Transactions on Information Theory **24** (1978), no. 1, 106–110. (Link)

[Pol74] J. M. Pollard, *Theorems on factorization and primality testing*, Proc. Cambridge Philos. Soc. **76** (1974), 521–528. (MR354514)

[Raj13] W. Raji, *An introductory course in elementary number theory*, 2013. (Link)

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Commun. ACM **21** (February 1978), no. 2, 120–126. (Link)

[Sha71] D. Shanks, *Class number, a theory of factorization, and genera*, Number Theory Institute (Proc. Sympos. Pure Math.), 1971, pp. 415–440. (MR316385)

[Sin09] A. Sinkov, *Elementary cryptanalysis*: *A mathematical approach*, Anneli Lax New Mathematical Library, vol. 22, Mathematical Association of America, 2009. (MR2530836)

[Was08] L. C. Washington, *Elliptic curves*: *Number theory and cryptography*, 2nd ed., Discrete Mathematics and its Applications, Chapman & Hall/CRC, 2008. (MR2404461)