The man pictured above is Alan Turing, the most important figure in the history of computer science. For decades, his fascinating life story was shrouded by government secrecy, societal taboo, and even his own deceptions.

At 24 Turing wrote a paper entitled *On Computable Numbers, with an Application to the Entscheidungsproblem*. The crux of the paper was an elegant way to model a computer in mathematical terms. This was a breakthrough, because it allowed the tools of mathematics to be brought to bear on questions of computation. For example, with his model in hand, Turing immediately proved that there exist problems that no computer can solve—no matter how ingenious the programmer. Turing’s paper is all the more remarkable, because it dates to 1936, a full decade before any computer actually existed.

The word “Entscheidungsproblem” in the title refers to one of the 28 mathematical problems posed by David Hilbert in 1900 as challenges to mathematicians of the 20th century. Turing knocked that one off in the same paper. And perhaps you’ve heard of the “Church-Turing thesis”? Same paper. So Turing was obviously a brilliant guy who generated lots of amazing ideas. But this lecture is about one of Turing’s less-amazing ideas. It involved codes. It involved number theory. It was sort of stupid.

1 A Theory of the Integers

*Number theory* is the study of the integers. Why anyone would want to study the integers is not immediately obvious. First of all, what’s to know? There’s 0, there’s 1, 2, 3 and so
on, and there’s the negatives. Which one don’t you understand? And who cares, anyway? After all, the mathematician G. H. Hardy wrote:

[Number theorists] may be justified in rejoicing that there is one science, at any rate, and that their own, whose very remoteness from ordinary human activities should keep it gentle and clean.

What most concerned Hardy was that number theory not be used in warfare; he was a pacifist. Good for him, but if number theory is remote from all human activity, then why study it?

Let’s look back to the fall of 1937. Nazi Germany was rearming under Adolf Hitler, world-shattering war looked imminent, and—like us—Alan Turing was pondering the usefulness of number theory. He foresaw that preserving military secrets would be vital in the coming conflict and proposed a way to encrypt communications using number theory. This was an idea that has ricocheted up to our own time. Today, number theory is the basis for numerous public-key cryptosystems, digital signature schemes, cryptographic hash functions, and digital cash systems. Every time you buy a book from Amazon, check your grades on WebSIS, or use a PayPal account, you are relying on number theoretic algorithms.

Soon after devising his code, Turing disappeared from public view, and half a century would pass before the world learned the full story of where he’d gone and what he did there. We’ll come back to Turing’s life in a little while; for now, let’s investigate the code Turing left behind. The details are uncertain, since he never formally published the idea, so we’ll consider a couple possibilities. But, first, we need some number theory.

All quantities discussed in this lecture are integers with a few exception that are noted explicitly.

2 Divisibility

We say that \( a \) divides \( b \) if there is an integer \( k \) such that \( ak = b \). This is denoted \( a \mid b \). For example, \( 7 \mid 63 \), because \( 7 \cdot 9 = 63 \). A consequence of this definition is that every number divides zero. If \( a \) divides \( b \), then \( b \) is a multiple of \( a \). For example, 63 is a multiple of 7.

Divisibility is a simple idea, but sufficient to dispel the notion that number theory lacks depth. The ancient Greeks considered a number perfect if it equalled the sum of its positive divisors, excluding itself. For example, \( 6 = 1+2+3 \) and \( 28 = 1+2+4+7+14 \) are perfect numbers. Euclid characterized all the even perfect numbers around 300 BC. But is there an odd perfect number? More than two thousand years later, we still don’t know! All numbers up to about \( 10^{300} \) have been ruled out, but no one has proved that there isn’t a odd perfect number waiting just over the horizon. Number theory is full of questions like this: easy to pose, but incredibly difficult to answer.

The lemma below states some basic facts about divisibility that are not difficult to prove:
Lemma 1. The following statements about divisibility hold.

