THE MANGA GUIDE™ TO
CRYPTOGRAPHY

MASAAKI MITANI
SHINICHI SATO
IDERO HINOKI
VERTE CORP.
PRAI SE FOR THE MANGA GUIDE SERIES

"Highly recommended."
—CHOICE MAGAZINE ON THE MANGA GUIDE TO DATABASES

"The Manga Guides definitely have a place on my bookshelf."
—SYTHESIAN MAGAZINE

"The art is charming and the humor engaging. A fun and fairly painless lesson on what
many consider to be a less-than-thrilling subject."
—SCHOOL LIBRARY JOURNAL ON THE MANGA GUIDE TO STATISTICS

"Stimulus for the next generation of scientists."
—SCIENTIFIC COMPUTING ON THE MANGA GUIDE TO MOLECULAR BIOLOGY

"The series is consistently good. A great way to introduce kids to the wonder and vastness of
the cosmos."
—DISCOVERY.COM

"Absolutely amazing for teaching complex ideas and
topics. Excellent primers for serious study of physics
and other fields."
—PHYSICS TODAY ON THE MANGA GUIDE TO PHYSICS

"A great fit of form and subject. Recommended."
—SHAKA USA MAGAZINE ON THE MANGA GUIDE TO PHYSICS

"I found the cartoon approach of this book so compelling
that I recommend that every teacher of introductory physics, in both high school and
college, consider using it."
—AMERICAN JOURNAL OF PHYSICS ON THE MANGA GUIDE TO PHYSICS

"This is really what a good math text should be like. Unlike
the majority of books on subjects like statistics, it doesn't
just present the material as a dry series of pointless
formulas. It presents statistics as something fun
and something enlightening."
—GOOD MATH, BAD MATH ON THE MANGA GUIDE TO STATISTICS

"A single tortured cry will escape the lips of every thirty-
something biochem major who reads The Manga Guide to
Molecular Biology. Why, oh why couldn't this have been
written when I was in college?"
—THE SAN FRANCISCO EXAMINER
"A lot of fun to read. The interactions between the characters are lighthearted, and the whole setting has a sort of quirkiness about it that makes you keep reading just for the joy of it."
—HACKDODY ON THE MANGA GUIDE TO ELECTRICITY

"The Manga Guide to Databases was the most enjoyable tech book I've ever read."
—RHO KITE, LINUX PRO MAGAZINE

"The Manga Guide to Electricity makes accessible a very intimidating subject, letting the reader have fun while still delivering the goods."
—CEDOMO

"If you want to introduce a subject that kids wouldn't normally be very interested in, give it an amusing storyline and wrap it in cartoons."
—MAKE ON THE MANGA GUIDE TO STATISTICS

"A clever blend that makes relativity easier to think about—even if you're no Einstein."
—STARDATE, UNIVERSITY OF TEXAS, ON THE MANGA GUIDE TO RELATIVITY

"This book does exactly what it is supposed to: offer a fun, interesting way to learn calculus concepts that would otherwise be extremely bland to memorize."
—EILLY TECH ON THE MANGA GUIDE TO CALCULUS

"Scientifically solid... entertainingly bizarre."
—CHAD ORZEL, SCIENCEBLOGS, ON THE MANGA GUIDE TO RELATIVITY

"Makes it possible for a 10-year-old to develop a decent working knowledge of a subject that sends most college students running for the hills."
—SKEPTICBLOG ON THE MANGA GUIDE TO MOLECULAR BIOLOGY

"The Manga Guide to the Universe does an excellent job of addressing some of the biggest science questions out there, exploring both the history of cosmology and the main riddles that still challenge physicists today."
—ABOUL.COM

"The Manga Guide to Calculus is an entertaining comic with colorful characters and a fun strategy to teach its readers calculus."
—DR. DEBBS
THE MANGA GUIDE™ TO CRYPTOGRAPHY
THE MANGA GUIDE™ TO
CRYPTOGRAPHY

MASAAKI MITANI,
SHINICHI SATO,
IDERO HINOKI AND
VERTE CORP.

Ohmsha

No Starch Press
Converting Data into Binary .................................................. 89
Generating a DES Ciphertext .................................................. 90
   Perform the Initial Permutation ........................................... 91
   Separate Data into Higher- and Lower-Order Bits .................. 92
   Feistel Rounds Variant .................................................. 92
   Swap the Higher- and Lower-Order Bits .............................. 95
   Perform the Final Permutation .......................................... 95
Decryption a DES Ciphertext ................................................. 97
   Perform the Initial Permutation ........................................... 97
   Separate Data into Higher- and Lower-Order Bits .................. 97
   Round 1 .................................................................. 97
   Round 2 .................................................................. 98
Generating DES Keys ............................................................. 100
   Perform Permutation Choice PC-1 ........................................ 101
   Rotate the Bits ................................................................ 101
   Perform Permutation Choice PC-2 ........................................ 102
   Rotate the Bits Again ...................................................... 103
   Compress the Bits .......................................................... 103
How Reduced DES Differs from Real DES ............................... 104

3
PUBLIC-KEY ENCRYPTION ...................................................... 105

The Basics of Public-Key Encryption ...................................... 106
   Mathematical Dependencies of Public-Key Encryption ........ 115
   One-Way Functions ....................................................... 116
   The Birth of RSA Encryption .......................................... 119
Prime Numbers and Integer Factorization ............................... 120
   Types of Primality Tests ................................................ 129
Modulo Operations .............................................................. 134
   Addition and Subtraction in Modulo Operations .................. 137
   Multiplication and Division of Modulo Operations ............... 146
Fermat's Little Theorem and Euler's Theorem ........................... 152
   Fermat: the Father of Number Theory ................................. 153
   Fermat's Method and Pseudoprime Numbers ....................... 155
   Euler's Theorem ......................................................... 156
   The Mathematician Euler ................................................ 157
   Euler Function on the Product of Two Prime Numbers .......... 158
The Structure of RSA Encryption .......................................... 161
   RSA Encryption: Encryption and Decryption ....................... 163
   Generating an RSA Encryption Key .................................... 165
   Verifying That the Public Key e and
   Private Key d Work in RSA Encryption .............................. 166
   How to Make Public Keys and Private Keys ....................... 167
   Generating a Ciphertext Using RSA ................................... 169
   Decrypting a Ciphertext Using RSA ................................... 171
Public-Key Encryption and Discrete Logarithm Problems: 173
Discrete Logarithm Problems: 174
Encryption and Decryption in ElGamal Encryption: 176
The Extended Euclidean Algorithm: 181
Calculating the Private Key in RSA Key Generation: 182

4
PRACTICAL APPLICATIONS OF ENCRYPTION: 185
Hybrid Encryption: 186
Hash Functions and Message Authentication Codes: 190
Pseudorandomness: 190
Tampering Countermeasures: 192
Hash Functions: 193
Identity Fraud: 194
Identity Fraud Countermeasures: 195
Structure of Message Authentication Codes: 196
Repudiation: 197
Drawbacks of Message Authentication Codes: 199
Digital Signatures: 200
Repudiation Measures: 200
The Workings of a Digital Signature: 201
Man-in-the-Middle Attack: 203
Using Certificates to Protect Against Man-in-the-Middle Attacks: 204
Public-Key Infrastructure: 206
Zero-Knowledge Interactive Proof: 217
Preparatory Stage: 217
Implementation Stage (Verification Process): 218
Methods of Identity Fraud: 220
Anti-Identity Fraud in a Zero-Knowledge Interactive Proof: 222

WHAT NEXT?: 224
Pseudorandom Numbers and Encryption Security: 224
PGP: 224
SSL/TLS: 225
Quantum Cryptography: 225
Biometrics: 225

INDEX: 227
When thinking about information societies, which rely on networks that function with the internet at their core, we can see how many aspects of our lives have become more convenient. In addition to now being able to search for information on the web and communicate via email, other network services like online shopping and internet banking have become widespread.

While the age of networking comes with many benefits, we also encounter phrases like, "safe and secure," "information security," "protection of personal information," "encryption," and so on that have a somewhat unpleasant ring to them. These words are everywhere—not a day goes by without seeing them. Why is such talk so prevalent?

Networks require a variety of information to be exchanged, and this includes information that could cause problems when in the hands of others and information that we wish to keep secret. We must take measures to protect information in order to prevent important data such as our credit card numbers, bank account numbers, medical histories, loan debts, and email addresses from easily leaking to outsiders. This information can be abused, and as such there is no doubt that protecting information is the most important issue that faces us in the age of networking. In our connected society, where there are so many unreliable factors, the ability to distinguish authentic information and provide protection from dangers such as fraud, forgery, revision, and interception is crucial. Cryptography is the core technology we use to create network services that the public can use safely.

Cryptographic technology has evolved rapidly in recent years. It is no longer just the domain of information security specialists, and is now necessary knowledge for the users themselves, who up until now have used convenient network services without a thought.

So how then does cryptography work? How does it actually provide information security and protect personal information?

Using manga, this book explains how cryptography works and what it does. The complex math is broken down into friendly explanations. We've done our best to make this an effortless learning experience where you can enjoy the story. Of course, there are also ciphertexts worked into the story, so do your best to decipher them while having fun. By the time you've finished reading this book, you'll no doubt have a thorough understanding of the fundamentals of cryptography and security.

Lastly, we'd like to express our most sincere gratitude to those who helped us publish this book, first to the folks at Ohmsia, Ltd. and also to Mr. Idero Hinoki who provided the illustrations.

MASAHIKI MITANI AND SHINICHI SATO
APRIL, 2007

Trying to copy this text. No Joy.
PROLOGUE
A CERTAIN CITY IN A CERTAIN PREFECTURE, 78TH PRECINCT

RUKA MEGURO

BIG BROTHER, BUY IT FOR MEEEE!

INSPECTOR JUN MEGURO

NO WAY! A HIGH SCHOOL GIRL DOESN'T NEED A COMPUTER THAT FANCY!

SLUUUUURP

BUT I WANT TO USE IT TO STUDY MATHEMATICS!

\[ ax + by = \gcd(a,b) \]

\[ \phi(p) = p - 1 \]

\[ \alpha^p \equiv 1 \pmod{p} \]

BUT I WANT IT!
AN ABACUS WILL DO JUST FINE FOR THAT!

SLURP

RING RING

CLICK

HELLO, 78TH PRECINCT.

WHAAAAT? STOLEN?!

GAK!!

MARBLE ART MUSEUM

ANY REPORT ON THE WEST HALL? YOU TWO GO AROUND TO THAT EXIT.

YES, SIR!

OH NO.

OH NO.

OH NO.

PROLOGUE 3
Hmmm, so a masterpiece— which was closely guarded— just suddenly vanished?

It's quite shocking.

The smiling Madonna is valued at 300 million yen!

Was the security guard even standing guard?

Was your security airtight?

Of course!

And when it came time to change guard shifts...

We made sure to exchange code words.

Mountain!

River!

Salute!

I see, I see. Right on!

This is elementary stuff...

Wobble.
The location of the painting depository is protected by a cipher to keep unauthorized people from finding it.

THE DEPOSITARY IS WATArehOtaUSEfITAVE

All right! No way an outsider could have figured that out!

That’s awful.

You call that security?

Click

Who are you?!

Click Click

Rio Yoneda, from the nightly news!
SO THE PAPER'S ALREADY GOTTEN WIND OF THE INCIDENT, EH?

WHAT'S WRONG WITH OUR SECURITY?!

YOUR CODE WORDS AND ENCRYPTION METHODS ARE JOKES!

WHAAAAAT?!

THE PAINTING DEPOSITORY IS WAREHOUSE 5, RIGHT?

THE DEPOSITORY IS WHAT IS USEFUL.

SQUEAK SQUEAK

SINCE THERE'S A PICTURE OF A JAPANESE RACCOON, WHICH IS CALLED A TANUKI, THE MESSAGE CAN BE READ BY REMOVING EACH INSTANCE OF THE WORD'S FIRST SYLLABLE-TA.*

ARE YOU THE PERPETRATOR?!

HOW DID YOU KNOW?

WHAT?! NO!

ANYONE COULD FIGURE IT OUT!

* THE TANUKI CIPHER IS A COMMON WORDPLAY GAME IN JAPAN. THE CIPHER IS BASED ON A PUN OF THE WORD TANUKI, WHICH CAN MEAN "RACCOON" OR "OMIT THE TA'S."
WAIT, REALLY?

INSPECTOR MEGURO! FORGET ABOUT HER, FIND THE REAL THIEF—AND QUICK!

THE PERP... THE PERP IS...

THE PERP IS RIGHT THERE!

ME?

REALLY? YOU'RE UNDER ARREST!

CHNK!

EEEEEK!
NO—ON THE WALL, BEHIND THE CURATOR!

HUH?

Ms. Cipher

LOL

TA-

DAH!

There wasn't a painting here before, but...
DON'T LOOK AT THE PAINTING—CHECK OUT ITS PLAQUE!

YOU'VE BEEN VISITED BY MS. CYpher.
I'VE TAKEN THE PAINTING.
NEXT I'LL TAKE VDVIRCU.
GOOD NIGHT.

MS. CYpher?! THE PHANTOM THIEF?!

AH!

IT SEEMS STRANGE TO SAY "GOOD NIGHT" GIVEN THAT IT'S DAYTIME, EH?

I WONDER WHAT THIS MESSAGE MEANS.

I WAS JUST WONDERING THE SAME THING.

NOT THAT!
I've taken the painting. Next I'll take VDVIRCU.

VDVIRCU—what's that? Is that English?

I don't know any foreign languages, so...

It's a ciphertext! A ciphertext telling us what will be stolen next!

Unlike with the museum's ciphertext, I'm not finding any meaningful words by omitting letters...

VDVIRCU

In that case, let's study cryptography and take down Ms. Cypher!

But this isn't some spy novel. Is cryptography really that useful?
In the computer age, cryptography has become essential to protecting against the falsification, destruction, and interception of information.
IF YOU DIDN'T HAVE CRYPTOGRAPHY, SOMEONE COULD INTERCEPT AND TAMPER WITH YOUR MESSAGES...

HEHEHE

I'LL JUST OVERWRITE THIS.

Bob

Alice

I LOVE YOU, ALICE!

Notorious eavesdropper intercepting messages

He hates me?!

Sender

Recipient

Internet

SHOCK!

I SEE! SO, IT'S BETTER TO SEND MAIL VIA POSTAL SERVICE THAN BY COMPUTER.

SMACK!

THAT'S NOT IT AT ALL!
WE HAVE SOMETHING THAT PROTECTS AGAINST THOSE DANGERS...

CRYPTOGRAPHY!

Notorious eavesdropper

Eve

AHHH! I CAN'T TAMPER WITH IT!

Bob

Alice

Sender → Encryption → Internet → Decryption → Recipient

Key k used for encryption

I love you!

7yelzi4njx3

Key k used for decryption

Oooh, so cryptography really is useful!

See? Starting to get interested?

7 The recipient doesn't have to be a person. It could instead be a hard disk, computer memory, or some other type of storage media.
LET'S STUDY CRYPTOGRAPHY! I'LL BE YOUR TEACHER!

ALL RIGHT! LET'S DO OUR BEST TO CAPTURE MS. CYFHER!

IN-INSPECTOR...

CURATOR, WE'LL HAVE THIS INCIDENT RESOLVED IN NO TIME. JUST LEAVE IT TO ME!

WAIT! GET ME OUT OF THESE HANDCUFFS!

UM...
1

THE FOUNDATIONS OF ENCRYPTION
WAS WAS YEAH!

SO WHAT'S A REPORTER LIKE YOU DOING HERE?

BIG BROTHER, IT'S TIME TO GET YOU UP TO SPEED ON THIS CASE! LISTEN UP!

THIS THEFT IS GOING TO BE A BIG STORY! I WANT TO COVER IT THOROUGHLY.
WHAT IS A CIPHER?

A BIG STORY... THE PRESSURE IS ON!

FIRST...

LET'S TALK ABOUT WHY THIS PHANTOM THIEF CALLS HERSELF "MS. CYFER".

MM...

THAT'S SIMPLE!

SHE STEALS OTHER PEOPLE'S TREASURES, SO SHE GOES BY THE NAME CYFER!*

TALK ABOUT SIMPLE...

cipher

NO, CYFER! IT'S A WAY OF WRITING A SECRET MESSAGE.

* THE WORD FOR TREASURE IS SARU IN JAPANESE, WHICH SOUNDS LIKE CIPHER.
I see, a cipher.

MAYDAY = DISTRESS SIGNAL

ROGER = TRANSMISSION RECEIVED

Oh!

I know some famous ones, like when a pilot says "MAYDAY" or "ROGER."

Close but not quite, those aren't actually ciphers.

Was was well, what are they?

If they aren't ciphers...

These kinds of words are only used among peers in a particular field. You could call these words jargon or procedure words.

Code

But what we're studying...

ARE CIPHERS!

Cipher

They're also known as codes.
Let's look at how ciphers work using Shannon's encryption model.

Claude Shannon (1916-2001) was a 20th-century English mathematician who came to be known as the father of information theory. In 1948, he wrote an article entitled "A Mathematical Theory of Communication" in which he coined the term 'bit' and developed several concepts key to cryptography.
SHANNON'S ENCRYPTION MODEL

Here's what the process looks like, broken down into steps.
You start with a plaintext \( m \), which is just an unencrypted, normal message.

Plaintext \( m \) = I love you!

You end up with ciphertext \( C \), which is the encrypted message.

Ciphertext \( C \) = yye02h4njw3

Encryption is the process that transforms the plaintext to a ciphertext:

\[
\begin{align*}
\text{Plaintext } m & \quad \rightarrow \quad \text{Ciphertext } C \\
I \text{ love you!} & \quad \rightarrow \quad \text{yye02h4njw3}
\end{align*}
\]

where \( E_k \) is encryption using an encryption key \( k \).

Decryption is the process that restores the plaintext from a ciphertext:

\[
\begin{align*}
\text{Ciphertext } C & \quad \rightarrow \quad \text{Plaintext } m \\
yye02h4njw3 & \quad \rightarrow \quad I \text{ love you!}
\end{align*}
\]

where \( D_k \) is decryption using a decryption key \( k \).
THE RELATIONSHIP BETWEEN THE ENCRYPTION KEY AND THE DECRYPTION KEY

The sender encrypts their message. Using the sender's plaintext message \( m \) and encryption key \( k \) on the encryption algorithm results in ciphertext \( C \).

\[
\text{Encryption key } k
\]
\[
\text{Plaintext } m \rightarrow \text{Encryption} \rightarrow \text{Ciphertext } C
\]

The recipient decodes the ciphertext. Ciphertext \( C \) is decrypted into plaintext \( m \) by using the ciphertext \( C \) and decryption key \( k \) on the decryption algorithm.

\[
\text{Decryption Key } k
\]
\[
\text{Ciphertext } C \rightarrow \text{Decryption} \rightarrow \text{Plaintext } m
\]
Let's look at another example. Take this ciphertext...

SVLBJTCFBVUJGVM

SQUEAK SQUEAK

If encryption with encryption key $k$ shifts each of the letters in the alphabet forward one space...

SVLBJTCFBVUJGVM

RUCAISBEAUTIFUL

What would the plaintext be?

RUKA IS BEAUTIFUL, RIGHT?

BWAHAHAHAHAHA!

AHAHAHA! RUKA? BEAUTIFUL? THAT'S TOTALLY OFF BASE!
She's absolutely correct!

Decryption with decryption key K moves each letter back one space to its original position.

Ruka-sensei is a little scary...

But can't a cipher like that be instantly decoded?

Information Security Handbook

Ooh, the weight of knowledge...

That's a very simple cipher. Ciphers have become much more sophisticated after all, if you want to pass secret messages, you have to deal with eavesdroppers trying to decipher them.

Let's take a look at some classic encryption methods to get a better understanding of this game of cat and mouse!
CLASSIC ENCRYPTION METHODS

Although classic encryption methods are no longer used because they're not secure, they're still useful for learning how encryption works. Let's look at how some of these historical ciphers worked in practice.

CAESAR CIPHER

In the previous example, you saw the Caesar cipher in action. The Caesar cipher uses an algorithm that creates the ciphertext by shifting each letter of plaintext n letters. By way of example, let's try to encrypt the title of the Japanese fairytale Momotaro. If $n = 3$, we would move each letter forward by three.

\[ ... L M N O P Q R ... \]

DO THE SAME WITH THE REST OF THE LETTERS:

O → R T → W A → D R → U

Once you've done this, a ciphertext is formed.

\[
\begin{align*}
\text{MOMOTARO} & \quad \text{Plaintext m} \\
\text{PRPRWDUR} & \quad \text{Ciphertext C}
\end{align*}
\]

Letters at the end of the alphabet just wrap around to the beginning.

\[
\begin{align*}
X & \rightarrow A \\
Y & \rightarrow B \\
Z & \rightarrow C
\end{align*}
\]

This cipher is named after the Roman soldier and statesman Julius Caesar (Gaius Julius, 100-44 BCE). Caesar used this cipher during the Gallic Wars to correspond with his allies unbeknownst to his rivals.

**Julius Caesar said, "The die is cast!" when he crossed the Rubicon.**

**WHAM**

**Really?!**

*In Japanese, die sounds the same as rhino.*
**SUBSTITUTION CIPHER**

It's pretty easy to decrypt a plain Caesar cipher, which is a type of substitution cipher. Substitution ciphers that replace one letter of the plaintext with another letter to make the ciphertext, as the Caesar cipher does, are called simple substitution ciphers.

Let's look at a simple substitution cipher to get a handle on it.

Assume that the 26 letters of the English alphabet have been converted as follows:

```
Conversion rule \( \sigma = \)

| 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 |
| Q | W | E | R | T | Y | U | I | O | P | A | S | D | F | G | H | J | K | L | Z | X | C | V | B | N | M |
```

Thus, a cipher is formed as follows:

```
\[ \text{MOMOTARO} \rightarrow \text{DG DZ QKG} \]

Plaintext \( m \)

Conversion in accordance with conversion rule \( \sigma \)

Ciphertext \( C \)
```

In this cipher, the conversion of letters is the algorithm, and the "letter-by-letter replacement method"—in other words, the conversion rule \( \sigma \)—is the encryption key \( k \).
POLYALPHABETIC CIPHER

A cipher that divides the plaintext into fixed-size blocks of \( n \) letters and varies the number of positions each letter shifts within each block is known as a polyalphabetic cipher. This makes a substitution cipher even sneakier.

Here we see conversion rule \( \delta \) (delta) in action for a block of four letters \( (n = 4) \):

Conversion rule \( \delta \):
- First letter \( \rightarrow \) Move by 2 letters
- Second letter \( \rightarrow \) Move by 5 letters
- Third letter \( \rightarrow \) Move by 3 letters
- Fourth letter \( \rightarrow \) Move by 1 letter

The resulting cipher works like this:

Plaintext \( m \)

\[
\begin{align*}
\text{MOMOTARO} \\
\text{MOMO} & \quad \text{TARO} \\
\text{OTPP} & \quad \text{VFUP}
\end{align*}
\]

Divide each group of \( n \) letters into blocks \( (n = 4) \).

Convert each block of \( n \) letters in accordance with conversion rule \( \delta \).

The process to decrypt the ciphertext works the same way, just in reverse.

NOW DECIPHER THE FOLLOWING, WHICH WAS CREATED BY USING THIS FORMULA: TZN8 KX EFCZWJHZO!

POKE

RUKA IS BEAUTIFUL! REALLY!
TRANPOSITION CIPHER

A cipher that divides the plaintext into fixed-size blocks of \( n \) letters and then changes the sequence of the letters in each block is called a transposition cipher. For example, if \( n = 4 \), the substitution rule \( \tau \) (tau) gives us the following:

\[
\tau = \begin{pmatrix}
1 & 2 & 3 & 4 \\
4 & 1 & 2 & 3
\end{pmatrix}
\]

Here's what is going on in this transposition cipher:

**Substitution rule** \( \tau = \)
- First letter \( \rightarrow \) Becomes second letter
- Second letter \( \rightarrow \) Becomes fourth letter
- Third letter \( \rightarrow \) Becomes first letter
- Fourth letter \( \rightarrow \) Becomes third letter

Thus, the following cipher is born:

Plaintext \( m \)

\[
\text{MOMOTARO}
\]

\[
\begin{array}{c}
\text{MOMO} \\
\text{TARO}
\end{array}
\]

Ciphertext \( c \)

\[
\begin{array}{c}
\text{OOMM} \\
\text{ATOR}
\end{array}
\]

Divide each group of \( n \) letters into blocks \( (n = 4) \).

Shift letter in each block according to substitution rule \( \tau \).

In this cipher, the rearrangement of the letters makes up the algorithm, and the number of letters per block and the substitution rule \( \tau \) are the encryption key.

\( \delta \) is the Greek letter delta, and \( \tau \) is the Greek letter tau.

I see.
Cipher Security

Although the Caesar cipher was invented more than 2,000 years ago...

The Caesar cipher's encryption algorithm...

Shifting plaintext letters by $n$ number of letters.

$\text{CDEFGIJLM}\rightarrow \text{ABCDERHIJ}$

We use the concepts of algorithms and keys...

To describe the same ideas in modern cryptography.

For instance,

$n = 3$ is the encryption key that Caesar used.

$\text{ABCD} \rightarrow \text{ABCDEF}$

The problem I showed you earlier—$\text{SVLBCDFAEHIJGM}$—is an example of encrypting with the Caesar cipher.

$\text{SVL} \leftrightarrow \text{RUKAISBEAUTIFUL}$

28  Chapter 1  The Foundations of Encryption
HMM...

The ancient Roman alphabet had only 23 letters.

It looks like there aren't enough encryption keys to be very secure...

ABCDFG
HILMNOP
QRSTVX
YZ

Could you shift through the letters over and over to make encryption more secure?

Like if you went through the alphabet 1,000 or 2,000 times...

No, just like the hands on a clock, you would revolve through the same series of letters.

Huff

So, let's see... 1,000, 2,000...

In other words, no matter how many letters you shift, there are only 22 possible encryption keys! We call the total number of possible keys in a cipher its key space.
SO, IF AN EAVESDROPPER NOTICED THE STRUCTURE OF A CAESAR CIPHER THAT USED THE ANCIENT ROMAN ALPHABET...

THEY COULD FIGURE OUT THE KEY AND DECODE THE CIPHER IN FEWER THAN 23 ATTEMPTS.

SO IT'S BETTER TO USE JAPANESE THAN IT IS TO USE THE ROMAN ALPHABET!

AHA!

TO PREVENT YOUR CIPHERTEXT FROM BEING DECODED...

IT'S BEST TO HAVE A VERY LARGE NUMBER OF KEYS.

NICE IDEA!

BECAUSE WE HAVE HIRAGANA, KATAKANA, AND KANJI, THERE ARE MORE THAN 10,000 KEYS.

LET'S TAKE A LOOK AT HOW MANY KEYS THERE ARE IN OTHER CIPHERS.
THE SUBSTITUTION CIPHER’S KEY SPACE

Let's think more about how many potential keys there might be for substitution ciphers and how easy such a cipher would be to break. You saw that a Caesar cipher using the ancient Roman alphabet had only 22 possible keys (a key space of 22), so breaking it wouldn’t be too hard. The key space of the Caesar cipher is limited because it depends on shifting the alphabet to determine the plaintext letter’s ciphertext substitution. In Caesar’s version of the Caesar cipher, the substitution letters are simply the alphabet shifted by three positions, but you could take a different approach by scrambling the positions of the letters entirely.

Instead of shifting the letters to determine letter substitutions, you could substitute each letter of the alphabet with any other letter as long as each plaintext letter corresponds with one unique substitution. Your plaintext letters would be the alphabet in order:

ABCD{EFGHIJKLMNOPQRSTUVWXYZ

Then one example of a substitution with a scrambled alphabet could be written out like this:

QWERTYUISDFGHJKLZXCVBNM

Each letter of the alphabet would be substituted with a corresponding letter in the scrambled version of the alphabet. You can simply substitute according to the position of the letters in the alphabet, so A would be substituted as Q, B as W, and so on. The scrambled alphabet would be the cipher’s key.

