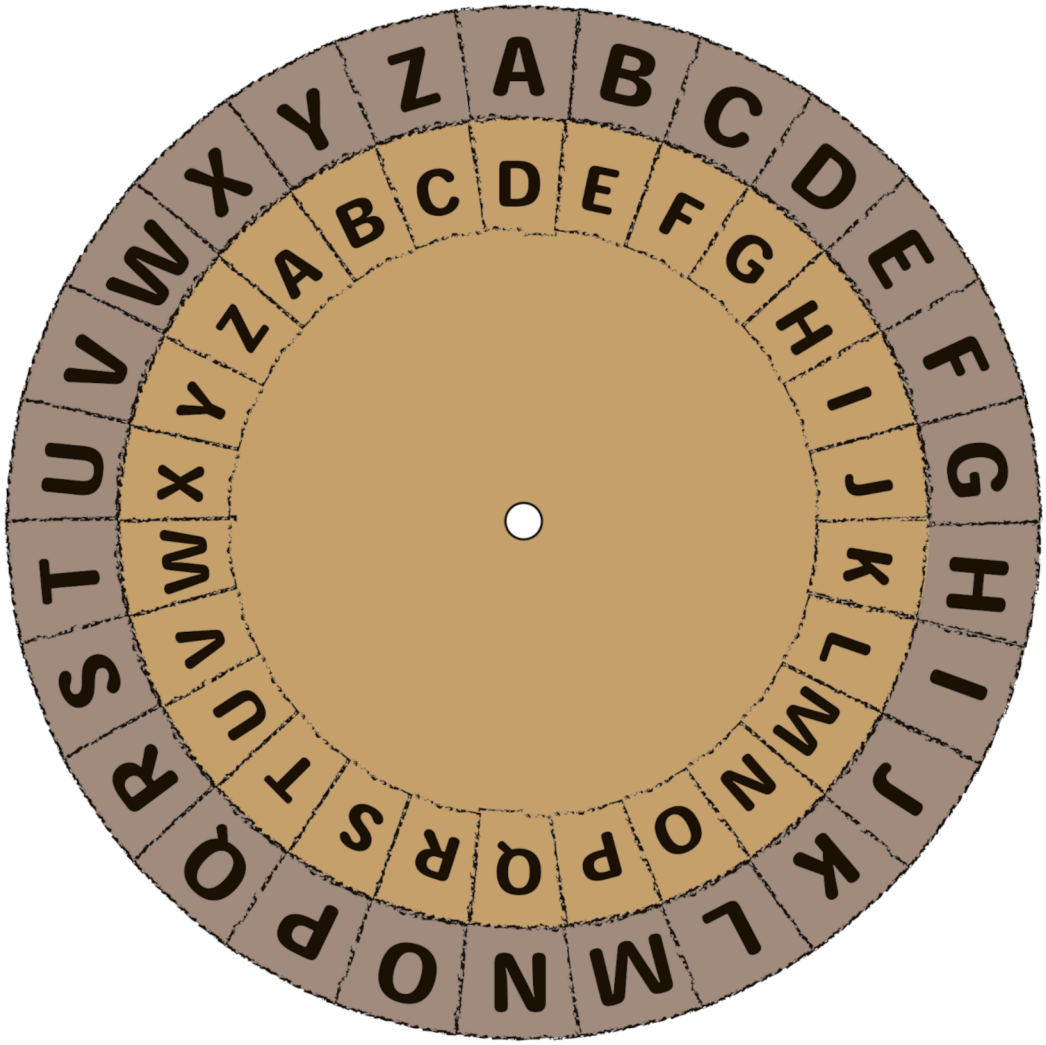
Caesar Cipher Breaking
A Concise Edition



Author: IPUSIRON

Caesar Cipher Breaking
A Concise Edition

Author: IPUSIRON

2025-07-20 Edition **Security Akademeia**

Table of Contents

Chapter 1 About This Book 5

1.1 The Original Book Has Full and Simplified Versions 5
1.2 What You Can Learn from This Book 5
1.3 Intended Audience 6
1.4 Prerequisites 6
1.5 Why Start Studying Classical Ciphers Now? 6
1.5.1 Many Unsolved Ciphertexts Still Remain 6
1.5.2 Classical Ciphers Satisfy Intellectual Curiosity 7
1.6 Note to Readers 7
1.7 Notes on This Book 7
1.8 Terminology 8
1.9 Disclaimer 8

Chapter 2 The Basics of the Caesar Cipher 9

2.1 What Is the Caesar Cipher? 9
2.2 What Is a Shift Cipher? 10
2.3 Shift Cipher Algorithm 11
2.3.1 Mathematical Expression of Correctness 12
2.3.2 Modify the Process to Use a Substitution Table 12
2.3.3 How to Call Each Algorithm 12
2.3.4 Focus on Input/Output Before Implementation Details 13
2.4 Applying ROT13 Twice Restores the Original Text 14
2.4.1 ROT13 Substitution Table 14
2.4.2 Applications of ROT13 15
2.5 Shift Ciphers Are a Type of Substitution Cipher 15
2.5.1 Shift Cipher Keys and Their Representations 15
2.6 Encrypting English Sentences with a Shift Cipher 16
2.7 Hands-on Practice with Caesar Cipher 16
2.7.1 Encryption Example 17
2.7.2 Decryption Example 17
Shift Ciphers and Mathematics 18

Chapter 3 Using Computers to Explore Shift Ciphers 19

3.1 Using the `tr` Command for Shift Ciphers 19
3.1.1 Testing Caesar Cipher with `tr` Command 19
3.1.2 Testing ROT13 20
3.2 Testing ROT13 with the `nkf` Command 20

3.3 Testing ROT13 with One-Liner Commands 21
3.4 Testing Caesar Cipher in Your Browser 22
3.5 Testing Caesar Cipher with Python 24
3.5.1 Extending ROT13 to Match the Character Set 25
3.6 Testing the ROT13 Encoder 26
3.7 Working with Shift Ciphers Using CrypTool 2 27
3.7.1 Using the Shift Cipher Wizard 28
3.7.2 Creating a Shift Cipher Project 31

Chapter 4 Breaking Caesar and Shift Ciphers 33
4.1 Breaking Shift Ciphers with Brute-Force Attacks 33
4.1.1 Manual Brute-Force Attack 33
4.2 Breaking Ciphers Through Frequency Analysis 35
4.2.1 Frequency Analysis Against Monoalphabetic Substitution Ciphers 35
4.2.2 Initial Letter Patterns in English Words 35
4.2.3 Analyzing Character Frequencies Without Word Boundaries 35
4.2.4 Breaking Shift Ciphers with Frequency Analysis in CrypTool 2 36
4.3 Identifying Shift Cipher Ciphertext 37
4.3.1 Using Index of Coincidence to Identify Monoalphabetic Ciphers 38
4.4 Verifying Correct Plaintext Recovery 39
4.4.1 Leveraging Dictionary Files 40
4.4.2 Performing Brute-Force Attack and Dictionary Matching with CrypTool 2 40

Chapter 5 Improving the Shift Cipher 50
5.1 Extending to the Vigenère Cipher 50
5.1.1 Vigenère Cipher Algorithm 51
5.1.2 Key Generation 52
5.1.3 Encryption Mechanism 53
5.1.4 Manual Encryption Example 53
5.2 Experimenting with the Vigenère Cipher Tool 53

Appendix A Letter Frequency in Alphabets 55
A.1 Letter Frequency Distribution 55
A.2 Letter Frequency Graph 56
A.3 Grouping by Letter Frequency 56
A.4 Letter Frequencies Across Languages 57

Afterword 58

Acknowledgments 58

About the Author **59**

IPUSIRON 59

Contact 60

License **61**

Chapter 1

About This Book

This book focuses on the Caesar cipher, one of the most famous and simplest classical ciphers, along with its natural extension—the shift cipher.

While conventional cryptography books typically devote only a few pages to these ciphers, this book dedicates an entire volume to exploring them in depth. You will fully understand both the Caesar cipher and the shift cipher, gaining insight into the essence of classical cryptography.

The shift cipher's simplicity makes it easy to apply, and it forms the foundation for several advanced classical and modern ciphers. By mastering these fundamental concepts, you'll develop a fresh perspective on modern cryptographic systems.

1.1 The Original Book Has Full and Simplified Versions

This book is an English translation of "How to Decipher Caesar Ciphers: Simplified Version," originally published in Japanese.

The Japanese edition comes in two versions:

- Full version: 16 chapters + 5 appendices with detailed explanations
  - `https://hack.booth.pm/items/5557030`
- Simplified version: 5 chapters + 1 appendix focusing on essentials
  - `https://www.amazon.co.jp/dp/B0FH1DT5RX`

This English edition, based on the simplified version, is ideal for readers who want to quickly grasp Caesar cipher mechanisms and decryption methods.

Depending on reader response and interest, we may consider translating the full version or creating an expanded edition in the future.

1.2 What You Can Learn from This Book

- Understanding the mechanisms of the Caesar cipher and shift ciphers
- Basic structure and concepts of classical cryptography
- Hands-on experience with encryption using Linux commands
- Cryptanalysis using CrypTool 2
- Interactive practice using web tools created by the author

1.3 Intended Audience

- Students (K-12) seeking independent study or research projects
- Beginners in cybersecurity
- Amateur cryptography enthusiasts
- Those interested in cryptographic theory
- People who want to revisit classical ciphers
- People who want to develop cryptanalysis skills
- Fans of the Caesar cipher
- Those seeking classical cryptography for general education

1.4 Prerequisites

No prior knowledge is required to read this book. You don't need advanced mathematics or programming skills.

1.5 Why Start Studying Classical Ciphers Now?

1.5.1 Many Unsolved Ciphertexts Still Remain

There are still many unsolved ciphertexts left in the world. Some are world-famous, while others were created by ordinary individuals and may seem insignificant from a historical standpoint.

It is commonly believed that the number of unsolved ciphertexts will not increase, but this is not actually the case. During the dismantling or organization of libraries or old houses, previously unknown documents, journals, and notes are sometimes discovered. Among these documents, there are occasionally ciphertexts.

If you're interested, try accessing the ciphertext database on HCPortal (Portal of Historical Ciphers). It contains a wide range of ciphertexts, including encrypted postcards and notes. Each entry is tagged as either "Solved" or "Not solved", making it easy to find unsolved ciphertexts.

- HCPortal
  - `https://hcportal.eu/`
- Ciphertext Database
  - `https://www.cryptograms.hcportal.eu/`

If you succeed in deciphering a famous unsolved ciphertext, it will undoubtedly bring you great recognition—possibly even beyond the cryptography and IT communities.

1.5.2 Classical Ciphers Satisfy Intellectual Curiosity

The final reason to study classical ciphers is that they satisfy intellectual curiosity.

Knowing about classical ciphers as trivia might come in handy two or three times in your life. Since they frequently appear in movies and novels, understanding them will enhance your enjoyment of these stories.

Classical ciphers can be created with nothing more than pen and paper, making them useful for secret communication when computers are unavailable. There have been documented cases of prisoners under surveillance using classical ciphers to communicate with each other or with people on the outside.

1.6 Note to Readers

We welcome your feedback, questions, and experimental reports about this book.

To share your thoughts publicly, please post on X (formerly Twitter) using:

- Keyword: "Caesar Cipher Breaking"
- Hashtag: #CaesarCipherBreaking

Public discussions benefit the entire reader community. When reporting issues or asking questions, please include relevant details such as error screenshots to facilitate troubleshooting.

- Resources:
  - FAQ and errata: Security Akademeia(`https://akademeia.info/`)
- Contact: If you discover errors not listed in the errata, please contact the author:
  - Email: ipusiron@gmail.com
  - X (Twitter): @ipusiron(`https://x.com/ipusiron`)

Verified corrections will be added to the errata promptly.

1.7 Notes on This Book

- We cannot respond to inquiries beyond the scope of this book, questions without specific section references, or issues arising from individual reader environments.
- URLs and other information in this book may change without notice.
- PDF viewers may introduce spaces or break URLs when copying. If this occurs, please type the URL manually.

1.8 Terminology

- "At the time of writing" refers to July 2025.

1.9 Disclaimer

The content of this book is provided for informational purposes only and does not guarantee fitness or accuracy for any specific purpose. Any use of this book's content must be at your own discretion and risk. The author assumes no liability for any resulting outcomes.

Chapter 2

The Basics of the Caesar Cipher

In this chapter, we will provide a detailed explanation of the Caesar cipher, which is the central theme of this book.

If we were to explain the Caesar cipher in a straightforward manner, it might simply come across as "a cipher with a simple mechanism."

However, having read this far, you will find this chapter especially valuable.

With an understanding of the essence of classical ciphers, you will be able to recognize the Caesar cipher's deeper significance.

2.1 What Is the Caesar Cipher?

The Caesar cipher is an encryption method that converts plaintext into ciphertext by shifting each letter three positions forward in alphabetical order.

This cipher is named after Gaius Julius Caesar—known simply as "Caesar" in English—the Roman military leader who famously employed it during the Gallic Wars to communicate with his allies while avoiding enemy interception.

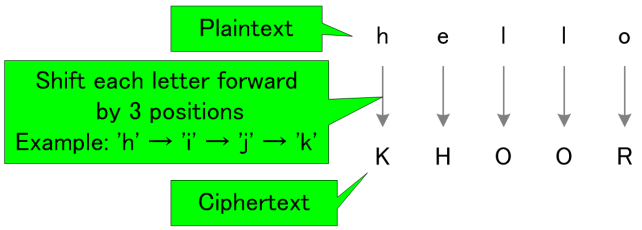
Figure 2.1 shows the **substitution table** for the Caesar cipher.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

▲ Figure 2.1 Substitution Table for the Caesar Cipher

Each letter in the plaintext is shifted three positions forward to generate the ciphertext. When this shift extends beyond 'Z', it wraps around to the beginning of the alphabet.

As an example, let us examine how the plaintext "hello" is encrypted using the Caesar cipher. Each letter is encrypted sequentially. See Figure 2.2.



▲ Figure 2.2 Encrypting "hello" using Caesar cipher

- Plaintext letter 'h' ⇒ Ciphertext letter 'K'
- Plaintext letter 'e' ⇒ Ciphertext letter 'H'
- Plaintext letter 'l' ⇒ Ciphertext letter 'O'
- Plaintext letter 'l' ⇒ Ciphertext letter 'O'
- Plaintext letter 'o' ⇒ Ciphertext letter 'R'

As a result, the ciphertext becomes **"KHOOR"**.