1. If $a \mid b$, then $a \mid bc$ for all $c$.
2. If $a \mid b$ and $b \mid c$, then $a \mid c$.
3. If $a \mid b$ and $a \mid c$, then $a \mid sb + tc$ for all $s$ and $t$.
4. For all $c \neq 0$, $a \mid b$ if and only if $ca \mid cb$.

Proof. We’ll only prove part (2); the other proofs are similar. Since $a \mid b$, there exists an integer $k_1$ such that $ak_1 = b$. Since $b \mid c$, there exists an integer $k_2$ such that $bk_2 = c$. Substituting $ak_1$ for $b$ in the second equation gives $ak_1k_2 = c$, which implies that $a \mid c$. \(\square\)

A number $p > 1$ with no positive divisors other than 1 and itself is called a prime. Every other number greater than 1 is called composite. The number 1 is considered neither prime nor composite. This is just a matter of definition, but reflects the fact that 1 does not behave like a prime in some important contexts, such as the Fundamental Theorem of Arithmetic.

Here is one more essential fact about divisibility and primes.

Theorem 2. Let $p$ be a prime. If $p \mid a_1a_2 \ldots a_n$, then $p$ divides some $a_i$.

For example, if you know that $19 \mid 403 \cdot 629$, then you know that either $19 \mid 403$ or $19 \mid 629$, though you might not know which! The proof of this theorem takes some work, which we’ll defer to avoid bogging down in minutiae here at the outset.

2.1 Turing’s Code (Version 1.0)

Now let’s look at Turing’s scheme for using number theory to encrypt messages. First, the message must be translated to a prime number. This step is not intended to make a message harder to read, so the details are not too important. Here is one approach: replace each letter of the message with two digits ($A = 01$, $B = 02$, $C = 03$, etc.), string all the digits together to form one huge number, and then append digits as needed to produce a prime. For example, the message “victory” could be translated this way:

“v i c t o r y”

$\rightarrow$ 22 09 03 20 15 18 25 13

Appending the digits 13 gives the number 2209032015182513, which is a prime.

Now here is how the encryption process works. In the description below, the variable $m$ denotes the unencoded message (which we want to keep secret), and $m'$ denotes the encrypted message (which the Nazis may intercept).
Famous Problems in Number Theory

**Fermat’s Last Theorem** Do there exist positive integers \(x, y,\) and \(z\) such that

\[x^n + y^n = z^n\]

for some integer \(n > 2\)? In a book he was reading around 1630, Fermat claimed to have a proof, but not enough space in the margin to write it down. Wiles finally solved the problem in 1994, after seven years of working in secrecy and isolation in his attic.

**Goldbach Conjecture** Is every even integer greater than or equal to 4 the sum of two primes? For example, \(4 = 2 + 2, 6 = 3 + 3, 8 = 3 + 5,\) etc. The conjecture holds for all numbers up to \(10^{16}\). In 1939 Schnirelman proved that every even number can be written as the sum of not more than 300,000 primes, which was a start. Today, we know that every even number is the sum of at most 6 primes.

**Twin Prime Conjecture** Are there infinitely many primes \(p\) such that \(p + 2\) is also a prime? In 1966 Chen showed that there are infinitely many primes \(p\) such that \(p + 2\) is the product of at most two primes. So the conjecture is known to be *almost* true!

**Prime Number Theorem** How are the primes distributed? Let \(\pi(x)\) denote the number of primes less than or equal to \(x\). Primes are very irregularly distributed, so this is a complicated function. However, the Prime Number Theorem states that \(\pi(x)\) is very nearly \(x/\ln x\). The theorem was conjectured by Legendre in 1798 and proved a century later by de la Vallée Poussin and Hadamard in 1896. However, after his death, a notebook of Gauss was found to contain the conjecture, which he apparently made in 1791 at age 14.