If we used our modern alphabet with 26 letters, we would have a large number of ways to potentially arrange those letters for a key:

\[ P_{26} = 26! = 26 \times 25 \times 24 \times \ldots \times 3 \times 2 \times 1 = 4.03291461 \times 10^{26} \]

The letter P is used to represent a permutation, which is a unique arrangement of letters in a sequence. The subscript number before the P means that there are 26 total characters, and the subscript number after the P means we are using all 26 of them. The exclamation mark means factorial, or the product of the integer and all the positive integers less than it. This formula makes sense if you think about it this way:

- The first letter could be assigned any of the 26 letters, so there are 26 to choose from.
- The second letter could be any of the remaining 25 letters, so there are 25 to choose from. For two letters, there are 26 \times 25 possibilities.
- The third letter could be any of the remaining 24 letters.
  And so on until...
- The last letter will be whichever letter remains—there is one option for the last letter.
Given $4.03291461 \times 10^{35}$ possible permutations of letters, if a computer were asked to generate all these arrangements until it found the correct key to break the cipher, then, even if it were to perform an exhaustive search at a rate of 100,000,000 keys per second, it could take a preposterous period of up to 128,000,000,000 years.

**PERMUTATION, COMBINATION, AND FREQUENCY ANALYSIS**

Let's look at permutations in more detail. A permutation, expressed as $P$, is a method of selecting $r$ things from $n$ possibilities and putting them into unique arrangements. By unique arrangements, we mean that, for instance, ABCDE is not the same as EDCBA. Those are considered two different arrangements.

If we selected 5 letters out of the 26 letters of the alphabet, $r$ would be 5 and $n$ would be 26. Then we would use this formula to find the number of arrangements of 5 letters selected from 26:

$$P_r = n! / (n-r)!$$

Thus,

$$P_5 = 26! / (26-5)! = 26! / 21!$$

Because all the factors in $21!$ are in $26!$, you can cancel out the numbers up to and including 21:

$$26! / 21! = 26 \times 25 \times 24 \times 23 \times 22 - 7,893,000$$

32. **CHAPTER 1 THE FOUNDATIONS OF ENCRYPTION**
In contrast, if \( r \) things are selected from \( n \) possibilities and order does not matter, this is called combination and is expressed as \( C \) in the combination formula:

\[
^\text{\( n \)} C_r = \frac{n!}{r!(n-r)!}
\]

If we're still selecting 5 letters out of 26 possible letters, we would get this:

\[
\frac{26!}{(21!\times5!)}
\]

The factors would cancel like in the permutation example, which would leave you with this:

\[
\frac{26\times25\times24\times23\times22}{1\times2\times3\times4\times5} = 65,780
\]

Because order does not matter, the arrangements ABCDE and EDCBA would be considered the same, meaning that there are fewer possible arrangements when using combinations rather than permutations.

Searching for the correct key within the large space of \( 4.03291461 \times 10^{26} \) permutations is theoretically possible, but in practical terms, it's a \textit{computationally infeasible problem}. That said, these types of ciphers are susceptible to a cryptographic attack known as frequency analysis, which assumes that the frequency of letters that appear in the plaintext and the frequency of letters that appear in the ciphertext are consistent.

For example, the letter \( E \) is the most common letter in English. That means if \( E \) is the most common letter in the encrypted text, you can reasonably assume that the corresponding letter in the plaintext is \( E \). Frequency analysis attacks are one of the most effective attacks on simple substitution ciphers.

**THE POLYALPHABETIC CIPHER'S KEY SPACE**

Let's assume that a given block is \( n \) characters long. Because we don't know how many spaces the first letter has been shifted, we make 26 attempts to figure it out. Each letter shift in the block is independent from the other shifts, so the second letter could also take 26 attempts, same with the third letter, and so on, until we arrive at the \( n \)th letter.

Using what we just learned about permutations, combinations, and frequency analysis, \( r \) would be block size \( n \), and the possible correct values for \( n \) would be the 26 letters of the alphabet and would apply to each letter. Because of this, the total number of keys is as follows:

\[
26 \times 26 \times \ldots \times 26 \times 26 = 26^n
\]
If \( n = 4 \), then the key space would be the following:

\[
26^4 = 26 \times 26 \times 26 \times 26 = 456,976
\]

As \( n \) grows larger, the number of keys rapidly increases. When \( n = 10 \), the number of possible keys surpasses 140 trillion.

---

**THE TRANPOSITION CIPHER’S KEY SPACE**

The total number of keys for a block of \( n \) characters is expressed as follows:

\[
P_n = n \times (n - 1) \times (n - 2) \times \ldots \times 3 \times 2 \times 1 = n!
\]

So when there are four letters in one block \( (n = 4) \), the total number of keys \( k \) is as follows:

\[
4! = 4 \times 3 \times 2 \times 1 = 24
\]

As with other ciphers, the key space increases as \( n \) becomes larger, and the security of the encryption becomes stronger. Note that \( n = 26 \) would give the same key space as the substitution cipher.
THEN COULD A SUBSTITUTION CIPHER WITH A LARGE ENOUGH KEY SPACE BE SECURE? IT IS SIMPLE...

SIMPLE SUBSTITUTION CIPHERS CAN BE EASILY CRACKED USING BASIC DECODING CLUES.

THE POLYALPHABETIC CIPHER SEEMS REALLY COMPLICATED.

GOLD BUG? IS THAT A PICTURE BOOK OF INSECTS?

I'M RICH!

IT'S A FAMOUS MYSTERY STORY THAT DEALS WITH CRYPTOGRAPHY.

COO CON, A STORY!

HERE'S PART OF "THE GOLD BUG" CIPHER:

53++3(05));b*;4(826)
4*;4*);806*;4(89
8*80));85;1*;1*88(88)

THE TRANSPOSITION CIPHER'S KEY SPACE 35
Simple substitution ciphers can be decoded using frequency analysis!

That's how the protagonist in "The Gold Bug" solves the cipher.

Because we know that the most common letter in English is e, and the most common word is the...

We can start by assuming that the most frequently used symbol, is e, and that the most frequently used combination of symbols, is the.

Without a doubt!

Brilliant!

I wish I could recruit him to the force!

When a ciphertext is long, these kinds of clues increase in number, making decoding easier.

But when you have a short sentence, decoding is tough!
**WHEN BREAKING CLASSICAL ENCRYPTION IS POSSIBLE**

Typically, breaking classical encryption is possible under these conditions:

1. When you understand the encryption algorithm
2. When there is data about the statistical properties of the encrypted plaintext; for example, the frequency with which a letter or word appears
3. When you have a large number of encrypted example sentences

**PERFECTLY SECURE ENCRYPTION**

By generating a random number to use as a one-time pad—a key that is as long as the plaintext, is used once, and is then discarded—you can produce a cipher that is computationally secure. More specifically, you would be generating a ciphertext by applying a string of random numbers to plaintext n that is the same length as the plaintext. This ciphertext cannot be decrypted without knowledge of the key. Unfortunately, a one-time pad is not practical because it is difficult to distribute the one-time pad to all the communicating parties ahead of time.

Gilbert Vernam devised a cipher that uses a one-time pad, the Vernam cipher, in 1917 and patented it a couple of years later. During World War II, Claude Shannon (see page 19) established that this cipher is unbreakable and in 1949 publicly published his mathematical proof.

Here is a simple example of a Vernam cipher. You start by converting the alphabet to numerical values in a process known as *character encoding*:
1. Convert each letter to a numerical value (e.g., A becomes 0).

### Plaintext

<table>
<thead>
<tr>
<th>M</th>
<th>O</th>
<th>M</th>
<th>O</th>
<th>T</th>
<th>A</th>
<th>R</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>14</td>
<td>12</td>
<td>14</td>
<td>19</td>
<td>0</td>
<td>17</td>
<td>14</td>
</tr>
</tbody>
</table>

2. Add each number in the sequence to the corresponding one-time use number.

### Random number sequence

<table>
<thead>
<tr>
<th>12</th>
<th>14</th>
<th>12</th>
<th>14</th>
<th>19</th>
<th>0</th>
<th>17</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>9</strong></td>
<td><strong>20</strong></td>
<td><strong>15</strong></td>
<td><strong>23</strong></td>
<td><strong>27</strong></td>
<td><strong>2</strong></td>
<td><strong>15</strong></td>
<td><strong>8</strong></td>
</tr>
<tr>
<td><strong>21</strong></td>
<td><strong>34</strong></td>
<td><strong>27</strong></td>
<td><strong>37</strong></td>
<td><strong>46</strong></td>
<td><strong>2</strong></td>
<td><strong>32</strong></td>
<td><strong>22</strong></td>
</tr>
</tbody>
</table>

### Encryption key

3. Calculate the remainder when each number is divided by 28.

### Ciphered number sequence

<table>
<thead>
<tr>
<th>21</th>
<th>34</th>
<th>27</th>
<th>37</th>
<th>46</th>
<th>2</th>
<th>32</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>26</strong></td>
<td><strong>26</strong></td>
<td><strong>26</strong></td>
<td><strong>26</strong></td>
<td><strong>26</strong></td>
<td><strong>26</strong></td>
<td><strong>26</strong></td>
<td><strong>26</strong></td>
</tr>
<tr>
<td><strong>21</strong></td>
<td><strong>8</strong></td>
<td><strong>1</strong></td>
<td><strong>11</strong></td>
<td><strong>20</strong></td>
<td><strong>2</strong></td>
<td><strong>6</strong></td>
<td><strong>22</strong></td>
</tr>
</tbody>
</table>

4. Convert the numbers to letters using the character encoding.

### Ciphered text

<table>
<thead>
<tr>
<th>V</th>
<th>I</th>
<th>B</th>
<th>L</th>
<th>U</th>
<th>C</th>
<th>G</th>
<th>W</th>
</tr>
</thead>
</table>
SECURE CIPHERS

1. Fail-safe ciphers
   A fail-safe cipher like Vernam's cipher is theoretically unbreakable.

2. Computationally secure ciphers
   Although computationally secure ciphers can be decoded, doing so is computationally infeasible because it's so time and labor intensive that it isn't worth trying. These ciphers are what we use today.
Even though the Vernam cipher is theoretically unbreakable, it is impractical to use it to encrypt and decrypt long messages. Sending 1,000 letters of plaintext would require a 1,000-character key.

Such a long ciphertext would take a lot of computational resources and time to compute. This is why we use computationally secure ciphers in real-world scenarios.

I don't think that Ms. Cipher's cipher is a Vernam cipher.

It was deliberately left behind, so it's probably not a complex cipher.

GROOOOOOAN

There's got to be a hint somewhere...

Ah! Look at the time!

I'm headed back to the newsroom.

UHNN
EVENING FALLS

THANKS FOR CHOOSING USAGI'S DINER!

DINNER TIME!

THANK YOU FOR WAITING.

TA-DA!

UNTIL NEXT TIME—

Usagi's Diner

DELICIOUS!

SLURP SLURP

RAMEN... USAGI... MS. CYFHER...

GOOD NIGHT...

* USAGI MEANS RABBIT OR BUNNY IN JAPANESE.
YOU BETTER EAT UP OR YOUR RAMEN WILL GET COLD!

I’VE GOT IT!

EAT NOW, TALK LATER.

NO!

I’VE SOLVED THE CIPHER!

I KNOW WHAT MS. CYpher PLANS TO STEAL NEXT!

WHa—WHAT?!
IF I TELL YOU WHAT SHE PLANS TO STEAL, WILL YOU BUY ME A COMPUTER?

ER, I GUESS SO...

THE BUNNY COSTUME AND THE PHRASE "GOOD NIGHT" WERE HINTS!

YOU’VE BEEN VISITED BY MS. CYpher. I’VE TAKEN THE PAINTING. NEXT I’LL TAKE VDVIRCU. GOOD NIGHT.

WHAT DO YOU MEAN?

WE WERE PUZZLED ABOUT WHY MS. CYpher SIGNED OFF WITH "GOOD NIGHT."

I THINK IT’S THE SECOND CLUE TO THE CIPHER. YOU SAY "GOOD NIGHT" WHEN YOU GO TO SLEEP, AND SLEEP HAS THE SAME NUMBER OF LETTERS AS BUNNY.

bunny

sleep

I SEE... SO...
IT'S A CAESAR CIPHER!
IF YOU SHIFT EACH LETTER IN BUNNY FORWARD BY 17 LETTERS...

bunny
↓↓↓↓↓↓↓↓↓↓↓↓
17 LETTERS

sleep

YOU GET SLEEP!

MOVING THE LETTERS 17 PLACES IS THE KEY!

YES!

SO IF WE MOVE EACH LETTER IN VDVIRCU 17 SPACES...

THE SOLUTION IS...

gulp
EMERALD!

THA—THAT'S RIGHT.

EMERALD...

AN EMERALD HAS BEEN STOLEN FROM THE INTERNATIONAL TREASURES EXHIBITION!

MS. CYRHER STRIKES AGAIN! LET'S HEAD TO THE SCENE!

BUT WHAT ABOUT...
SINCE THE POLICEMAN GAVE US THE ANSWER, YOUR NEW COMPUTER WILL HAVE TO WAIT.
Symmetric-Key Algorithms
We've been had again!

Emerald
THAT EMERALD COSTS THREE BILLION YEN!"

WAS THE EXHIBITION CASE LOCKED?

OF COURSE! BUT--

IT MUST HAVE BEEN OPENED AT SOME POINT, BECAUSE YOUR PRECIOUS GEM WAS REELED RIGHT UP FROM THE CEILING.

Hi!

YOU AGAIN!

HOW DO YOU KNOW THE THIEF'S MODUS OPERANDI?

Fwip!

THIS WAS DELIVERED TO THE NEWSROOM!
WHAT?
WHAT IS IT?

CHECK OUT MY CATCH OF THE DAY:
YOUR PRECIOUS EMERALD ❤️

-MS. CYpher

PS: IF YOU WANT TO
MEET ME IN THE FLESH,
THEN FIGURE THIS OUT!

SO IT'S A NEW
CIPHERTEXT...

HMM....

I'LL GET RUKA'S
HELP—PRONTO!

WAIT! WHAT ABOUT
THE EMERALD?

WAIT FOR ME!
WE KNOW WHO THE THIEF IS!

CATCH ME IF YOU CAN!

WELL, HURRY UP AND BRING BACK MY EMBEERALD!

WE WILL! ONCE WE FIGURE OUT HER REAL NAME AND WHEREABOUTS...

ZOOOOOM

AH

WAIT UP!

HOW IRRESPONSIBLE!
00110001 00101011 00110001 00111101 00110000

This time it's a bunch of 1s and 0s. What could it mean?

Aha!

It's binary!
We usually use the base-10 decimal number system, but the binary system is in base 2, which uses only two numbers to count.

When you count past 1 in binary, you run out of digits and must carry the 1 to the next largest place value. For example, the number 2 is 10 in binary.

A group of eight bits in a row is called a *byte*. Each bit represents a 0 or 1, so a byte has a total of $2^8 = 256$ possible combinations. Each combination represents a different piece of information.

<table>
<thead>
<tr>
<th>Binary number</th>
<th>Decimal number</th>
<th>Hexadecimal number</th>
<th>Binary number</th>
<th>Decimal number</th>
<th>Hexadecimal number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
<td>1001</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
<td>1010</td>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
<td>1011</td>
<td>11</td>
<td>B</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
<td>1100</td>
<td>12</td>
<td>C</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
<td>1101</td>
<td>13</td>
<td>D</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
<td>1110</td>
<td>14</td>
<td>E</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
<td>1111</td>
<td>15</td>
<td>F</td>
</tr>
</tbody>
</table>

As numbers become larger, the number of digits required to represent them in binary grows rapidly, so *hexadecimal notation* is often used instead. Hexadecimal notation is in base 16, using digits 0 through 9 and letters A through F, as shown in this table. This means that after you count up to 9, you use A instead of 10, B instead of 11, and so on up to F. The prefix 0x is added to hexadecimal numbers, so for example, the decimal number 10 is 0xA in hexadecimal.

Unlike historical ciphers, which operate on letters, modern encryption methods operate on binary numbers.
So in the note left by Ms. Cypher, these binary digits represent letters?

Well, let's start by splitting the first eight binary digits into four-digit sets. Then we'll convert them into hexadecimal notation. When you split a byte, the left four digits are called the higher-order bits and the right four are the lower-order bits.

0011/0001

\[ \downarrow \quad \downarrow \]

3 1

31

What does that mean?

The hexadecimal notation 0x31 can be converted to a letter using a character-encoding system called ASCII, which is used by many computers. As you can see in the following table, 0x31 represents the number 1.

---

**Lower-order bits**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>sp</td>
<td>0</td>
<td>G</td>
<td>p</td>
<td>q</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>~</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
</tr>
<tr>
<td>4</td>
<td>ETX</td>
<td>DC4</td>
<td>$</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAE</td>
<td>%</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>^</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>_</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>0</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>1</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
<td>SUB</td>
<td>2</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
<td>ESC</td>
<td>3</td>
<td>K</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
<td>FS</td>
<td>4</td>
<td>L</td>
<td>|</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
<td>GS</td>
<td>5</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
<td>RS</td>
<td>6</td>
<td>N</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>US</td>
<td>7</td>
<td>O</td>
<td>del</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

54 Chapter 2: Symmetric-Key Algorithms
SO IF WE CONVERT THE REST OF THE CODE MS. CYPHER LEFT, WE GET...

<table>
<thead>
<tr>
<th>Binary number</th>
<th>Hexadecimal number</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>00110001</td>
<td>31</td>
<td>1</td>
</tr>
<tr>
<td>00101011</td>
<td>2B</td>
<td>+</td>
</tr>
<tr>
<td>00110001</td>
<td>31</td>
<td>1</td>
</tr>
<tr>
<td>00111101</td>
<td>3D</td>
<td>=</td>
</tr>
<tr>
<td>00110000</td>
<td>30</td>
<td>0</td>
</tr>
</tbody>
</table>

1 + 1 = 0

BUT DOESN'T 1 + 1 = 2?

IS MS. CYPHER AN IDIOT?

OF COURSE SHE'S NOT!

THIS IS AN XOR OPERATION...

ALSO CALLED AN EXCLUSIVE OR OPERATION.

IT'S A LOGICAL OPERATION USED IN CRYPTOGRAPHY.

WAAA!

XO SAUCE IS A CHINESE CONDIMIENT.

SOAR?

SORE?

NOT SO FAST!

* XO SAUCE IS A CHINESE CONDIMIENT.
MS. CYPER HAS USED A BITWISE OPERATION, WHICH IS A CALCULATION THAT WORKS ONLY ON BINARY NUMBERS 0 AND 1. ALL COMPUTER CALCULATIONS ARE BITWISE OPERATIONS!

HERE'S A FIGURE TO ILLUSTRATE. A AND B ARE INPUTS FOR THE BITWISE OPERATIONS, AND FOR EACH OPERATION, WE GET AN OUTPUT THAT IS A BITWISE NUMBER (EITHER 1 OR 0).

AND operations are sometimes referred to as conjunction logic, and OR operations are sometimes referred to as disjunction logic. XOR operations are a combination of conjunction and disjunction logic.

<table>
<thead>
<tr>
<th>OR operation</th>
<th>AND operation</th>
<th>NOT operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A + B )</td>
<td>( A \cdot B )</td>
<td>( \overline{A} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 0 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

If either A or B is a 1, the output is 1.

If both A and B are 1, the output is 1.

The output is the opposite of A. If A is 1, the output is 0; if A is 0, the output is 1.

NAND operation | NOR operation | XOR operation |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( A \oplus B )</td>
<td>( \overline{A + B} )</td>
<td>( \overline{A + B} \oplus A \overline{B} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 0 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

If either A or B is 0, the output is 1; otherwise it is 0.

If both A and B are 0, the output is 1; otherwise it is 0.

If A and B are different, the output is 1; otherwise it is 0.

So...

When you use the XOR operation on two different values, the output is 1. If the input values are the same, the output is 0.

The symbol for XOR is $\oplus$ and is used like this: $1 \oplus 0 = 1, 1 \oplus 1 = 0$.

Okay, but how is this useful?
As an example, let's perform an XOR operation using the plaintext 1101 and the encryption key 1001.

\[
1101 \oplus 1001 = 0100
\]

Plain text  Encryption key

The result of the operation is the ciphertext 0100. Next, we'll perform an XOR operation using the ciphertext 0100 and the decryption key 1001.

\[
0100 \oplus 1001 = 1101
\]

Ciphertext  Decryption key

The result of the operation is the plaintext 1101. When you perform the XOR operation using the ciphertext 0100 and the plaintext 1101, you get the encryption/decryption key:

\[
0100 \oplus 1101 = 1001
\]

Ciphertext  Plaintext  Decryption key = Encryption key

This means that as long as we have two of the three pieces of data—the plaintext, encryption/decryption key, or ciphertext—we can derive the remaining piece of data.
AND THAT MEANS...?

RIGHT! YOU USE AN XOR OPERATION TO PERFORM BOTH ENCRYPTION AND DECRYPTION!

**ENCRYPTION**

Plaintext + Encryption key = Ciphertext

I LOVE YOU! + → = SERIYA

**DECryption**

Ciphertext + Decryption key = Plaintext

SERIYA + → = I LOVE YOU!

Encryption key = Decryption key

indeed!

AND IF WE USE THE SUBSTITUTION CIPHER AND PERMUTATION, WHICH WERE INTRODUCED IN CHAPTER 1, AND THEN USE AN XOR OPERATION...

**Symmetric-key algorithm** = Substitution cipher process + Permutation process + XOR operation

WE CAN IMPLEMENT A SYMMETRIC-KEY ALGORITHM, WHICH IS A MODERN ENCRYPTION TECHNIQUE!

**Symmetric-key?**

IT'S A TYPE OF ALGORITHM THAT USES THE SAME KEY FOR ENCRYPTION AND DECRYPTION.

SO NOW WE JUST NEED TO LEARN ABOUT SYMMETRIC-KEY ALGORITHMS!

YOU'LL GET THE HANG OF IT.

I HOPE I CAN KEEP UP.
Symmetric-Key Algorithms

This is what a symmetric-key algorithm looks like!

Sender

It won't open!

Eavesdropper

Recipient

The cipher's sender and recipient use the same key...

Hmmm...

That's unknown to anyone else to perform both encryption and decryption.

Symmetric-Key Algorithms 59
Because both parties share the same key, symmetric-key cryptography is also known as shared-key cryptography or secret-key cryptography. Most traditional encryption schemes use a shared key.

If three people are communicating using a symmetric-key algorithm...

It's three, right?

How many keys do you think they would need?

Ding! Ding!

Big brother!

If four people are using a symmetric-key algorithm, how many keys would they need?

Umm...
IF THREE PEOPLE NEED THREE KEYS...

WOMP WOOOOOMP.

FOUR PEOPLE WOULD NEED FOUR, RIGHT?

IT'S AN EASY MISTAKE TO MAKE, BUT THE ANSWER IS SIX.

WHEN n USERS ARE USING SHARED KEYS TO TRANSMIT ENCRYPTED CORRESPONDENCE...

\[ n \binom{2}{2} = \frac{n(n-1)}{2} \]

YOU CAN CALCULATE THE NUMBER OF KEYS WITH THIS FORMULA.
Oooh, so if there were 100 people...

Umm...

R—really?

I'm good at mental math!

You'd need 4,950 keys!

Calculate \( \binom{100}{2} \) and

\[
\frac{100 \times (100 - 1)}{2} = 4950
\]

Is what you get!

Ahh—That was delicious!

Here's your bill.

Yes, I'd like to pay.

So could you encrypt and send a secret email using a symmetric-key algorithm?
YES, BUT DELIVERING THE SHARED KEY TO THE PERSON YOU'RE COMMUNICATING WITH IS AN ISSUE.

CAN'T YOU JUST SEND IT IN THE EMAIL?

IF THE KEY IS STOLEN, IT'S USELESS.

THEN WHAT IF YOU ENCRYPT THAT KEY WITH ANOTHER SHARED KEY?

HOW WOULD YOU DELIVER THAT OTHER KEY?

AND, UM...

Encrypt the other key using yet another key...

Then encrypt that key yet again?

But then what would I do with that key?

By email...
In the end, would you share the key in person?
The safest option would be to have someone you trust deliver it, right?

But think about it: you'd have trouble delivering the shared key to everyone involved in a communication.

So we don't! Instead, public-key cryptography uses two keys: a public key that can be distributed to everyone and a private key that is kept by one person.

We'll learn about that in Chapter 3!

Features of a symmetric-key algorithm

- To keep the key secure, users must be careful how they store and exchange it.
- Usually, encryption and decryption are fast, so it's convenient for large quantities of data.
- Because users would have to create a large number of keys, it isn't suited for communications among large groups.

For now, we'll focus on symmetric-key algorithms.

But many communication methods use a combination of symmetric-key and public-key encryption.

* Public-key encryption uses separate but mathematically paired keys for encryption and decryption. This is different from symmetric-key encryption, where the same key is used to both encrypt and decrypt a message.
There are two main types of symmetric-key algorithms:

Stream Ciphers

Block Ciphers

What's the difference?

A stream cipher encodes each bit or byte individually in succession.

A block cipher splits plaintext and cryptographic data into sections of uniform lengths (blocks). Then it performs encryption and decryption one block at a time.

I'll explain that in a moment.

First, let's look at stream ciphers.
In a stream cipher, the key is a long, pseudorandom sequence of numbers. You can encrypt or decrypt with the key by performing XOR operations on each bit or byte data, as shown in the following figure. Stream ciphers are generally less computationally intensive than block ciphers. Some examples of stream ciphers include RC4 and SEAL.

![Diagram showing the process of stream cipher encryption](image-url)
What's a pseudorandom number sequence?

It's a series of numbers that appear completely random.

Incidentally, both stream and block ciphers are vulnerable to brute-force attacks, in which an attacker can uncover the correct key by trying all possible keys.

This means that security is guaranteed only up to a point, depending on how much computational power an attacker has.

Can't you just make a truly random sequence?

If you made the key a string of random numbers that's as long as or longer than the plaintext, you could use a one-time pad to have a secure cipher.

Yes, but it's impractical to create a key that large and to generate a truly random number sequence.

Hey! Where did Big Brother go?

Now that you mention it...

Whee!
Unlike a stream cipher, which encrypts each bit in succession, a block cipher splits data into blocks of uniform length and then encrypts each block, as shown here.

The length of a given block depends on the structure of the cipher used. Block sizes of 64, 128, or 256 bits are the most common for ciphers. Because 1 byte is composed of 8 contiguous bits, it would be reasonable to use 64-bit blocks because you could use 8 characters to satisfy the block size. Using 64-bit blocks allows for faster calculations on a computer.

<table>
<thead>
<tr>
<th>Name of cipher</th>
<th>Block length (bits)</th>
<th>Key length (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>AES</td>
<td>128</td>
<td>128 192 256</td>
</tr>
</tbody>
</table>
Here's how you would create a block with a length of 64 bits.

Remember: digits that have the prefix 0x are numbers in hexadecimal notation.

The value 0x08 is added to the end of the block so that the block length is 64 bits (8 bytes). The padding is 0x08 because that is the amount of padding that should be removed when the message is decrypted. This is just one way to pad blocks.
NO MATTER WHAT THE PLAINTEXT IS, IT CAN BE ADJUSTED TO BE A MULTIPLE OF THE BLOCK LENGTH.

padding

WHEN THE PLAINTEXT IS LONGER AND THERE ARE MORE BLOCKS...

DO YOU ENCRYPT THEM ALL INDIVIDUALLY?

THAT'S ONE WAY OF DEALING WITH THEM!

ECB: ELECTRIC CODE BOOK

THE WAY A BLOCK CIPHER IS ENCRYPTED OR DECRYPTED DEPENDS ON ITS MODE OF OPERATION, WHICH IS THE ALGORITHM WE USE FOR THE CIPHER.

ECB MODE TREATS EACH BLOCK AS AN INDEPENDENT UNIT AND ENCRYPTS AND DECRYPTS EACH BLOCK INDIVIDUALLY.

BUT IF THE PLAINTEXT INCLUDES IDENTICAL BLOCKS...

WON'T THE CIPHERTEXT ALSO HAVE IDENTICAL BLOCKS AND BE INSECURE?

FOR EXAMPLE, IF THE PHRASE "MY LOVE," APPEARS TWICE IN THE PLAINTEXT...