Note that plaintext is written in lowercase and ciphertext in uppercase, following a traditional convention in classical cryptography. This distinction makes it easy to identify which is which at a glance. However, not all texts follow this convention, so context remains important.

Although the Caesar cipher is simple, it demonstrates three fundamental elements of modern cryptography:

- Algorithm
- Key (3 in the Caesar cipher)
- Substitution

2.2 What Is a Shift Cipher?

The Caesar cipher typically shifts letters by 3 positions, but it isn't limited to 3. For instance, you can shift by 1, 10, or any other number of positions. Shifting by 26 positions is effectively the same as no shift at all, since the remainder when divided by 26 is zero.

A cipher that shifts letters by any arbitrary number is called a **shift cipher**, with the shift amount defined as **n**. This value **n** serves as the key, shared only between sender and receiver.

Due to their cyclic nature, shift ciphers are often called **ROT ciphers**, from the English word *rotate*. A shift of **n** positions is denoted as **ROTn**.

For example, a 1-position shift is written as **ROT1**. See Figure 2.3 for its substitution table.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A

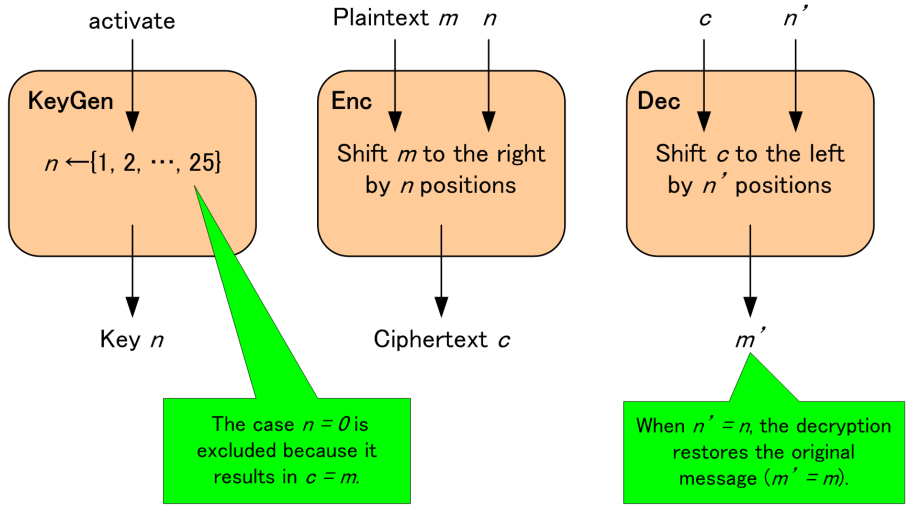
▲ Figure 2.3 Substitution Table for ROT1

The Caesar cipher is a shift cipher with a shift of 3, denoted as ROT3.

Since shift ciphers and the Caesar cipher share the same mechanism, some sources use these terms interchangeably. Context determines which term is more appropriate.

2.3 Shift Cipher Algorithm

From an algorithmic perspective, an encryption scheme comprises three fundamental algorithms: **KeyGen**, **Enc**, and **Dec**. Here, **KeyGen** denotes the key generation algorithm, **Enc** denotes the encryption algorithm, and **Dec** denotes the decryption algorithm[*1].



▲ Figure 2.4 Shift Cipher Algorithm

The key generation algorithm generates a key, while the encryption algorithm converts plaintext into ciphertext. The decryption algorithm decrypts the ciphertext.

The term key here refers to the data that controls the transformation process in both encryption and decryption algorithms. In cryptographic systems, keys are critically important and must be kept secret from anyone other than the sender and receiver to maintain secure communication. If a key is compromised, the ciphertext can be decrypted by anyone, exposing the plaintext.

[*1] These three algorithms—**KeyGen**, **Enc**, and **Dec**—are not exclusive to shift ciphers but form the foundational framework of modern cryptographic schemes.

2.3.1 Mathematical Expression of Correctness

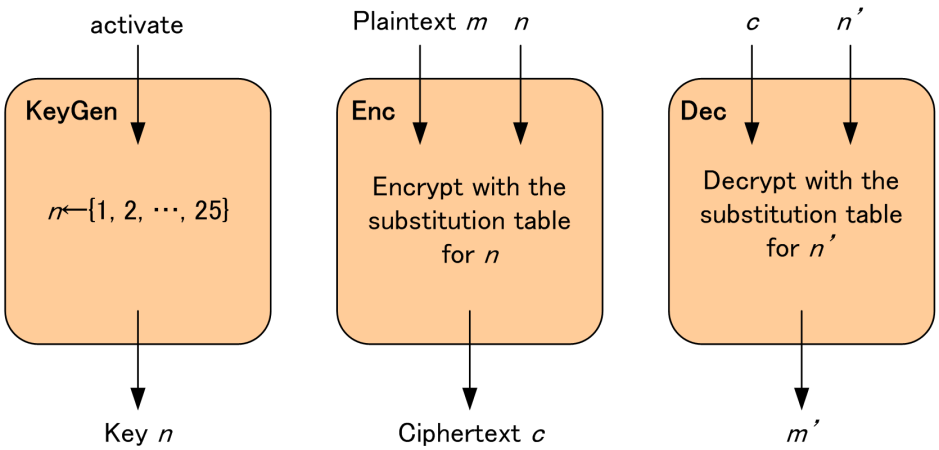
Correctness refers to the property that decryption accurately recovers the original plaintext from the ciphertext. An encryption scheme must guarantee this correctness. If the original plaintext cannot be recovered even when using the correct ciphertext and key, the encryption fails to fulfill its fundamental purpose of secure communication.

As mentioned earlier, the shift cipher is defined by three algorithms. Using these algorithms to express correctness means that for any message m and key, the following equation must hold:

$$m = Dec(key, Enc(key, m))$$

2.3.2 Modify the Process to Use a Substitution Table

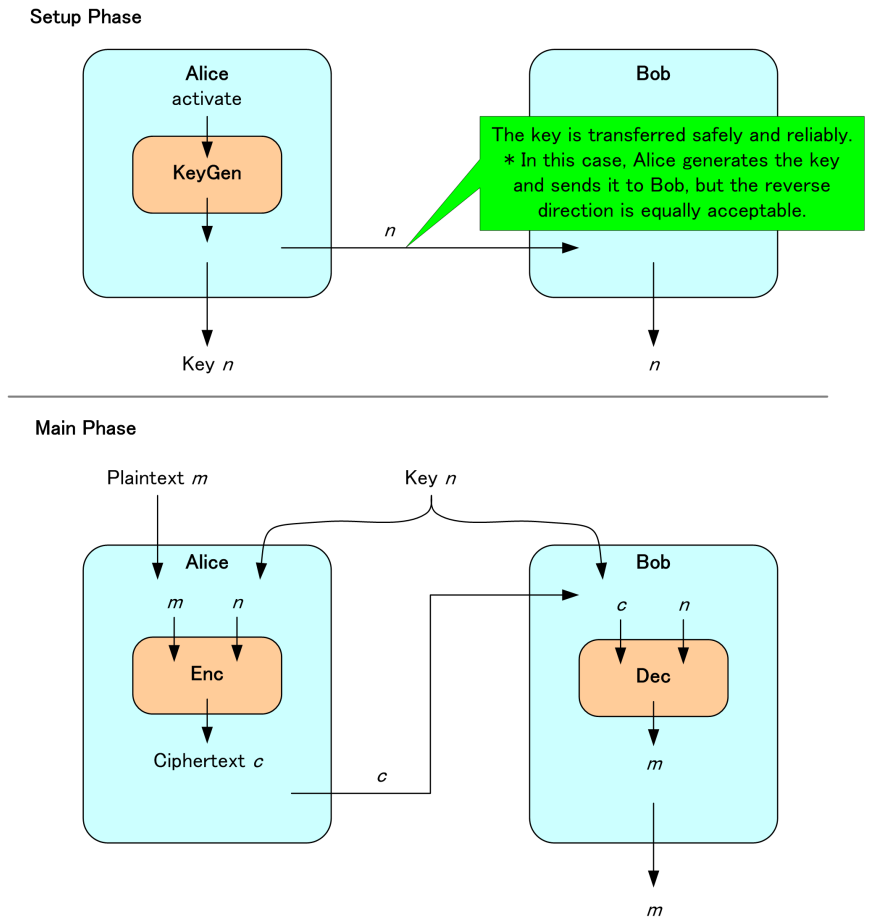
The modified algorithm using a substitution table is shown in Figure 2.5.



▲ Figure 2.5 Shift Cipher Algorithm <Substitution Table Version>

2.3.3 How to Call Each Algorithm

Figure 2.6 shows the algorithm execution sequence. Alice and Bob call **KeyGen** in the preparation stage, then **Enc** and **Dec** in the main stage.



▲ Figure 2.6 Calling the algorithm by Alice and Bob

2.3.4 Focus on Input/Output Before Implementation Details

Rather than diving into implementation details, first understand each algorithm's inputs and outputs. All encryption algorithms—classical or modern—follow the same basic input/output pattern:

- **KeyGen** outputs a key (denoted n)
- **Enc** inputs plaintext m and key n, outputs ciphertext c
- **Dec** inputs ciphertext c and key n', outputs decrypted result m'
  - When the key is correct: $m' = m$
  - This property is called **correctness**

Notice how these symbols connect the algorithms:

n links **KeyGen** to **Enc**, while c links **Enc** to **Dec**. Without these shared symbols, the algorithms would be independent.

Understanding these input/output relationships first makes the implementation de-

tails much clearer. Apply this approach when learning any new cryptographic scheme.

2.4 Applying ROT13 Twice Restores the Original Text

ROT13 is a shift cipher that uses a key of $n = 13$. Since the alphabet has 26 letters and $13 = 26/2$, ROT13 has a unique property: encryption and decryption are identical operations.

The substitution table for ROT13 is shown in Figure 2.7.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M

▲ Figure 2.7 Substitution Table for ROT13

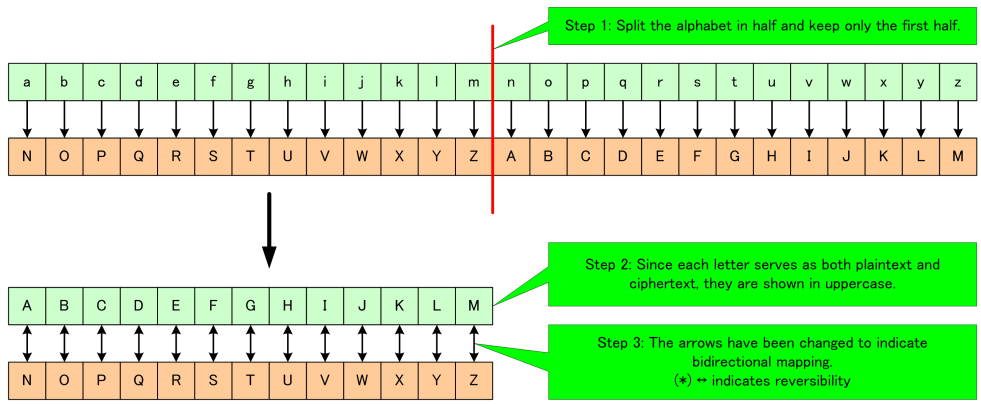
Notice that each letter maps to another letter exactly 13 positions away.

The second row is formed by swapping the first half $(a - m)$ with the second half $(n - z)$. This symmetry means that applying ROT13 twice returns the original text:

$$\text{ROT13}(\text{ROT13}(m)) = m$$

2.4.1 ROT13 Substitution Table

Figure 2.8 shows the bijective mapping where each letter pairs with another exactly 13 positions away.



▲ Figure 2.8 ROT13 Character Mapping

Notice that shifting 13 positions left equals shifting 13 positions right. Since 13 is half of 26, this creates ROT13's defining characteristic: **self-inverse property**.

Mathematically, since $13 = -13 \pmod{26}$:

- Encryption: $(x + 13) \pmod{26}$

- Decryption: $(x - 13) \pmod{26} = (x + 13) \pmod{26}$

Therefore, applying ROT13 twice returns each letter to its original position—ROT13 is its own inverse.

Figure 2.9 demonstrates this self-inverse property in action.

Plaintext	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Encrypted once	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
Encrypted twice	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

▲ Figure 2.9 ROT13 Applied Twice: Full Cycle

2.4.2 Applications of ROT13

ROT13 does not provide genuine confidentiality but is used to temporarily obscure text:

- To hide puzzle answers from immediate view
- To obscure potentially offensive content
- To conceal geocaching location hints
  - Geocaching is a GPS-based treasure-hunting game
  - `https://www.geocaching.com/play`

2.5 Shift Ciphers Are a Type of Substitution Cipher

The key in a shift cipher specifies the shift distance. This creates a one-to-one character mapping—effectively a substitution table. Since shift ciphers work by substituting each letter with another, they are a special case of substitution ciphers.

2.5.1 Shift Cipher Keys and Their Representations

Earlier, we established that shift cipher key = shift amount = substitution table[*2].

Shift values range from 0 to 25. Why exactly 26 values? Because the alphabet has 26 letters:

- Shifting by 26 returns to the original position
- Shifting by 27 equals shifting by 1

We can represent each shift value with a letter, creating a **key character** system (Table 2.1)[*3].

[*2] The equals sign '=' indicates conceptual equivalence, not numerical equality.

[*3] Alternative mappings exist (e.g., 'A'=1, 'B'=2, ..., 'Z'=26), but 'A'=0 ensures no change when shift is zero.

▼ Table 2.1 Shift Values and Key Characters

Shift Value	Key Character
0	'A'
1	'B'
2	'C'
3	'D'
…	…
25	'Z'

For example, ROT3 uses the key character 'D', corresponding to a shift value of 3. In the substitution table, this means that 'A' in plaintext maps to 'D' in ciphertext, and so on. The key character defines how the alphabet is shifted, which in turn determines the substitution mapping.

This principle applies to all shift ciphers: once you know the key character, the entire substitution pattern is fixed.

Therefore:

$$\text{Key} \Leftrightarrow \text{Shift Value} \Leftrightarrow \text{Key Character}$$

$$\text{Substitution Table} \Leftrightarrow \text{Letter-to-Letter Mapping}$$

2.6 Encrypting English Sentences with a Shift Cipher

Encrypting individual English words using a shift cipher works without problems.