**Primality Testing** Is there an efficient way to determine whether \(n\) is prime? An amazing simple, yet efficient method was finally discovered in 2002 by Agrawal, Kayal, and Saxena. Their paper began with a quote from Gauss emphasizing the importance and antiquity of the problem even in his time—two centuries ago.

**Factoring** Given the product of two large primes \(n = pq,\) is there an efficient way to recover the primes \(p\) and \(q?\) The best known algorithm is the “number field sieve”, which runs in time proportional to:

\[e^{1.9(\ln n)^{1/3}(\ln \ln n)^{2/3}}\]

This is infeasible when \(n\) has a couple hundred digits or more.
Beforehand The sender and receiver agree on a secret key, which is a large prime \( p \).

Encryption The sender encrypts the message \( m \) by computing:

\[
m' = m \cdot p
\]

Decryption The receiver decrypts \( m' \) by computing:

\[
\frac{m'}{p} = \frac{m \cdot p}{p} = m
\]

For example, suppose that the secret key is the prime number 22801763489 and the message \( m \) is “victory”. Then the encrypted message is:

\[
m' = m \cdot p
\]
\[
= 2209032015182513 \cdot 22801763489
\]
\[
= 50369825549820718594667857
\]

Turing’s code raises a couple immediate questions.

1. How can the sender and receiver ensure that \( m \) and \( p \) are prime numbers, as required?

The general problem of determining whether a large number is prime or composite has been studied for centuries, and reasonably good primality tests were known even in Turing’s time. In 2002, Manindra Agrawal, Neeraj Kayal, and Nitin Saxena announced a primality test that is guaranteed to work on a number \( n \) in about \((\log n)^{12}\) steps. This definitively placed primality testing in the class of “easy” computational problems at last. Amazingly, the description of their algorithm is only thirteen lines!

2. Is Turing’s code secure?

The Nazis see only the encrypted message \( m' = m \cdot p \), so recovering the original message \( m \) requires “factoring” \( m' \). Despite immense efforts, no really efficient factoring algorithm has ever been found. It appears to be a fundamentally difficult problem, though a breakthrough is not impossible. In effect, Turing’s code puts to practical use his discovery that there are limits to the power of computation. Thus, provided \( m \) and \( p \) are sufficiently large, the Nazis seem to be out of luck!

Nevertheless, there is a major flaw in Turing’s code. Can you find it? We’ll reveal the weakness after saying a bit more about divisibility.
2.2 The Division Algorithm

As you learned in elementary school, if one number does not evenly divide another, then there is a “remainder” left over. More precisely, if you divide \( n \) by \( d \), then you get a quotient \( q \) and a remainder \( r \). This basic fact is the subject of a useful theorem:

**Theorem 3 (Division Algorithm).** Let \( n \) and \( d \) be integers such that \( d > 0 \). Then there exists a unique pair of integers \( q \) and \( r \) such that \( n = qd + r \) and \( 0 \leq r < d \).

**Proof.** We must prove that the integers \( q \) and \( r \) exist and that they are unique.

For existence, we use the well-ordering principle. First, we show that the equation \( n = qd + r \) holds for some \( r \geq 0 \). If \( n \) is positive, then the equation holds when \( q = 0 \) and \( r = n \). If \( n \) is not positive, then the equation holds when \( q = n \) and \( r = n(1 - d) \geq 0 \). Thus, by the well-ordering principle, there must exist a smallest \( r \geq 0 \) such that the equation holds. Furthermore, \( r \) must be less than \( d \); otherwise, \( b = (q + 1)d + (r - d) \) would be another solution with a smaller nonnegative remainder, contradicting the choice of \( r \).