MY LOVE, MY LOVE;

Encryption

"/\x6b6b" /\x6b6b"

THEN THE CIPHERTEXT "/\x6b6b" IS REPEATED. THIS WOULD INADVERTENTLY PROVIDE A CLUE FOR CRACKING THE CIPHER, WOULDN'T IT?

CBC: CIPHER BLOCK CHAINING

RIGHT! TO COMBAT THIS, WE USE A MORE SECURE MODE CALLED CIPHER BLOCK CHAINING.

WHAT ON EARTH IS THAT?
CBC MODE

CBC mode outputs different ciphertexts even in cases where the plaintext is the same.

Once the plaintext is cut into blocks, the first block is XORed with an initialization vector (an additional cipher block, which is a value known to everyone), and then it's encrypted using a block cipher with a key. The next plaintext block is XORed with the most recently encrypted block and then also encrypted using the block cipher with a key.

This process is done on all of the blocks, making it practically impossible for any connection to be made between blocks. In the following figure, the previous ciphertext block $C_{i-1}$ and the next plaintext block $P_i$ are XORed to produce $X_i$. In other words, $X_i$ is calculated by using the following formula:

$$X_i = C_{i-1} \oplus P_i$$

With this method, even if there are blocks of identical plaintext data, there are no identical blocks in the ciphertext.

CBC and ECB are two of the most well-known modes of operation. ECB is no longer used because it is insecure. Other common modes are OFB (output feedback) and CFB (cipher feedback).

THIS IS GETTING TOUGH.
The first encryption standard that was widely adopted for business purposes was DES...

But it was actually based on another cryptosystem!

The first standardized civilian encryption system was DES (Data Encryption Standard). It was based on an earlier system, the Lucifer cipher, which was developed in the early 1970s by Horst Feistel at IBM.

Cool!

Can you give me an example of how DES works?

Sure!

Let's start by taking a look at Feistel ciphers, which DES was based on. The ciphers you've seen so far have been simple...

With just a few steps performed on each encrypted block, in practice, ciphers perform a longer series of steps.
THE BASIC CONFIGURATION OF A FEISTEL CIPHER

Each set of steps is called a round, and rounds are performed multiple times on a block. The series of steps performed each round is the round function.

The Feistel cipher uses multiple rounds and subkeys that are derived from one initial key. The following figure outlines Feistel's method of encryption; we'll discuss the details of each round and how the subkeys are created later.

1. Divide the plaintext into blocks of 64 bits.
2. Perform an initial permutation (IP) using an algorithm for each 64-bit block that substitutes and rearranges the bits so that even bits are in the left half of the block and odd bits are in the right half. The block is split in half so that the 32 even bits now on the left side are $R_0$, and the left 32 odd bits are $L_0$.
3. Use the round function $f$ with the encryption key $K$. We'll go into more detail about $f$ later. For these examples, you just need to know that $f$ is an algorithm that rearranges and substitutes bits in a block.
4. Perform an XOR operation on $R_0$ and $L_0$ to create a new right 32-bit block ($R_1$) to replace the original right block. Take the original $R_0$ and make it into the 32-bit block on the left ($L_1$).
5. After completing one round of steps 3 and 4, repeat these steps for rounds 2 through 15.
6. Reconstruct the left 32-bit block ($L_{16}$) and the right 32-bit block ($R_{16}$) into one 64-bit block.
7. Finally, perform a reversal of the initial permutation (IP'), and one block of encryption is complete.

The structure of DES is similar to this!
INVOLUTION

An involution is a function that is its own inverse, which means that if you apply the function twice, you will return the input to its original state. For example, consider a function in which 1 becomes 4, 2 becomes 3, 3 becomes 2, and 4 becomes 1. The values in the list after the conversion are the same as the values before.

<table>
<thead>
<tr>
<th>First conversion</th>
<th>Second conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

By applying the function twice, we return to all of the original values in the same order.
**DES Cipher**

**Is DES the same as the Lucifer Cipher?**

No, Lucifer was an earlier Feistel network.

DES has a standard key size of 64 bits, while Lucifer had variants from 48 to 128 bits.

In fact, the DES key size is 64 bits, but it actually uses a key of only 56 bits. The remaining 8 bits were designed to perform a parity check.

A parity check detects data errors that arise from noise, reading mistakes, and so on. In the key, it is no longer used in modern ciphers, though.

**But the longer the key is, the more secure the cipher will be, right?**

The National Security Agency (NSA) determines the standards, and it restricted the number of key choices.

The key choices have been limited to fewer than 100 quadrillion* keys.

So we only have 56 bits ($2^{56}$).

---

* One hundred million = $10^8$, one trillion = $10^{12}$, ten quadrillion = $10^{15}$, one hundred quintillion = $10^{24}$
It's because if the number of key choices becomes too large...

The NSA won't be able to intercept ciphertexts, right?

Probably.

The NSA does engage in global intelligence.

It's constantly cracking ciphers and probing important classified information.

Shudder.

Oh... scary.

But as I was saying, in DES, subkeys are derived from the encryption key...

And then used as inputs in the round function $f$. 
GENERATING DES ENCRYPTION SUBKEYS

In DES, a different key is used for each round of encryption, so 16 subkeys—$K_1, K_2, K_3, \ldots, K_{16}$—need to be created from the encryption key for each of the 16 rounds. This is the method used to derive subkeys:

1. A permutation called permuted choice 1 (PC-1) is performed on the 56 non-parity bits of the initial key. Permuted choice 1 is a special permutation method that transposes each of the 56 bits into specific positions. We won't cover the details of it here.

2. The 56 bits are divided into a 28-bit left block ($C_0$) and a 28-bit right block ($D_0$).

3. The bits in $C_0$ and $D_0$ are left-circular shifted to create $C_1$ and $D_1$. A left-circular shift moves every bit a certain number of places to the left, and the bits at the beginning of the block are shifted to the very end, as though the block's beginning and end were connected as in a circle.

4. $C_1$ and $D_1$ are joined. Then subkey $K_1$ is made through another permutation, permuted choice 2 (PC-2). PC-2 applies all 64 bits of $C_1$ and $D_1$ except for the 8 parity bits, resulting in a compressed 48-bit subkey.

5. Steps 3 and 4 are repeated for each round to generate each subkey $K_1$ through $K_{16}$. Each of these subkeys will then be used for each round in DES.

To generate the decryption subkeys, the bits are right-circular shifted instead. When the initial key is used for decryption, the subkeys are obtained in reverse order from $K_{16}$ to $K_1$. 

DES CIPHERS 77
THE DES ROUND FUNCTION \( f \)

In a round function, you use an S-box (substitution box) to perform substitutions on an input to make a new output. Each S-box in this diagram contains a different permutation for creating the substitutions.

1. The DES function only works on 48-bit blocks, so the rightmost block of data, which is only 32 bits, needs to be expanded using a 48-bit expansion permutation \( E \). This results in the output \( ER_{n-1} \), which is a 48-bit block.

2. Perform an XOR operation on the data and the subkey.

3. Separate the result of the operation into eight sets of 6-bit blocks each.

4. Substitute each 6-bit set of data with 4 bits using S-boxes 1 through 8.

5. Combine the S-box output data sequentially to produce a 32-bit block.

6. Finally, apply permutation \( P \) to the data to yield the output of function \( f \).
THE FULL STRUCTURE OF DES ENCRYPTION AND DECRYPTION

The full DES encryption and decryption processes with all the steps we've covered are shown here. The plaintext encryption process and the cipher decryption process are opposite each other.

The number inside each 1... indicates the block size in bits.

IN THE EARLY 1970s...

LUCIFER INVOLVED A BLOCK LENGTH OF 64 BITS AND A KEY SIZE OF 112 BITS.
3-DES ENCRYPTION AND AES ENCRYPTION

But once I review it...

I'll undoubtedly master it!

Yeah!

That's the spirit!

See?

Pat pat

First you'll have to deepen your understanding of the simplified version, so keep at it!

Blush

But can DES still be used safely now? It's been around since the 1970s!
### Yes... DES has become insecure due to advancement in computers and now can be easily cracked.

### The Drawbacks of DES

- **Short Key Size:** The shorter the key, the smaller the number of possible keys available and the more feasible it is to uncover the secret key.
- **Outdated Design:** The structural design of DES is considered obsolete and is no longer used. Nonrigorous implementations of cryptography commonly appear on the market and are easier to crack than already established ones.

### How is it done?

- **Slow**
- **Absolutely...**
- **Impossible.**
- **Can even I do it?**

### Here are a few methods for cracking a block cipher!

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exhaustive search algorithm</strong></td>
<td>Searching for the key by trying every possible key (brute forcing)</td>
</tr>
<tr>
<td><strong>Differential cryptanalysis</strong></td>
<td>Searching for vulnerabilities in the cipher by looking for any correlation between its inputs and outputs</td>
</tr>
<tr>
<td><strong>Linear cryptanalysis</strong></td>
<td>Searching for vulnerabilities in the cipher by looking for mathematical weaknesses in the cipher structure</td>
</tr>
</tbody>
</table>
IN THE 1990s, DES ENCRYPTION...

BECAME CRACKABLE USING COMPUTING POWER TO BRUTE-FORCE THE KEY.

HUM? BUT I WORKED SO HARD TO LEARN IT!

IT'S HOPELESS!

GAAAAAHHH!

THANKFULLY, 3-DES TOOK ITS PLACE!

3-DES IS ALSO KNOWN AS TRIPLE-DES.

BASED ON THE NAME, IT USES DES, RIGHT?

EXACTLY. IT PERFORMS ENCRYPTION BY USING THREE SHARED KEYS. THE FIRST ENCRYPTS THE PLAINTEXT, THE SECOND DECRYPTS, AND THE THIRD Encrypts AGAIN.

Plaintext

DES encryption

DES ciphertext 1

DES decryption

DES ciphertext 2

DES encryption

DES ciphertext 3

Key 1

[56-bit encryption key]

Key 2

[56-bit decryption key]

Key 3

[56-bit encryption key]
When the first encryption and decryption rounds use the same key:

The result is the same as DES no matter what key is used for the last encryption. That's because Ciphertext 2 would end up the same as the plaintext. In other words, you're only encrypting the plaintext once.

If you use the same key in both rounds of encryption:

And a different key for decryption, you end up with a key size of 56 x 2 = 112 bits!

112 bits

If the first two keys or all three keys are identical, the real key size is 56 bits.

This 3-DES formula is known as EDE (Encrypt-Decrypt-Encrypt) mode.

168 bits!

If you use different keys throughout, you end up with a key size of 56 x 3 = 168 bits.
In 1997, the US National Institute of Standards and Technology made a public call for a better algorithm that would become the new world standard for encryption. This algorithm would eventually be called AES.

AES stands for Advanced Encryption Standard.

IBM, NTT, and other world enterprises and organizations entered submissions.

The winning submission was...

The Rijndael Algorithm!

Now, it's used to encrypt the American Government's confidential documents!
AN OUTLINE OF AES

In 2000, Rijndael was renamed AES and published as a Federal Information Processing Standard (FIPS). The name Rijndael is derived from the names of its developers, Joan Daemen and Vincent Rijmen, researchers at Belgium’s Catholic University of Leuven.

As shown in the following table, in AES there are three possible key sizes.

<table>
<thead>
<tr>
<th>Type</th>
<th>Key size (bits)</th>
<th>Block size (bits)</th>
<th>Number of rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>128</td>
<td>128</td>
<td>10</td>
</tr>
<tr>
<td>AES-192</td>
<td>192</td>
<td>128</td>
<td>12</td>
</tr>
<tr>
<td>AES-256</td>
<td>256</td>
<td>128</td>
<td>14</td>
</tr>
</tbody>
</table>

The encryption strength increases with respect to the key size and the number of operations required. The block cipher structure of AES doesn’t use a Feistel network but instead uses an SPN (substitution-permutation network). To give a superficial explanation of what an SPN is, imagine the substitution portion as an XOR operation that is performed on each block and subkey while the permutation is some function that simply scrambles the input bits. These two are performed simultaneously, usually for some number of rounds (just as in a Feistel network).

From here on out, we’ll leave DES behind and discuss AES.
A year or so?

It would take trillions of years!

Though the structure of the S-box is known and researchers have examined it to find vulnerabilities...

AES remains secure from cryptanalysis attacks.

Then AES will be secure even in the future, huh?

Well, the techniques used in cryptanalysis are also evolving!

There are some who say that AES may only be secure for another 10 years or so. That's just speculation, though.
HMM...

SO, BIG BROTHER...

SINCE I'VE TAUGHT YOU SO MUCH ABOUT ENCRYPTION, I'M GRADUALLY EARNING THAT COMPUTER, RIGHT?

AHHH...

I'VE BEEN THINKING THE SAME THING AND MADE SOME ARRANGEMENTS...

ホント!?

REALLY!?

OOOOH

AHHHH!

HERE!
This computer is way more useful than an abacus!

Isn't it great?!

Tremble tremble

This is a calculator!

I want a computer!

You're the worst brother ever!

Now then, read the following explanation of simplified DES and try out some practical applications of encryption and decryption!

Aghh

In Chapter 3, the key that has been secret until now will be revealed, so enjoy!
Simplified DES Encryption and Decryption

How are encryption and decryption performed in DES? We will work on a reduced version of the DES algorithm to get an idea of what real DES is like. The adaptation of DES that we'll use here to help you solidify your understanding of the concepts imitates DES but with fewer steps and smaller block sizes (input/output sizes).

Converting Data into Binary

Because modern-day algorithms—including DES and others—deal with binary data, we need to be able to convert plaintext messages composed of letters and numbers into binary. Table 2-1 illustrates 16 characters converted to 4-bit binary numbers.

<table>
<thead>
<tr>
<th>Characters</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0000</td>
</tr>
<tr>
<td>B</td>
<td>0001</td>
</tr>
<tr>
<td>C</td>
<td>0010</td>
</tr>
<tr>
<td>D</td>
<td>0011</td>
</tr>
<tr>
<td>E</td>
<td>0100</td>
</tr>
<tr>
<td>F</td>
<td>0101</td>
</tr>
<tr>
<td>G</td>
<td>0110</td>
</tr>
<tr>
<td>H</td>
<td>0111</td>
</tr>
<tr>
<td>I</td>
<td>1000</td>
</tr>
<tr>
<td>J</td>
<td>1001</td>
</tr>
<tr>
<td>K</td>
<td>1010</td>
</tr>
<tr>
<td>L</td>
<td>1011</td>
</tr>
<tr>
<td>M</td>
<td>1100</td>
</tr>
<tr>
<td>N</td>
<td>1101</td>
</tr>
<tr>
<td>O</td>
<td>1110</td>
</tr>
<tr>
<td>discarded character</td>
<td>1111</td>
</tr>
</tbody>
</table>

The encoding in Table 2-1 is a simple conversion we will use for the purposes of this book and is not based on any particular encoding used in cryptography. Likewise, the following conversions do not reflect actual binary representations of letters and numbers done in practice (which we saw earlier with ASCII encoding), but rather serve as simple examples of how characters may be converted to binary numbers.
Generating a DES Ciphertext

In DES encryption, a block is 64 bits. However, for the purposes of this book, we will use an 8-bit block in our reduced DES cipher example, as shown in Figure 2-1. Our reduced DES cipher algorithm is composed of two steps—a key schedule (which generates a larger key from the key we use) and actual encryption using the generated keys. DES is not the only block cipher to require a key schedule procedure. It is common for block ciphers to perform a key schedule prior to or in parallel with encryption and decryption.

Figure 2-1: Simplified process of generating a DES ciphertext (excluding the key scheduler)
Though it is not shown in Figure 2-1, the plaintext we want to encrypt is first converted into 0s and 1s, using Table 2-1, so that we can work with binary data.

The two binary keys we’ll use to perform reduced DES are as follows:

\[ K_1 = (110001)_2, \ K_2 = (111000)_2 \]

**NOTE:** The notations \( (\_)_2 \) and \( (\_)_n \) represent binary digits and decimal numbers, respectively.

These keys have already been generated through the key schedule. We’ll discuss how a key is generated through the key schedule later; for now, let’s focus on the encryption process of truncated DES.

We’ll encrypt the character string “MC” into a ciphertext using the truncated version of DES, so we’ll start by expressing these two inputted characters as a block of 8 bits. Based on Table 2-1, “MC” in binary is expressed as 11000010.

**PERFORM THE INITIAL PERMUTATION**

We first convert the plaintext to binary, and then we perform the initial permutation on the binary data according to Table 2-2. IP permutes the 8-bit binary data by rearranging the input bit positions.

<table>
<thead>
<tr>
<th>Input bits (original position)</th>
<th>( b_1 )</th>
<th>( b_2 )</th>
<th>( b_3 )</th>
<th>( b_4 )</th>
<th>( b_5 )</th>
<th>( b_6 )</th>
<th>( b_7 )</th>
<th>( b_8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output bits (using original input positions)</td>
<td>( b_2 )</td>
<td>( b_4 )</td>
<td>( b_6 )</td>
<td>( b_8 )</td>
<td>( b_3 )</td>
<td>( b_5 )</td>
<td>( b_7 )</td>
<td>( b_1 )</td>
</tr>
</tbody>
</table>

In Table 2-2, the subscript of each bit indicates its original position, so these numbers are in order in the first row. Then the bit is translated, so for example, the first bit of input data \( (b_1) \) is transposed to be the fifth bit of output data, and the second bit of input data \( (b_2) \) is transposed to be the first bit of the output data. Figure 2-2 shows “MC”, represented in binary, undergoing the initial permutation.

**Figure 2-2: Initial permutation of the plaintext “MC”**
SEPARATE DATA INTO HIGHER- AND LOWER-ORDER BITS

We separate the permutation's output data into higher-order and lower-order bits. The higher-order bits are denoted \( L_0 \), and the lower-order bits are denoted \( R_0 \). Based on Figure 2-2, this yields the following:

\[
\begin{align*}
L_0 & = (1000)_2 \\
R_0 & = (1001)_2
\end{align*}
\]

FEISTEL ROUNDS VARIANT

Now we begin going through rounds in a Feistel network. The Feistel network performs a specific set of steps that "mix bits" to create a new \( L \) and \( R \). Since we're using a reduced form of DES, we'll mix the bits using a less rigorous method than with real DES. (In real DES, the network mixes the bits in a way that ensures pseudorandomness.) The following represents what a round of the Feistel network looks like in notation form:

\[
\begin{align*}
L_i & = R_{i-1} \\
R_i & = L_{i-1} \oplus f(K_i, R_{i-1})
\end{align*}
\]

Each generation of a new \( L \) and \( R \) is a round, and \( i \) is equal to the round number. The process represented in Figure 2-2 is one round.

\( l_i \) will be equal to \( R_0 \), but the process to derive \( R_1 \) requires much more work; \( R_2 \) and key \( K_1 \) are input into function \( f \). In our reduced DES, there will be only 2 rounds, but real DES has 16 rounds.

Let's walk through how \( f(K_i, R_i) \) works.

ROUND 1

The first operation in \( f \) is to apply the expansion function \( E \) from Table 2-3 on \( R_0 \).

**Table 2-3: Expansion Function E**

<table>
<thead>
<tr>
<th>Input bits (original position)</th>
<th>( b_1 )</th>
<th>( b_2 )</th>
<th>( b_3 )</th>
<th>( b_4 )</th>
<th>( b_5 )</th>
<th>( b_6 )</th>
<th>( b_7 )</th>
<th>( b_8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output bits (using original input positions)</td>
<td>( b_1 )</td>
<td>( b_2 )</td>
<td>( b_3 )</td>
<td>( b_4 )</td>
<td>( b_5 )</td>
<td>( b_6 )</td>
<td>( b_7 )</td>
<td>( b_8 )</td>
</tr>
</tbody>
</table>

This means that we duplicate the third and fourth bits of \( R_0 \) and append them to the beginning of \( R_0 \) to form a new \( R_0 \) that is 8 bits long:

\[
R_0 = (1001)_2 \oplus \underbrace{(00000000)}_{\text{3 times}} = (011001)_2
\]

We next XOR the expansion permutation \( R_0 \) and the key \( K_1 = (110001)_2 \), as shown in Figure 2-3.
The process shown in Figure 2-3 can be represented by the following equation:

\[ E(R_0) \bigoplus K_1 = (011001)_2 \bigoplus (110001)_2 = (101000)_2 \]

Now we put \( R_0 \bigoplus K_1 \) through the substitution box (S-box) in Table 2-4, which compresses and substitutes the bits from \( R_0 \bigoplus K_1 \) to create a 4-bit output. We'll label the bits from \( R_0 \bigoplus K_1 \) as \( b_1, b_2, b_3, b_4 \).

**Table 2-4: Substitution Box (S-box)**

<table>
<thead>
<tr>
<th>Row numbers</th>
<th>Column numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td>1</td>
<td>0 15 7</td>
</tr>
<tr>
<td>2</td>
<td>1 14 8</td>
</tr>
<tr>
<td>3</td>
<td>15 12 8</td>
</tr>
</tbody>
</table>

In the S-box shown in Table 2-4, four substitutions are available and labeled with row numbers starting with 0 and going up to 3. We determine which substitution to use based on the value of the bits \( b_1, b_2, b_3, b_4 \). We convert the first and last bits, \( b_1 \) and \( b_4 \), to their base-10 equivalent and use those two numbers to determine which row of the S-box we use. We then also convert the remaining bits, \( b_2, b_3 \), to base 10 to determine which column we will use.

For example, in \((101000)_2\), we select row 2 because \( b_1 \) is 1 and \( b_4 \) is 0 and \((10)_2 = (2)_{10}\). Then, bits \( b_2, b_3 \) are also converted to base 10 to determine the column number we select. Since \((011000)_2 = (4)_{10}\), we select column 4 in this case, which contains the value \((13)_{10}\) (the position that is boxed in Table 2-4). After we find the corresponding table entry, we convert it into binary: \((1101)_2\).
The resulting binary, \((1100)_2\), then undergoes the permutation \(PS\) (Table 2-5).

<table>
<thead>
<tr>
<th>Input bits (original position)</th>
<th>(b_1)</th>
<th>(b_2)</th>
<th>(b_3)</th>
<th>(b_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output bits (using original input positions)</td>
<td>(b_1)</td>
<td>(b_4)</td>
<td>(b_3)</td>
<td>(b_2)</td>
</tr>
</tbody>
</table>

When the permutation \(PS\) from Table 2-5 is applied to our working example, the process looks like Figure 2-4.

\[ R_0 \text{ xor } K_1 \rightarrow \begin{array}{cccc} 1 & 0 & 1 & 0 & 0 \end{array} \]

\[ S\text{-box } [R_0 \text{ xor } K_1] \rightarrow \]

\[ \begin{array}{cccc} 1 & 1 & 0 & 1 \end{array} \rightarrow \begin{array}{cccc} 0 & 1 & 1 & 1 \end{array} \]

\[ f(R_0, K_1) \rightarrow \begin{array}{cccc} 0 & 1 & 1 & 1 \end{array} \]

**Figure 2-4: The calculation process of PS**

The result is \((1111)_2\). We've now finished the permutation process. That is to say, we've done all the steps involved in the function \(f\). All of these steps together are written with this notation:

\[ f(K_1, R_0) = (0111)_2 \]

This is the last step in the first round of the Feistel network (see Figure 2-1). In this step, we switch the sides of the bits to create \(L_1\) and \(R_1\). The four lower-order bits on the right side \((R_0)\) that we got in “Separate Data into Higher- and Lower-Order Bits” on page 92 are assigned to \(L_1\) so that they become the higher-order bits. \(L_0\) is assigned to \(R_1\) and becomes the lower-order bits. We can summarize the flip by assigning \(L_1\) to the equation that produced \(K_1\) and assigning \(R_1\) to the equation in “Perform the Initial Permutation” on page 91. Using the equation from “Feistel Rounds Variant” on page 92, we will do the following:

\[ L_1 = R_0 \oplus (1001)_2 \]

\[ R_1 = L_0 \oplus f(K_1, R_0) = (1000)_2 \oplus (0111)_2 = (1111)_2 \]

In order to complete a second round, we repeat these steps on \(R_1\) and \(L_1\). Since the steps in round 2 are identical to those in round 1, we'll only show the calculations involved, without detailed explanations.
ROUND 2

The expansion function from Table 2-3 is applied to $R_i$:

$$R_i = (1111)_2 \rightarrow (1111011)_2$$

We then XOR $R_i$ with key $K_2 = (111000)_2$:

$$R_i \oplus K_2 = (1111011)_2 \oplus (111000)_2 = (000111)_2$$

Then we use the permutation in Table 2-5 to rearrange the position of the bits, resulting in this:

$$\begin{align*}
(0100)_2 \oplus (0001)_2
\end{align*}$$

We'll apply the equation we use to express XORing $R_i$ and $K_2$ and process the result with the S-box:

$$f(K_2, R_i) = (0001)_2$$

For the final step of round 2 (see Figure 2-1 again), we will do the following:

$$L_2 = R_i \oplus (1111)_2$$
$$R_2 = L_2 \oplus f(K_2, R_i) = (1001)_2 \oplus (0001)_2 = (1000)_2$$

SWAP THE HIGHER- AND LOWER-ORDER BITS

After round 2 is completed, we swap the higher-order bits $L_2$ and lower-order bits $R_2$ again to get $L_2'$ and $R_2'$:

$$L_2' = R_2 = (1000)_2$$
$$R_2' = L_2 = (1111)_2$$

This swap is shown in Figure 2-5.

![Figure 2-5: Swapping the higher- and lower-order bits](image)

PERFORM THE FINAL PERMUTATION

Based on the final permutation $FP$ in Table 2-6, we transpose each bit to a new position.
For example, if you look at Table 2-2, you'll see that the fifth input bit (b₅) becomes the seventh bit in the output. When the seventh bit (b₇) is processed in Table 2-6, it becomes the fifth bit in the output, which means the bit reverts to its initial position. Figure 2-6 shows the initial and final permutations.

**Figure 2-6: The process of putting an 8-bit block through IP and FP. The initial permutation IP and final permutation FP are inverses of one another.**

The process shown in Table 2-6 results in $L_{\alpha}$ and $R_{\alpha}$, which when combined result in our 8-bit ciphertext:

$L_{\alpha} = (1000)_2$

$R_{\alpha} = (1111)_2$

The resulting ciphertext is $(11101010)_2$, as shown in Figure 2-7.

**Figure 2-7: The resulting ciphertext from the final permutation FP**
DECYPHERING A DES CIPHERTEXT

Now, we'll try to decrypt the DES ciphertext from Figure 2-7 into its original plaintext. For decryption, you apply the same steps shown in Figure 2-1.

PERFORM THE INITIAL PERMUTATION

The ciphertext is made up of its left and right sides, \( L_0 \) and \( R_0 \). In the initial permutation, both sides of the ciphertext's bits (11101010) were rearranged according to the initial permutation chart in Table 2-2.

SEPARATE DATA INTO HIGHER- AND LOWER-ORDER BITS

The permuted bits are separated into four higher-order bits (left side) \( L_0 \) and four lower-order bits (right side) \( R_0 \). These bits are the same as \( L_1 \) and \( R_1 \). In equation form, this is the opposite of the final permutation when encrypting with DES:

\[
L_0 = (1000)_2 = (L_1') \\
R_0 = (1111)_2 = (R_1')
\]

Figure 2-8 shows the initial permutation and separation processes together.

![Diagram](image)

Figure 2-8: Output after applying the initial permutation IP to the ciphertext

ROUND 1

The first round of decryption inverts the last round of encryption, so we start by applying the expansion function \( E \) from Table 2-3 on \( R_0 \). The expansion function duplicates the first two bits of \( R_0 \) and attaches them to the end of \( R_0 \):

\[
R_0 = (1111)_2 \uplus (1111)_2 \oplus R_0 = (111111)_2
\]

We calculate the XOR of \( R_0 \) and key \( K_2 \). As a reminder, \( K_2 \) is \((110000)_2\).

\[
R_0 \oplus K_2 = (111111)_2 \oplus (110000)_2 = (001111)_2
\]

We then compress and substitute the result from the previous step \((000111)_2\) using the S-box in Table 2-4. We select the first and last bits of the result, \( (000111)_2 \); combine those two bits into one binary number, \( (01)_2 \);
and convert the number to decimal, \(01)_{2} = (1)_{10}\). We then convert the remaining four bits from the original binary number \((0011)_{2}\) to decimal, \((0011)_{2} = (3)_{10}\).