However, encrypting complete English sentences introduces complications. This is because English sentences contain non-alphabetic characters (characters other than letters of the alphabet).

These non-alphabetic characters include spaces, commas, periods, exclamation marks, question marks, and other punctuation. Since the Caesar cipher handles only alphabetic characters, it excludes all non-alphabetic characters from the plaintext during encryption. As a result, the output ciphertext consists solely of letters, without the excluded characters.

To properly restore the original sentence after decryption, you must carefully reinsert the missing spaces and punctuation.

2.7 Hands-on Practice with Caesar Cipher

Let's experience the encryption process using the Caesar cipher. We'll use key letter 'D', which corresponds to a shift of 3.

2.7.1 Encryption Example

As an example, we'll use the following line from the beginning of Edgar Allan Poe's poem *"The Sleeper"*[*4][*5].

> At midnight, in the month of June, I stand beneath the mystic moon.

To simplify our discussion, we'll follow these conditions:

- Keep all spaces and punctuation; do **not** remove non-alphabetic characters
- Write all ciphertext in **uppercase**

Now, let's walk through the encryption steps:

Step 1: Convert all plaintext to lowercase.

This gives us (before capitalization):

"at midnight, in the month of june, i stand beneath the mystic moon."

Step 2: Apply the substitution table for key letter 'D'. We substitute each letter according to the plaintext row of the table.

The resulting ciphertext is:

"DW PLGQLJKW, LQ WKH PRQWK RI MXQH, L VWDQG EHQHDWK WKH PBVWLF PRRQ."

2.7.2 Decryption Example

Now let's go through the decryption process:

Step 1: Decrypt using the substitution table for key letter 'D'. For decryption, we use the ciphertext row (second row) of the table. This gives us:

"at midnight, in the month of june, i stand beneath the mystic moon."

Step 2: Verify the message makes sense and apply proper capitalization. In the final step, we apply proper capitalization as follows:

- The first letter of the sentence
- The pronoun 'I'
- Proper nouns (months, countries, names)

The final result is:

"At midnight, in the month of June, I stand beneath the mystic moon."

This result perfectly matches the original plaintext.

[*4] As poetry allows various interpretations, one translation might be: "On a certain midnight in June, under a mysterious moon, I stood."

[*5] Poe's works are available at PoeStories.com (`https://poestories.com/read/ovalportrait`). "The Sleeper" can be found at `https://poestories.com/read/sleeper`.

Shift Ciphers and Mathematics

This section explains how to express shift cipher operations mathematically. To begin with, we assign each letter a numerical value from 0 to 25:

$$A = 0, B = 1, ..., Z = 25$$

Let T be the plaintext character, K the key character, and C the ciphertext character. The shift cipher operation is then:

$$T + K \equiv C \pmod{26}$$

For encryption, C is unknown while T and K are known. Since only one variable is unknown, we can solve for C.

For decryption, T is unknown while C and K are known. We solve for T similarly.

Since substitution ciphers (including shift ciphers) replace letters systematically, we can view the substitution table as a bijective mapping from the alphabet set to itself.

This mathematical perspective clarifies the fundamental mechanisms of classical ciphers and often directly informs their implementation in code.

Chapter 3

Using Computers to Explore Shift Ciphers

We've explained how shift ciphers work and practiced manual encryption and decryption. Before advancing to breaking Caesar and shift ciphers, let's gain more hands-on experience.

In this chapter, we'll use computers to explore shift ciphers through commands, programs, and tools.

3.1 Using the `tr` Command for Shift Ciphers

You can implement shift ciphers easily by piping `echo` and `tr` commands.

The `echo` command outputs text to standard output, while `tr` (translate) command transforms characters from standard input and sends the result to standard output. By combining these commands with a pipe (|), we can create functional shift ciphers in a single line.

3.1.1 Testing Caesar Cipher with `tr` Command

Using lowercase for plaintext and uppercase for ciphertext, we can encrypt "hello world" as follows:

```
ipusiron@parrot:~$ echo "hello world" | tr a-z D-ZA-C
KHOOR ZRUOG  # Encrypted result
```

To decrypt the ciphertext:

```
ipusiron@parrot:~$ echo "KHOOR ZRUOG" | tr D-ZA-C a-z
hello world  # Decrypted result
```

The output confirms perfect recovery of the original plaintext "hello world".

In classical cryptography, plaintext is conventionally written in lowercase and ciphertext in uppercase. The above `tr` command follows this convention.

To handle both cases, use:

```
ipusiron@parrot:~$ echo "hello world" | tr 'a-zA-Z' 'd-za-cD-ZA-C'
KHOOR ZRUOG  # Encrypted result
ipusiron@parrot:~$ echo "KHOOR ZRUOG" | tr 'd-za-cD-ZA-C' 'a-zA-Z'
hello world  # Decrypted result
```

3.1.2 Testing ROT13

We can implement ROT13 using the same command structure. Let's verify ROT13's self-inverse property using "i love you." as our plaintext.

Step 1: Standard encryption and decryption.

```
ipusiron@parrot:~$ echo "i love you." | tr a-z N-ZA-M
V YBIR LBH.  # Encrypted text
ipusiron@parrot:~$ echo "V YBIR LBH." | tr N-ZA-M a-z
i love you.  # Decrypted text
```

Step 2: Apply the same ROT13 command twice. To clearly show we're using identical commands, we'll use the case-inclusive `tr` pattern.

However, to make it clear that the same command is being used, we will use a version of the `tr` command that supports both lowercase and uppercase letters in its arguments.

```
ipusiron@parrot:~$ echo "i love you." | tr 'a-zA-Z' 'n-za-mN-ZA-M'
V YBIR LBH.  # First ROT13
ipusiron@parrot:~$ echo "V YBIR LBH." | tr 'a-zA-Z' 'n-za-mN-ZA-M'
i love you.  # Second ROT13 returns to original
```

Note: In Step 2, we apply the same transformation twice. Since ROT13 shifts by exactly half the alphabet (13 positions), it acts as its own inverse.

3.2 Testing ROT13 with the `nkf` Command

The `nkf` command is a filter tool used primarily on UNIX systems to convert text encoding and newline formats.

e.g., It was originally developed for handling Japanese encodings such as Shift_JIS and EUC-JP.

Step 1: Prepare a file named `plain.txt` containing the plaintext. Here, we assume the plaintext consists of lowercase letters.

```
ipusiron@parrot:~$ cat plain.txt
i love you.
```

Step 2: By adding the `-r` option to the `nkf` command, you can perform ROT13 encryption.

```
ipusiron@parrot:~$ nkf -r plain.txt
v ybir lbh.
```

The output is the same as the result of ROT13 encryption using the `tr` command. However, since the text is still in lowercase, let's pass it to the `tr` command to convert it to uppercase.

```
ipusiron@parrot:~$ nkf -r plain.txt | tr '[:lower:]' '[:upper:]'
V YBIR LBH.
```

Now we have obtained the ciphertext.

For the next step where we'll try decryption, let's save the ciphertext message to a file called `cipher.txt`.

```
ipusiron@parrot:~$ nkf -r plain.txt | tr '[:lower:]' '[:upper:]' > cipher.tx →
t
```

Step 3: Since ROT13 returns to the original text when applied twice, decryption is simply another encryption. We'll use the `-r` option of the `nkf` command once more.

```
ipusiron@parrot:~$ nkf -r cipher.txt
I LOVE YOU.
ipusiron@parrot:~$ nkf -r cipher.txt | tr '[:upper:]' '[:lower:]'
i love you.
```

We have successfully confirmed that the original plaintext message has been restored.

3.3 Testing ROT13 with One-Liner Commands

One-liner commands execute complete operations in a single line. Python's `codecs` module includes built-in ROT13 support through its encoding functions.

We'll use `codecs.encode()` for encryption and `codecs.decode()` for decryption, with "i love you." as our test string.

Step 1: Encrypt using ROT13:

```
ipusiron@parrot:~$ python3 -c "import codecs; print(codecs.encode(\"i love y →
ou.\", \"rot-13\"))"
v ybir lbh.  # Encrypted text
```

Step 2: Decrypt using ROT13: Here we use `decode()` to show that Python treats ROT13 as a reversible encoding.

```
ipusiron@parrot:~$ python3 -c "import codecs; print(codecs.decode(\"v ybir l →
bh.\", \"rot-13\"))"
i love you.  # Decrypted text
```

3.4 Testing Caesar Cipher in Your Browser

I've created an interactive Caesar Cipher Wheel Tool for experimenting with shift ciphers:

- GitHub repository
  - `https://github.com/ipusiron/caesar-cipher-wheel`
- Live demo
  - `https://ipusiron.github.io/caesar-cipher-wheel/`

Set Shift to 1 for ROT1, or 3 for ROT3 (traditional Caesar cipher). Figure 3.1 demonstrates the tool's interface.

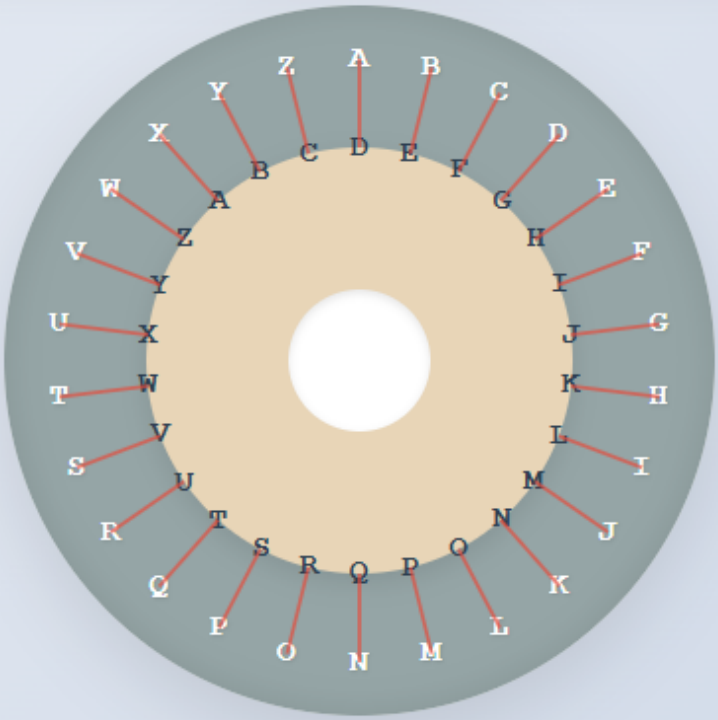
Caesar Cipher Wheel Tool

hello world.

Shift: 3

Encrypt Decrypt

Show correspondence lines:



Result:

KHOOR ZRUOG.

GitHubリポジトリはこちら（ipusiron/caesar-cipher-wheel）

3.5 Testing Caesar Cipher with Python

Caesar and shift ciphers are fundamental encryption methods with implementations readily available online.

This section uses code adapted from *Cracking Codes with Python* by Al Sweigart (Japanese translation: 『Python でいかにして暗号を破るか』, Socym). The book covers shift ciphers across two chapters:

- Chapter 5: Encryption and decryption implementation
- Chapter 6: Brute-force attack techniques

- Resources:
  - Original book: `https://nostarch.com/crackingcodes`
  - Japanese guide: `https://akademeia.info/?page_id=94`
  - Japanese publisher: `https://www.socym.co.jp/book/1216`

Our implementation is a modified version of the book's code (BSD licensed) that gracefully handles non-alphabetic characters.

- Usage:
  - `message`: string to encrypt/decrypt
  - `key`: shift value (0-25)
  - `mode`: `"encrypt"` or `"decrypt"`

▼ List 3.1 caesarCipher.py

```
 1: # https://www.nostarch.com/crackingcodes (BSD Licensed)
 2: # The string to be encrypted/decrypted:
 3: message = 'This is my secret message.'
 4:
 5: # The encryption/decryption key:
 6: key = 13
 7:
 8: # Whether the program encrypts or decrypts:
 9: mode = 'encrypt' # Set to either 'encrypt' or 'decrypt'.
10:
11: # Every possible symbol that can be encrypted:
12: SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' \
13:         'abcdefghijklmnopqrstuvwxyz' \
14:         '1234567890 !?.'
15:
16: def caesar(message, key):
17:   # Stores the encrypted/decrypted form of the message:
18:   translated = ""
19:   for symbol in message:
20:     # Note: Only symbols in the `SYMBOLS` string can be encr→
    ypted/decrypted.
21:     if symbol in SYMBOLS:
22:       symbolIndex = SYMBOLS.find(symbol)
23:
24:       # Perform encryption/decryption:
25:       if mode == 'encrypt':
26:         translatedIndex = symbolIndex + key
27:       elif mode == 'decrypt':
```

```
28:         translatedIndex = symbolIndex - key
29:
30:       # Handle wrap-around, if needed:
31:       if translatedIndex >= len(SYMBOLS):
32:         translatedIndex = translatedIndex - len(SYMBOLS)
33:       elif translatedIndex < 0:
34:         translatedIndex = translatedIndex + len(SYMBOLS)
35:
36:       translated = translated + SYMBOLS[translatedIndex]
37:     else:
38:       # Append the symbol without encrypting/decrypting:
39:       translated = translated + symbol
40:   return translated
41:
42: def main():
43:   # Output the translated string:
44:   print(caesar(message, key))
45:
46: if __name__ == '__main__':
47:   main()
```

Try running the program in your local Python environment or online:

- paiza.IO (no registration)
  - `https://paiza.io/en/languages/python3`
- Google Colab (Google account required)
  - `https://colab.research.google.com/`
- CodeChef (no registration)
  - `https://www.codechef.com/ide`

When you execute List 3.1 as is, it outputs: `"guv6Jv6Jz!J6rp5r7Jzr66ntrM"`.

Notice that spaces, punctuation, and case are all preserved during encryption.

To decrypt, set `message` to the ciphertext and `mode` to `"decrypt"`. The output will be the original message: `"This is my secret message."`.

3.5.1 Extending ROT13 to Match the Character Set

In the List 3.1 program, the key was set to 13. Try verifying whether encrypting twice returns the original text, as with ROT13.

"This is my secret message."
-> "guv6Jv6Jz!J6rp5r7Jzr66ntrM" (first encryption)
-> "t89EW9EW?KWE53D5FW?5EE175Z" (second encryption)

In this experiment, applying encryption twice did **not** return the original text.

The reason is that the key doesn't match the character set size. Traditional ROT13 uses a 26-letter alphabet, with a key of 13 (half of 26) ensuring that double encryption restores the original message.