Now we show uniqueness. Suppose that that there exist two different pairs of integers \( q_1, r_1 \) and \( q_2, r_2 \) such that:

\[
\begin{align*}
  n &= q_1d + r_1 \quad \text{(where } 0 \leq r_1 < d) \\
  n &= q_2d + r_2 \quad \text{(where } 0 \leq r_2 < d)
\end{align*}
\]

Subtracting the second equation from the first gives:

\[
0 = (q_1 - q_2)d + (r_1 - r_2)
\]

The absolute difference between the remainders \( r_1 \) and \( r_2 \) must be less than \( d \), since \( 0 \leq r_1, r_2 < d \). This implies that the absolute value of \( (q_1 - q_2)d \) must also be less than \( d \), which means that \( q_1 - q_2 = 0 \). But then the equation above implies that \( r_1 - r_2 = 0 \) as well. Therefore, the pairs \( q_1, r_1 \) and \( q_2, r_2 \) are actually the same, which is a contradiction. So the quotient and remainder are unique. \( \square \)

This theorem is traditionally called the “Division Algorithm”, even though it is really a statement about the long-division procedure. As an example, suppose that \( a = 10 \) and \( b = 2716 \). Then the quotient is \( q = 271 \) and the remainder is \( r = 6 \), since \( 2716 = 271 \cdot 10 + 6 \).

The remainder \( r \) in the Division Algorithm is often denoted \( b \mod a \). In other words, \( b \mod a \) is the remainder when \( b \) is divided by \( a \). For example, \( 32 \mod 5 \) is the remainder when 32 is divided by 5, which is 2. Similarly, \( -11 \mod 7 = 3 \), since \( -11 = (-2) \cdot 7 + 3 \).

Some people use the notation “mod” (which is short for “modulo”) instead of “rem”. This is unfortunate, because “mod” has been used by mathematicians for centuries in a confusingly similar context, and we shall do so here as well. Until those people are, shall we say, liquidated, you’ll have to cope with the confusion.

Many programming languages have a remainder or modulus operator. For example, the expression “32 % 5” evaluates to 2 in Java, C, and C++. However, all these languages treat negative numbers strangely.
2.3 Breaking Turing’s Code

Let’s consider what happens when the sender transmits a second message using Turing’s code and the same key. This gives the Nazis two encrypted messages to look at:

\[ m'_1 = m_1 \cdot p \quad \text{and} \quad m'_2 = m_2 \cdot p \]

The greatest common divisor (gcd) of \( a \) and \( b \) is the largest integer \( c \) such that \( c \mid a \) and \( c \mid b \). For example, the greatest common divisor of 9 and 15 is \( \gcd(9, 15) = 3 \). In this case, the greatest common divisor of the two encrypted messages, \( m'_1 \) and \( m'_2 \), is the secret key \( p \). And, as we’ll see, the gcd of two numbers can be computed very efficiently. So after the second message is sent, the Nazis can read recover the secret key and read every message!

It is difficult to believe a mathematician as brilliant as Turing could overlook such a glaring problem. One possible explanation is that he had a slightly different system in mind, one based on modular arithmetic.

3 Modular Arithmetic

On page 1 of his masterpiece on number theory, *Disquisitiones Arithmeticae*, C. F. Gauss introduced the notion of “congruence”. Now, Gauss is another guy who managed to cough up a half-decent idea every now and then, so let’s take a look at this one. Gauss said that \( a \) is congruent to \( b \) modulo \( c \) if \( c \mid (a - b) \). This is denoted \( a \equiv b \pmod{c} \). For example:

\[ 29 \equiv 15 \pmod{7} \quad \text{because} \quad 7 \mid (29 - 15). \]

Intuitively, the \( \equiv \) symbol is sort of like an = sign, and the \( \pmod{7} \) describes the specific sense in which 29 is equal-ish to 15. Thus, even though \( \pmod{7} \) appears on the right side, it is in no sense more strongly associated with the 15 than the 29; in fact, it is actually associated with the \( \equiv \) sign.