In Table 2-4, we find where the two decimal values intersect. The column number is 3 and the row is 1, so the intersecting value is \((4)_{10}\) (the position that is circled in Table 2-4). The intersecting value is converted into binary to produce \((0100)_{2}\). Then we use the permutation \(PS\) from Table 2-5 on this binary number, so \((0100)_{2}\) becomes \((0001)_{2}\). Finally, this function can be expressed as follows:

\[
f(K_{r}, R_{o}) = (0001)_{2}
\]

So far, we've found the output for the four higher-order bits \(L_{1}\) and calculated \(f(K_{r}, R_{o}) = (0001)_{2}\). With this information, we can also derive \(R_{1}\). We first take \(L_{1}\) and set it equal to \(R_{o}\). Then, we derive \(R_{1}\) by XORing \(L_{o}\) and \(f(K_{r}, R_{o})\).

We'll find the output for the four lower-order bits \(R_{1}\) as follows:

\[
L_{1} = R_{o} = (1111)_{2}
\]

\[
R_{1} = L_{o} \oplus f(K_{r}, R_{o}) = (1000)_{2} \oplus (0001)_{2} = (1001)_{2}
\]

Now that we have finished the first round, we'll perform the second round.

**ROUND 2**

As we did in all the previous rounds, we first find \(f(K_{r}, R_{o})\) by applying the expansion function \(E\) from Table 2-3 on \(R_{1}\):

\[
E = (1001)_{2} \oplus (011001)_{2}
\]

We calculate the XOR of \(R_{1}\) and the key \(K_{1}\). The key \(K_{1}\) is equal to \((110001)_{2}\).

\[
R_{1} \oplus K_{1} = (011001)_{2} \oplus (110001)_{2} = (101000)_{2}
\]

Then we put \(R_{1} \oplus K_{1}\) through the S-box using the permutation \(PS\) from Table 2-5. Doing so rearranges the position of the binary number's bits so that it becomes the following:

\[
(1101)_{2} \oplus (0111)_{2}
\]

Finally, this whole process is expressed in equation form as this:

\[
f(K_{r}, R_{o}) = (0111)_{2}
\]

In Figure 2-12, the output of round 2 is the four higher-order bits (left side) \(L_{2}\) and four lower-order bits (right side) \(R_{2}\), which are found using the following equations:

\[
L_{2} = R_{1} = (1001)_{2}
\]

\[
R_{2} = L_{1} \oplus f(K_{r}, R_{1}) = (1111)_{2} \oplus (0111)_{2} = (1000)_{2}
\]

\(L_{2}\) is set equal to \(R_{1}\), which in this case is the binary value 1001. \(R_{2}\) is found by XORing \(L_{1}\) and \(f(K_{r}, R_{1})\).
In the final stage of the process in Figure 2-1, we swap the higher-order bits, $L_{x'}$, and lower-order bits, $R_{x'}$, to find $L_{x''}$ and $R_{x''}$, as seen in Figure 2-9.

\[
\begin{array}{c|c}
L_x & R_x \\
\hline
1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
L_{x''} & R_{x''} \\
\end{array}
\]

**Figure 2-9: Swapping the left and right bits**

This swap is represented by the following equations:

\[
\begin{align*}
L_{x''} &= R_x = (1000)_2 \\
R_{x''} &= L_x = (1001)_2
\end{align*}
\]

$L_{x''}$ and $R_{x''}$ are found by performing the final permutation $FP$ from Table 2-6 on $L_{x'}$ and $R_{x'}$. Using Table 2-1, $L_{x''}$ is then converted from binary to the character “M”, and $R_{x''}$ is converted to “C”.

\[
\begin{align*}
L_{x''} &= (1100)_2 \\
R_{x''} &= (0010)_2
\end{align*}
\]

You can see this process in Figure 2-10.

**Figure 2-10: The final permutation performed on the bits of $L_{x''}$ and $R_{x''}$**

The plaintext is 1100 0010, which is the character string “MC”.

We can see that the resulting 8-bit output data corresponds with the original plaintext, so the ciphertext has been decrypted. As you can see, decryption is the same process as encryption, just with the order of keys reversed. The relationship between encryption and decryption processes is shown in Figure 2-11.
GENERATING DES KEYS

Now that we have seen how encryption and decryption are performed using reduced DES, let's discuss how keys are generated from the key schedule. Reduced DES requires an input of an 8-bit key $K_n$, and the key given by the user generates two 8-bit keys, which are then used in the encryption and decryption process.

$$K_n = (10011001)_2$$

The key schedule, which is the procedure for generating the keys $K_1$ and $K_2$, is shown in Figure 2-12.
Figure 2-12: Generating encryption keys and decryption keys

**PERFORM PERMUTED CHOICE PC-1**

The symmetric key $K_i$ is permuted according to PC-1.

$$K_i' = (00110101)_2$$

The bits are transposed, as shown in Table 2-7.

<table>
<thead>
<tr>
<th>TABLE 2-7: PERMUTED CHOICE PC-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input bits (original position)</td>
</tr>
<tr>
<td>Output bits (using original input positions)</td>
</tr>
<tr>
<td>b_2</td>
</tr>
</tbody>
</table>

**GENERATING DES KEYS**
When the transposition in Table 2-7 is applied to $K_0$, we acquire $K'_0$, as shown in Figure 2-13.

![Figure 2-13: Permutation choice PC-1](image)

Here, $K'_0$ is divided into $C_0$, which is made up of the four higher-order bits, and $D_0$, the key's four lower-order bits, and is expressed as follows:

\[
C_0 = (0011) \_{\text{b}} \\
D_0 = (0101) \_{\text{b}}
\]

**ROTATE THE BITS**

$C_0$ and $D_0$ are each rotated one bit to the left, as Figure 2-14 shows, and the results are expressed as $C_1$ and $D_1$. During round 1, bits are all shifted by one position.

\[
C_1 = (0110) \_{\text{b}} \\
D_1 = (1010) \_{\text{b}}
\]

The bits in $C_0$ and $D_0$ are rotated independently to produce $C_1$ and $D_1$, as shown in Figure 2-14.

![Figure 2-14: Rotating $C_0$ and $D_0$ one bit to the left](image)

**PERFORM PERMUTED CHOICE PC-2**

Based on permuted choice PC-2, $C_1$ and $D_1$ are compressed from 8 bits to 6 bits, yielding the key $K_1$, which is used in round 1 of encryption.

\[
K_1 = (110001) \_{\text{b}}
\]
Permuted choice PC-2 is shown in Table 2-8.

**TABLE 2-8: PERMUTED CHOICE PC-2**

<table>
<thead>
<tr>
<th>Position of input bits</th>
<th>$b_1$</th>
<th>$b_2$</th>
<th>$b_3$</th>
<th>$b_4$</th>
<th>$b_5$</th>
<th>$b_6$</th>
<th>$b_7$</th>
<th>$b_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output labeled with original input bits</td>
<td>$b_5$</td>
<td>$b_3$</td>
<td>$b_1$</td>
<td>$b_8$</td>
<td>$b_6$</td>
<td>$b_4$</td>
<td>$b_7$</td>
<td>$b_8$</td>
</tr>
</tbody>
</table>

Figure 2-15 shows the results of applying permuted choice PC-2 from Table 2-8 to $C_1$ and $D_1$.

**Figure 2-15: Permuted choice PC-2**

Repeating the key schedule steps will yield new keys to use for reduced DES.

**ROTATE THE BITS AGAIN**

Now we are going to generate $K_r$. Each bit of $C_1$ and $D_1$ is rotated to the left by two bits, as shown in Figure 2-16.

**Figure 2-16: Rotating $C_1$ and $D_1$ two bits to the left**

This results in $C_2$ and $D_2$:

- $C_2 = (1001)_2$
- $D_2 = (1010)_2$

**COMPRESS THE BITS**

Based on permuted choice PC-2 in Table 2-8, $C_2$ and $D_2$ are compressed from a total of 8 bits to 6 bits. This yields key $K_r$, which is used in the second round of encryption:

- $K_r = (111000)_2$
Figure 2-17 shows how permuted choice PC-2 compresses the bits of $C_2$ and $D_2$.

Figure 2-17: Using permuted choice PC-2 to produce $K_2$

**HOW REDUCED DES DIFFERS FROM REAL DES**

We've learned how reduced DES works. As mentioned at the beginning of this chapter, real DES used in practice is more complex.

**DES (and its reduced version) is no longer secure and should not be used in any practical implementations.**

Reduced DES is a modified DES that encrypts and decrypts only 8-bit messages using an 8-bit key, while real DES encrypts and decrypts 64-bit messages with a 56-bit key. This means that all of the procedures done in real DES use a different expansion function, $f$ function, and S-box to accommodate for the size of the inputs. In addition, real DES performs 16 rounds rather than the 2 rounds of the Feistel network. The key schedule is also different, because real DES generates 16 keys and performs only one circular bit shift rather than two.

Although we didn't cover real DES, this section should still give you an idea of just how involved, exhaustive, and particular the steps in modern-day encryption schemes are.
3

PUBLIC-KEY ENCRYPTION
THE BASICS OF PUBLIC-KEY ENCRYPTION

LOOKS LIKE YOU GOT YOUR LAPTOP.

WHEE!

THANKS, BIG BROTHER!

SHOOT, NOW I'M BROKE!

EMPTY

WE'RE HAVING NOTHING BUT RAMEN FOR A WHILE...

YOU LIVE OFF RAMEN ANYWAY...

106 CHAPTER 3 PUBLIC-KEY ENCRYPTION
When you have a computer, you can do all sorts of things...

My new best friend!❤

Email, shopping, games...

Wait a minute... I thought you were gonna use it to study?

Hehehe!

I also do a lot of online shopping! It's so convenient to pay by card!

Books
Music
Shoes

Isn't that dangerous? I mean, what if your credit card info is stolen?

That's why we use public-key encryption—to prevent your transactions from being intercepted!
Isn't it lovely?
Well... I guess.

Give me your credit card for a sec.

What do you plan to do?

I'll just enter the information...

Click click click...

And presto!

Wha—what have you done?

Check it out!
You can view the public key used for encrypting messages to a website you are currently communicating with in your web browser. If you’re using Google Chrome, click the lock symbol in the address bar, and then click the Valid option under the Certificate heading. Switch to the Details tab and select Public key. You would also click the lock symbol in Firefox and Safari to view your SSL certificate.

WHOA! THAT’S AN AWFULLY LONG HEXADECIMAL STRING!

THAT’S THE PUBLIC KEY?!

SURE IS!
AND IT MAKES SECURE COMMUNICATION POSSIBLE!

OH! WELL, THAT'S A RELIANCE!

HAHA!

SHE STILL HAS YOUR CARD...
Now let's learn about asymmetric cryptography...

Which most people use regularly on the Internet without realizing!

Then why is it okay for the key to be discoverable in this case?

Because we're using an encryption scheme in which the public key is only one of the keys!

The other key is the private key!
IN OTHER WORDS, WHEN WE WERE SHOPPING JUST NOW, THE CUSTOMER DATA WAS ENCRYPTED WITH A PUBLIC KEY...

PRIVATE KEY

AND WAS DECRYPTED WITH A PRIVATE KEY BY THE RETAILER.

The recipient (the retailer) makes the public key $P_k$ (the encryption key) open to the public but keeps the private key $S_k$ (the decryption key) secret.

THEN THE KEY USED FOR ENCRYPTION AND THE KEY USED FOR DECRYPTION ARE DIFFERENT?

YEP! AND THIS IS KNOWN AS ASYMMETRIC ENCRYPTION.

THE ENCRYPTION SCHEME IS ONE PART OF PUBLIC-KEY CRYPTOGRAPHY, WHICH IS ALSO CALLED ASYMMETRIC CRYPTOGRAPHY.
In the public-key encryption scheme, even if $n$ users were communicating with encryption, the total number of keys involved is only $2n$. That’s because each user’s public key from their key pair is what others will use to encrypt a message to that person. When there are 1,000 users in a symmetric-key scheme, the number of keys is calculated as follows (using the equation on page 81):

$$\binom{1000}{2} = \frac{1000 \times (1000 - 1)}{2}$$

This means that you need 499,500 keys for a symmetric-key scheme. But for public-key encryption, the key calculation is just $2 \times 1,000$, which means you need only 2,000 keys.

Since you can encrypt using a publicly accessible key...

Key delivery isn’t an issue—unlike with a symmetric key.
THAT'S EXCELLENT!

SO WHAT IF WE WERE TO JUST STOP USING SYMMETRIC KEYS AND ONLY USE PUBLIC-KEY CRYPTOGRAPHY?

BUT AREN'T PUBLIC KEYS INCREDIBLY LONG?

THE KEY SIZE OF A SYMMETRIC KEY IS 64, 128, OR 256 BITS, BUT PUBLIC KEYS CAN BE EVEN LONGER AT 1,024 BITS!

YES, THE KEYS ARE MUCH LONGER FOR PUBLIC-KEY ENCRYPTION, BUT THAT ISN'T THE MAIN REASON OTHER TYPES OF CRYPTOGRAPHY ARE USED. PUBLIC-KEY CRYPTOGRAPHY REQUIRES A LOT OF CALCULATIONS—MANY MORE THAN IN OTHER TYPES OF CRYPTOGRAPHY!

WE'LL STUDY THIS IN CHAPTER 4.

BUT IN THAT CASE, WE OFTEN USE HYBRID METHODS...

WHAT ABOUT WHEN YOU ENCRYPT A LONG SENTENCE?

COMBINING SYMMETRIC-KEY AND PUBLIC-KEY ENCRYPTION.

ALSO, WHEN A MESSAGE IS ENCRYPTED USING A SYMMETRIC-KEY ALGORITHM, IT'S CLEAR YOU'RE COMMUNICATING WITH A MUTUAL PARTY, BUT...

HMM

Huh?
When you use a public key, there's no way for the recipient to verify the identity of the sender!

Identity theft

Phishing scam

Falsification

For that reason, we use a form of identity verification known as authentication.

Then can I have you start by teaching us the key types and structures?

We'll learn about this in Chapter 4 as well.

Like symmetric-key methods, public-key cryptosystems have various schemes.
Mathematical Dependencies of Public-Key Encryption

Public-key encryption schemes rely on two types of mathematical problems:

- The integer factorization problem is used by RSA encryption, Rabin encryption, and other schemes.
- The discrete logarithm problem is used by ElGamal encryption, DSA authentication, and so on.

I need to know math?!

Don't worry! I'll teach you the math behind encryption starting from the basics.

You can think of encryption as a hard-to-solve math problem where security increases with the difficulty of the problem. The more time needed to solve a problem, the harder the problem is.

Why do we have to learn such difficult math?

You can't make me!

To understand the structure of a cryptosystem, you need to know the math behind it!

First I'll explain one-way functions, which are needed to guarantee the security of public-key encryption.
ONE-WAY FUNCTIONS

A one-way function is a function $F$ for which it is easy to compute $F(x)$ for a given input $x$ but not easy to find the input $x$ given the calculation of $F(x)$. Let's look at an example of a one-way function.

INTEGER FACTORIZATION PROBLEM

Multiplying two large prime numbers and finding the resulting composite number is simple. On the other hand, attempting to find the original two primes is exceedingly difficult when you are given only the composite number. The process of deriving the original primes based on the composite number is known as the integer factorization problem (explained later in “Prime Numbers and Integer Factorization” on page 120).

DISCRETE LOGARITHM PROBLEM

Consider the following equation, which uses modular arithmetic:

$$a^x = y \pmod{p}$$

When you know $a$ and $x$, finding $y$ is comparatively simple. But if you know $a$ and $y$, finding $x$, the logarithm of $y$, is extraordinarily difficult. This is the discrete logarithm problem.
BY THE WAY, WHY IS THE ONE-WAY FUNCTION NECESSARY?

IN A SYMMETRIC-KEY ALGORITHM, DOESN'T DECRYPTION INVOLVE REVERSING THE ENCRYPTION PROCESS?

WAIT A MINUTE!

IN PUBLIC-KEY CRYPTOGRAPHY, WITHOUT THE ONE-WAY FUNCTION, THERE'S THE DANGER OF SOMEONE INFERRING THE PRIVATE KEY.

IF YOU USE A ONE-WAY FUNCTION, DOESN'T DECRYPTION BECOME IMPOSSIBLE?

YOU JUST USE THE PRIVATE KEY TO DECrypt, DON'T YOU?

CORRECT!

PRIVATE KEYS DO HAVE THAT FUNCTIONALITY.

A ONE-WAY FUNCTION IS ALSO KNOWN AS A TRAPDOOR FUNCTION. YOU CAN FIND THE INPUT USING THE PRIVATE KEY, SO THE FUNCTION ONLY APPEARS TO BE ONE-WAY TO THOSE WITHOUT THE KEY.
If you leave through an automatically locking door, then without a key, you can't get back in. A function with this type of structure is known as a trapdoor function.

Without a key, you can leave the room.
But without a key, you can't enter the room.
If you have a key, you can enter the room.

Let's take a look at the birth of RSA encryption, which is a public-key encryption scheme, as well as how to use the mathematics behind it!
THE BIRTH OF RSA ENCRYPTION

RSA was introduced in 1977 as the world’s first public-key encryption system. Its name derives from the initials of the three American researchers who developed it: Rivest, Shamir, and Adleman.

RSA’s strength lies in the difficulty of solving the integer factorization problem developers published in a science magazine in 1977. The problem involves applying integer factorization to a given number to decipher a message.

Here is the 129-digit natural number they presented:

\[
114381625757888867669235779761466120102182967212
4236262561842935706603524573338978394067123568395870
50589990751475992900268709543541
\]

Not until 1994 was the number’s integer factorization calculated and the message deciphered. Doing so took some 1,800 computers. It may seem as though 17 years is a long time to find the factors of a number, but one of the RSA developers, Rivest, had anticipated that it would take a thousand years, so the creators were actually surprised the cipher was cracked when it was.

Incidentally, the decoded message was “THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE.”

The numbers currently used in RSA encryption are more than 300 digits long in base 10 and, even with computers, would take an astronomical amount of time to perform integer factorization on.
PRIME NUMBERS AND INTEGER FACTORIZATION

IT'S MATH TIME!

FIRST, SOME MATERIALS! THESE WILL HELP YOU UNDERSTAND RSA.

"FUN PRIME NUMBERS"?

THERE'S NOTHING FUN ABOUT THIS!

I GAVE UP ON ARITHMETIC AS SOON AS WE STARTED DIVIDING FRACTIONS.
In the math behind RSA, you only have to work with natural numbers.

There aren't any irrational numbers or fractions—keep this in mind!

What? Really?

Let's do it!

Time for a problem!

Say you have 30 oranges.

How would you divide them equally among a group of children, with no remainder?
SIMPLE!

HERE'S THE RELATIONSHIP BETWEEN THE NUMBER OF PEOPLE AND THE NUMBER OF ORANGES:

<table>
<thead>
<tr>
<th>Number of children</th>
<th>Oranges per child</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
</tr>
</tbody>
</table>

THAT'S IT!

A NUMBER THAT DIVIDES PEOPLE, ORANGES, AND OTHER THINGS WITH NO REMAINDER...

IS KNOWN AS A DIVISOR OR A FACTOR.

THE DIVISORS (OR FACTORS) OF 30 ARE 1, 2, 3, 5, 6, 10, 15, AND 30—EIGHT IN ALL.

AND THEN THERE ARE SOME NATURAL NUMBERS THAT ARE DIVISIBLE BY NOTHING BUT 1 AND THEMSELVES—THOSE ARE KNOWN AS PRIME NUMBERS.

IS 1 A PRIME NUMBER?

1

A FUNDAMENTAL RULE OF MATHEMATICS IS THAT 1 ISN'T A PRIME NUMBER.

SORRY, 1.

LET'S TAKE A LOOK AT THE PRIME NUMBERS UP TO 20.
<table>
<thead>
<tr>
<th>Number</th>
<th>Divisibility Characteristics</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Divisible only by itself (2) and 1</td>
<td>Prime number</td>
</tr>
<tr>
<td>3</td>
<td>Divisible only by itself (3) and 1</td>
<td>Prime number</td>
</tr>
<tr>
<td>4</td>
<td>Divisible by 2</td>
<td>Not prime</td>
</tr>
<tr>
<td>5</td>
<td>Divisible only by itself (5) and 1</td>
<td>Prime number</td>
</tr>
<tr>
<td>6</td>
<td>Divisible by 2 and 3</td>
<td>Not prime</td>
</tr>
<tr>
<td>7</td>
<td>Divisible only by itself (7) and 1</td>
<td>Prime number</td>
</tr>
<tr>
<td>8</td>
<td>Divisible by 2 and 4</td>
<td>Not prime</td>
</tr>
<tr>
<td>9</td>
<td>Divisible by 3</td>
<td>Not prime</td>
</tr>
<tr>
<td>10</td>
<td>Divisible by 2 and 5</td>
<td>Not prime</td>
</tr>
<tr>
<td>11</td>
<td>Divisible only by itself (11) and 1</td>
<td>Prime number</td>
</tr>
<tr>
<td>12</td>
<td>Divisible by 2, 3, 4, and 6</td>
<td>Not prime</td>
</tr>
<tr>
<td>13</td>
<td>Divisible only by itself (13) and 1</td>
<td>Prime number</td>
</tr>
<tr>
<td>14</td>
<td>Divisible by 2 and 7</td>
<td>Not prime</td>
</tr>
<tr>
<td>15</td>
<td>Divisible by 3 and 5</td>
<td>Not prime</td>
</tr>
<tr>
<td>16</td>
<td>Divisible by 2, 4, and 8</td>
<td>Not prime</td>
</tr>
<tr>
<td>17</td>
<td>Divisible only by itself (17) and 1</td>
<td>Prime number</td>
</tr>
<tr>
<td>18</td>
<td>Divisible by 2, 3, 6, and 9</td>
<td>Not prime</td>
</tr>
<tr>
<td>19</td>
<td>Divisible only by itself (19) and 1</td>
<td>Prime number</td>
</tr>
<tr>
<td>20</td>
<td>Divisible by 2, 4, 5, and 10</td>
<td>Not prime</td>
</tr>
</tbody>
</table>

Nonprime numbers are called composite numbers, and they can be expressed as the product of one or more primes.

This is called integer factorization.
INTEGER FACTORIZATION FOR ANY COMPOSITE NUMBER HAS ONLY ONE RESULT. ACCORDING TO THE UNIQUE FACTORIZATION THEOREM, EACH FACTORIZATION FOR A COMPOSITE NUMBER IS ALSO UNIQUE TO THAT NUMBER. TO PRESERVE THE INTEGER FACTORIZATION'S UNIQUENESS, WE DON'T CATEGORIZE 1 AS A PRIME NUMBER.

WHAAAAAAAT?!?

IF 1 WERE PRIME, NUMBERS COULD HAVE MORE THAN ONE FACTORIZATION. FOR EXAMPLE, THE FACTORS OF 6 WOULD BE BOTH 2 × 3 AND 2 × 2 × 1.

DO WE HAVE TO INSPECT EACH AND EVERY NUMBER TO DETERMINE WHETHER IT'S PRIME?

FORTUNATELY, WE CAN USE A METHOD KNOWN AS THE SIEVE OF ERATOSTHENES.

HERE'S HOW IT WORKS!

WHEN A NATURAL NUMBER N ISN'T DIVISIBLE BY ALL OF THE PRIME NUMBERS LESS THAN \sqrt{N}, THE NATURAL NUMBER N IS A PRIME NUMBER.
THINK OF N AS BEING EQUAL TO pq.

\[ p \leq \sqrt{N} \text{ OR } q \leq \sqrt{N} \]

IF N CAN BE EXPRESSED AS THE PRODUCT OF THE TWO PRIME NUMBERS pq, AT LEAST p OR q MUST BE LESS THAN \( \sqrt{N} \).

I SEE!

\[ p > \sqrt{N} \text{ AND } q > \sqrt{N} \]

\[ pq > N \]

SO IF BOTH p AND q ARE LARGER THAN \( \sqrt{N} \), THEIR PRODUCT WOULD HAVE TO BE GREATER THAN N.

SO WHAT'S ERA... WHATEVER IT IS YOU SAID?

I'M AN ANCIENT GREEK SCHOLAR!

I WAS THE FIRST PERSON TO CALCULATE THE SIZE OF EARTH!
The Sieve of Eratosthenes is an efficient method of determining whether a number is prime.

Look at this printout.

These boxes contain the numbers from 1 to 400.

Since \((400 - 20) = 20\), the prime numbers less than 400 can't be divided by prime numbers greater than 20.

The set \(\{2, 3, 5, 7, 11, 13, 17, 19\}\) comprises all of the prime numbers less than 20.

First, since 2 is a prime number, leave it alone.

Then blot out all multiples of 2!
GOT IT!

NEXT, BLIT OUT ALL MULTIPLES OF 3 WITH THE EXCEPTION OF 3.

AFTER THAT, DO THE SAME THING FOR ALL PRIME NUMBERS THROUGH 19.

REALLY?
Look! Now we’re left with every prime number through 400!

Prime numbers:

Simple, eh?

Yeah, simple.
TYPES OF PRIMALITY TESTS

The sieve of Eratosthenes is a reliable method for finding prime numbers. When determining whether a very large number is prime, however, using this method takes a great deal of time. Instead, we can use Fermat’s method. It isn’t completely reliable because it determines prime numbers probabilistically, but it is faster. We’ll cover Fermat’s method in depth later, but for now, you just need to know that this method has a small chance of misidentifying a nonprime number $n$ (a composite number) as prime.

We could also use the Miller-Rabin method, which improves upon Fermat’s method. In a given test, the probability of misidentifying a composite number as prime is less than with the Fermat method. In fact, this method can determine prime numbers with near certainty.

NUMBERS THAT ARE PROBABLE PRIMES ARE KNOWN AS PSEUDOPRIMES.
Next, let's check whether integer factorization is as difficult as it seems!

RIO, perform an integer factorization on 35.

That's easy!

5 x 7!

Correct!

And the prime factorization of 77?

7 x 11.

Right again!

Now it's your turn, big brother.

Perform a prime factorization on 1,001!
HOW ABOUT A PRIME FACTORIZATION OF 9,991!

GURHH

WHY DO I GET ALL THE TOUGHS ONS?!

SPROING

1,001 IS 7 x 11 x 13, RIGHT?

CORRECT!

OH!

RAMEN'S HERE! THANKS FOR WAITING!
IT'S EASY IF YOU USE A FACTORIZATION FORMULA!

Since \( x^3 + y^3 = (x + y)(x^2 - xy + y^2) \), first think of 1,001 as \( 10^3 + 1^3 \).

So, \( 1,001 = 10^3 + 1^3 \)
\begin{align*}
&= (10 + 1) \times (100 - 10 + 1) \\
&= 11 \times 91 \\
&= 11 \times 7 \times 13
\end{align*}

OH, I SEE!

LOOK!

DO YOU REALLY?

9,991 is 103 \times 97!

\( x^2 - y^2 = (x + y)(x - y), \) so 9,991 = \( 100^2 - 3^2 \)
\begin{align*}
&= (100 + 3) \times (100 - 3) \\
&= 103 \times 97
\end{align*}

OOOCH!

WELL DONE!

DID YOU SEE THAT?

SO 10,001 WOULD BE?

YOU DIDN'T SOLVE IT...
Potential divisors of 10,00! are prime numbers less than 1000. Let's consider each of the numbers in the set (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97) to find potential divisors.

When we try them all out, we discover that 10,00! = 73 x 137.

I see!

I think we'd make great friends.

Thanks for your business!
MODULO OPERATIONS

Next, let's take a look at the remainder of integer division.

You need to become familiar with modulo operations to understand RSA encryption.

You mean like the remainder division you learn in elementary school?

$15 \div 7 = 2$ with a remainder of 1

SLUUUURP!

In modular arithmetic, we would express the same equation like this.

$15 = 1 \pmod{7}$

The equation represents the remainder when you divide 15 by 7.

What does $\pmod{}$ mean?
IT'S AN ABBREVIATION FOR MODULO.

$15 \equiv 1 \pmod{7}$

This equation means that when 15 and 1 are divided by 7, their remainder is the same. In other words, 15 and 1 modulo 7 are congruent.