However, List 3.1 recognizes 66 characters:

- 26 uppercase letters
- 26 lowercase letters
- 10 digits

- 4 symbols

Therefore, we need a key of 33 (half of 66) for the self-inverse property.

Try this yourself:

"This is my secret message."
-> "1ABLdBLdFRdL!0K!MdF!LL8.!g" (first encryption)
-> "This is my secret message." (second encryption)

3.6 Testing the ROT13 Encoder

I've created an interactive ROT13 Encoder web tool:

- GitHub repository
  - `https://github.com/ipusiron/rot13-encoder`
- Live demo
  - `https://ipusiron.github.io/rot13-encoder/`

Figure 3.2 shows the algorithm execution sequence.

▲ Figure 3.2 Testing the ROT13 Encoder

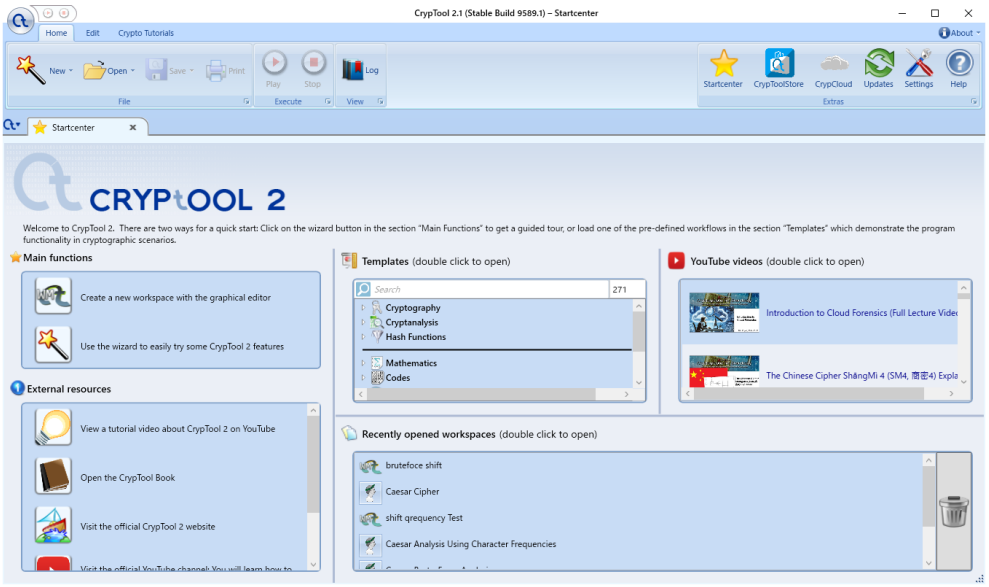
3.7 Working with Shift Ciphers Using CrypTool 2

CrypTool 2 is a comprehensive GUI-based tool that supports various cryptographic algorithms. Its intuitive interface makes cryptographic operations accessible without programming knowledge. With CrypTool 2, you can focus on encryption, decryption, and cryptanalysis rather than implementation details.

The CrypTool Portal (`https://www.cryptool.org/en/`) offers four cryptographic learning platforms:

- CrypTool-Online (CTO)
  - Browser-based platform
  - Individual tools for various ciphers
  - Ideal for quick experiments
- CrypTool 1 (CT1)
  - Original version from 1998
  - Windows only
  - Started as corporate security training software, now open source
- CrypTool 2 (CT2)
  - Modern Windows GUI version
  - Visual workflow designer for cryptographic operations
  - Extensive plugin architecture
- JCrypTool (JCT)
  - Java-based, cross-platform (Windows, macOS, Linux)
  - Different UI approach from CrypTool 2
  - Version 1.0.0 released November 2020
  - Over 100 plugins available

For this book's demonstrations, we'll use CrypTool 2.1 on Windows.
Figure 3.3 shows the CrypTool 2.1 interface upon startup.



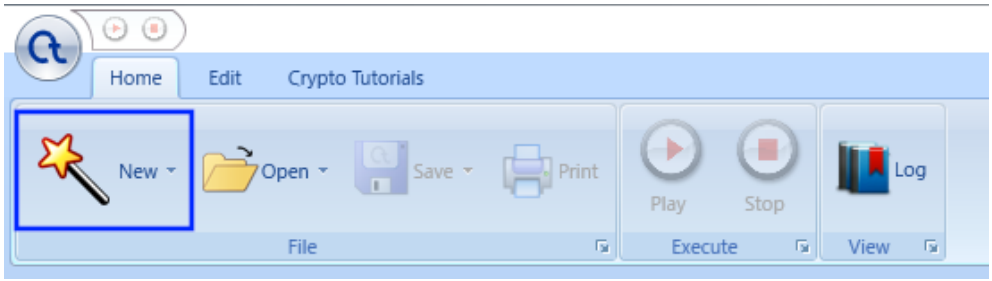
▲ Figure 3.3 CrypTool 2.1 startup screen

3.7.1 Using the Shift Cipher Wizard

CrypTool 2 includes a wizard for experimenting with shift ciphers. This section explains how to launch and use it.

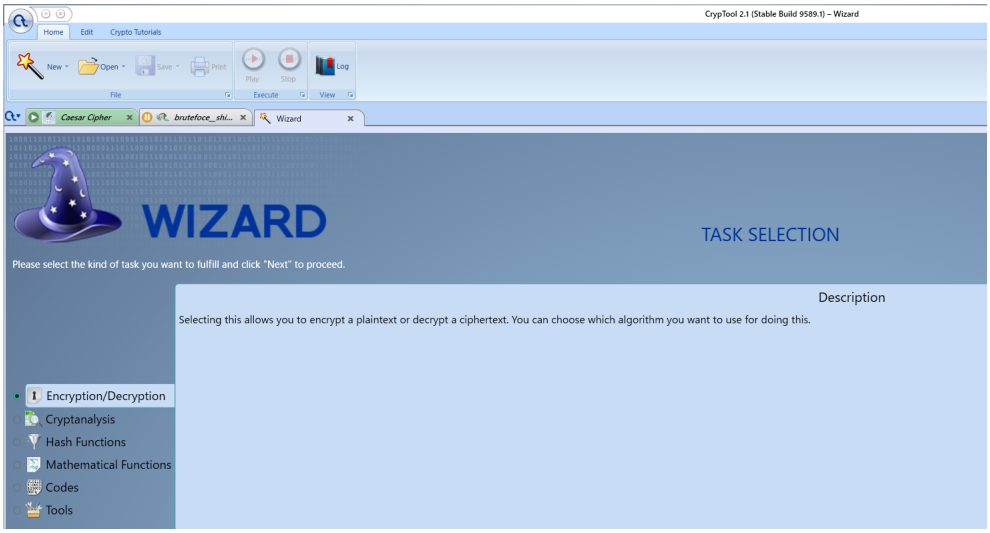
Note: CrypTool 2 uses "Caesar Cipher" and "Shift Cipher" interchangeably.

Step 1: Click the [New] icon on the CrypTool 2 home screen, or click the dropdown arrow next to it and select "Wizard". Refer to Figure 3.2.



▲ Figure 3.4 Launching the wizard from the home screen

Step 2: The Wizard tab opens. In the "TASK SELECTION" screen, select "Encryption/Decryption" and click [Next]. Refer to Figure 3.5.



▲ Figure 3.5 "TASK SELECTION" screen

Step 3: In the "AGE SELECTION" screen, select "Classic Encryption/Decryption" and click [Next].

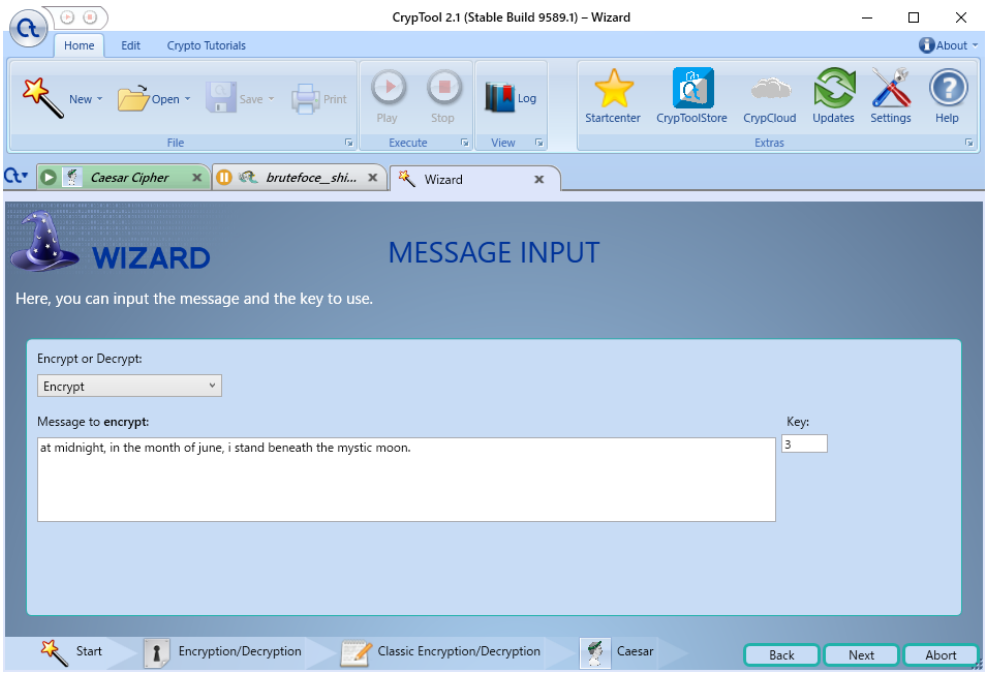
Step 4: In the "ALGORITHM SELECTION" screen, select "Caesar" and click [Next].

Step 5: The "MESSAGE INPUT" screen appears. Select "Encrypt" to encrypt or "Decrypt" to decrypt. Since we're encrypting, select "Encrypt".

Enter your message in the text box. Here, we use the text from the previous chapter: "At midnight, in the month of June, I stand beneath the mystic moon."

Enter the shift value in the Key field on the right. We'll keep the default value of 3.

Once all settings are complete, click [Next]. Refer to Figure 3.6.

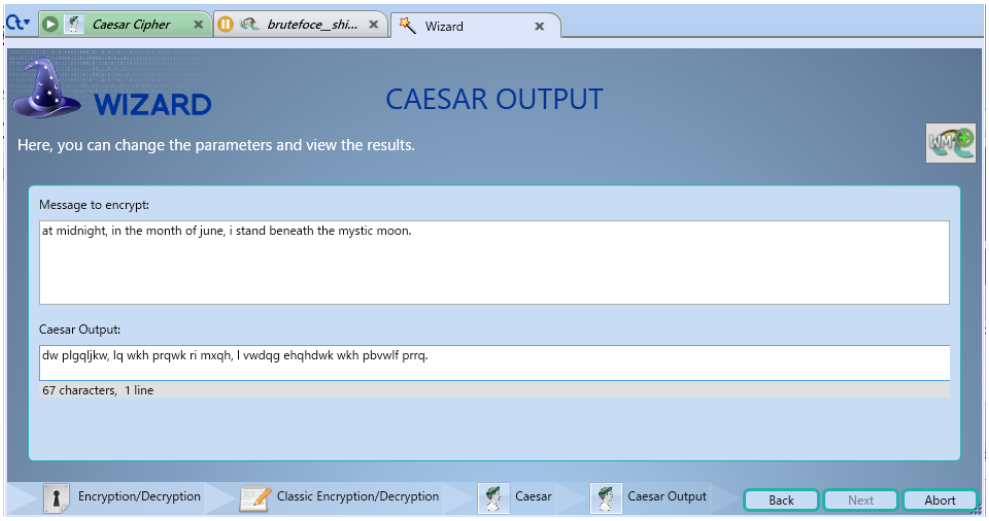


▲ Figure 3.6 "MESSAGE INPUT" screen

Step 6: The CAESAR OUTPUT screen displays the original message and its encrypted form. The resulting ciphertext is:

`"dw plgqljkw, lq wkh prqwk ri mxqh, l vwdqg ehqhdwk wkh pbvwlf prrq."`

Figure 3.6 shows the "CAESAR OUTPUT" screen.



▲ Figure 3.7 "CAESAR OUTPUT" screen

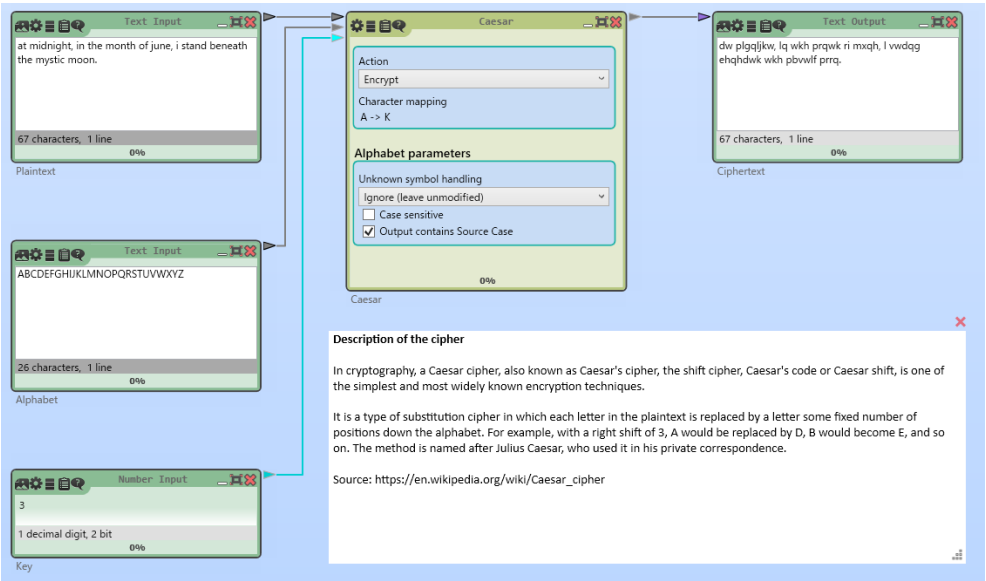
To encrypt another message, click [Back] to return to MESSAGE INPUT. To exit, click [Abort].

When finished, close the wizard tab.