In typical modular arithmetic, $a = b \pmod{N}$ is also known as a modulo operation. This equation is read as "a and b are congruent modulo N." You can also use $\equiv$ instead of $=$ to note congruence.

But if it's just integer calculation, it's simple...

Why complicate things with modular arithmetic?

Because it has a number of very specific advantages for encryption!

Really?

Like what?
SEVEN-CAR FERRIS WHEEL

HERE, I'VE CONVERTED A MODULO 7 OPERATION INTO A DIAGRAM.

SINCE THE PROBLEM INVOLVES FINDING THE REMAINDER AFTER WE DIVIDE BY 7, WE'RE ONLY WORKING WITH THE NUMBERS 0 THROUGH 6.

THAT'S THE FIRST ADVANTAGE!

THE SURPLUS (REMAINDER) WILL ALWAYS BE SMALLER THAN THE DIVISOR N (MODULO N OR MOD N).

SO OUR RESULTING VALUE IS LIMITED TO A SPECIFIC RANGE.
ADDITION AND SUBTRACTION IN MODULO OPERATIONS

Let’s use the Ferris wheel to model a modulo operation. Each of the seven cars is labeled with a number from 0 to 6. The cars correspond with numbered positions: we start with 0 at the top, and the positions are numbered 1 to 6 clockwise. Position 3 is the exit, and position 4 is the entrance.

Initially, car 0 is in position 0, car 1 is in position 1, and so forth. When we perform addition, the cars are rotated clockwise.

First, take a look at car 0. After moving $1/7$ of a rotation, car 0 moves from position 0 to position 1. This is defined as $+1$ (the addition of 1). When the wheel makes $2/7$ of a rotation, car 0 moves from position 0 to position 2. This is $+2$ (the addition of 2). When the wheel has made $7/7$ of a rotation—in other words, one full rotation—car 0 returns once again to position 0. This is $+7$, which is the equivalent of 0—in other words, it is the same as if the wheel had not moved at all.

1/7 rotation = +1
2/7 rotation = +2
7/7 rotation = +7 = 0
This Ferris wheel model can be used to explain modulo addition. For example, consider $5 + 6$. In the initial state, the 5 in $5 + 6$ corresponds to car 5. When this car moves by $6/7$ of a rotation, where do you think it will end up?

If we move the car six positions clockwise, we find that it ends up in position 4. In other words, we end up with the following equation:

$$5 + 6 = 4 \pmod{7}$$

When $a$ is the car in its initial state (the car's initial position) and is moved $b/7$ of a rotation, its final position is the solution to the addition problem. We can review all of the different additions here.

<table>
<thead>
<tr>
<th>$a + b$ MODULO 7</th>
</tr>
</thead>
</table>
| \begin{array}{cccccc}
| a \backslash b | 0 | 1 | 2 | 3 | 4 | 5 | 6 \\
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 \\
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 0 \\
| 2 | 2 | 3 | 4 | 5 | 6 | 0 | 1 \\
| 3 | 3 | 4 | 5 | 6 | 0 | 1 | 2 \\
| 4 | 4 | 5 | 6 | 0 | 1 | 2 | 3 \\
| 5 | 5 | 6 | 0 | 1 | 2 | 3 | 4 \\
| 6 | 6 | 0 | 1 | 2 | 3 | 4 | 5 \\
| \end{array} |

Next, let's use the Ferris wheel to explore subtraction.
First, take a look at car 0 in the left Ferris wheel as we put it through its paces. After moving counterclockwise by $1/7$ of a rotation, car 0 moves from position 0 to position 6. This is defined as $-1$ (a subtraction of 1). After moving counterclockwise by $2/7$ of a rotation, car 0 moves from position 0 to position 5. This is $-2$ (a subtraction of 2). If it moves a full rotation counterclockwise, car 0 returns to position 0. This is $-7$, which is also the equivalent of the car having not moved at all.

This Ferris wheel model can be used to explain all instances of subtraction. For example, let's consider the problem $3 - 4$ using the Ferris wheel on the right.

In its initial state, the 3 in the expression $3 - 4$ is car 3. After car 3 moves $4/7$ of a rotation counterclockwise, it arrives at position 6.

In other words, this is the equivalent of the following equation:

$$3 - 4 = 6 \pmod{7}$$

When $a$ is the car in its initial state (the car's initial position) and is moved counterclockwise $b/7$ of a rotation, its final position is the solution to our subtraction problem. The different combinations of subtracting $a$ and $b$ are shown in the table.

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a - b \pmod{7}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

**Modulo Operations** 139
MODULAR ADDITION AND SUBTRACTION ARE ACTUALLY PRETTY SIMPLE, RIGHT?

THAT WAS DELICIOUS!

PIECE OF CAKE!

EVEN I CAN DO IT!

FIGURING OUT WHAT TIME IT IS ON A CLOCK AND WHICH DAY OF THE WEEK IT IS ON A CALENDAR ARE ALSO TYPES OF MODULO OPERATIONS.

NOW LET'S USE MODULO OPERATIONS TO PERFORM MULTIPLICATION AND DIVISION!

CMON!

SO IF YOU WERE CALCULATING, SAY, 5 \times 4, WOULD YOU DO IT LIKE THIS?

5 \times 4 = 7 \times 2 + 6,

SO 5 \times 4 = 6 (MOD 7)

THAT'S RIGHT!

THUMBS UP!

HERE'S A TABLE FOR A MODULO 7 MULTIPLICATION!
The numbers in this table seem pretty chaotic, though...

But look! In the columns in which neither \(a\) nor \(b\) is equal to 0...

Each number between 1 and 6 appears only once in each column and row!

Great point!

Here's how modulo 8 multiplication works.
When neither a nor b is 0, columns 1, 3, 5, and 7 include all of the numbers between 1 and 7...

But this doesn't apply to rows 2, 4, and 6...

And there are random zeros?!?

For multiplication to yield zero when zero isn't part of the equation—that seems weird, right?

In this case, the rule that "when a \times b = 0, either a or b has to equal 0" no longer holds true.

Why does this happen?

Well... it happens in situations when modulo b and either a or b aren't coprime.

Jump on in, the water's coprime...
Two numbers are said to be coprime when they share no common factors (common divisors) except 1.

For example, in addition to 1, the numbers 8 and 2 share the common factor (common divisor) 2, so they aren't coprime. The numbers 4 and 6 in the modulo 8 multiplication table likewise share the common divisor 2 with 8, so they aren't coprime.

On the other hand, 1, 3, 5, and 7 are coprime to 8. You can determine whether two integers are coprime by confirming that the greatest common divisor of both is 1.

Thus, all prime numbers are coprime to integers that are not multiples of themselves. This property allows us to uncover prime numbers when using the sieve of Eratosthenes method.

So that means it's okay if a number that takes the modulo is prime?

HMM...

So how would we solve the equation 3 ÷ 5?

Exactly!

And if you have a prime number like 7, you can also divide it!
JUST CONVERT THE DIVISION INTO MULTIPLICATION!

\[ a \div b = a \times (1 / b) \]

Dividing \( a \) by \( b \) is the same as multiplying \( a \) by the reciprocal of \( b \). The reciprocal is also known as the inverse element.

I SEE!

BUT WAIT, HOW DO YOU FIND THE RECIPROCAL?

Huh?

3 \times (1/3) = 1

For instance, the reciprocal of 3 is 1/3. The reciprocal of a number is a number that yields 1 when multiplied by the original number.

LET'S TAKE ANOTHER LOOK AT THE MODULO 7 MULTIPLICATION TABLE AND FIND ALL INSTANCES OF 1.

The reciprocal of 1 is 1, the reciprocal of 2 is 4, the reciprocal of 3 is 5, the reciprocal of 4 is 2, the reciprocal of 5 is 3, and the reciprocal of 6 is 6.
So you would calculate $3 + 5$ like this?

3 divided by 5 is equivalent to 3 times the reciprocal of 5, which is 3:

$3 + 5 = 3 \times \frac{1}{5} = 9$

$9 + 7 = 7 + 2 = 2 \pmod{7}$

In other words, $3 + 5 = 2 \pmod{7}$.

Great job!

Yeah! Great!

There's a division table, too.

<table>
<thead>
<tr>
<th>$a \div b \mod 7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

We can also use the Ferris wheel model to explain multiplication and division!
MULTIPLICATION AND DIVISION OF MODULO OPERATIONS

Let's start by explaining multiplication using the seven-car Ferris wheel model. It's best to think of multiplication in terms of rotational speed.

In the initial state, car 0 is in position 0, car 1 is in position 1, and so on—all of the car numbers and position numbers match.

If in 1 minute, a car moves $1/7$ of a rotation (in other words, in 7 minutes, it moves one full rotation), car 0’s position after 3 minutes would be as follows:

$$1 \text{ (speed)} \times 3 \text{ (minutes)} = 3 \text{ (position after rotation)}$$

When the car moves at a rate of $5/7$ of a rotation per minute, what position would it be in after 6 minutes?

$$5 \times 6 = 30$$

$$30 = 7 \times 4 + 2 \ (30 \div 7 = 4 \text{ with a remainder of } 2)$$

Therefore, $5 \times 6 = 2 \pmod{7}$.

In other words, after the value of 30 is divided by 7, we have 4 rotations with a remainder of 2. In this case, 4 rotations is equivalent to 0 rotations when dealing with modulo, so you only have to pay attention to the remainder of 2.
Think of division as the opposite of multiplication. Given the final position and the rotational speed, we determine the time the car was in motion using an inverse operation.

With a speed of 1/7 of a rotation per minute, car 0 begins at position 1 and finally ends up in position 5. Let's determine how many minutes the car was in motion.

\[ \frac{5 \text{ (final position)}}{1 \text{ (speed)}} = 5 \text{ (minutes)} \]

So, it was in motion for 5 minutes. Were it to move for 12 minutes, or 19 minutes, it would end up in the same position, since each of these rotations is equal to 5 added to a multiple of 7 \((5 + 7n)\). However, because the times in our mod 7 calculation range only from 0 to 6 minutes, we can't visually represent the passage of 12 or 19 minutes. They would appear identical to 5 minutes. (See the Ferris wheel at the bottom left.)

At a rate of 2/7 of a rotation per minute (in other words, two full rotations per 7 minutes), car 0 begins at position 0 and finally arrives at position 5. Using the division table, we can determine the number of minutes it was in motion as follows:

\[ \frac{5 \text{ (final position)}}{2 \text{ (speed)}} = 6 \text{ (minutes)} \]

The answer is that it rotated for 6 minutes, but what's the best way to explain this?

Consider this: the final position was 5, but there was actually one surplus rotation. In other words, in reality the final position was \(2 \times 6 = 12\), which in mod 7 is expressed as 5.

Therefore, since \(12 \div 2 = 6\), the answer is that the car rotated for 6 minutes. (See the Ferris wheel at the right.)
Now we're all clear on how the four arithmetic operations can be performed using modulo operations, right?

The four arithmetic operations are addition, subtraction, multiplication, and division.

Is that a big deal?

Once you've mastered these operations, you're ready to take on the mathematics of encryption and decryption!

No problem!

We can't use all integers with these operations, right?

Division doesn't work for nonnegative integers.
Modulo operations on a prime number \( p \) are still subject to the commutative, associative, and distributive properties, so any operation will always result in a number in the set \( \{0, 1, \ldots, p-1\} \). This set is equivalent to the cars of the Ferris wheel in our example; when there were 7 cars, they were numbered 0 through 6.

**Examples of the commutative property:**
\[
\begin{align*}
    a + b &= b + a \\
    ab &= ba
\end{align*}
\]

**Examples of the associative property:**
\[
\begin{align*}
    (a + b) + c &= a + (b + c) \\
    (ab)c &= a(bc)
\end{align*}
\]

**Example of the distributive property:**
\[
    a(b + c) = ab + ac
\]

A field is the set of elements (numbers in our case) resulting from modulo and arithmetic operations.

Rational numbers are one example of a field, since all rational numbers can be factors in an operation. There is an infinite number of rational numbers, which makes the rational numbers an infinite field. In contrast, the factors of the modulo operation for prime number \( p \) are \( 0, 1, \ldots, p-1 \), meaning that \( p \) is a finite field because there is a limited number of numbers in the field.

*The Japanese word for field sounds similar to the word for a type of fish.*
GREAT! NOW WE'RE FINISHED WITH MODULO OPERATIONS, RIGHT?

THERE'S ONE LAST THING!

MORE? KEEP AT IT!

HERE'S A POWER TABLE.

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
\alpha \times 7^6 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 2 & 4 & 1 & 2 & 4 & 1 \\
3 & 3 & 2 & 6 & 4 & 5 & 1 \\
4 & 4 & 2 & 1 & 4 & 2 & 1 \\
5 & 5 & 4 & 6 & 2 & 3 & 1 \\
6 & 6 & 1 & 6 & 1 & 6 & 1 \\
\hline
\end{array}
\]

TO CALCULATE THE VALUES IN THE TABLE, TAKE THE NUMBER \( \alpha \times 7^6 \) AND DIVIDE IT BY 7 TO GET THE REMAINDER. FOR EXAMPLE, TAKING ANY VALUE OF \( \alpha \) BETWEEN 1 AND 6 TO THE 6TH POWER AND THEN DIVIDING THE RESULT BY 7 WILL ALWAYS RESULT IN A REMAINDER OF 1, WHICH IS REPRESENTED IN THE LAST COLUMN OF THE TABLE.

\[
\begin{align*}
1^6 & = 1 = 0 \times 7 + 1 \\
2^6 & = 64 = 9 \times 7 + 1 \\
3^6 & = 729 = 104 \times 7 + 1 \\
4^6 & = 4,096 = 585 \times 7 + 1 \\
5^6 & = 15,625 = 2,232 \times 7 + 1 \\
6^6 & = 46,656 = 6,666 \times 7 + 1
\end{align*}
\]

THIS MODULO OPERATION RESULTS IN VALUES THAT ARE PRETTY SPREAD OUT...
indeed!

modulo operations are also used to generate pseudorandom numbers.

isn't this strange?!

according to this table, any number taken to the 6th power yields 1?

exactly!

and this is connected to the next thing we'll study: fermat's little theorem!
FERMAT'S LITTLE THEOREM AND EULER'S THEOREM

NOW,
LET'S MOVE ON TO FERMAT'S LITTLE THEOREM!

FERMAT'S LITTLE THEOREM:
WHEN n IS A PRIME NUMBER AND YOU HAVE AN INTEGER α THAT IS COPRIME TO n (AN INTEGER THAT IS NOT A MULTIPLE OF n), THE FOLLOWING FORMULA HOLDS TRUE:
\[ a^{n-1} = 1 \pmod{n} \]
IN OTHER WORDS, IF YOU TAKE α TO THE POWER OF n - 1 AND DIVIDE IT BY n, THE REMAINDER IS 1.

WE USED FERMAT'S LITTLE THEOREM IN PRIMALITY TESTS, REMEMBER?
IT'S AN ABSOLUTELY ESSENTIAL FOUNDATION TO HAVE FOR LEARNING EULER'S THEOREM.

SO THIS IS THE LAW WE SAW DEMONSTRATED IN MODULO 7, WHEN RAISING NUMBERS 1 THROUGH 6 TO THE 6TH POWER YIELDS 1.

<table>
<thead>
<tr>
<th>a</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

192  CHAPTER 3  PUBLIC-KEY ENCRYPTION
FERMAT: THE FATHER OF NUMBER THEORY

Pierre de Fermat (1601–1665) was a 17th-century French lawyer and prominent mathematician. He made great contributions to the field of number theory, including but not limited to creating the modulo operation.

He created not only Fermat's little theorem but also what is known as Fermat's big theorem (or last theorem).

Fermat's big theorem states that given a natural number n whose value is 3 or more, no three natural numbers (x, y, z) can solve the equation \( x^n + y^n = z^n \). Interestingly, Fermat himself left no proof of this theorem.

At first glance, the contents of the theorem look so simple that it seems like a middle schooler could solve it. I'm sure you're familiar with the Pythagorean theorem, which states that if the lengths of the sides of a right triangle are a, b, and c, then \( a^2 + b^2 = c^2 \). Fermat's theorem concerns an equation for when the 2 in \( a^2 + b^2 = c^2 \) is replaced by a value of 3 or more.

Proof of the big theorem came 330 years after Fermat's death, in 1995, thanks to British mathematician Andrew Wiles (1953–).

FERMAT HAD APPARENTLY WRITTEN THIS IN HIS NOTEBOOK:

I thought I would write a proof of my big theorem here, but there isn't enough space in the margin!
Let's use Fermat's little theorem to verify prime numbers!

Take the contraposition of Fermat's little theorem. If $n$ and $a$ are coprime and $a^{n-1} \equiv 1 \pmod{n}$ is true, then we can say that $n$ is not a prime number.

What's contraposition?

A contraposition contrasts with a proposition. For example, if we had the proposition "If A, then B," the contraposition would be "If not B, then not A."

A B

If A, then B

Not B, then not A

The contraposition of a correct proposition is always true.

I see... wait, this isn't a manga.

If the proposition "All manga are fun" is true, then the contraposition "If it is not fun, it is not a manga" is also true.

Using this as a method of determining prime numbers...

This is known as Fermat's method!
FERMAT'S METHOD AND PSEUDOPRIME NUMBERS

To perform a primality test using Fermat's method, use this equation:

\[ a^{n-1} \equiv 1 \pmod{n}, \]

where \( n \), the number you are performing the primality test on, and \( a \) are coprime.

Although Fermat's method is very efficient at testing for primality, even if the number \( n \) passes the primality test for several of its coprimes, that doesn't necessarily prove \( n \) is a prime number.

With Fermat's method, there are situations in which a number may be probabilistically determined to be prime even when it really isn't prime. We call such values \textit{pseudoprime numbers}.

For example, when \( n = 3,215,031,751 \), it passes the primality test for the coprimes 2, 3, 5, and 7:

\[
\begin{align*}
2^{3215031750} &\equiv 1 \pmod{3,215,031,751} \\
3^{3215031750} &\equiv 1 \pmod{3,215,031,751} \\
5^{3215031750} &\equiv 1 \pmod{3,215,031,751} \\
7^{3215031750} &\equiv 1 \pmod{3,215,031,751}
\end{align*}
\]

However, 3,215,031,751 is not actually a prime number. You can see this when you perform integer factorization:

\[ 3,215,031,751 = 151 \times 2,515 \times 28,351 \]

However, out of the numbers \( n \) less than 250 billion, 3,215,031,751 is the only nonprime that passes the Fermat method's primality test for the four prime numbers 2, 3, 5, and 7.
Euler's Theorem

Given a natural number \( n \) and a coprime integer \( \alpha \), we present the following equation:

\[ \alpha^{\phi(n)} \equiv 1 \pmod{n} \]

The \( \phi(n) \) is known as Euler's totient function. Euler's totient function gives the number of integers between 1 and \( n \) that are coprime to \( n \).

In addition, if we multiply \( \alpha^{\phi(n)} \) by \( \alpha \), we get \( \alpha^{\phi(n)+1} \), and from this we can extrapolate the following equation:

\[ \alpha^{\phi(n)+1} \equiv \alpha \pmod{n} \]

When \( k \) is an integer, \( \alpha^{\phi(n)} \equiv \alpha \pmod{n} \) will result in 1, as we saw in this equation:

\[ \alpha^{\phi(n)} \equiv 1 \pmod{n} \]

This might not be apparent at first sight, but for example, when \( k = 1 \),

\[ \alpha^{\phi(n)} = \alpha^{1 \cdot \phi(n)} = \alpha^{\phi(n)} \]

We can express an even number as \( k\phi(n) + 1 \), since adding 1 to any odd number will result in an even number. Because we know that multiplying both sides of Euler's theorem equation by \( \alpha \) results in

\[ \alpha^{\phi(n)+1} = \alpha \pmod{n}, \]

that means modding \( \alpha^{\phi(n)+1} \) by \( n \) will also result in \( \alpha \). You should get the following formulas:

\[ \alpha^{\phi(n)} = 1 \pmod{n} \]
\[ \alpha^{\phi(n)+1} = \alpha \pmod{n} \]

This means that when \( \alpha \) is any integer between 1 and \( (n-1) \), the following formula will apply:

\[ \alpha^{\phi(n)-1} = \alpha \pmod{n} \]

156. Chapter 3 Public-Key Encryption
Given $\phi(7)$, the numbers 1, 2, 3, 4, 5, 6 are coprime to 7.

So $\phi(7) = 6$, right?

If $n$ is a prime number, then the only number that isn't coprime to $n$ between the natural numbers 1 and $n$ is $n$ itself.

In other words, $\phi(n) = n - 1$.

When $n$ is a prime number, then $\phi(n) = (n - 1)$, which we can substitute into the equation so that $a^{\phi(n)} = a^{n-1} \equiv 1 \pmod{n}$, which is consistent with Fermat's Little Theorem and the gray column of the table on page 152.

The Mathematician Euler

Leonhard Euler (1707-1783) was a prominent Swiss mathematician of the 18th century. He not only made huge contributions to the broad field of mathematics but also contributed to the fields of physics and astronomy. His most well-known contribution to mathematics is known as Euler’s formula:

$$e^{i\theta} = \cos\theta + i\sin\theta$$

We won’t cover it in detail, but this formula demonstrates the relationship between the complex exponential function $e^{i\theta}$ and the trigonometric functions $\cos\theta$ and $\sin\theta$ through the imaginary number $i = \sqrt{-1}$.

I love stars, too.

What kind of guy was this Euler?
Euler function on the product of two prime numbers

$N$ is the product of two prime numbers $p$ and $q$. Here, in order to derive the Euler function of $N$ using $p$ and $q$, we need to first count the integers that are not coprime to $N$. Since $p$ and $q$ are prime numbers, we know that multiples of $p$ and multiples of $q$ make up all the numbers that are not coprime to $N$.

1. Because the multiples of $p$ between 1 and $qp$ are $p, 2p, 3p, \ldots, \text{ up to } qp$, there are $q$ number of multiples of $p$.

2. Because the multiples of $q$ between 1 and $qp$ are $q, 2q, 3q, \ldots, \text{ up to } pq$, there are $p$ number of multiples of $q$.

3. Of the multiples of $q$ and $p$, only $qp$ and $pq$ are shared by both $p$ and $q$, but because $qp$ and $pq$ are both equal to $N$, they are the same number.

For a prime number $n$, $\phi(n) = n - 1$, but because $N$ isn't prime and the multiples of the prime numbers $q$ and $p$ are not coprime with it, we need to subtract the number of multiples of $p$ and $q$ from $N$ instead. Through our analysis from steps 1 and 2, we found that there are $q + p$ number of integers not coprime with $N$. In other words,

$$\phi(N) = N - p - q + 1$$

Since $N = pq$, we can substitute it into the equation to get this:

$$\phi(N) = pq - p - q + 1 = (p - 1)(q - 1)$$

This means that $\phi(pq) = \phi(p)\phi(q)$. If we apply Euler's theorem to $p$ and $q$ individually, we would get these equations:

$$a^{\phi(p)} \equiv 1 \pmod{p}$$

$$a^{\phi(q)} \equiv 1 \pmod{q}$$

We can combine these equations to derive another equation. The smallest multiple $\{p - 1\}$ and $\{q - 1\}$ share is known as their lowest common multiple, which we'll represent as $L$. This means that when $\{p - 1\}$ is multiplied by some integer $m$, it equals $L$, and when $\{q - 1\}$ is multiplied by some integer $n$, it also equals $L$. Therefore, $L = m(p - 1) = n(q - 1)$.

With $a^{\phi} \equiv 1 \pmod{p}$, if we raise both sides by $m$, we get

$$a^{\phi(m)} \equiv 1^m \pmod{p}$$

$$a^L \equiv 1 \pmod{p}$$

155. Chapter 3: Public-Key Encryption
If we do the same with $a^{q-1} = 1 \pmod{q}$, we get the same result:

$$a^{q-1} = 1 \pmod{q}$$

$$a^q = 1 \pmod{q}$$

We can combine these two equations to get $a^\phi = 1 \pmod{pq}$. Because $N$ is equal to $pq$, we would get the following equation:

$$a^\phi = 1 \pmod{N}$$

In other words, $\phi$ functions identically to Euler's theorem $\phi(N)$. In addition, because the product of two arbitrary positive integers is equal to the product of the lowest common multiple and the greatest common divisor (which we'll call $G$), we would get the equation $(p-1)(q-1) = LG$, and we can derive the following:

$$L = \frac{(p-1)(q-1)}{G}$$

For example, let's say $p = 3$ and $q = 5$, so $N = pq$ is 15. Therefore, $(p-1)$ is 2, $(q-1)$ is 4, and $\phi(N) = (p-1)(q-1)$ is 8. The lowest common multiple $L$ of 2 and 4 is 4, and the greatest common divisor $G$ is 2. In this case, for any natural number $a$ that is coprime to 15 and for any $k$ that is a nonnegative integer, the following formula applies:

$$a^{8k} = 1 \pmod{15}$$

In other words, until we take $a$ to the $\phi(N)$th power, the greatest common divisor $G = 1$ sometimes appears periodically for every $L$ numbers. According to the Euler's theorem equation, for all of the integers $a$ between 1 and $(N-1)$, this applies:

$$a^{\phi(N)} = 1 \pmod{N}$$

You can see this in the gray columns of this table.

<p>| $\phi = 8$ when $N = 15$, $\phi(15) = 8$, $L = 4$, $G = 2$ |
|---|---|---|---|---|---|---|---|---|</p>
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>9</td>
<td>12</td>
<td>6</td>
<td>3</td>
<td>9</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>6</td>
<td>9</td>
<td>6</td>
<td>9</td>
<td>6</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>9</td>
<td>3</td>
<td>6</td>
<td>12</td>
<td>9</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>4</td>
<td>7</td>
<td>4</td>
<td>13</td>
<td>4</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>1</td>
<td>14</td>
<td>1</td>
<td>14</td>
<td>1</td>
<td>14</td>
<td>1</td>
</tr>
</tbody>
</table>
I HATE EULER AND FERMAT...

IT'S OKAY, IT'S OKAY! "EVEN A JOURNEY OF A THOUSAND MILES BEGINS WITH THE FIRST STEP!"

JUST TAKE IT AT YOUR OWN PACE.

I GUESS... YOU'RE RIGHT.

INDEED. INDEED.

BIG BROTHER...

OH LOOK, A CONSTELLATION!

THAT'S IT FOR THE BASICS OF ENCRYPTION MATHEMATICS!

NEXT, WE'LL TAKE A LOOK AT REAL PUBLIC-KEY RSA ENCRYPTIONS!
THE STRUCTURE OF RSA ENCRYPTION

LIBERATED FROM MATH AT LAST!

WHAT DO YOU MEAN? WE'RE JUST GETTING STARTED!

THESE SNACKS ARE HERE TO TEACH YOU ABOUT ENCRYPTION.

SWEET, SWEET SUGAR.

I CAN FINALLY RELAX.

WHAAAT!

THE SECRET TO RSA ENCRYPTION KEYS...

IS ACTUALLY THIS!

HUH? A PORK BUN?
There's a secret trick to public keys!

Public-key encryption has two layers.

I do like pork buns...

So now we're finally learning how RSA encryption works, right?

Never mind...

Let's first look at the differences between RSA encryption and decryption.
RSA ENCRYPTION: ENCRYPTION AND DECRYPTION

With a plaintext message $m$ and ciphertext $C$, encryption is expressed as follows:

$$C = m^e \pmod{N}$$

In other words, if we raise $m$ (the plaintext) to the power of $e$ (a public key) and divide it by $N$ (another public key), we’re left with $C$ (the ciphertext).

Decryption is expressed as follows:

$$m = C^d \pmod{N}$$

In other words, in decryption, $C$ raised to the power of $d$ (the private key) divided by $N$ yields $m$. Here, $N$ is made up of two different and large prime numbers $p$ and $q$.

DO YOU KNOW WHY WE CAN’T DECODE THE FIRST EQUATION, EVEN THOUGH THE PUBLIC KEYS $e$ AND $N$ AND THE CIPHERTEXT $C$ ARE KNOWN?

IF YOU THINK OF $m$ IN TERMS OF AN UNKNOWN VALUE $x$ IN AN EQUATION...

I DON’T WANNA HEAR IT!
IF YOU KNOW EVERYTHING BUT $x$, WHICH IS $n$, AND $d$, YOU THINK IT'D BE POSSIBLE TO DECRYPT THE CIPHERTEXT.

RIGHT?

IF YOU UNDERSTAND EULER'S TOTIENT FUNCTION $\phi(n)$, YOU CAN CALCULATE THE RESULT USING EULER'S THEOREM!

BUT WHEN CALCULATING $\phi(n)$, YOU HAVE TO PERFORM INTEGER FACTORIZATION ON $n$, RIGHT?

LISTEN CLOSELY!

BUT SINCE IT WOULD TAKE AN ENORMOUS AMOUNT OF TIME TO DERIVE $x$ BY SUBSTITUTE N NUMERICAL VALUES FOR $d$ ONE BY ONE,

IT WOULD BE NEARLY IMPOSSIBLE TO SOLVE THE EQUATION FROM A PRACTICAL STANDPOINT. THIS IS WHAT MAKES RSA A COMPUTATIONALLY SECURE ENCRYPTION.

IN OTHER WORDS, WE END UP WITH AN INTEGER FACTORIZATION PROBLEM THAT IS DIFFICULT TO SOLVE MATHEMATICALLY!

THAT'S WHY IT'S HARD TO DECRYPT!

THE ENCRYPTION KEY AND DECRYPTION KEY CARRY OUT IMPORTANT ROLES IN A PUBLIC KEY. NEXT UP, LET'S LEARN ABOUT THE METHODS FOR GENERATING THEM STEP-BY-STEP!
GENERATING AN RSA ENCRYPTION KEY

We'll first generate the public key, and then we'll generate the private key.

PUBLIC-KEY GENERATION

Public-key generation involves several steps.

1. Select two different, large prime numbers, \( p \) and \( q \), at random and calculate \( N = pq \).

2. Solve Euler's function \( \phi(N) = \phi(pq) = \phi(p)\phi(q) = (p-1)(q-1) \).

After you solve Euler's function, \( p \) and \( q \) become unnecessary! You should discard them so that other parties don't find them. This is crucial because if an adversary finds \( p \) and \( q \), it is trivial to then calculate \( d \) and break RSA encryption.

3. Select a random positive integer \( e \) such that \( e \) is coprime with \( N \):
   \[ 1 < e < \phi(n) \]

   However, select \( e \) such that \( m^e > N \) if \( m^e \leq N \), we wouldn't be able to use the modulo operation since we would get \( m^e = C \), where \( C \) is the ciphertext. We wouldn't be able to mod by \( N \) to scramble the equation. Because we don't know what messages we'll be encrypting, it is best to choose a reasonably large \( e \) (in the past, 65,537 has commonly been used as a value for \( e \)).

PRIVATE-KEY GENERATION

Find a positive integer \( d \) that satisfies the following equations:

\[ d = e^{-1} \pmod{\phi(N)} \]
\[ ed = 1 \pmod{\phi(N)} \]

where \( d \) is less than \( \phi(N) \) but larger than both \( p \) and \( q \).

Now the public key is the pair \((n, e)\), and the private key is \( d \).
d. Is equivalent to the reciprocal (inverse element) of e in multiplication when modding by $\phi(N)$.

In other words, you need to solve for the decryption key $d$ and the encryption key $e$ as a pair!

Verifying that the public key $e$ and private key $d$ work in RSA encryption

Since we know that $ed = 1 \pmod{\phi(n)}$, we can subtract 1 from each side of the equation to get $ed - 1 = 0 \pmod{\phi(n)}$. In other words, $ed - 1$ leaves no remainder when modded by $\phi(n)$, meaning that it is a multiple of $\phi(n)$. So, you get:

$$ed - 1 = k \phi(n),$$

where $k$ is a nonnegative integer. In addition, we can manipulate the equation to be this:

$$ed = k \phi(n) + 1$$

Therefore, if we take the earlier equation $a^{k+1} = a \pmod{N}$ (see page 169), we can express all of the natural numbers of $m$ (the plaintext) from 1 to $(N - 1)$ in the following equation:

$$m^{k+1} = m \pmod{N}$$

Because $ed = k\phi(n) + 1$, we can substitute the exponent, so when the ciphertext is raised to the private key, we get the following:

$$m^{ed} = m^{k\phi(n)+1} = m \pmod{N}$$

Since the ciphertext $C = m^e$, $m^{ed} = C$, it becomes clear that $d$ allows us to decrypt plaintext $m$.

Private key $d$ and public key $e$ cancel each other to reveal the plaintext!

Let's practice creating public and private keys!
HOW TO MAKE PUBLIC KEYS AND PRIVATE KEYS

Now, given two prime numbers $p = 5$ and $q = 11$, let's solve for public key $N$ and private key $d$.

**STEP 1**

The product of $p$ and $q$ is $N$.

$$N = pq = 5 \times 11 = 55$$

**STEP 2**

Solve the Euler function of $N$, $\phi(N)$.

$$\phi(55) = (5 - 1) \times (11 - 1) = 4 \times 10 = 40$$

**STEP 3**

Choose a random integer $e$ such that $e$ is coprime to $N$ and $1 < e < \phi(N)$. The list of candidates that fulfill these requirements for our current example is $\{1, 3, 7, 9, 11, 13, 17, 19\}$. We'll use $e = 17$, so the public key is $(55, 17)$.

**STEP 4**

Solve for the private key $d$. We know that $e = 17$, and we are using modulo $\phi(N)(40)$. The equation we need to solve is

$$d = e - 1 \mod \phi(N)$$

$$ed = 1 \mod \phi(N)$$

$$17d = 1 \mod 40$$

We can see that $d$ multiplied by 17 modulo 40 must equal 1. From the extended Euclidean algorithm, which we'll see later, we get that $d = 33$.

For now, let's verify that:

$$17d = 1 \mod 40$$

$$17 \times 33 = 1 \mod 40$$

$$561 = 1 \mod 40$$

where

$$561 = 1 + 40k$$

$$561 = 1 + (40 \times 14) \text{ for } k = 14$$

$$561 = 1 + 560$$

This shows that $d = 33$ is valid; thus the private key is $d = 33$. 

THE STRUCTURE OF RSA ENCRYPTION 167
STEP 5

Solve for the decryption key $d$, which is the inverse of encryption key $e$. We know that $e = 17$ and are modding by 40 ($\phi(55) = 40$).

$$ed = k\phi(n) + 1 \pmod{40}, \text{ so } 17d = 40k + 1$$

When you rearrange this equation, you get

$$\frac{40k + 1}{17}$$

Because we know $d$ must be an integer, the right side of the equation can't leave a remainder. So we can search for solutions in which $40k + 1$ is a multiple of 17 and find that one solution is $k = 11$.

Furthermore, you can calculate what $d$ is in relation to each $e$ acquired in step 4 by solving the same formula, $ed = 40k + 1$. You would get the following $d$ and $e$ pairs:

$(e = 3, d = 27), (e = 7, d = 23), (e = 9, d = 9),$
$(e = 11, d = 11), (e = 17, d = 33), (e = 19, d = 19)$

In general, the encryption key and decryption key should be different integers ($e \neq d$), and larger integers are preferred for encryption key $e$; $e = 17$ and $d = 33$ are just examples we will use here.
GENERATING A CIPHERTEXT USING RSA

To begin, we'll look at the procedure for using a public key to generate a ciphertext.

To make this a concrete example, we'll encrypt the four-letter plaintext GOLF using the encryption key $e = 17$ that we calculated earlier.

**STEP 1**

First, use the table at the right to encode the characters as integers. This is non-standard character encoding, but it is convenient for this exercise.

<table>
<thead>
<tr>
<th>Character_encoding</th>
<th>Character</th>
<th>Code</th>
<th>Character</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>K</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>L</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>M</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>3</td>
<td>N</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>4</td>
<td>O</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>5</td>
<td>P</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>6</td>
<td>Q</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>7</td>
<td>R</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>8</td>
<td>S</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>9</td>
<td>T</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>10</td>
<td>U</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>11</td>
<td>V</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>12</td>
<td>W</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>13</td>
<td>X</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>14</td>
<td>Y</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>15</td>
<td>Z</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>16</td>
<td>Space</td>
<td>63</td>
<td></td>
</tr>
</tbody>
</table>

**STEP 2**

Convert the integers to 6-bit binary.

```
32  40  37  31
```

```
100000 101000 100101 011111
```

**STEP 3**

Express the binary data as a nonnegative integer less than $(N - 1)$. In this example, since $N = 55$ and $N - 1 = 55 - 1 = 54$, the data is divided into 5-bit units. In other words, the greatest value that can be expressed in 5 bits is 31, which is below 54 and fulfills the conditions. Of course, we could use 3- or 4-bit sections instead, but using larger-bit sections is more efficient.

We append the final 0 to pad the last section because there is an insufficient number of bits to make the section a complete 5-bit unit. Here, we're using 0, but other padding digits could be used instead, depending on the encryption scheme.

**STEP 4**

Convert the binary data into decimal notation.

```
16  10  4  21  30
```

The structure of RSA encryption. 169
STEP 5

Using the encryption keys \((N = 55, e = 17)\), perform the encryption. In concrete terms, this involves raising the data in decimal notation to the 17th power, dividing by 55, and searching for the remainder. By performing these calculations for each integer as follows . . .

\[
16^{17} \pmod{55}, \quad 10^{17} \pmod{55}, \quad 4^{17} \pmod{55}, \quad 21^{17} \pmod{55}, \quad 30^{17} \pmod{55}
\]

. . . we obtain the encrypted data. For example, we can calculate this for 16 by finding the solution to \(16^{17} \pmod{55}\). We can reduce \(16^{17}\) by expanding the exponent as follows:

\[
16^{17} = 16^2 \times 16^3 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16 \pmod{55}
\]

Because \(16^2 = 256 = 36 \pmod{55}\), we can replace each \(16^2\) with 36:

\[
= 36 \times 36 \times 36 \times 36 \times 36 \times 36 \times 36 \times 36 \times 36 \pmod{55}
\]

which is equal to

\[
= 36^2 \times 36^2 \times 36^2 \times 36^2 \times 16 \pmod{55}
\]

And since \(36^2 = 1,296 = 31 \pmod{55}\), we can further reduce the equation:

\[
= 31 \times 31 \times 31 \times 31 \times 16 \pmod{55}
\]

\[
= 31^2 \times 31^2 \times 16 \pmod{55}
\]

Then because \(31^2 = 961 = 26 \pmod{55}\), we get

\[
= 26 \times 26 \times 16 \pmod{55}
\]

\[
= 26^2 \times 16 \pmod{55}
\]

And finally, because \(26^2 = 676 = 16 \pmod{55}\), we get

\[
= 16 \times 16 \pmod{55}
\]

\[
= 36 \pmod{55}
\]

The remaining encryption is performed in the same way, and we record the results as follows:

\[
10^{17} \pmod{55} = 10
\]

\[
4^{17} \pmod{55} = 49
\]

\[
21^{17} \pmod{55} = 21
\]

\[
30^{17} \pmod{55} = 35
\]

Thus, the ciphertext expressed in decimal notation is as follows:

\[
36 \quad 10 \quad 49 \quad 21 \quad 35
\]
Then we can encode the decimal notation into letters:

\[
\begin{array}{cccccc}
36 & 10 & 49 & 21 & 35 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
K & k & X & v & J
\end{array}
\]

So the ciphertext is KkXvJ.

**DECYP廷ING A CIPHERTEXT USING RSA**

Now let's go over the process of using the private key to decrypt a ciphertext into plaintext. We'll use the private encryption key \(d = 33\) in our concrete example.

**STEP 1**

Using decryption key \(d = 33\), calculate \(C^d \mod \lambda(n)\). This involves raising each number in the decimal-form ciphertext \((36, 10, 49, 21, 35)\) to the 33rd power, dividing by 55, and seeking out the remainder to obtain the plaintext data as follows:

\[
36^{33} \mod 55, \ 10^{33} \mod 55, \ 49^{33} \mod 55, \ 21^{33} \mod 55, \ 35^{33} \mod 55
\]

To calculate this for the first decimal digit 36, adopt the same technique used in step 5 of generating the ciphertext, as follows:

\[
36^{33} = 36^3 \times 36^3 \times 36^3 \times 36^3 \times 36^3 \times 36^3 \times 36^3 \times 36^3 \times 36^3 \times 36^3 \times 36^3 \times 36^3 \times 36^3 \times 36^3 \times 36^3 \times \ (mod \ 55)
\]

\[
= 46,656 \times 46,656 \times 46,656 \times 46,656 \times 46,656 \times 46,656 \times 46,656 \times 46,656 \times 46,656 \times 46,656 \times 46,656 \times 46,656 \times 46,656 \times 46,656 \times 46,656 \times \ (mod \ 55)
\]

\[
= 16 \times 16 \times 16 \times 16 \times 16 \times 16 \times 16 \times 16 \times 16 \times 16 \times 16 \times 16 \times 16 \times 16 \times \ (mod \ 55)
\]

\[
= 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times 16^2 \times (mod \ 55)
\]

\[
= 967,458,816 \ (mod \ 55)
\]

\[
= 16 \ (mod \ 55)
\]

The remaining encrypted data \((10, 49, 21, 35)\) is calculated in the same way, and we record the results as follows:

\[
10^{33} \mod 55 = 10
\]

\[
49^{33} \mod 55 = 4
\]

\[
21^{33} \mod 55 = 21
\]

\[
35^{33} \mod 55 = 30
\]

Thus, we would get the plaintext data represented in decimal notation as follows:

\[
16 \ 10 \ 4 \ 21 \ 30
\]
STEP 2

Next, convert the decimal numbers you have obtained into 5-bit binary data:

\[
\begin{array}{cccccc}
16 & 10 & 4 & 21 & 30 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
10000 & 01010 & 00100 & 10101 & 11110 \\
\end{array}
\]

STEP 3

To make the bits correspond with the character encoding in the table on page 169, divide the binary data into 8-bit units:

Because the last 0 was used as padding, it won't fit into a 6-bit unit. You can eliminate it.

STEP 4

Convert each 6-bit binary unit into an integer:

\[
\begin{array}{cccc}
10000 & 10100 & 10010 & 011111 \\
\downarrow & \downarrow & \downarrow & \downarrow \\
32 & 40 & 37 & 31 \\
\end{array}
\]

STEP 5

Using the table on page 169, encode the integer data into characters:

\[
\begin{array}{cccc}
32 & 40 & 37 & 31 \\
\downarrow & \downarrow & \downarrow & \downarrow \\
G & O & L & F \\
\end{array}
\]

Now you've completed the decryption process.
PUBLIC-KEY ENCRYPTION AND DISCRETE LOGARITHM PROBLEMS

DO YOU GET RSA ENCRYPTION NOW?

HA... HA HA. OF COURSE...!

BY THE WAY, FOR PUBLIC-KEY ENCRYPTION,

YOU DON'T USE JUST INTEGER FACTORIZATION LIKE IN RSA ENCRYPTION, RIGHT? WHAT ELSE IS THERE TO IT?

YES, THAT'S RIGHT. I'LL ALSO GIVE A SIMPLE EXPLANATION OF ELGAMAL ENCRYPTION, WHICH HAS ITS FOUNDATION IN THE DISCRETE LOGARITHM PROBLEM.

SIMPLE? YEAH RIGHT!

UGH, MORE COMPLICATED VOCAB...

BUT IT REALLY IS!

FIRST READ THE FOLLOWING EXPLANATION...

AND LEARN ABOUT THE THEORETICAL FOUNDATIONS OF ENCRYPTION!
DISCRETE LOGARITHM PROBLEMS

Take another look at the modulo 7 power table. This table contains all the powers between 1 and 6 and shows the results of modding them by 7. We can express the possible results of a modulo 7 operation as a finite field comprising these members:

\[ \{0, 1, 2, 3, 4, 5, 6\} \]

We'll represent this finite field in group notation. When we have a number \( n \) and a field of numbers related to \( n \), we can represent the whole group as \( Z_n^* \). In this case, \( n \) is 7, so we get \( Z_7^* = \{0, 1, 2, 3, 4, 5, 6\} \). You'll notice that every number in this set is coprime to 7 except 0. We'll represent the group of coprimes in \( Z_n^* \) as \( Z_n^* \).

You'll notice in the table to the right that in the row showing the power of 3 mod 7, the values from 1 to 6 each appear in the row only once. This means \( Z_7^* \) with the exception of 0 is expressed in the power of 3 modulo 7. When we have a number \( n \), if the power of \( a \) modulo \( n \) contains every coprime of \( n \) in \( Z_n^* \), then \( a \) is called \( n \)'s primitive root.

Every prime number \( p \) has primitive roots, and the number of primitive roots is \( \phi(p - 1) \). For modulo 7, the result of this equation would be

\[ \phi(7 - 1) = \phi(6) = \phi(2 \times 3) = (2 - 1) \times (3 - 1) = 2 \]

This means that modulo 7 has two primitive roots. Thus, aside from 3, there must be one other numerical value that is a primitive root, and, if we look at the \( a^k \) modulo 7 table, we can ascertain that it is 5. Where \( p \) is a prime number, \( a \) represents a primitive root modulo \( p \), and \( k \) is a nonnegative integer but also fulfills the equation \( k \leq p - 1 \), we can express any member of \( Z_p \) as a modulo operation like so:

\[ a^k = \mathbb{Z}_p \text{ (mod } p) \]

With some rearrangement of this equation, the exponent \( k \) of primitive root \( a \) can also be expressed as the following equation:

\[ k = \log_a \mathbb{Z}_p \text{ (mod } p) \]
When this is the case, \( k \) is known as a *discrete logarithm* with base \( a \).

Let's quickly go over log notation. For example, the equation \(2^3 = 8\) is the same as

\[ 3 = \log_2 2 = \log_2 2^3 = \log_2 8 \]

This is similar to rephrasing the expression "2 to the power of 3 is 8" to be "to get 8, you must multiply 2 by itself 3 times."

As is also explained on page 116, when you have the equation

\[ a^k = Z \pmod{p} \]

it isn't difficult to find \( Z \) when \( a, k, \) and \( p \) are known, but finding the *discrete logarithm* \( k \) is incredibly difficult even if you know \( a, Z, \) and \( p \). This is the *discrete logarithm problem*.
ENCRIPTION AND DECRYPTION IN ELGAMAL ENCRYPTION

In this cipher, the sender is Ruka, and the recipient is Ran.

1. Recipient Ran prepares a large prime number $q$ and the primitive root $\alpha$.

2. Recipient Ran randomly chooses a private key $d$, performs the following calculation, and makes the public keys $g$, $\alpha$, and $q$ available to others.

   $$g = ad \pmod{q}$$

3. Sender Ruka selects a random number $r$ and calculates $C_1 = \alpha^r \pmod{q}$.

   In addition, she calculates $C_2 = m \times gr \pmod{q}$ for the plaintext message $m$. 
4. Sender Ruka sends $C_1$ and $C_2$ to Ran.

5. The recipient Ran uses private key $d$ to calculate the following equation and decrypt the message:

$$m = \frac{C_1^a}{C_2^a} \pmod{q}$$

We have

$$C_1^a - \left(\alpha^a\right)^d - \left(\alpha^a\right)^r - \alpha^d - g^r$$

so

$$\frac{C_1^a}{C_2^a} = \frac{m \times g^r}{g^r} - m$$

is the decryption of $m$.

WOW! WE'VE DECRYPTED THE PLAINTEXT $m$!

AMAZING!
1. Ruka and Ran share the large primary number $p$ and the primitive root $\alpha$, neither of which is private.

2. Ruka selects a secret random number $c$ and then sends $\alpha^c \pmod{p}$ to Ran. On the other end, Ran selects a private random number $d$ and then sends $\alpha^d \pmod{p}$ to Ruka.

3. The shared key is $\alpha^{cd}$. Ruka obtains the shared key by calculating $(\alpha^c)^d = \alpha^{cd} \pmod{p}$ from private key $c$, and Ran obtains the shared key by calculating $(\alpha^d)^c = \alpha^{cd} \pmod{p}$ from private key $d$.

The two of them are able to share a key.

OHHH, THAT'S EASY TO UNDERSTAND!

ON TOP OF THAT, ELGAMAL ENCRYPTION IS REALLY STRONG!
ELGAMAL IS ALSO USED AS A DIGITAL SIGNATURE SCHEME...

AND IS USED IN THE FAIRLY NEW FIELD OF ELLIPTIC CURVE CRYPTOGRAPHY!

DIGITAL SIGNATURE? THAT TECHNOLOGY IS USED IN AUTHENTICATION, RIGHT?

WE'LL LEARN ABOUT THAT IN CHAPTER 4!

BING! BING!

OH! MY EMAIL!

LOOK! IT'S A MESSAGE FROM RAN!

FROM: RAN UZUKI
TO: RUKA
SUBJECT: BYE-BYE!

I HOPE TODAY'S LESSON WASN'T TOO TOUGH! I'VE MOVED OVERSEAS, BUT DON'T WORRY—WE CAN STILL EMAIL!

YOUR CONVERSATIONS MEANT SO MUCH TO ME, SO PLEASE KEEP IN TOUCH!

HOPE YOU CATCH THE THIEF AND GET THE EMERALD BACK!

RAN ( ^=0 ^=)
I'm sad and lonely too, but more importantly, I'm hungry!

Let's go get some ramen!

Let's do that!

You all know who Ms. Cypher is now, right?!
THE EXTENDED EUCLIDEAN ALGORITHM

The Euclidean algorithm derives the greatest common divisor of two natural numbers more efficiently than does base factorization. The procedure for identifying the greatest common divisor of two natural numbers $a$ and $b$ (where $a > b$) using the Euclidean algorithm is as follows:

1. Divide $a$ by $b$ to find the remainder $r$.
2. When $r = 0$, the greatest common divisor is $b$, and the process is finished.
3. When $r \neq 0$, replace $a$ and $b$ with $b$ and $r$, and perform the first step again.

In other words, repeat steps 1–3, and when you arrive at a remainder of 0, the divided number is the greatest common divisor.

For example, let's solve for the greatest common divisor of 1,365 and 77.

Although the Euclidean algorithm deals with remainders, its proof is conventionally written without using modular arithmetic, so we'll do the same:

\[
\begin{align*}
1,365 & = 17 \times 77 + 56 \\
77 & = 1 \times 56 + 21 \\
56 & = 2 \times 21 + 14 \\
21 & = 1 \times 14 + 7 \\
14 & = 2 \times 7 + 0
\end{align*}
\]

1,365 ÷ 77 = 17 with a remainder of 56
77 ÷ 56 = 1 with a remainder of 21
56 ÷ 21 = 2 with a remainder of 14
21 ÷ 14 = 1 with a remainder of 7
14 ÷ 7 = 2 with a remainder of 0

So, the greatest common divisor is 7.

Next, we'll locate the greatest common divisor for the coprimes 20 and 17.

\[
\begin{align*}
20 & = 1 \times 17 + 3 \\
17 & = 5 \times 3 + 2 \\
3 & = 1 \times 2 + 1 \\
2 & = 2 \times 1 + 0
\end{align*}
\]

Of course, since the greatest common divisor is 1 for coprimes, the Euclidean algorithm seems unnecessary. However, the extended Euclidean algorithm is very useful to us.

The extended Euclidean algorithm is used to solve for the modular multiplicative inverse, which is one of the calculations required in RSA. It aims to solve for the coefficients of Bezout's identity, which is the following equation.

\[ax + by = d,\]

where $d$ is the greatest common divisor of $a$ and $b$ ($x$ and $y$ are the coefficients we aim to solve for). To solve for $x$ and $y$, we must perform a few more steps after Euclid's algorithm (hence the name extended Euclidean algorithm).

First, we'll transpose the first three equations into the following equations:

\[
\begin{align*}
20 - 1 \times 17 & = 3 \\
17 - 5 \times 3 & = 2 \\
3 - 1 \times 2 & = 1
\end{align*}
\]
Since 2 is equal to \((17 - 5 \times 3)\), as seen in the earlier equations, we’ll take the last equation we created and substitute it for 2.

\[
3 - 1 \times 2 = 1 \\
3 - 1 \times (17 - 5 \times 3) = 1
\]

Then manipulate the equation by distributing and combining like terms. The goal of these substitutions is to express everything on the left side in terms of \(ax + by\) where \(a = 20\) and \(b = 17\).

\[
3 - 1 \times (17 - 5 \times 3) = 1 \\
3 + (-1 \times 17) + [-1(-5 \times 3)] = 1 \\
[-1 \times 17] + 3 + (5 \times 3) = 1 \\
(-1 \times 17) + (6 \times 3) = 1
\]

Next, we’ll substitute \((20 - 1 \times 17)\) for 3:

\[
[-1 \times 17] + 6 \times (20 - 1 \times 17) = 1 \\
(-1 \times 17) + (6 \times 20) + (-6 \times 17) = 1 \\
(6 \times 20) + (-7 \times 17) = 1
\]

The result of this series of procedures is rewritten as follows:

\[20(6) + 17(-7) = 1\]

This is exactly what we were looking for. The value \(d = 1\) because \(a\) and \(b\) are coprimes, \(a = 20\) and \(b = 17\) as expected, and we now have the values of \(x\) and \(y\).

The solution to the extended Euclidean algorithm here is \(x = 6\), \(y = -7\).

**Calculating the Private Key in RSA Key Generation**

Here we’ll show how we got \(d = 33\) in our key generation example, which requires calculating inverse elements in a modulo operation.

In our example, we were trying to solve for the private key \(d\) and had this equation:

\[17d = 1 \pmod{40}\]

To solve this, you can rearrange the equation into Bezout’s identity \((ax + by = 1)\) and use the extended Euclidean algorithm. We’ll arrange the equation so that \(a = 17\), \(b = 40\), and \(x = d\) to get

\[17x + 40y = 1\]

Now, to solve for \(x\), let’s first perform Euclid’s algorithm:

\[40 = (2 \times 17) + 6 \]
\[17 = (2 \times 6) + 5 \]
\[6 = (1 \times 5) + 1 \]
\[5 = (5 \times 1) + 0\]
Now we'll perform the extended Euclidean algorithm. Let's start with the second-to-last equation from our calculations for Euclid's algorithm:

$$6 - 1 \times 5 = 1$$

Now substitute the third-to-last equation into the second-to-last equation:

$$6 - 1 \times (17 - 2 \times 6) = 1$$

Now substitute in the first equation as well:

$$(40 - 2 \times 17) - 1 \times (17 - 2 \times (40 - 2 \times 17)) = 1$$

Group the numbers into similar terms. In this case, group them into 17s and 40s:

$$-7 \times 17 + 3 \times 40 = 1$$

Given that we had $17x + 40y = 1$ and were solving for $x$, we can see that our private key $d = x = -7 = 33 \pmod{40}$ became $-7 + 40 = 33$.

Now that you've completed this hands-on example, you should have the confidence to solve for other private keys for RSA key generation!
4

PRACTICAL APPLICATIONS OF ENCRYPTION
HYBRID ENCRYPTION

**Usagi’s Diner**

**Ramen**

**Hiring!**

**Hiring!**

**Start Today!!**

IS THE RAMEN READY YET...?

THEY MUST BE SHORT-STAFFED SINCE RAN LEFT...

I'M RAN!

Well, while we wait, I'll teach you about Hybrid Encryption.

By "Hybrid" do you mean a "combination"—as in a fusion of multiple methods?

I hate studying on an empty stomach.
Precisely! It's a method of encryption that accounts for the weaknesses of both symmetric-key algorithms and public-key cryptography.

The calculations for symmetric-key algorithms are fast, but exchanging keys is an issue. The calculations involved in public-key cryptography take a lot of time, but exchanging keys to implement a public-key scheme is simple.

Let's see how hybrid encryption solves these issues by encrypting a shared key for a symmetric algorithm using public-key cryptography. Once the encrypted key is exchanged, the public-key algorithm is no longer needed, and future messages are encrypted using the symmetric algorithm with the shared key.

The ramen is taking forever...

Symmetric algorithm

Public-key encryption process

Hybrid Encryption 187
As demonstrated in the previous figure, public-key encryption is used only to encrypt and decrypt the shared key so that the symmetric-key algorithm can be used to encrypt and decrypt the actual messages that are exchanged. Since the shared key is encrypted using a public key, the greatest weakness of a shared key—the delivery of the key—is avoided.