3.7.2 Creating a Shift Cipher Project

Let's create a workspace project for shift cipher encryption. Since the decryption project uses essentially the same structure, we'll focus on encryption here. For decryption details, see "4.2.4 Breaking Shift Ciphers with Frequency Analysis in CrypTool 2".

Figure 3.8 shows the completed shift cipher project looks.



▲ Figure 3.8 Shift cipher encryption project

The Caesar component, located in the center, plays the central role in the project. Although named "Caesar," it handles shift cipher processing.

When building your own project, first place the Caesar component on the workspace. Find the Caesar component under "Classic Ciphers" in the left component list and drag it to the workspace. Then connect various components to the inputs and outputs to complete the project.

The Caesar component accepts these connectors:

- Input
  - **Text input**: Plaintext
  - **External alphabet input**: String containing the alphabet to use. If unconnected, uses internal alphabet
  - **Shift value**: Shift number (key) as integer
- Output
  - **Text output**: Ciphertext

Connect the Text Input, Text Output, and Number Input components, found under Tools > Data input/output.

Caesar component settings:

- Action: Select "Encrypt" or "Decrypt"
- Alphabet: Specifies alphabet when External alphabet input is unconnected
- Alphabet parameters: Configure non-alphabetic character handling
  - Unknown symbol handling: "Ignore" (output as-is), "Remove" (delete), or "Replace with '?'"
  - Case sensitive: Check to distinguish uppercase/lowercase
  - Output contains source case: Check to preserve original casing

After placing components and making connections, click the Play icon in the Execute toolbar section to start. Results appear in the Text Output component.

Click Stop to halt the process.

Encrypting "at midnight, in the month of june, i stand beneath the mystic moon." with key 3 produces:
"dw plgqljkw, lq wkh prqwk ri mxqh, l vwdqg ehqhdwk wkh pbvwlf prrq."

This matches the wizard output, confirming proper implementation.

Chapter 4

Breaking Caesar and Shift Ciphers

This chapter explains in detail how to break shift cipher ciphertexts without knowing the key. Here we address the central challenge indicated by this book's title.

We begin with manual cryptanalysis to develop a thorough understanding of the breaking process. We then implement computer-based methods for efficient and systematic attacks.

4.1 Breaking Shift Ciphers with Brute-Force Attacks

With only 26 possible keys (one for each letter), shift ciphers are vulnerable to brute-force attacks.

4.1.1 Manual Brute-Force Attack

Let's break the shift cipher ciphertext "KZDVWCZVJCZBURERIIFN".

Step 1: Analyze the ciphertext characteristics. The ciphertext is a continuous string without spaces or punctuation. This suggests word boundaries have been removed during encryption.

Step 2: Execute brute-force decryption. Test all 26 possible shift values (0-25). For shift value n, decrypt by shifting left n positions (or right 26-n positions).

See Table 4.1 for all results.

▼ Table 4.1 Decryption Results for All Shift Values

Shift Value n	Key Character	Decrypted Result
0	'A'	kzdvwczvjczbvreriifn
1	'B'	jycuvbyuibyauqdqhhem
2	'C'	ixbtuaxthaxztpcpggdl
3	'D'	hwastzwsgzwysoboffck
4	'E'	gvzrsyvrfyvxrnaneebj
5	'F'	fuyqrxuqexuwqmzmddai
6	'G'	etxpqwtpdwtvplylcczh
7	'H'	dswopvsocvsuokxkbbyg
8	'I'	crvnournburtnjwjaaxf
9	'J'	bqumntqmatqsmivizzwe
10	'K'	aptlmsplzsprlhuhyyvd
11	'L'	zosklrokyroqkgtgxxuc
12	'M'	ynrjkqnjxqnpjfsfwwtb
13	'N'	xmqijpmiwpmoierevvsa
14	'O'	wlphiolhvolnhdqduurz
15	'P'	vkoghnkgunkmgcpcttqy
16	'Q'	ujnfgmjftmjlfbobsspx
17	'R'	timeflieslikeanarrow
18	'S'	shldekhdrkhjdzmzqqnv
19	'T'	rgkcdjgcqjgicylyppmu
20	'U'	qfjbcifbpifhbxkxoolt
21	'V'	peiabheaohegawjwnnks
22	'W'	odhzagdzngdfzvivmmjr
23	'X'	ncgyzfcymfceyuhulliq
24	'Y'	mbfxyebxlebdxtgtkkhp
25	'Z'	laewxdawkdacwsfsjjgo

To identify the correct plaintext, apply these three approaches:

- **Approach 1: Common word detection**
  - Search for frequent English words ("the", "and", "that")
  - Their presence indicates a likely correct shift value
- **Approach 2: Pattern elimination**
  - Exclude results with unnatural consonant clusters (e.g., "kzdv")
  - Remember: word boundaries may be removed ("goodbye" from "good bye")
- **Approach 3: Rare letter analysis**
  - Check context around uncommon letters ('z', 'q')
  - Verify if they form valid English words

Applying these approaches reveals that shift value 17 produces "timeflieslikeanarrow". Adding spaces and proper capitalization yields: "Time flies like an arrow". This English proverb confirms we've found the correct plaintext with key shift n = 17.

4.2 Breaking Ciphers Through Frequency Analysis

Frequency analysis identifies plaintext by studying character distribution in the ciphertext. This technique devastates classical ciphers, particularly monoalphabetic substitutions like shift ciphers. Since shift ciphers preserve letter frequencies (only shifting positions), frequency analysis remains highly effective.

4.2.1 Frequency Analysis Against Monoalphabetic Substitution Ciphers

In monoalphabetic substitution ciphers, each plaintext character maps to a fixed ciphertext character. This preserves character frequencies—the frequency distribution remains identical, just with different letters. This preservation allows cryptanalysts to match ciphertext characters to plaintext characters.

Longer ciphertexts yield better results, as larger samples converge toward standard English letter frequencies.

However, texts deliberately avoiding certain letters[*1] defy standard frequency patterns. Cryptanalysts must consider such edge cases.

Letter and word frequencies vary by language, era, and text type. New words emerge while others become obsolete. Sample selection significantly affects results. Consequently, frequency tables differ across reference sources.

4.2.2 Initial Letter Patterns in English Words

Classical cipher plaintexts typically use natural language. Natural languages exhibit predictable character distributions—here we examine English.

If you have an English dictionary, close it and view the fore edge[*2]. You'll see index tabs (black bands) marking each letter section. Band width directly reflects word frequency: wider bands contain more words starting with that letter.

Letters like 's' and 'c' have wide bands, while 'x', 'y', and 'z' are so rare they often share a single tab.

When ciphertexts preserve word boundaries, initial letter frequencies become powerful cryptanalytic tools. These patterns hold true across virtually all English texts.

4.2.3 Analyzing Character Frequencies Without Word Boundaries

Without spaces to mark word boundaries, we must rely on character frequency patterns.

[*1] Ernest Vincent Wright's "Gadsby" (50,000+ words) contains no words with the letter 'e'.

[*2] The edge opposite the spine, also called the fore edge.

English exhibits predictable character distributions:

- Most common: e, t, a, o, i, n
- Least common: z, q, x, j

$$\text{Character Frequency} = \frac{\text{Occurrences}}{\text{Total Characters}}$$

This ratio reveals how often each character appears in typical English text. See Appendix 1 for detailed frequency tables.

Since frequency patterns persist regardless of spacing, this technique breaks both spaced and unspaced ciphertexts equally well.

4.2.4 Breaking Shift Ciphers with Frequency Analysis in CrypTool 2

CrypTool 2 provides the Caesar Analyzer component (found under Cryptanalysis > Specific) for frequency-based cryptanalysis. This component breaks shift ciphers by analyzing character frequencies, supporting unigrams, bigrams, trigrams, and higher n-grams.

The Caesar Analyzer operates as follows:

Step 1: Decrypt the ciphertext with all possible keys (brute force).

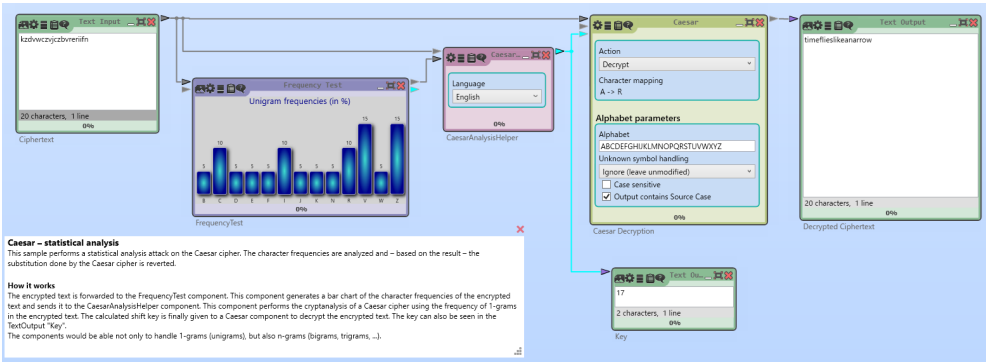
Step 2: Compare each result's character frequencies against known language patterns.

Step 3: Select the most linguistically probable plaintext and output its key.

Component connections:

- Inputs
  - **Encrypted text**: The shift cipher ciphertext
  - **Frequency List**: Language statistics from Frequency Test component
- Output
  - **Key**: The most probable decryption key

Build your project as shown in Figure 4.1:



▲ Figure 4.1 Frequency analysis project setup

Component configuration and roles:

- **Text Input** ("Ciphertext") – Top left
  - Input: Encrypted text to analyze
- **Frequency Test** ("FrequencyTest")
  - Function: Analyzes character frequencies
  - Output: Bar chart visualization
- **Caesar Analyzer** ("CaesarAnalysisHelper")
  - Function: Determines most probable shift value
- **Caesar** ("Caesar Decryption")
  - Action: Decrypt
  - Alphabet: A-Z
  - Settings: Enable "Output contains Source Case"
- **Text Output** ("Key")
  - Displays: Identified shift value
- **Text Output** ("Decrypted Ciphertext") – Far right
  - Displays: Final decrypted text

Workflow:

1. FrequencyTest analyzes unigram frequencies and generates a bar chart
2. CaesarAnalysisHelper uses these frequencies to determine the most probable key
3. Caesar component decrypts the ciphertext using the identified key
4. Both the key and decrypted text are displayed in their respective outputs

Testing with the ciphertext from "4.1.1 Manual Brute-Force Attack" correctly identified shift value 17, despite its brevity.

4.3 Identifying Shift Cipher Ciphertext

In cryptanalysis, identifying the encryption method is the crucial first step.

When analyzing unknown ciphertext, use frequency analysis to generate its character distribution. Compare this against shifted versions of standard English frequen-

cies. A match indicates a likely shift cipher; no match suggests a different encryption method.

4.3.1 Using Index of Coincidence to Identify Monoalphabetic Ciphers

The Index of Coincidence (IC) measures the probability that two randomly selected characters match. This statistical tool helps analyze texts and identify encryption types.

Typical IC values:

- English text: ~6.7%
- Random text: ~3.8%

In practical cryptanalysis, calculating and applying IC matters more than understanding its theoretical foundations.

The IC method reliably identifies monoalphabetic substitution ciphers because character substitution preserves the original IC value—the frequency pattern remains intact, just with different letters.

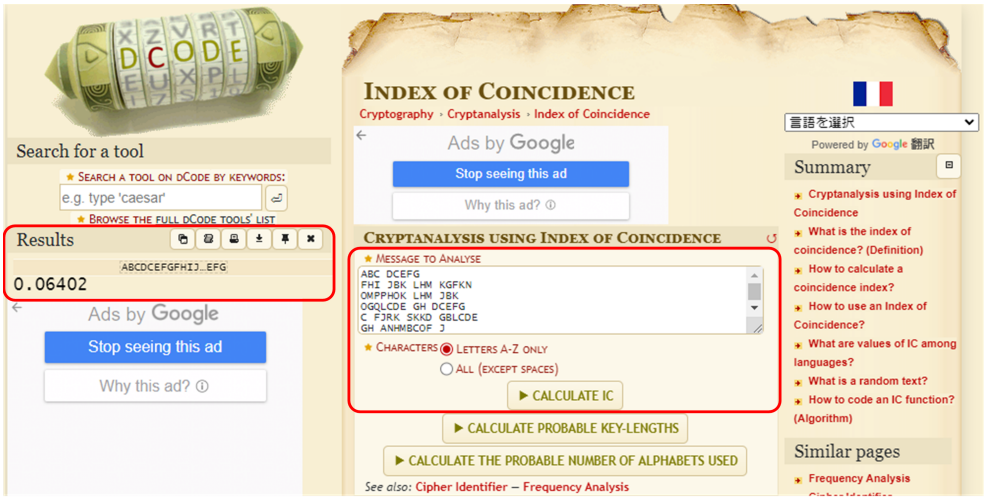
Next, we'll calculate the Index of Coincidence.

Step 1: Access the Index of Coincidence calculator at dCode.fr: `https://www.dcode.fr/index-coincidence`

Step 2: Enter the message to analyze in "MESSAGE TO ANALYSE". We'll use the provided sample ciphertext (shown here without line breaks):

> ABC DCEFGFHI JBK LHM KGFKNOMPPHOK LHM JBKOGQLCDE GH DCEFGC FJRK SKKD GBLCDEGH ANHMBCOF JNCGGNK. FJRKSKKD HMG SBKJTCDEBHJQO GFCOJAGKBDHHD AHB GFKIJEHDO. EHHQ DCEFG

Select "LETTER A-Z ONLY" under CHARACTERS. Click [CALCULATE IC] to display the results. Figure 4.2 shows the process of calculating the Index of Coincidence.



▲ Figure 4.2 Calculating the Index of Coincidence

Step 3: The IC value is 0.06402 (6.4%). This closely matches English's typical IC of 6.7%, strongly indicating a monoalphabetic substitution cipher.

4.4 Verifying Correct Plaintext Recovery

Cryptanalysis typically generates multiple plaintext candidates, but only one represents the true message. This message will be in natural language—English in our context.

The correct plaintext exhibits natural English characteristics and meaningful content, distinguishable through linguistic flow and contextual coherence.

While "4.1.1 Manual Brute-Force Attack" demonstrated manual techniques, here we leverage computational methods to evaluate candidates:

- **Approach 1: Word-Based Analysis**
  - With spaces preserved:
    - High-frequency words ("the", "and", "is") indicate probable correctness
  - Without spaces:
    - Word boundary detection becomes challenging[*3]
    - Longer word matches increase confidence
- **Approach 2: Frequency Analysis**
  - Compare character distributions against standard English frequencies
- **Approach 3: Bigram/Trigram Analysis**
  - Common pairs ("th", "er", "in") suggest natural English

[*3] Distinguishing standalone "the" from fragments like "...th" + "e..." requires context.