So, now let's look at a real-life example of encryption using ramen orders.

The message is encrypted using a shared key, and the shared key is encrypted using a public key!

Sending the encrypted shared key and the message as ciphertext...

THREE RAMEN IT IS!

THREE RAMEN PLEASE!

I HAVE A MESSAGE!

HERE, I'M SENDING THE PUBLIC KEY!

These two are a pair.

Three Rams are a pair.

AND NOW I'M SENDING IT!

GOT IT!

THREE RAMEN IT IS!

THANK YOU FOR YOUR BUSINESS!
I see! So that's how we can exchange encrypted messages in a fast, highly efficient manner!

I wish the ramen would come out in a fast, highly efficient manner, too...

Hybrid encryption is used on the internet.

Several types of hybrid encryption exist online, including pretty good privacy (PGP), which is used for encrypting email, and SSL/TLS, which is used to encrypt web pages.

Sorry for the wait!
HASH FUNCTIONS AND MESSAGE AUTHENTICATION CODES

FALSIFICATION

WHY IS HE SO ANGRY?

RUMBLE, RUMBLE, RUMBLE

THAT FACE...

GLARE

INSPECTOR, HURRY UP AND ARREST THAT CRIMINAL!

SURE, BUT...

I DON'T KNOW WHERE SHE IS?

I'M Ms. CYpher!
WE LOST 29 BOWLS OF RAMEN THE OTHER DAY!

RAMEN?!

WE RECEIVED A DELIVERY ORDER VIA EMAIL, BUT...

Sato-san in District 1

DELIVER ONE BOWL OF RAMEN.

3D BOWLS OF RAMEN ASAP!

I'M OVERWRITING IT!

Attacker

THE MESSAGE WAS TAMPERED WITH!

ARRGGHHH!

SCARY...

CAN'T YOU PREVENT MESSAGE TAMPERING USING ENCRYPTION?

OH DEAR...
TAMPERING COUNTERMEASURES

TO STOP TAMPERING, YOU SHOULD USE HASH FUNCTIONS!

HASH? IS THAT LIKE BEEF HASH?

HOLD THE DROOL, I DON'T THINK THERE'S BEEF INVOLVED.

"HASH" REFERS TO SOMETHING THAT'S CHOPPED UP.

SOUNDS DELICIOUS.

OOH, I SEE!

INSTEAD OF BEEF, HASH FUNCTIONS CHOP AND MANGLE MESSAGES INTO TINY PIECES TO PRODUCE HASH VALUES.

WHAT'S A HASH VALUE?

MORE IMPORTANTLY, IS IT EDIBLE?

IT'S A VALUE CALCULATED FROM THE MESSAGE. IT'S LIKE A FINGERPRINT YOU WOULD USE TO IDENTIFY A PERSON IN A CRIMINAL INVESTIGATION.

WE USE HASHES TO CHECK THAT MESSAGES HAVEN'T BEEN FALSEIFIED.

THEY'RE NOT FOOD!
HASH FUNCTIONS

A hash value identifies a message, like a fingerprint. The hash value is a small, fixed size, whereas the input is often larger than the hash. A hash function is used to calculate a hash value from a message.

The sender transmits a hash value with a message to guarantee the message's integrity. A message has integrity when the recipient is assured that the message hasn't been falsified. The recipient recalculates the hash using the same hash function as the sender and then compares that calculated hash to the hash attached to the message. If the values are the same, the message hasn't been falsified. This process, shown in the following figure, is called performing a checksum.

A hash function is a one-way function. The original message can't be derived from the hash value. This property is referred to as irreversibility.

Hash functions need to be resistant to collision attacks, which guess the hashes for messages. Because a hash needs to be unique to a message, the hash function can't produce the same hash for different messages. When a hash function fulfills this requirement, it has strong collision resistance. In addition to being resistant to hash collisions, a hash function shouldn't produce similar values for similar messages. For example, if we were to give the input 01001 to a hash function, it should output a different value than if we used 01000 as the input, which differs by only 1 bit. Real-life hash functions have a larger input than output, which means collisions are unavoidable. Therefore, they don't have strong collision resistance. However, hash functions can be developed to make finding an input that matches an output as difficult as possible. This is accomplished by using a large output and sets of calculations that are thought to be resistant to reverse engineering. Some hash functions developed with these requirements in mind are MD5, SHA-1, SHA-256, and SHA-512, but these functions are not foolproof. For example, researchers discovered that MD5 and SHA-1 are vulnerable to collision, so they are no longer safe to use.
IDENTITY FRAUD

YOU SHOULD USE HASH FUNCTIONS TO PREVENT MESSAGE TAMPERING.

I SEE...

BUT...

THAT ALONE ISN'T GOOD ENOUGH!

WE LOST 30 BOWLS OF RAMEN AGAIN TODAY!

Attacker

I'M GONNA CAUSE TROUBLE!

EEEK!

Deliver 10 orders of Ramen to Miyami in District 11!
Run 5 orders of Ramen to Sato in District 2!
7 orders of Ramen for Suzuki in District 3!
5 orders of Ramen for Tsun in District 5, Tanaka!

SOMEONE'S BEEN IMPERSONATING CUSTOMERS!

UH...

AREN'T THERE WAYS TO PREVENT IDENTITY FRAUD?

SO THAT'S WHY HE WAS MAD...
IDENTITY FRAUD COUNTERMEASURES

YOU CAN USE A MESSAGE AUTHENTICATION CODE!

MAC: MESSAGE AUTHENTICATION CODE

MESSAGE AUTHENTICATION?

WHAT'S THAT?

WHEN YOU HAVE A MESSAGE FROM THE ACTUAL SENDER...

IT AUTHENTICATES THAT THE RECEIVED MESSAGE HAS COME FROM THE CORRECT PERSON AND ISN'T FROM A FRAUDULENT SENDER ENGAGED IN IDENTITY THEFT.

WHOA!

HOW DOES IT AUTHENTICATE MESSAGES?
STRUCTURE OF MESSAGE AUTHENTICATION CODES

A message authentication code is a procedure that confirms a message's authenticity and certifies a message. Let's look at the following figure to learn how a message authentication code can be produced.

The sender transmits a message and the MAC value (also called a tag) derived from their message. The MAC value is used to perform a checksum in the same way as with a hash value.

The sender delivers a message with its corresponding MAC value to a recipient. The recipient calculates a MAC value from the message they received and compares their MAC with the original MAC value from the sender. If the two MAC values match, the recipient is assured that the integrity of the message is intact and that the message is from the sender.

**NOTE:** The recipient knows that the message is from the sender because it is assumed that only the sender and recipient have the key. If the recipient didn't create the tag (the MAC value), then it must be from the sender.

Think of a message authentication code as a one-way hash function that is associated with the shared key. MAC values are similar to hash functions because they possess the same mechanisms and confirm the authenticity of a message by calculating and comparing values on the sender's and recipient's ends.

When the two parties calculate the MAC value, they use a key that is shared only between themselves. By calculating the MAC value from the message, the receiving party can be assured that the sender of the message is in fact the person who has the same shared key.

196 CHAPTER 4 PRACTICAL APPLICATIONS OF ENCRYPTION
MAC values provide integrity for a message. When the two MAC values are identical, the message has not been tampered with en route from the sender. When the two MAC values differ, the message has potentially been falsified, and the sender may not be the one who shared the key.

A message authentication code works in this way, but as with public-key encryption, there are issues with the security of the shared key.

Message authentication codes are also used for SSL/TLS, which is used by international banks for money transfers and in online shopping.
Attacker

Send the Ramen!

I didn't ask for any!
WHAAA—?!?

So they're the original sender, but they won't admit it.

How nasty!

Can't we use a message authentication code?

Unfortunately, there are limits to what a message authentication code can do.
DRAWBACKS OF MESSAGE AUTHENTICATION CODES

Repudiation is the ability to deny being the sender of a message. For example, a message and MAC value are sent from A to B, and afterward A claims, "I didn't send this message to B. B made this up." There's no way to disprove A's statement, and even if B enlisted the help of a third party to find out the truth, this third party wouldn't have a way to determine whether the message and MAC value were generated by A or by B.

When a message is sent from A to B, B can't verify to a third party C that the message was sent from A. This is because the message and MAC value can be generated by either A or B. In other words, C is unable to determine whether the MAC value was generated by A or B.
**DIGITAL SIGNATURES**

**REPUDIATION MEASURES**

**HOW DO YOU DEFEND AGAINST REPUDIATION?**

**YOU USE A DIGITAL SIGNATURE!**

**A DIGITAL SIGNATURE IS THE OPPOSITE OF ENCRYPTION IN PUBLIC-KEY ENCRYPTION.**

**DOING THIS WILL ALSO ALLOW THIRD PARTIES TO VERIFY YOUR IDENTITY.**

**HOW DOES IT WORK?**

**MESSAGE AUTHENTICATION CODES PROVIDE INTEGRITY IN PRIVATE-KEY/SYMMETRIC ENCRYPTION, WHEREAS A DIGITAL SIGNATURE PROVIDES INTEGRITY IN PUBLIC-KEY ENCRYPTION.**

**THE TABLE SHOWS ENCRYPTION AND DIGITAL SIGNATURES IN PUBLIC-KEY CRYPTOGRAPHY.**

<table>
<thead>
<tr>
<th>Encrypting message</th>
<th>Encryption using sender's public key</th>
<th>Ciphertext</th>
<th>Decryption using recipient's private key</th>
<th>Digital signature</th>
<th>Decryption using sender's public key</th>
<th>Signature</th>
<th>Encryption using sender's private key</th>
</tr>
</thead>
</table>

[200] **CHAPTER 4: PRACTICAL APPLICATIONS OF ENCRYPTION**
THE WORKINGS OF A DIGITAL SIGNATURE

When using a digital signature, the sender signs a message that has been encrypted using their own private key. This signature is sent to the recipient along with the message.

The recipient decrypts the signature using the sender's public key to obtain the message. Then the recipient compares the decrypted message to the sent message.

If the two are the same, the message is verified as authentic, and the sender is thought to be legitimate. Because the sender's public key is used for decryption, a third party can also verify the signature. When a third party is able to verify that a message came from a specific sender, we have nonrepudiation—the sender can't deny they sent a message.

Let's look at some examples of how a digital signature works. The following figure has been simplified to make the general concept of a digital signature easier to understand.

In this figure, the message shown is immediately encrypted and signed. In reality, it takes time to sign an entire message with public-key cryptography.
It is common to first use a hash function to turn the message into a hash value and then convert the hash into a signature, as shown in the following figure.

A digital signature is also used to generate a server certificate that authenticates the legitimacy of an SSL/TLS server. A certificate is something that appends a digital signature to a public key (in this case, the server's public key). Certificates also append digital signatures to software downloads to protect software from being falsified.
MAN-IN-THE-MIDDLE ATTACK

Let's say we have a sender A and a receiver B. We'll have Mr. Sato of District 1 be A and Usagi's Diner be B. Sender A corresponds with B using encryption, but first he has to receive B's public key. A person in the middle, an attacker, intercepts the public key as it is transmitted from B to A and then sends their own public key to A instead.

Since the encrypted message sent by A has been encrypted using the attacker's public key, the attacker can decode the message using their own private key. The attacker alters the contents of the message, encrypts it using B's public key, and sends it. Once the attacker is done, B has no idea that the message was tampered with and has no way to check for falsification. Let's see what this looks like.

ILL ORDER A BOWL OF RAMEN USING ENCRYPTION.

Mr. Sato in District 1

I ORDER OF RAMEN, PLEASE!

Attacker

Attacker's public key

Message is encrypted using the attacker's public key

Fraudulent message is encrypted using Usagi's Diner's public key

Usagi's Diner

30 ORDERS OF RAMEN, QUICK!

I'LL CHANGE THE KEY AND THE MESSAGE.

Attacker sends their own public key in place of Usagi's Diner's public key

EEEK!
USING CERTIFICATES TO PROTECT AGAINST MAN-IN-THE-MIDDLE ATTACKS

A certificate attaches the digital signature of a public key and that key to an identity. A certificate is published by a certificate authority. When someone wants to make a public key available, they register their public key using a certificate authority and simultaneously request publication of the certificate.

Based on this request, the certificate authority verifies the authenticity of the user, and, if the public key meets the certificate authority's standards, the certificate authority generates a certificate for the public-key and digital-signature set. Sometimes the user creates the public- and private-key set, while other times the certificate authority generates these during registration.

The process of verifying the certificate guarantees that the public key indeed belongs to a specific user. The certificate authority, a trusted third party, verifies that the public key is correct on behalf of the user. Now let's look at the procedure shown in the figure and outlined in these six steps.

1. User A requests that the certificate authority publish the user's public key.

2. After the certificate authority confirms the identity of User A, the certificate authority produces a digital signature for User A's public key, and the certificate is published. The published certificate affixes a digital signature to User A's public key by means of the certificate authority. The certificate is usually made up of the public key of User A (PKA) and the digital signature of PKA.

3. The certificate authority stores the certificate in a repository (a data storehouse).

4. User B downloads User A's certificate from the repository.

5. User B decrypts the digital signature using the certificate authority's public key.

6. The decrypted key is compared with the public key attached to the certificate. If the two keys are the same, the public key included with the certificate is verified as belonging to User A.

By means of the certificate-issuing procedure, User B can obtain User A's verified public key. By using User A’s secured public key, User B can verify that
the message was encrypted with User A's private key and affixed with a digital signature.

The validated message will simultaneously meet the following three conditions:

1. The message has not been falsified.
2. The message has not been generated by a third party posing as User A.
3. User A can't repudiate the message. In other words, User A can't deny they generated the message.

As long as the legitimacy of a public key is verified, any message affixed with the public key's digital signature is assured to meet these three conditions. Let's finish up by discussing the mechanisms behind public-key infrastructure (PKI).
PUBLIC-KEY INFRASTRUCTURE

WE'RE ALMOST DONE STUDYING ENCRYPTION!

BUT CAN WE REALLY RELY ON THOSE CERTIFICATES?

HOW CAN WE KNOW A CERTIFICATE ACTUALLY WAS PUBLISHED BY A CERTIFICATE AUTHORITY?

CAN WE REALLY TRUST THE CERTIFICATE AUTHORITY?

I STILL HAVE SOME DOUBTS.

HOW DO WE KNOW INFORMATION IS AUTHENTIC?

FOR EXAMPLE... LET'S THINK ABOUT IT IN TERMS OF CURRENCY.

TA-DA!

OOOOH, A ¥10,000 NOTE! RUKA'S LOADED!
IN REALITY, THIS IS JUST A SLIP OF PAPER.

IF THE VALUE OF MONEY INFLATES...

HUMP! THE 10,000 YEN NOTE THAT WAS IN MY WALLET IS GONE...

IF MONEY LOSES VALUE, IT REALLY WILL BE JUST A SLIP OF PAPER.

BUT NOTES ARE ISSUED BY THE BANK OF JAPAN, WHICH WE CAN TRUST, RIGHT?

SO IT WAS MINE! WHEN DID YOU EVEN...

HERE, I'LL GIVE IT BACK TO YOU.

THAT'S NOT ALL!

THE BANK IS CRUCIAL TO MAINTAINING THE STABILITY OF PAPER MONEY'S VALUE SO CITIZENS CAN FEEL SECURE USING CURRENCY.

AGREED!

IT'S IMPORTANT TO KEEP THE INFRASTRUCTURE OF SOCIETY IN MIND.

Only the Bank of Japan produces paper currency in Japan (no one else is allowed to create money).

The Bank of Japan enacts a variety of measures to ensure that the value of money (paper currency) doesn't decline.

The nation defends against forgery and uses laws to defend society's trust in money.
Just as social infrastructure maintains the security and authenticity of money, public-key infrastructure (PKI) guarantees information security and authenticity for public-key encryption.

In other words, because of PKI, we can use public-key encryption to exchange emails, do business over the Internet, and perform other actions with peace of mind.
TO BEGIN, WE HAVE USER A AND USER B, WHO ARE EXCHANGING INFORMATION.

USER A

USER B

BUT WE DON'T HAVE ANY IDEA WHEN OR HOW THEIR INFORMATION EXCHANGE COULD BE ATTACKED, RIGHT?

UNCERTAIN

LET'S VISUALIZE THE FOUR ELEMENTS OF THE CERTIFICATE AUTHORITY AND REPOSITORY.

Certificate Authority

Request to store certificate

Certificate

Download certificate containing the key

User A

Encryption using private key

Digital signature + Message

Decryption using public key

User B

IF I'M USER B, WHAT PROCEDURE DO I NEED TO FOLLOW TO SECURELY RECEIVE USER A'S MESSAGE?

LET'S TAKE A LOOK AT THE STEPS IN ORDER!

PUBLIC-KEY INFRASTRUCTURE 209
User B wants to receive User A's message without fear of falsification, identity fraud, or repudiation.

Using his own private key, User A attaches a digital signature he has generated to the message. Then he sends the message to User B.

Using User A's public key, User B verifies the digital signature on the received message. If the decryption of the digital signature with User A's public key matches the message sent, the message is legitimate.

But how do we know this public key actually belongs to User A? It's suspicious!

Hmm, how would you verify that?

That's it! The trusted certificate authority can verify the public key!

User A registers their public key with the certificate authority, which publishes a certificate.
The certificate is composed of User A's public key and the certificate authority's digital signature.

The certificate authority stores the certificate in a repository.

User B downloads User A's certificate from the repository.

User B decrypts the digital signature included in User A's certificate. User B then compares the public key and digital-signature key. If the two keys are the same, User B has verified that the message came from User A.

When verification is complete, we know that User A's public key is legitimate and the message User B received is also legitimate (in other words, we've ruled out falsification, identity fraud, and repudiation).

Since the public key is registered with the certificate authority by User A, anyone can verify User A's identity.

I GOT THE MESSAGE!
NOW WE CAN RELAX KNOWING OUR EXCHANGES OF INFORMATION ARE SECURE.

PHEW!

CERTIFICATES AND PUBLIC KEYS ARE STANDARDS CREATED BY OUR INFORMATION SOCIETY TO PROTECT IMPORTANT INTERACTIONS.

WE COULDN'T SECURELY EXCHANGE INFORMATION WITHOUT THE STANDARDS AND RESOURCES CREATED BY OUR INFORMATION SOCIETY.

LET'S SEE HOW THE PKI REQUEST PROCEDURE WORKS BY LOOKING AT THE FIGURE BELOW.

I SEE!

YES!

When the decryption of the received application and the public key are verified, the certificate authority will grant User A the certificate.
But isn't it a bit of a pain for User B to have to go through all of this?

It might be too much for me...

No worries!

The process is automated by hardware built into web browsers, private software registration cards, card readers, and the like.

So what's a certificate look like?

Like this!

Here's an example of a certificate in a web browser. Web browsers have the functionality to view certificates from websites. You'll be able to see the certificate's version information, serial number, signature algorithm, issuer, and other such information.

Well, our introductory course on encryption is done!

Good work!

It's finally over!
WHAAAAAAT?!

OH! IT SAYS IT'S FROM RAN, ALSO KNOWN AS MS. CYpher!

FROM: RAN (MS. CYpher)
TO: RUKA
SUBJECT: ARE YOU WELL?

I'M RETURNING THE SMILING MADONNA AND THE PRECIOUS EMERALD.
(I'VE STRUCK A DEAL WITH THE INSURANCE COMPANY.)

BY THE WAY, I'M CURRENTLY IN THE COUNTRY OF LIBERTY, AND I HAVE A BIG JOB COMING UP VERY SOON!

HERE'S A HINT, IN BINARY:
00000010 00001100 00010101
00011000 00011001 00000110
00010010

WELL THEN, SEE YA!

STRUCK A DEAL WITH THE INSURANCE COMPANY?

THE STOLEN GOODS MUST HAVE BEEN INSURED!

MS. CYpher MUST HAVE RECEIVED A PORTION OF THE INSURANCE POLICY IN EXCHANGE FOR RETURNING THE TREASURES.

AND FOR THE INSURANCE COMPANY, THIS WAS A SMALLER LOSS THAN PAYING OUT THE FULL AMOUNT!

NOT BAD...
WHAT A SCOOP! I'VE GOTTA GET BACK TO THE NEWSPAPER!

IF WE PERFORM AN XOR OPERATION ON THE BINARY HINT, WE'LL GET OUR ANSWER! SEE PAGE 226 FOR THE SOLUTION!

BY THE WAY, WHAT DO YOU THINK MS. CYpher WILL STEAL THIS TIME?

SHE'S ABROAD, RIGHT?
PAY ATTENTION, BIG BROTHER.

SEVERAL DAYS LATER...

MARBLE ART MUSEUM

DIRECTOR, DO YOU REALLY WANT ME TO TAKE DOWN THIS PAINTING?
OF COURSE!

WE'RE EXCHANGING IT FOR THE SMILING MADONNA.

THE SMILING MADONNA

... THE PRECIOUS STONE HAS BEEN RETURNED TOO, SO FOR NOW, ALL IS WELL!

THAT'S GREAT! THIS IS WONDERFUL!

FROM HERE ON, LET'S BE ATTENTIVE WHEN IT COMES TO INFORMATION SECURITY SO WE CAN BUILD A SAFER SOCIETY!

ALL RIGHT!
ZERO-KNOWLEDGE INTERACTIVE PROOF

In order to use a credit card, you need to share information to verify the authenticity of the card, but credit card theft is common. If your credit information is stolen while you are authenticating your card, you could be charged for items you don’t recall purchasing (because you didn’t purchase them!). In the same way, you risk revealing private information to outsiders whenever you need to verify your identity. Because of these risks, we need methods that ensure confidentiality can’t be breached (also known as a zero-knowledge protocols) and methods that verify a person’s identity to others (also known as authenticity).

In an effort to respond to these necessities, Goldwasser, Micali, and Rackoff introduced the concept of the zero-knowledge interactive proof in 1985. The zero-knowledge proof is a means by which a third party (for example, a credit card company) can verify the authenticity of an individual’s card while ensuring that the private information attached to a card is not leaked. For example, a card has a randomized decimal password of more than 100 digits. To authenticate a card, a company or individual charging to the card would need to verify that the password is authentic without the ability to see the card’s password. This might seem impossible, but we can do this using the mathematical theory behind encryption. Let's look at how this method works in detail by breaking the process down into two stages: the preparatory stage and the implementation stage.

PREPARATORY STAGE

The setup of a zero-knowledge proof first requires an honest verifier (such as a credit card company). Let's look at an example.

CREATE A COMPOSITE NUMBER N THAT IS PUBLICLY AVAILABLE TO ALL USERS

The verifier prepares two prime numbers \( p, q \) and calculates their product, the composite number \( N \). That is, they use the following equation:

\[
N = pq
\]

The prime numbers \( p \) and \( q \) are secret. In actual practice, a proof would use large prime numbers comprising around 80 digits, but here, we'll use the two-digit prime numbers \( p = 13 \) and \( q = 19 \) for simplicity's sake. The product of these two prime numbers is

\[
N = 13 \times 19 = 247
\]

This is a three-digit composite number, but as noted, the factorization of \( N \) in practice would generate a number so large that it would be impossible for any computer to calculate. This very large composite number \( N \) is made available to all users.

REGISTER EACH USER’S ID WITH THE VERIFIER

Each user has an ID that is a publicly available numerical value associated with each user's public key. These IDs enable users to engage in one-to-one correspondence. Each user registers their ID with the verifier. In this book, we'll identify IDs with subscripts. For example, User A's ID is \( ID_A \).
THE VERIFIER CALCULATES AND SENDS EACH USER A PRIVATE KEY

The verifier calculates the user's registered ID modulo the square root of N. When dealing with real numbers, calculating a square root is simple; however, because we need to use integers, the square root calculation is complex unless the prime numbers p and q of composite number N are known. The zero-knowledge proof is built upon this computationally complex calculation.

In this example, only the verifier knows the prime numbers 13 and 19, and since only the verifier can calculate the square root of each ID registered by the users, confidentiality is not breached. Let's look at an example in which ID_A is 101. In this case, the square root is 71.

\[\sqrt{101} \pmod{247} = 71\]

When we calculate in reverse and square 71, we get \(71^2 \pmod{247} = 101\). The value 71 is User A's private key (S_A), which is secretly delivered to User A. Generally, the following relationship exists between ID_A and private key S_A:

\[\sqrt{\text{ID}_A \pmod{N}} = S_A\]

\[(S_A)^2 \pmod{N} = \text{ID}_A\]

In practice, the private key will be a number of more than 100 digits, so the private key isn't a number that User A can memorize. However, the purpose of the private key (S_A) is not to verify the identity of User A but rather to verify the authenticity of User A's card. Therefore, User A doesn't need to memorize S_A as one would memorize the PIN for an ATM card. Now that User A has a number to verify that their card is authentic, let's look at how User A would use this card.

IMPLEMENTATION STAGE (VERIFICATION PROCESS)

Now we'll demonstrate the verification process for when a prover such as User A wants to authenticate their identity to User B. If User A is trying to buy something with their credit card, this process shows that the card User A possesses is genuine.

USER A SENDS A VERIFICATION REQUEST TO USER B

First, User A selects a random number r_A, squares it, and divides it by the composite number N to find the remainder, y_A. In equation form, the remainder y_A is calculated as follows:

\[y_A = (r_A)^2 \pmod{N}\]

The remainder y_A is sent to User B. Say for example that User A chooses the random number 50. The remainder y_A is calculated as follows:

\[y_A = 50^2 - 2,500 - 30 \pmod{247}\]

Thus, User A sends 30 to User B.
Next, User A takes the product of the private key \( S_A \) they received from the verifier and the random number \( r_A \). Then User A divides the product by the composite number \( N \) to find the remainder, \( z_A \):

\[
z_A = S_A r_A \pmod{N}
\]

Once User A calculates \( z_A \), they send the value to User B. If we use the example where the random number \( r_A \) is 50, we get the following equation:

\[
z_A = 71 \times 50 \equiv 92 \pmod{247}
\]

And 92 is sent to User B.

**User B Verifies User A's Authenticity**

User B squares the \( z_A \) they received from User A and divides by the composite number \( N \) to find the remainder \( v_A \):

\[
v_A = (z_A)^2 \pmod{N} = (S_A r_A)^2 \pmod{N}
\]

In our example, \( z_A = 92 \), so we get the following equation:

\[
v_A = 92^2 - 8,164 - 66 \pmod{247}
\]

Next, User B calculates the value of \( v_A \) divided by \( y_A \), which User B received from User A earlier in the process:

\[
w_A = \frac{v_A}{y_A} \pmod{N} = v_A \cdot (y_A^{-1}) \pmod{N}
\]

All of the calculations are operations using the composite number \( N \), and \( y_A^{-1} \) expresses the inverse element (reciprocal) of \( y_A \). This means that \( y_A^{-1} \) is a value that satisfies the following equation:

\[
y_A \cdot (y_A^{-1}) \equiv 1 \pmod{N}
\]

In this example, \( v_A = 66, y_A = 30, \) and \( y_A^{-1} = 30^{-1} \pmod{247} = 140; \) therefore, we can calculate \( w_A \) as follows:

\[
w_A = \frac{66}{30} \pmod{247} = 66 \cdot 30^{-1} \pmod{247} = 66 \cdot 140 \pmod{247} = 101
\]

Lo and behold, this is User A's ID \( ID_A \).

Using this process, User B can verify the authenticity of User A's identity. We can find \( ID_A \) using this process because squaring User A's private key \( S_A \) yields \( ID_A \). That is to say, we can derive \( (S_A)^2 \) using the previous equations as follows.
$w_a$ is found by the equation $w_a = v_a \cdot y_a \pmod{N}$.
Recall the equations found for $v_a$ and $y_a$:

\[
v_a = (z_a)^i \pmod{N} - (S_a t_a)^i \pmod{N}
\]
\[
y_a = (r_a)^i \pmod{N}
\]

Substitute these into the equation for $w_a$ to get

\[
w_a = \frac{(S_a t_a)^i}{(r_a)^i} - (S_a)^i - ID_a
\]

In other words, $w_a$ is equal to User A's ID, ID_a, as you can see in Figure 4-1.