4.4.1 Leveraging Dictionary Files

Computers can automate Approach 1 through dictionary files.

A dictionary file is a plain text file containing one word per line. This simple format enables efficient word lookup during cryptanalysis.

English dictionary files help verify whether decrypted text forms valid English.

Common dictionary locations on UNIX-like systems:

- `/usr/share/dict`
- `/usr/dict/words`
- Password tool directories:
  - ParrotOS: `/usr/share/john/password.lst`
  - Debian: `/usr/share/opendict/dictionaries`
  - macOS: `/Library/Dictionaries`

Beyond system dictionaries, specialized wordlists serve different cryptanalytic purposes:

- **General dictionaries**: Animal names, European names, technical terms
- **Password lists**: Common choices ("12345678", "password1"), leaked databases
- **Filtered sets**: Excluding single letters ('a', 'I'), focusing on meaningful words
- **Pattern-based**: Numeric sequences (PIN analysis), alphanumeric combinations (≤ 8 characters)
- etc.

4.4.2 Performing Brute-Force Attack and Dictionary Matching with CrypTool 2

Let's experience Approach 1 using CrypTool 2.

CrypTool 2 includes a template called "Caesar Brute-Force Analysis," which allows you to experience decrypting Caesar ciphers. This template performs a brute-force attack and then applies dictionary matching to identify the correct plaintext.

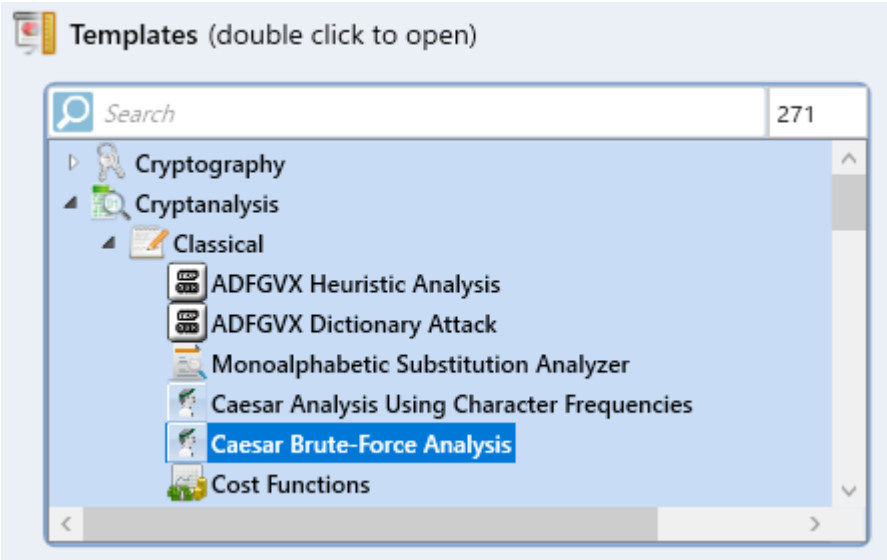
If the decrypted result contains even one word found in the dictionary, that plaintext is output. If it does not contain any dictionary words, it is discarded. This dictionary check is performed on all decryption results.

Since the tool does not include a function to automatically narrow down the optimal solution, multiple valid plaintext candidates may be output. Human judgment is required to select the final correct plaintext from the candidates.

Now, let's actually try this process in CrypTool 2. This will deepen your practical understanding of Caesar cipher decryption.

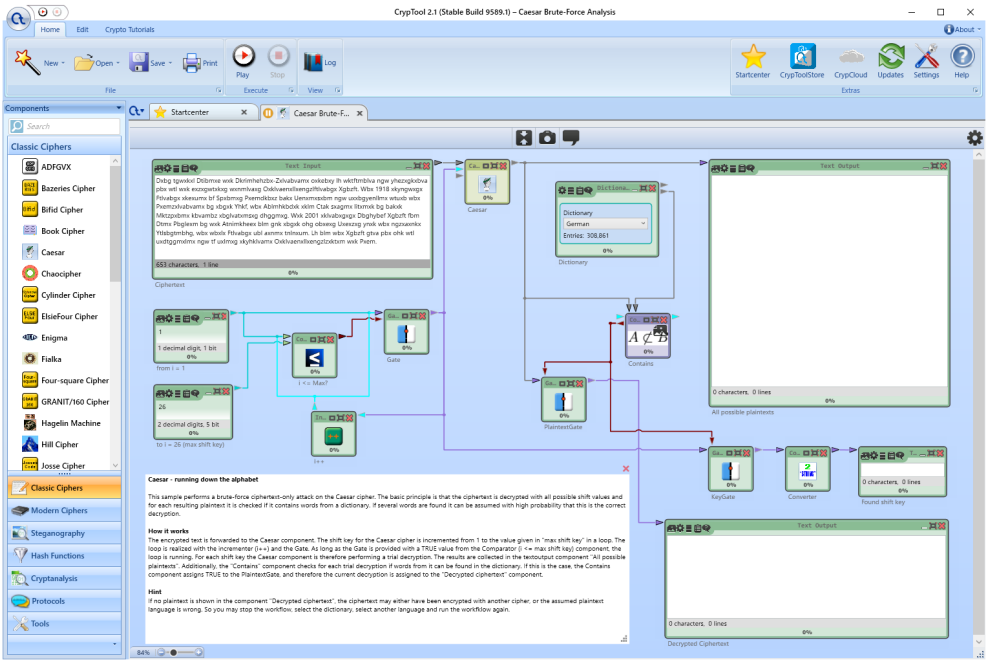
Step 1: Launch CrypTool 2. Look for the "Templates" section in the center of the main screen. Navigate to "Cryptanalysis" > "Classical" > "Caesar Brute-Force

Analysis" and double-click it. Refer to Figure 4.3.



▲ Figure 4.3 Select the "Caesar Brute-Force Analysis" template

Step 2: A project for performing a brute-force attack on a Caesar cipher will open in the workspace. See Figure 4.4.



▲ Figure 4.4 Project for performing a brute-force attack on a Caesar cipher

We will explain each component and the process flow later. For now, let's simply

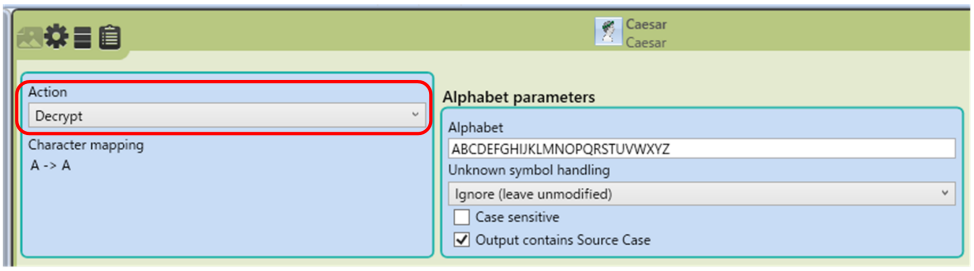
try running the project.

In the upper-left corner, you'll find the Ciphertext component, which already contains a sample encrypted message. We'll proceed using this ciphertext.

Step 3: Check the behavior of the Caesar component. Double-click the Caesar component and click the gear icon to open its settings. If the Action is set to Encrypt, change it to Decrypt[*4].

Note that component settings can only be modified when the project is in the Stop state.

Refer to Figure 4.5.



▲ Figure 4.5 Change Action to Decrypt

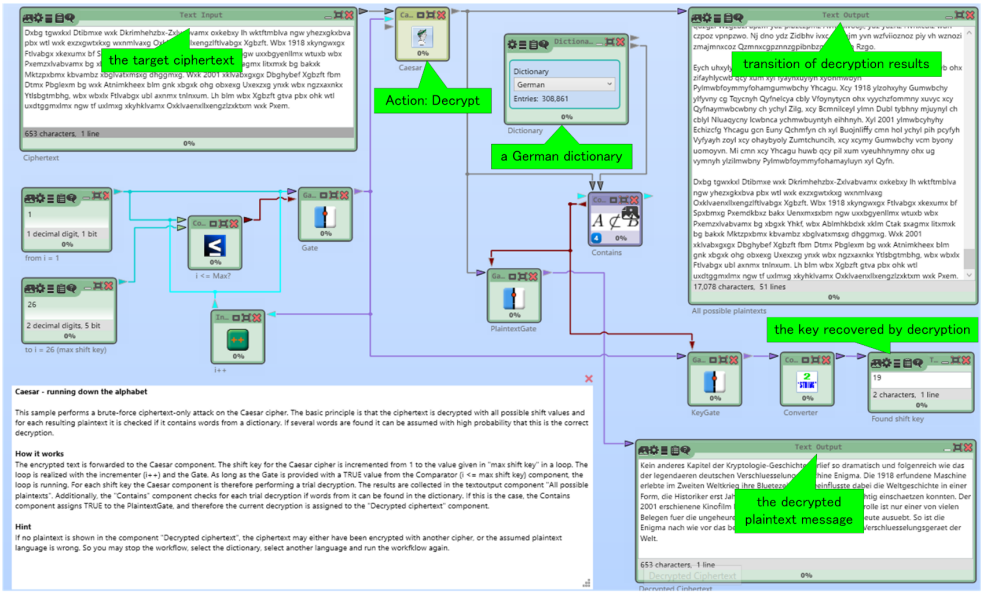
After finishing the settings, click the minimize icon in the upper-right corner of the window (the icon to the left of the ×). This will return you to the workspace.

Step 4: Now let's decrypt the sample ciphertext.

After returning to the workspace, press the [Play] button to start the decryption process. Plaintext candidates will begin to appear in the "All possible plaintexts" component in the upper right. Once the decryption reaches a certain point, the process will pause, and the final decrypted result will be displayed in the "Decrypted Ciphertext" component in the lower right.

Just above that, the "Found shift key" component displays the key (shift value) used for decryption. See Figure 4.6.

[*4] Although you can still obtain the plaintext message even if it remains set to Encrypt, the key will be incorrect. The decryption algorithm for Caesar cipher can technically be replaced with the encryption algorithm, so decryption will still succeed. However, since the key used is for encryption, you must subtract it from 26 to get the correct decryption key.



▲ Figure 4.6 Brute-force attacking the sample ciphertext

The ciphertext message (653 characters) used for decryption is as follows:

> Dxbg tgwxkxl Dtibmxe wxk Dkrimhehzbx-Zxlvabvamx oxkebxy lh wktftmblva ngw yhezxgkxbva pbx wtl wxk exzxgwtxkxg wxnmlvaxg Oxklvaenxllxengzlftlvabgx Xgbzft. Wbx 1918 xkyngwxgx Ftlvabgx xkexumx bf Spxbmxg Pxemdkbxz bakx Uenxmxsxbm ngw uxxbgyenllmx wtuxb wbx Pxemzxlvabvamx bg xbgxk Yhkf, wbx Ablmhkbdxk xklm Ctak sxagmx litxmxk bg bakxk Mktzpxbmx kbvambz xbglvatxmsxg dhggmxg. Wxk 2001 xklvabxgxgx Dbghybef Xgbzft fbm Dtmx Pbglexm bg wxk Atnimkheex blm gnk xbgxk ohg obxexg Uxexzxg ynxk wbx ngzxaxnkx Ytlsbgtmbhg, wbx wbxlx Ftlvabgx ubl axnmx tnlnxum. Lh blm wbx Xgbzft gtva pbx ohk wtl uxdtggmxlmx ngw tf uxlmxg xkyhklvamx Oxklvaenxllxengzlzxktxm wxk Pxem.

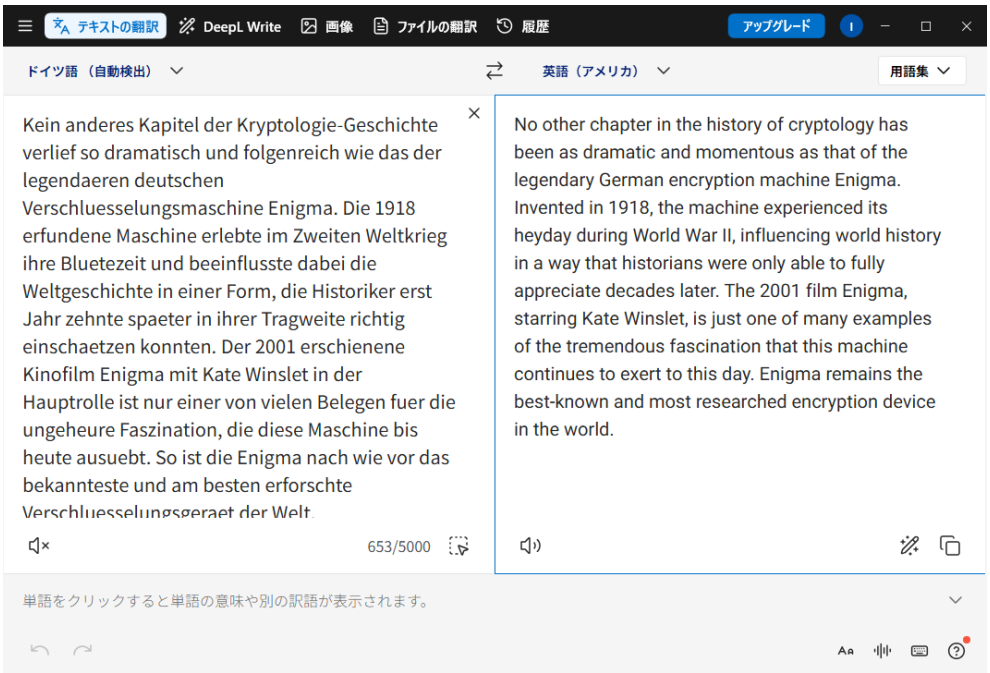
The plaintext message obtained after running the project is as follows:

> Kein anderes Kapitel der Kryptologie-Geschichte verlief so dramatisch und folgenreich wie das der legendaeren deutschen Verschluesselungsmaschine Enigma. Die 1918 erfundene Maschine erlebte im Zweiten Weltkrieg ihre Bluetezeit und beeinflusste dabei die Weltgeschichte in einer Form, die Historiker erst Jahr zehnte spaeter in ihrer Tragweite richtig einschaetzen konnten. Der 2001 erschienene Kinofilm Enigma mit Kate Winslet in der Hauptrolle ist nur einer von vielen Belegen fuer die ungeheure Faszination, die diese Maschine bis heute ausuebt. So ist die Enigma nach wie vor das bekannteste und am besten erforschte Verschluesselungsgeraet der Welt.

It is immediately obvious that the decrypted text is not in English.

The Dictionary component in the upper center of the CrypTool 2 workspace was set to "German" when the template was opened, suggesting that the original plaintext is in German.

When we entered the decrypted message into the DeepL translation tool, it was automatically detected as German. The translation result was about the Enigma machine. See Figure 4.7.



▲ Figure 4.7 Result after translating with DeepL

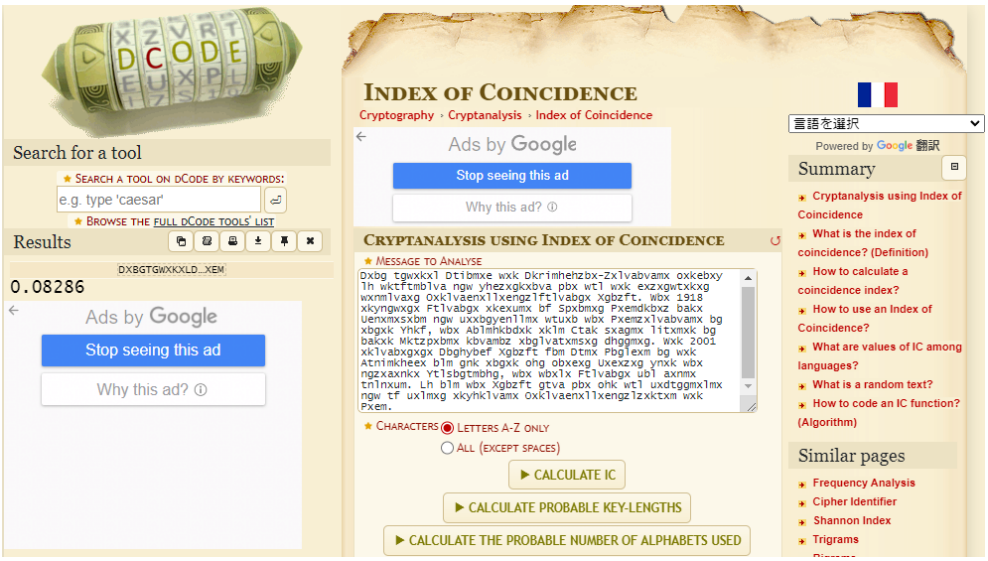
This concludes the explanation of how to use the "Caesar Brute-Force Analysis" template.

To end use of this template, press the [Stop] button to reset the system. This will clear the current session's settings and data.

Since we will continue with further explanations, please stop the project but do not close it.

Step 5: While not a direct method of decryption, checking the index of coincidence can also be useful.

For example, if you calculate the index of coincidence using `https://www.dcode.fr/index-coincidence`, you can confirm that the index of coincidence of the ciphertext is 8.2%. See Figure 4.8.



▲ Figure 4.8 Calculate the index of coincidence of the ciphertext

When you then calculate the index of coincidence of the decrypted result, it also comes out to 8.2%.

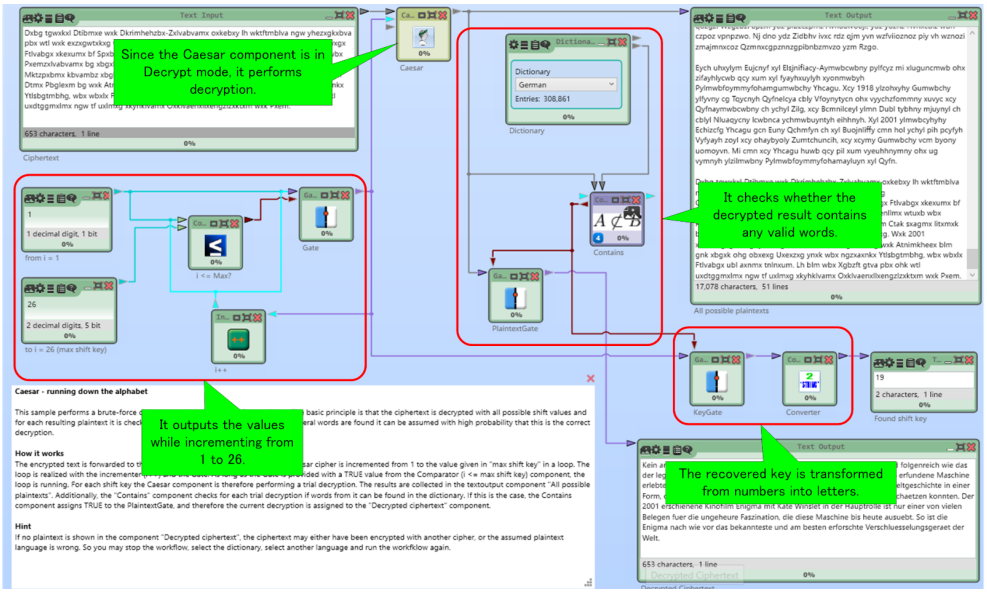
The index of coincidence of the ciphertext matches that of the decrypted result.

Since this was a Caesar cipher ciphertext, this is a natural outcome. This is because "the Caesar cipher is a type of monoalphabetic substitution cipher" and "the index of coincidence of a monoalphabetic substitution cipher's ciphertext equals that of the plaintext."

The index of coincidence for English is 5.7%, and for German it is 7.3%.

The calculated value of 8.2% is therefore closer to German than to English.

Step 6: From here on, we will explain the processing of each part of this project. See Figure 4.9.



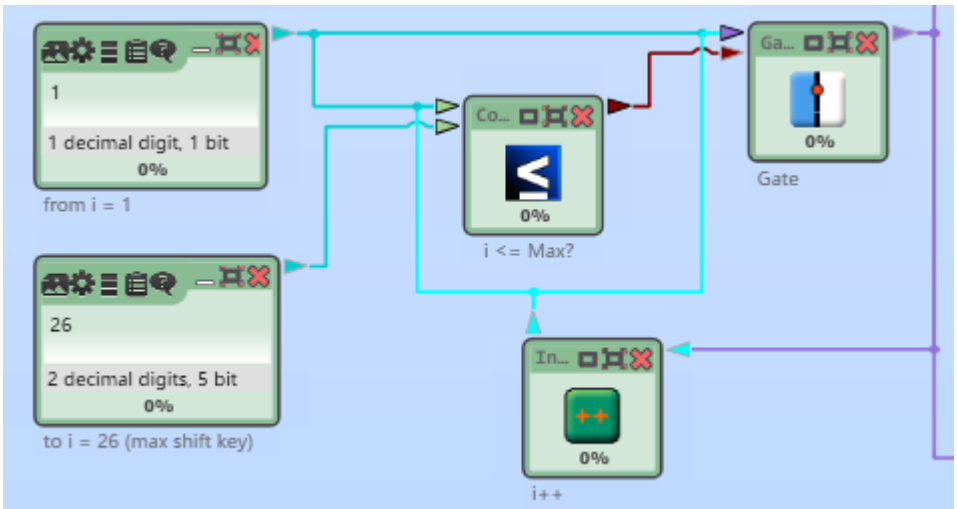
▲ Figure 4.9 Roles of each part

If the transformed message obtained during the decryption process contains real words, it will be output as the correct plaintext message.

Based on this condition, only one message met the criteria this time, so only that one was output as the decryption result.

Let's look at the left section of the project.

The main role of this part is to increment numbers from 1 to 26 and output their values. See Figure 4.10.



▲ Figure 4.10 Outputting numbers while incrementing

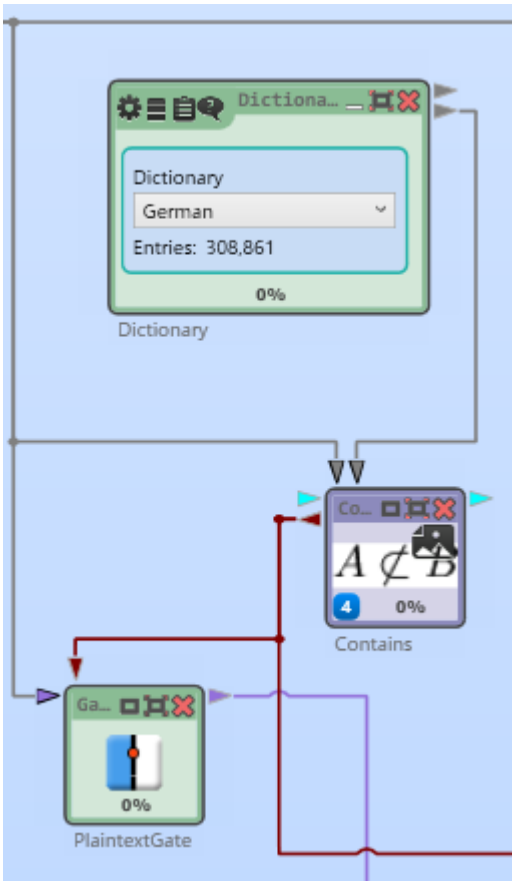
It is composed of the following components. Although it may look complicated,

you can think of it as a for loop commonly used in programming.

- Number Input component
  - Initial value of 1 and maximum value of 26 are provided.
- Inc Dec component
  - Since Mode is set to Increment and Inc/Dec value is 1, it increments by 1.
- Comparators component
  - Performs comparisons.
  - In this case, it uses the "<=" comparison operator.
- Gate component
  - Passes data when the specified Trigger is met.
  - The Trigger is set to true value.

Now focus on the center section.

The role of this section is to determine whether the decryption result contains any words. If it does, the message is output from the PlaintextGate. See Figure 4.11.



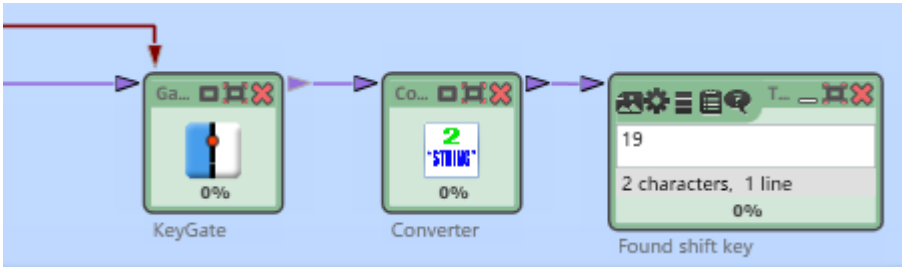
▲ Figure 4.11 Determining whether the decrypted result contains words

It is composed of the following components:

- Dictionary component
  - A German dictionary (containing over 300,000 words) is specified.
- Contains component
  - Search Type is set to Hashtable, with "Count each word only once" checked.
- Gate component
  - Passes data when the specified Trigger is met.
  - The Trigger is set to true value.

Finally, look at the right section.

The role of this part is to convert the decryption result key (numeric value) to a string for display. This is simply a process to display it in the "Found shift key" Text Output component. See Figure 4.12.

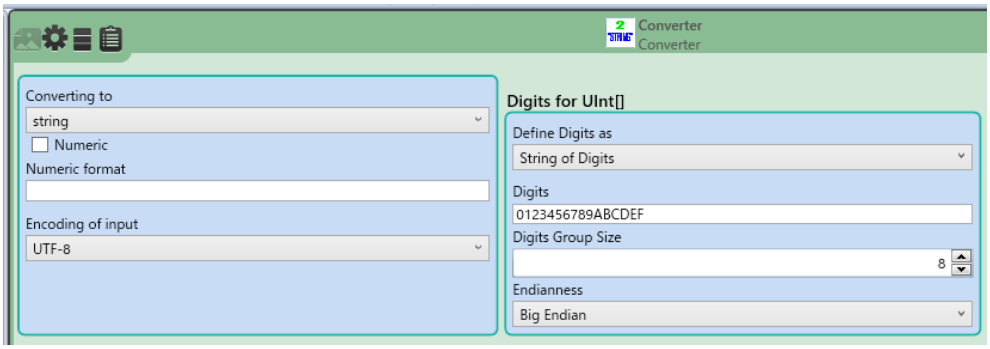


▲ Figure 4.12 Converting numbers to strings

It is composed of the following components:

- Gate component
  - Passes data when the specified Trigger is met.
  - The Trigger is set to true value.
- Converter component
  - "Converting to" is set to string.
  - For Digits for UInt[], "Define Digits as" is set to "String of Digits", Digits is set to "0123456789ABCDEF", and Endianness is set to "Big Endian".
- Text Output component
  - Named "Found shift key".
  - Displays the identified key.

Refer to Figure 4.13 for the Converter component settings.



▲ Figure 4.13 Converter component settings

This concludes the explanation of the "Caesar Brute-Force Analysis" template.

When closing the workspace tab or exiting CrypTool, if there are changes to the project, a dialog will appear asking whether to overwrite the "Caesar_ExhaustiveKeySearch.cwm" file. If you do not want to save the changes, click the [No] button.

If you want to decrypt an English ciphertext, enter the ciphertext into the Ciphertext component and change the language setting of the Dictionary component to "English". This allows you to determine the plaintext based on whether it contains English words.

Chapter 5

Improving the Shift Cipher

In the previous chapter, we successfully deciphered a shift cipher.

The fundamental problem that reduces the confidentiality of the shift cipher is the limited number of available keys. Because there are so few keys, it can be easily deciphered through brute-force attacks or statistical analysis.

With this fact in mind, this chapter introduces specific improvements to enhance the security of the shift cipher. Through these improvements, we aim to increase the confidentiality of the shift cipher and enable more secure communication.

5.1 Extending to the Vigenère Cipher

The **Vigenère table** is a matrix of 26 different Caesar-style substitution rows, each corresponding to a different key letter from A to Z. In the table, the leftmost column shows the key letters, and each row presents a shifted version of the alphabet starting from that key.

The full table is shown in Figure 5.1.