![Figure 4-1: Verification process using a zero-knowledge interactive proof](image)

**METHODS OF IDENTITY FRAUD**

Building on the same example, let's consider how User X, an attacker, might try to steal User A's identity. If User A can verify themselves to User X while ensuring that there is only a very small probability User X can learn anything about User A's secret key, then the scheme is a valid zero-knowledge protocol. Otherwise, the protocol is faulty.

The zero-knowledge protocol requires that the prover (in this case, User A or User X) know the secret key to be able to prove their identity to the verifier (User B), so the protocol protects against identity attacks in which the adversary doesn't have knowledge of the secret key. User X, the adversary trying to identify as User A, doesn't know anything about User A's private key $S_a$. In an attempt to trick User B, User X selects two numbers $e$ and $f$ that have the following relationship with $ID_a$:

\[e^3 = ID_a \cdot f \pmod{N}\]
We'll discuss this equation in more detail later.

In the scenario from Figure 4-1, the legitimate User A first sends User B $y_A$ and $z_A$ to initiate a verification request. When attempting identity fraud, User X mimics this verification request by sending $f$ and $e$ to User B as though they were User A's credentials.

Let's use $e = 25$ and $f = 82$ to see how User X would try to mimic User A's ID ($\text{ID}_A$).

Once User X sends $e$ and $f$ in place of $z_A$ and $y_A$, respectively, User B uses $e$ and $f$ to calculate $w_A$ as if it were a legitimate request. Because User X generated $e$ and $f$ to have a specific relationship with $\text{ID}_A$, $w_A$ will be identical to User A's public ID ($\text{ID}_A = 101$), and User B will mistakenly authenticate User X as User A.

Let's see how User B's calculations work with $e$ and $f$.

First, let's look at $e$'s relationship with $y_A$. First, we know that $z_A = e$, so we can substitute $z_A$ for $e$ in the following formula:

$$v_A = (z_A)^2 \pmod{N}$$

We find this relationship:

$$v_A = e^2 - 25^2 - 131 \pmod{247}$$

Next, we would substitute $v_A = e^2$ and $f = y_A$ into this equation:

$$w_A = \frac{v_A}{y_A} \pmod{N}$$

Doing so yields the following:

$$w_A = \frac{e^2}{y_A} = e^2 \cdot f^{-1} = v_A \cdot 82^{-1} \pmod{247}$$

Here, from $82^{-1} \pmod{247} = 244$, we get $w_A = 131 \times 244 \pmod{247} = 101$.

Using this method, User X can commit identity fraud on User A's public ID, without knowing User A's private key $S_A$, or the random number $r_A$. This process is illustrated in Figure 4-2.

To get around the verification process in Figure 4-1, User X needs to find values that fulfill the relationship $f = e^2 \pmod{N}$ using only User A's public ID. To do this, User X squares $e$ and divides it by $\text{ID}_A$ to find $f$:

$$f = \frac{e^2}{\text{ID}_A} \pmod{N}$$

Once User X calculates $e$ and $f$, they can pretend to be User A without even knowing the random number $r_A$, committing a security breach. Let's look at a way to turn this protocol into one in which only User A, or someone with actual knowledge of the secret key $S_A$, can be verified as User A.
ANTI-IDENTITY FRAUD IN A ZERO-KNOWLEDGE INTERACTIVE PROOF

Our protocol is flawed because identity fraud is possible, so let's look at how one can defend against an attack. To do this, we'll incorporate a randomly chosen challenge bit $b$ and modify the prover and verifier's protocol, as shown in Figure 4-3.

When User A delivers $y_a$ to User B, User B randomly chooses a value of 0 or 1 as the challenge bit $b$ and sends $b$ to User A. The prover then sends $z_a$ to User B, where $z_a = r_a \times (S_a)^{i_a}$. User B performs a check on the returned value to confirm whether the sender is a legitimate user. This check requires calculating $z_a^b$ using the two methods $z_a^b = r_a^b \times (ID_a)^i$ and $z_a \times z_a$ (where $z_a$ is the value that User A sent). User B then compares the results of the two methods of calculating $z_a^b$ to determine whether they are equal: $z_a \times z_a = r_a^b \times (ID_a)^i$? If they are equal, then the verifier accepts the prover's claim to be User A.

This entire protocol is run $t$ times to ensure that the verifier trusts the prover, because if $b = 0$ (which occurs 50 percent of the time if User B chooses he challenge bit randomly), the prover may simply return $r_a$. This will trick the verifier into accepting the answer even though the prover doesn't have knowledge...
of $S_A$. The number of times the challenge bit check is performed depends on the level of secrecy one wants to achieve, where the probability a user will falsely impersonate someone is $1/(2^t)$. Performing the check $t = \log(N)$ times isn't uncommon.

As long as User X doesn't have the secret key $S_A$ and we run this experiment enough times to satisfy the verifier, this protocol can authenticate a user and address the risk of identity fraud. The verifier also doesn't learn any information about $S_A$ from the protocol because $z_A$ is calculated from $S_i r_A^b \pmod{N}$, which multiplies $S_A$ by a random number.

![Protocol Diagram]

\[ z_A = r_A S_i^b \pmod{N} \]

Performs the following:
\[ \left( r_i S_i^b \right)^y \pmod{N} = z_A x_i \]
\[ y_i N = z_i x_i \]
and accepts if they are equal

**Figure 4-3: Valid verification process that prevents identity fraud**

This protocol uses a small number of computations, so it's not uncommon to see this method on a token card of some sort.
WHAT NEXT?

Though we've covered the main concepts of cryptography in this book, we've forgone exploring several important topics, including the future of cryptography. If you're curious about where cryptography will develop from here or what areas you could explore next, the following sections will give you an idea of where to start.

PSEUDORANDOM NUMBERS AND ENCRYPTION SECURITY

A sequence of random numbers is one of the foundational pieces underlying security. For example, in public-key encryption, the key used to encrypt and decrypt information is generated using a random number. To ensure a good encryption scheme, one must use a random number generator whose results are impossible or very hard to predict (that is, one that generates a random number that can't be predicted based on previous numbers).

Pseudorandom numbers must be distinguished from the concept of physically random numbers (also called truly random numbers). Pseudorandom numbers are produced from a fixed formula (called a pseudorandom number generator), so they are deterministic or predictable by individuals with enough computing power. Some examples of pseudorandom number generators include the linear congruential generator, the middle square method, the M-sequences, the BBS (Blum-Blum-Shub) method, and the one-way hash function.

In contrast, a physically random number is generated based on a natural physical phenomenon, implements a perfectly random sequence of numbers, and can generate an infinite chaotic sequence of numbers. For example, a physical random number can be generated from the noise produced when electricity is distributed to a semiconductor circuit.

Despite the fact that most pseudorandom numbers are not secure, most systems use them instead of physically random numbers because many pseudorandom number generators require an inordinate amount of computing power to distinguish them from random number generators. In addition, the cost of using a truly random number generator is magnitudes higher than that of a pseudorandom number generator. We can't treat pseudorandom numbers as though they are truly random numbers, but in most cases, you can reasonably assume the adversary isn't capable of harnessing that much computing power, so pseudorandom generators suffice. Some top-secret or mission-critical systems may warrant the use of a random number generator, but these are rare.

PGP

The initials PGP stand for Pretty Good Privacy. PGP is a widely used type of encryption software that was invented in 1991 by Philip Zimmermann.

PGP is a vital piece of functionality in nearly all modern encryption software. It makes certificate generation possible in symmetric-key algorithms (AES, 3-DES, and so on), public-key encryption (RSA, ElGamal, and so on), digital signatures (RSA, DSA), and one-way hash functions (MD5, SHA-1, RIPEMD-160, and so on).
SSL/TLS

Communication protocols are used in online shopping and other forms of internet correspondence. When a protocol is used, a message authentication code certifies and authenticates the contents of a correspondence. Two security protocols that are often used for this purpose are SSL (secure socket layer) and TLS (transport layer security). For example, when a credit card number or other secure information is sent through a web browser, SSL or TLS is used to encrypt the data and prevent it from being intercepted. To determine whether you are using SSL/TLS in online communication, you can inspect the URL in your internet browser. Non-SSL/TLS URLs begin with http://, while URLs beginning with https:// indicate SSL/TLS communication.

Like SSL/TLS, the protocols called SMTP (Simple Mail Transfer Protocol) and POP3 (Post Office Protocol) also encrypt emails using SSL/TLS to protect information. SMTP is used to encrypt mail that is being sent out, while POP3 is used for received mail.

QUANTUM CRYPTOGRAPHY

Quantum cryptography is considered an absolutely secure method of encryption. Quantum cryptography makes use of light pulses that contain more than 10,000 photons, which are the smallest unit of measurement for light. A single photon contains 1 bit of information, and the photon's polarity (the direction in which the electromagnetic waves oscillate) is used to distinguish whether the bit is a 0 or a 1. Photons follow Heisenberg's uncertainty principle, which states that a quantum system can't be observed without changing the system's state. This means the data contained in photons can't be copied by an attacker because, if a photon were observed by the attacker, the photon's polarization would be changed and any suspicious activity would be detected. This means that quantum encryption can't be intercepted. When quantum cryptography is combined with a one-time pad that uses a single-use encryption key, cryptographers can create fail-safe encryption.

BIOMETRICS

Biometrics are biological features unique to an individual that are used to confirm a person's identity. Some examples of biometrics include fingerprints, palm veins, facial structures, iris patterns, palm prints, DNA (genes), and so on. Verification systems are seeing more everyday usage, one prominent example being the fingerprint unlock mechanism in smartphones.
IF WE XOR THE HINT ON PAGE 214:
00000010 00001100 00010101 00011100 00000110 00001010
AND THE WORD LIBERTY IN ASCII BINARY:
001110 001110 010110 010001 010010 001100
WE GET
010110 010101 011100 011100 011010 011010
WHICH IS NEW YORK IN ASCII BINARY.
INDEX

A
addition (modulo operations), 137–138
Advanced Encryption Standard (AES), 84–87
algorithms, 20. See also symmetric-key algorithms
Euclidean algorithm, 161–162
exhaustive search algorithm, 81
extended Euclidean algorithm, 181–183
anti-identity fraud, 222–223
associative property (modulo operations), 149
asymmetric cryptography, 110–113. See also public-key encryption authentication, 114, 217. See also message authentication codes

B
BBS (Blum-Blum-Shub) method, 224
binary digits, 48–58
ASCII, 54
bits, 53
byte, 53
converting plaintext into binary, 89
hexadecimal notation, 53
logical operators and bitwise operation, 56
AND operations, 56
OR operations, 56
XOR operation, 55–58
biometrics, 225
bits, 53
origin of term, 19
reduced DES compressing, 101–104
higher-order bits, 92, 95, 97, 99
lower-order bits, 92, 95, 97, 99
rotating, 102, 103
block ciphers, 65
CBC mode, 71
ECB mode, 70
hexadecimal notation, 68–69
key size, 68–69
methods for breaking, 81
padding, 69
Caesar cipher, 24
codes versus, 18
computationally secure, 39
DES ciphers
decryption process, 79
disadvantages of, 81
cryptography process, 79
Feistel ciphers, 73–74, 92–95
involution, 74
key size, 75
NSA standards, 75–76
parity check, 75
reduced DES versus, 104
round function f, 78
subkeys, 77
cipher block chaining (CBC) mode, 71
cipher feedback (CFB), 71
ciphers, 17
block ciphers, 65
CBC mode, 71
ECB mode, 70
hexadecimal notation, 68–69
key size, 68–69
methods for breaking, 81
padding, 69
Caesar cipher, 24
codes versus, 18
computationally secure, 39
DES ciphers
decryption process, 79
disadvantages of, 81
cryptography process, 79
Feistel ciphers, 73–74, 92–95
involution, 74
key size, 75
NSA standards, 75–76
parity check, 75
reduced DES versus, 104
round function f, 78
subkeys, 77
cipher block chaining (CBC) mode, 71
cipher feedback (CFB), 71
reduced DES (continued)
relationship between
encryption key and
decryption key, 21–23
security, 28–46
breaking classical
cipher encryption,
37–40
combinations, 33
frequency analysis,
33
permutations, 32
polyalphabetic
cipher key
space, 33–34
secure ciphers,
39–40
substitution cipher's
key space, 31–32
transposition
cipher key
space, 34–37
stream ciphers, 66–67
substitution cipher, 25,
31–32
Tanuki cipher, 6
transposition cipher, 27,
34–37
Vernam cipher, 37–39
ciphertext, 20. See also
names of specific
encryption methods
breaking classic encryption,
37–40
Caesar cipher, 24
polyalphabetic cipher, 26
simple substitution
cipher, 25
substitution cipher, 25
transposition cipher, 27
RSA encryption
decrypting cipher-
text, 171–172
generating cipher-
text, 169–171
classic encryption
breaking, 37–40
perfectly secure en-
cryption, 37–38
secure ciphers,
39–40
Caesar cipher, 24
polyalphabetic cipher, 26
simple substitution
cipher, 25
substitution cipher, 25
transposition cipher, 27
codes (jargon; procedure
words), 18
collision attacks, 193
combinations, 33
commutative property (mod-
ulo operations), 149
composite numbers
creating, 217
prime numbers and,
123–124
computationally infeasible
problem, 33
computationally secure
ciphers, 39
transposition, 154
conversion rule \(\delta\) (delta), 26
conversion rule \(\sigma\) (sigma), 25
coprime numbers, 143
Euler's totient
function, 156
primality tests, 155
credit cards
SSL/TLS encryption, 226
zero-knowledge Interactive
proof, 217–218
Cryptography. See also
public-key encryption;
symmetric-key
algorithms
asymmetric cryptography,
110–113
importance of, 11–13
quantum
cryptography, 225
currency (money), 207

Daemen,Joan, 85
Data Encryption Standard
ciphers. See DES
ciphers
decryption key
generating, 79, 101,
165–166
relationship between
encryption key and,
21–23
decryption process, 20
DES ciphers, 79
reduced DES
expansion function,
97–98
final permutation, 99
initial permutation, 97
relationship between
encryption process
to, 100
separating data into
higher-order
and lower-order
bits, 97
swapping higher-
and lower-order
bits, 99
RSA, 163–164, 171–172
delta \(\delta\) conversion rule, 26
DES (Data Encryption Standard)ciphers, 72
decryption process, 79
disadvantages of, 81
cryptography, 79
Feistel ciphers, 73–74,
92–95
involution, 74
key size, 75
NSA standards, 75–76
parity check, 75
reduced DES
converting data into
binary, 89
decryption process,
97–100

228 INDEX
DES versus, 104
cryptography process, 90–96
key schedule, 90, 100–104
reduced DES versus, 104
round function, 78
subkeys, 77
differential cryptanalysis, 81
Diffie–Hellman key exchange method, 178
digital signatures, 200–202
certificates and, 204–205
digital signature scheme, 179
man-in-the-middle attacks, 203–205
PGP and, 224
discrete logarithm problem, 115, 173–175
ElGamal encryption, 176–180
extended Euclidean algorithm, 181–183
one-way functions, 116
distributive property (modulo operations), 149
division (modulo operations), 143–147
divisors (factors), 122

E
electric code book (ECB) mode, 70
ElGamal encryption (discrete logarithm problem), 176–180
Encrypt-Encrypt-Encrypt (EDE) mode, 83
encryption, 20. See also
public-key encryption; symmetric-key algorithms
ciphers, 17
block ciphers, 68–71
Caesar cipher, 24
classic encryption, 24–27, 37–40
codes versus, 18
combinations, 33
DES ciphers, 72–81
encryption model, 19–23
frequency analysis, 33
permutations, 32
polyalphabetic cipher, 26, 33–34
relationship between encryption key and decryption key, 21–23
secure, 39–40
security, 28–46
stream ciphers, 65–67
substitution cipher, 25, 31–32
transposition cipher, 27, 34–37
digital signatures, 200–205
ElGamal encryption, 176–180
hash functions, 190–194
hybrid encryption, 185–189
message authentication codes, 195–199
public-key infrastructure, 206–214
RSA encryption, 161–164
decrypting ciphertext, 171–172
generating ciphertext, 169–171
origin of, 119
private-key generation, 165–168
public-key generation, 165–168
SSL/TLS encryption, 189, 197, 225
zero-knowledge interactive proof, 217–223
encryption key, 20

F
factors (divisors), 122
fall-safe ciphers, 39
falsification, 189
Federal Information Processing Standard (FIPS), 85
Feistel, Horst, 72
Feistel ciphers (Feistel network), 73–74, 92–95

G
generating, 79, 101,
165–166
relationship between
decryption key and,
21–23
encryption process
DES ciphers, 79
reduced DES, 90–91
Feistel network, 92–95
final permutation, 95–96
initial permutation, 91
relationship between
decryption process and, 100
separating data into
higher-order and lower-order bits, 92
swapping higher-order and lower-order bits, 95
Shannon's encryption model, 19–23
Entosthenes of Cyrene, 125
Euclidean algorithm, 181–182
Euler, Leonhard, 157
Euler's formula, 157
Euler's totient function (theorem), 156–160
exhaustive search algorithm, 81
extended Euclidean algorithm (discrete logarithm problem), 191–193
Fermat, Pierre de, 153
Fermat's big (last) theorem, 153
Fermat's little theorem (Fermat's method), 152–155
Ferris wheel model (modulo operations), 136–139
addition and subtraction, 136–139
multiplication and division, 146–147
fields (modulo operations), 149
final permutation (FP), 95–96, 99
finite fields, 148
FIPS (Federal Information Processing Standard), 85
FP (final permutation), 95–96, 99
fractions, 121
frequency analysis, 33

G
The Gold Bug (Poe), 35–36

H
hash functions, 190–194, 224
Heisenberg's uncertainty principle, 225
higher-order bits (reduced DES decryption process), 92, 97
separating data into, 92, 97
swapping with lower-order bits, 95, 99
hybrid encryption, 185–186
public-key encryption process, 187–189
symmetric-key algorithms, 187

I
identity fraud
countermeasures, 195–199
zero-knowledge interactive proof, 220–223
implementation stage (zero-knowledge interactive proof), 218–220
infinite fields (modulo operations), 149
initialization, 207
initialization vector, 71
initial permutation (IP), 91, 97
integer factorization problem, 115
Euler's Theorem, 156–160
Fermat's little theorem, 152–155
modulo operations, 134–137
addition, 137–138
associative property, 149
commutative property, 149
distributive property, 149
division, 143–147
fields, 149
multiplication, 140–143, 146–147
power table, 150–151
subtraction, 139
one-way functions, 116
prime numbers and, 120–123
composite numbers and, 123–124
primality tests, 129–133
sieve of Eratosthenes, 124–129
RSA encryption, 119, 161–172
integers, 121
Integrity (messages), 193, 197
inverse element (reciprocal), 144–145
involution (DES ciphers), 74
IP (initial permutation), 91, 97
irrational numbers, 121
irreversibility property, 193
J
jargon (codes), 18
K
key schedule (reduced DES), 90, 100–101
compressing bits, 103–104
PC-1, 101–102
PC-2, 102–103
rotating bits, 102, 103
key size
DES ciphers, 75
3-DES, 83–84
key space, 29
polyalphabetic cipher, 33–34
substitution cipher, 31–32
transposition cipher, 34–37
L
letter-by-letter replacement method, 25
linear congruential generator, 224
linear cryptanalysis, 81
lower-order bits
reduced DES decryption process
separating data into, 97
swapping with higher-order bits, 99
reduced DES encryption process
separating data into, 92
swapping with higher-order bits, 95
lowest common multiple, 158
M

MACs (message authentication codes), 195–199
man-in-the-middle attacks, 203–205
“A Mathematical Theory of Communication” article (Shannon), 19
MD5 hash function, 193
message authentication codes (MACs), 195–199
middle square method, 224
Miller-Rabin method (primality tests), 129
modulo operations, 134–137
addition, 137–138
associative property, 149
commutative property, 149
distributive property, 149
division, 143–147
fields, 149
multiplication, 140–143, 146–147
power table, 150–151
subtraction, 139
M-series pseudorandom number generator, 224
multiplication (modulo operations), 140–143, 146–147

N

National Security Agency (NSA) standards (DES ciphers), 75–76
natural numbers, 121, 181–182
nonnegative integers, 121

O

OFB (output feedback), 71
one-time pad, 37, 225
one-way functions, 115–118
discrete logarithm problem, 116
integer factorization problem, 115
output feedback (OFB), 71
P
padding (block ciphers), 69
parity check (DES ciphers), 75
PC-1 (permutation 1)
(DES keys), 77, 101-102
PC-2 (permutation 2)
(DES keys), 77, 102-103
permutations, 32
final permutation, 95, 96, 99
initial permutation, 91, 97
permuted choice 1 (PC-1)
(DES keys), 77, 101-102
permuted choice 2 (PC-2)
(DES keys), 77, 102-103
PGP (Pretty Good Privacy), 189, 224
photons (quantum cryptography), 225
physically random numbers (true random numbers), 224
PKI (public-key infrastructure), 206–214
plaintext, 20. See also names of specific encryption methods
breaking classic encryption, 37–40
Caesar cipher, 24
converting into binary, 89
describing ciphers, 26
simple substitution cipher, 25
substitution cipher, 25
transposition cipher, 27
Poe, Edgar Allen, 35–36
polyalphabetic cipher, 26,
33–34
POPG (Post Office Protocol), 225
power table (modulo operations), 150–151
preparatory stage (zero-knowledge interactive proof), 217–218
Pretty Good Privacy (PGP), 189, 224
primality tests, 129–133
prime numbers, 120–123
composite numbers and, 123–124
coprime numbers, 143
Rabin's totient function, 156
primality tests, 155
primality tests, 129–133
sieve of Eratosthenes, 124–129
primitive roots, 174
private key
calculating and sending to user, 218
making, 167–168
RSA encryption
calculating, 182–183
generating, 165–168
verifying compatibility, 166
procedure words (codes), 18
proposition, 154
prove (zero-knowledge interactive proof), 218, 219
pseudoprimes, 129, 155
pseudorandom number generator, 224
pseudorandom numbers, 66–67, 151, 155, 224
public key
making, 167–168
RSA encryption
generating, 165–168
verifying compatibility, 166

INDEX 231
public-key encryption, 61, 106–109
asymmetric cryptography, 110–113
authentication, 114
discrete logarithm problem, 115, 173–175
ElGamal encryption, 176–180
extended Euclidean algorithm, 181–183
hybrid encryption, 187–189
integer factorization, 115
Euler’s Theorem, 156–160
Fermat’s Little Theorem, 152–155
modulo operations, 134–151
prime numbers and, 120–133
RSA encryption, 119, 161–172
one-way functions, 115–118
PGP and, 224
viewing public key, 109
public-key infrastructure (PKI), 206–214

Q
quantum cryptography, 225

R
rational numbers, 121, 149
reciprocal (inverse element), 144–145
reduced DES
converting data into binary, 89
decryption process
expansion function, 97–98
final permutation, 99
initial permutation, 97
relationship between encryption process and, 100
separating data into higher-order and lower-order bits, 97
swapping higher-order and lower-order bits, 99
DES versus, 104
cryptography process, 90–91
Feistel network, 92–95
final permutation, 95–96
initial permutation, 91
relationship between decryption process and, 100
separating data into higher-order and lower-order bits, 92
swapping higher-order and lower-order bits, 95
key schedule, 90, 100–101
compressing bits, 103–104
PC-1, 101–102
PC-2, 102–103
rotating bits, 102, 103
repository, 204, 209–210
repudiation, 197, 199–200
Rijmen, Vincent, 85
Rijndael algorithm (Advanced Encryption Standard), 84–87
round function $f$ (DES ciphers), 78
RSA encryption, 161–164
calculating private key generation, 182–183
decrypting ciphertext, 171–172

generating ciphertext, 168–171
origin of, 119
private key generation, 165–168
public key generation, 165–168
S
S-box (substitution box), 78, 83
secret-key cryptography.
See symmetric-key algorithms
secure ciphers, 39–40
secure socket layer/transport layer security (SSL/TLS) encryption, 189, 197, 225
server certificates (certificates), 202, 204, 209, 210, 213
SHA-1 hash function, 193
Shannon, Claude, 19, 37
Shannon’s encryption model, 19–23
shared-key cryptography.
See symmetric-key algorithms
shared keys, 59–64
sieve of Eratosthenes, 124–129
sigma ($\sigma$) conversion rule, 25
simple substitution ciphers, 25
SMTP (Simple Mail Transfer Protocol) protocol, 225
SPN (substitution-permutation network), 85
SSL/TLS (secure socket layer/transport layer security) encryption, 189, 197, 225
stream ciphers, 65–67
strong collision resistance (hash functions), 193
subkeys (DES ciphers), 77
INDEX 233
ABOUT THE AUTHORS

MASAAKI MITANI

Masaaki Mitani is from the town of Setoda, formerly known as Toyota, in Onomachi of the Hiroshima prefecture. He graduated in 1974 from the Department of Electrical and Electronic Engineering in the School of Engineering at the Tokyo Institute of Technology. After working as an assistant professor at his alma mater, he is now a professor at Tokyo Denki University’s School of Engineering in the Department of Information and Communication Engineering. He specializes in digital signal processing engineering, communication engineering, and educational engineering. He has a PhD in Engineering.

PRINCIPAL WORKS

- An Introduction to Digital Signal Processing (Ohmsha, Ltd.)
- Learning Digital Signal Processing using Scilab (CQ Publishing)
- A Friendly Introduction to Electronic Circuits, Introduction (1) to Diodes and Transistors (Micronet)
- Re-Learning Signal Mathematics (CQ Publishing)
- Immediately Applicable Fourier Transform (Kodansha Ltd.)

SHINICHI SATO

Shinichi Sato is from the town of Date in the Fukushima prefecture. In 1990, he earned his masters degree in electrical engineering at Tokyo Denki University’s graduate student program. He worked on designing imaging machinery at a private corporation before conducting research related to biotics in a private institution’s medical faculty as an assistant researcher. He is now an assistant professor at Tokyo Denki University’s School of Engineering in the Department of Information and Communication Engineering. He specializes in signal processing engineering and educational engineering.
PRODUCTION TEAM FOR THE JAPANESE EDITION

Production: Verte Corp.
Editing: Tsugumi Endo
DTP: Satoshi Aral

HOW THIS BOOK WAS MADE

The Manga Guide series is a co-publication of No Starch Press and Ohms ha, Ltd. of Tokyo, Japan, one of Japan’s oldest and most respected scientific and technical book publishers. Each title in the best-selling Manga Guide series is the product of the combined work of a manga illustrator, scenario writer, and expert scientist or mathematician. Once each title is translated into English, we rewrite and edit the translation as necessary and have an expert review each volume. The result is the English version you hold in your hands.
MORE MANGA GUIDES

Find more Manga Guides at your favorite bookstore, and learn more about the series at https://www.nostarch.com/manga/.
DECODE THE MYSTERIES OF CRYPTOGRAPHY

CYPHERS HAVE BECOME MUCH MORE SOPHISTICATED. AFTER ALL, IF YOU WANT TO PASS SECRET MESSAGES, YOU HAVE TO DEAL WITH EAVESDROPPERS TRYING TO DECRYPT THEM.

LET'S TAKE A LOOK AT SOME CLASSIC ENCRYPTION METHODS TO GET A BETTER UNDERSTANDING OF THIS GAME OF CAT AND MOUSE!

Inspector Jun Meguro has been tasked with tracking down a crafty art thief who leaves clues in the form of perplexing cryptographic puzzles. Lucky for him, his sister Ruko is a math whiz who is willing to teach him about cryptography so that he can crack the case. But can they decode the thief's ciphers before she steals again?

As you follow Jun and Ruko's quest to bring the cipher-wielding thief to justice, you'll learn real cryptography and the math behind it, including:

- The foundations of encryption, like the Caesar and Vernam ciphers
- How substitution ciphers, public-key cryptography, and digital signatures work
- How to use binary, logic operations, and modular arithmetic to turn plaintext messages into numeric values that can be encrypted
- How the Data Encryption Standard (DES) cipher works
- How to encrypt messages with hash functions, message authentication codes (MACs), and public key infrastructures (PKIs)

The Manga Guide to Cryptography is the perfect introduction to cryptography for programmers, security professionals, and aspiring cryptographers.

Find more manga guides at www.nostarch.com/manga

SHUEISHA INC. / COMPUTERS/SECURITY