Key	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

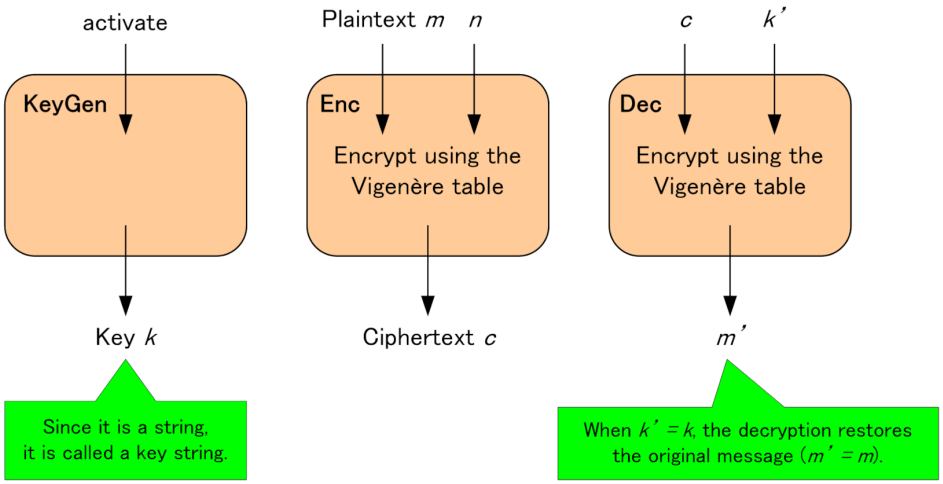
▲ Figure 5.1 The Vigenère table

In the 16th century, French cryptographer Blaise de Vigenère developed a polyalphabetic cipher using the Vigenère table. Unlike the traditional shift cipher that uses a single key letter, the Vigenère cipher uses a string of letters as the key. We will refer to this as the key string.

By using this key string, the total number of possible keys increases dramatically, making the cipher significantly more resistant to brute-force attacks.

5.1.1 Vigenère Cipher Algorithm

The algorithm for the Vigenère cipher is shown in Figure 5.2.



▲ Figure 5.2 Vigenère Cipher Algorithm

5.1.2 Key Generation

The key generation algorithm, **KeyGen**, generates the key string used in the encryption process. This generated key string is secretly shared between the sender and receiver and is used to maintain the security of encrypted communication.

The length of the key string directly affects its security. If the key string is too short, repeating patterns are more likely to occur, increasing the risk of decryption. On the other hand, if the key string is too long, it becomes difficult to manage, so a practical balance must be achieved. Typically, a key string of moderate length is chosen, balancing usability and security.

If the key string consists of a single character, the cipher becomes equivalent to a shift cipher.

The key string can be based on a specific keyword (called a *key keyword*), but using a random key increases security.

If a new random key is selected for each encryption, and the key length equals or exceeds the length of the plaintext, then while the ciphertext still reveals that the message uses alphabetic characters, it leaks absolutely no information about the original plaintext. This level of security is equivalent to the **one-time pad**, which is considered theoretically unbreakable.

Thus, the Vigenère cipher can be seen as a general form of monoalphabetic substitution ciphers. When the key string is only one character long, it becomes a shift cipher; when the key length equals or exceeds the plaintext length, it becomes a one-time pad. Furthermore, when the shift cipher's key is fixed to 3, it becomes the Caesar cipher.

5.1.3 Encryption Mechanism

Encryption in the Vigenère cipher involves applying the shift cipher to each character of the plaintext. For example, the first character of the plaintext is encrypted using the first character of the key string as the shift key. The same process is applied to the second character, the third, and so on.

However, if the key string is shorter than the plaintext, the key string is repeated as many times as necessary to match the length of the plaintext. This repeated version of the key is called the **extended key**.

In summary, the n-th character of the plaintext is encrypted using the n-th character of the extended key as the shift key.

5.1.4 Manual Encryption Example

For example, suppose the plaintext is "hello world." and the key string is "STAR". After removing spaces and punctuation from the plaintext, we get "helloworld". To match the length of the plaintext (10 characters), we generate an extended key from the key string (4 characters), resulting in "STARSTARST" (10 characters). This is formed by concatenating `"STAR" || "STAR" || "ST"`.

We then encrypt the plaintext using the extended key with the shift cipher. During this process, we use the Vigenère table to convert each character. The resulting ciphertext is "ZXLCGPOIDW".

See Table 5.1.

▼ Table 5.1 Vigenère Cipher Encryption Example

Plaintext	h	e	l	l	o	w	o	r	l	d
Extended Key	S	T	A	R	S	T	A	R	S	T
Ciphertext	Z	X	L	C	G	P	O	I	D	W

5.2 Experimenting with the Vigenère Cipher Tool

The Vigenère Cipher Tool is a web-based application for encrypting and decrypting text using the Vigenère cipher. It is designed to help users visually understand the encryption process.

- GitHub Page
  - `https://github.com/ipusiron/vigenere-cipher-tool`
- Demo Page
  - `https://ipusiron.github.io/vigenere-cipher-tool/`

Try entering the example we encrypted manually earlier into the tool and verify that it produces the same ciphertext.

▲ Figure 5.3　Result of performing the same encryption with the Vigenère Cipher Tool

Appendix A

Letter Frequency in Alphabets

The characteristics of letter frequency (how often letters appear) are often effective in deciphering classical ciphers. Moreover, letter frequency can help identify or narrow down the language (or writing system) used in the encrypted message.

For example, in Spanish, the relative pronoun "que" is frequently used, so the letter 'q' appears more often compared to other languages. In German texts, letters like 'e' and 'n' are highly frequent. Each language has its own skewed distribution where certain letters appear more frequently than others. In other words, every language has a unique semiotic fingerprint.

By effectively leveraging these biases in letter frequency, cryptanalysis can become significantly more powerful.

A.1 Letter Frequency Distribution

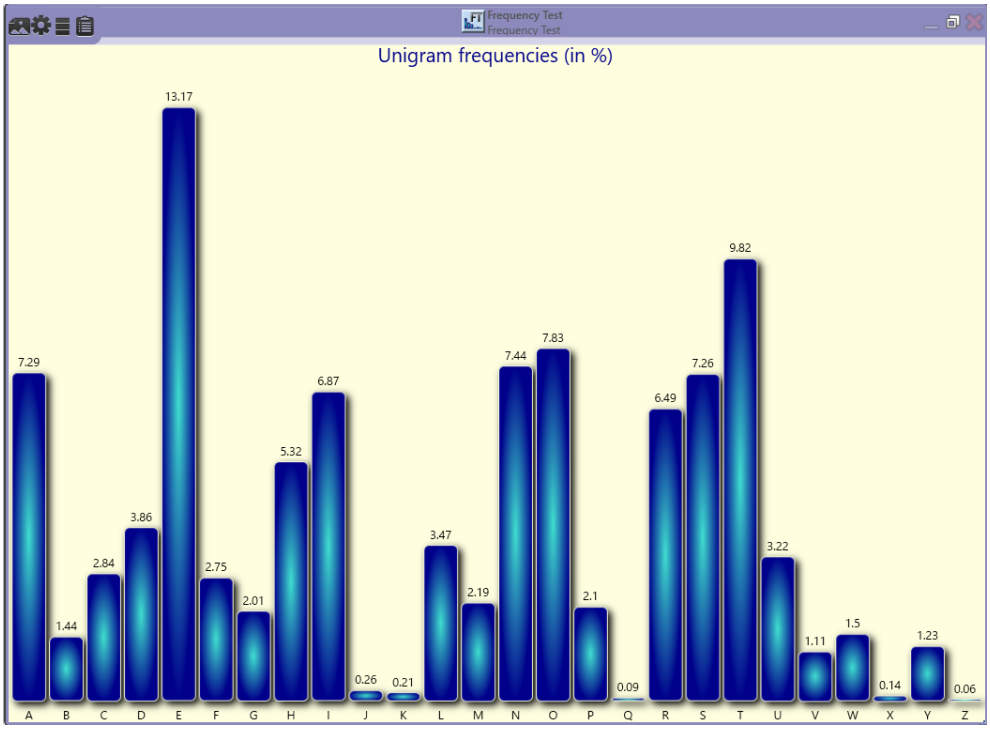
In English, each letter appears with the probabilities shown in Table 1.1. These values represent the typical letter frequency in general English texts.

▼ Table A.1 Letter Frequency Distribution

Letter	Frequency	Letter	Frequency
A	0.082	N	0.067
B	0.015	O	0.075
C	0.028	P	0.019
D	0.043	Q	0.001
E	0.127	R	0.060
F	0.022	S	0.063
G	0.020	T	0.091
H	0.061	U	0.028
I	0.070	V	0.010
J	0.002	W	0.023
K	0.008	X	0.001
L	0.040	Y	0.020
M	0.024	Z	0.001

A.2 Letter Frequency Graph

CrypTool 2 provides a built-in template for classical cipher analysis called the **Friedman Test for Classical Ciphers**. Using the sample provided with this template, we obtain the following results. The sample consists of 8,334 characters of English text.



▲ Figure A.1 Letter Frequency Graph

The letter 'E' appears with striking prominence. Its frequency is 13.17%, which is quite close to the 12.7% shown in Table 1.1.

Next comes 'T', followed by 'O', 'N', 'A', 'S', 'I', and 'R'. While the exact order doesn't perfectly match the frequencies in Table 1.1, it is largely consistent.

A.3 Grouping by Letter Frequency

Letters can be classified into six groups based on their frequency of appearance, as shown in Table 1.2.

▼ Table A.2 Hierarchical Classification of Letter Frequencies

Group	Letters	Frequency Range	Remarks
Most Frequent	E	0.12	Key for analysis
High Frequency	T, A, O, I, N, S, H, R	0.06-0.09	Common letters
Medium Frequency	D, L, U, C, M	0.03-0.04	-
Low Frequency	F, Y, W, G, P, B, V	0.015-0.028	-
Rare	J, K, Q, X, Z	Below 0.01	Very rare

A.4 Letter Frequencies Across Languages

Wikipedia's "Letter frequencies" page (`https://en.wikipedia.org/wiki/Letter_frequency`) provides information on letter frequencies for various languages.

When letters are ordered by frequency, we get the sequences shown in Table 1.3.

▼ Table A.3 Most Frequent Letters and Mnemonics by Language

Language	Letter Sequence
English	ETAOIN SHRDLU
Spanish	EAOSR NIDLT
French	ESAIT NRUOL
German	ENISR ATDHU
Italian	EAION LRTSC

You don't need to memorize every sequence, but it's useful to remember **ETAOIN** for English. This is a well-known mnemonic that aids memory and is commonly used in cryptanalysis.

While not every English text will follow this exact order of frequency, most will roughly align with it.

Incidentally, the phrase **ETAOIN SHRDLU** also became famous in the era of hot metal typesetting. It originated from typesetters' habits and often appeared in printed English materials as a placeholder. It is documented in resources such as the *Oxford English Dictionary* and *Random House Webster's Unabridged Dictionary.*

- Merriam-Webster Online Dictionary
  - `https://www.merriam-webster.com`
  - "ETAOIN SHRDLU" explanation page
    - `https://www.merriam-webster.com/dictionary/etaoin%20shrdlu`

This string also corresponds to the first two vertical columns of keys on the left side of a Linotype machine keyboard, traceable by hand.

Afterword

Thank you for reading *Caesar Cipher Breaking - A Concise Edition*.

In recent years, classical ciphers have often taken a backseat in cryptography books—especially in Japan, where this trend is particularly noticeable. For commercial publications, prioritizing sales is inevitable, so this outcome is understandable.

In contrast, this book is a self-published work, which means I could freely focus on the content I truly wanted to share without worrying about profitability. For a long time, I had wanted to write a book that focuses on a single classical cipher, rather than covering all of them broadly. From the beginning, I knew the first topic would be the Caesar cipher. As such, this book marks the first volume in the Classical Cipher Series, with Caesar as its starting point.

The Caesar cipher is simple in structure, yet it embodies the essence of substitution ciphers. Once you understand this essence, you will no longer be intimidated by more complex substitution ciphers.

This book aims not only to explain the cipher itself but also to convey the charm of classical cryptography. I hope that this small volume will help many readers discover the fascination of classical ciphers and contribute to renewed interest in this field.

Finally, I have a small request. I would be grateful for any honest feedback or impressions you may have about this book. Whether through posts on X or reviews on Amazon—any form is welcome. Your candid opinions will be incredibly helpful for my future writing.

Thank you again for your support, and I hope you continue to enjoy the world of cryptography.

Acknowledgments

The cover design of this book was originally created by Shizuka (@lily_mar32) and partially modified by the author.

I would like to express my sincere gratitude.

About the Author

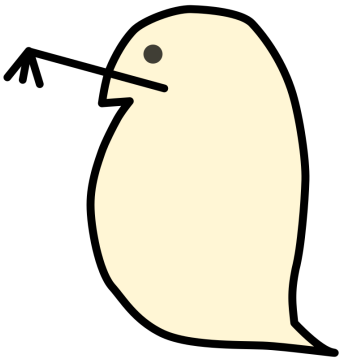


Figure: Author Photo

IPUSIRON

IPUSIRON is a Japanese technical writer and educator who helps people around the world understand security in a simple, approachable way.

With **over 30 books published in Japanese** on cryptography, hacking, and cybersecurity, he has established himself as a leading voice in Japan's security community. His bestsellers include:

- *The Complete Guide to Cryptography (*暗号技術のすべて*)*
- *How to Build a Hacking Lab – Complete Edition (*ハッキング・ラボのつくりかた 完全版*)*
- *The Textbook of White-Hat Hackers (*ホワイトハッカーの教科書*)*
- *The Hacker's School: A Handbook of Lockpicking (*ハッカーの学校 鍵開けの教科書*)*

As an experienced translator and editorial supervisor, he has brought important cryptographic literature to Japanese readers:

- Supervising translator of *Serious Cryptography* by Jean-Philippe Aumasson
- Translator of *Codebreaking: A Practical Guide* by Elonka Dunin & Klaus Schmeh
- Translator of *Cracking Codes with Python* by Al Sweigart

He founded **Security Akademeia** (akademeia.info), where he shares hands-on tutorials, cryptographic tools, and insights from the Japanese security community.

This book marks the first English installment of his Classical Cipher

Series, bringing historically significant ciphers to a global audience.

Contact

- Email: ipusiron@gmail.com
- X: `https://x.com/ipusiron`
- Website: `https://akademeia.info/`
- GitHub: `https://github.com/ipusiron`

Feel free to reach out or follow!

License

Caesar Cipher Breaking - A Concise Edition

Jul. 20, 2025 A Concise Edition（PDF）v1.0.0

Author IPUSIRON
Editor IPUSIRON
Publisher Security Akademeia