3.1 Congruence and Remainders

There is a strong relationship between congruence and remainders. In particular, *two numbers are congruent modulo c if and only if they have the same remainder when divided by c.* For example, 19 and 32 are congruent modulo 13, because both leave a remainder of 6 when divided by 13. We state this as a Lemma:

Lemma 4.

\[ a \equiv b \pmod{c} \quad \text{if and only if} \quad (a \rem c) = (b \rem c) \]
We’ll prove this in a moment, but an example is probably more convincing. Some integers are listed on the first line of the table below. The second line lists the remainders when those integers are divided by 3.

<table>
<thead>
<tr>
<th>#</th>
<th>...</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td># rem 3</td>
<td>...</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>...</td>
</tr>
</tbody>
</table>

Notice that numbers on the first line which differ by a multiple of 3 have the same remainder. For example, -2 and 4 differ by 6 and both leave remainder 1. This is precisely what the lemma asserts for $c = 3$; $a \equiv b \pmod{3}$ means that $a$ and $b$ differ by a multiple of 3, and $(a \text{ rem } 3) = (b \text{ rem } 3)$ means that they leave the same remainder when divided by 3.

**Proof.** By the division algorithm, there exist unique pairs of integers $q_1, r_1$ and $q_2, r_2$ such that:

\[
a = q_1c + r_1 \quad (\text{where } 0 \leq r_1 < c)
\]
\[
b = q_2c + r_2 \quad (\text{where } 0 \leq r_2 < c)
\]

In these terms, $(a \text{ rem } c) = r_1$ and $(b \text{ rem } c) = r_2$. Subtracting the second equation from the first gives:

\[
a - b = (q_1 - q_2)c + (r_1 - r_2) \quad (\text{where } -c < r_1 - r_2 < c)
\]

Now $a \equiv b \pmod{c}$ if and only if $c$ divides the left side. This is true if and only if $c$ divides the right side, which holds if and only if $r_1 - r_2$ is a multiple of $c$. Given the bounds on $r_1 = r_2$, this happens precisely when $r_1 = r_2$, which is equivalent to $(a \text{ rem } c) = (b \text{ rem } c)$.

Mathematics done with congruences instead of traditional equations is usually called “modular arithmetic.” If anything, the importance of modular arithmetic has grown since the time of Gauss. For him, congruences were a reasoning tool. These days, computer hardware works with fixed-sized chunks of data, so the arbitrarily large integers that can come up in ordinary arithmetic are problematic. A standard solution is to design computers to do modular arithmetic instead. For example, a computer with 64-bit internal registers typically does integer arithmetic modulo $2^{64}$. Thus, an instruction to add the contents of registers $A$ and $B$ actually computes $(A + B) \text{ rem } 2^{64}$.

### 3.2 Facts about rem and mod

Many familiar rules remain valid when one works modulo an integer $n \geq 1$. For example, we can add a constant to both sides of a congruence:

\[
a \equiv b \pmod{n} \quad \text{implies} \quad a + c \equiv b + c \pmod{n}
\]
Whenever you are unsure about a relationship involving congruences or remainders, go back to the definitions. For example, \( a \equiv b \pmod{n} \) means that \( n \mid (a - b) \). We can rewrite this as \( n \mid ((a + c) - (b + c)) \), which means that \( a + c \equiv b + c \pmod{n} \) as we claimed above.

There is one glaring difference between traditional arithmetic and modular arithmetic. You can cancel multiplicative terms on opposite sides of an ordinary equation:

\[
a \neq b \implies a = b \quad \text{(provided } c \neq 0)\]

However, you can not always cancel such multiplicative terms in a congruence. Here is an example where a true statement becomes a false statement after cancelling:

\[
2 \cdot 3 \equiv 4 \cdot 3 \pmod{6} \quad \leftarrow \text{This is an error!}
\]

We’ll revisit this issue of cancellation. Meanwhile, let’s get more practice with \texttt{rem} and \texttt{mod} by proving some basic facts:

**Lemma 5.** The following assertions hold for all \( n \geq 1 \):

1. If \( a_1 \equiv b_1 \pmod{n} \) and \( a_2 \equiv b_2 \pmod{n} \), then \( a_1 a_2 \equiv b_1 b_2 \pmod{n} \).
2. \( (a \texttt{ rem } n) \equiv a \pmod{n} \)
3. \( (a_1 \texttt{ rem } n) \cdot (a_2 \texttt{ rem } n) \cdots (a_k \texttt{ rem } n) \equiv a_1 a_2 \cdots a_k \pmod{n} \)

**Proof.** We prove each part separately.

1. The conditions \( a_1 \equiv b_1 \pmod{n} \) and \( a_2 \equiv b_2 \pmod{n} \) are equivalent to the assertions \( n \mid (a_1 - b_1) \) and \( n \mid (a_2 - b_2) \). By part 3 of Theorem 1, we know that:

\[
n \mid a_2(a_1 - b_1) + b_1(a_2 - b_2)
\]

Simplifying the right side gives \( n \mid a_1 a_2 - b_1 b_2 \). Thus, \( a_1 a_2 \equiv b_1 b_2 \pmod{n} \) as claimed.

2. Recall that \( a \texttt{ rem } n \) is equal to \( a - qn \) for some quotient \( q \). We can reason as follows:

\[
\begin{align*}
\text{(i)} & \quad n \mid qn \\
\Rightarrow & \quad n \mid a - (a - qn) \\
\Rightarrow & \quad n \mid a - (a \texttt{ rem } n)
\end{align*}
\]

The last statement is equivalent to \( (a \texttt{ rem } n) \equiv a \pmod{n} \).

3. (sketch) We can extend the congruence in part 1 to \( k \) variables using induction. This general assertion that products are congruent if their terms are congruent, together with part 2, proves the claim.
3.3 Turing’s Code (Version 2.0)

In 1940, France had fallen before Hitler’s army, and Britain alone stood against the Nazis in western Europe. British resistance depended on a steady flow of supplies brought across the north Atlantic from the United States by convoys of ships. These convoys were engaged in a cat-and-mouse game with German “U-boat” submarines, which prowled the Atlantic, trying to sink supply ships and starve Britain into submission. The outcome of this struggle pivoted on a balance of information: could the Germans locate convoys better than the Allies could locate U-boats or vice versa?

Germany lost.

But a critical reason behind Germany’s loss was made public only in 1974: the British had broken Germany’s naval code, Enigma. Through much of the war, the Allies were able to route convoys around German submarines by listening into German communications. The British government didn’t explain how Enigma was broken until 1996. When the analysis was finally released (by the US), the author was none other than Alan Turing. In 1939 he had joined the secret British codebreaking effort at Bletchley Park. There, he played a central role in cracking the German’s Enigma code and thus in preventing Britain from falling into Hitler’s hands.

Governments are always tight-lipped about cryptography, but the half-century of official silence about Turing’s role in breaking Enigma and saving Britain may have been something more, perhaps related to Turing’s life after the war, what the government did to him, and his tragic end.

We’ll come back to Turing’s story shortly. Meanwhile, let’s consider an alternative interpretation of his code. Perhaps we had the basic idea right (multiply the message by the key), but erred in using conventional arithmetic instead of modular arithmetic. Maybe this is what Turing meant:

**Beforehand** The sender and receiver agree on a large prime $p$, which may be made public. They also agree on a secret key $k \in \{1, 2, \ldots, p - 1\}$.

**Encryption** The message $m$ can be any integer in the set $\{1, 2, \ldots, p - 1\}$. The sender encrypts the message $m$ to produce $m'$ by computing:

$$m' = mk \text{ rem } p$$

**Decryption** The receiver decrypts $m'$ by finding a message $m$ such that equation (*) holds.

The decryption step is troubling. How can the receiver find a message $m$ satisfying equation (*) except by trying every possibility? That could be a lot of effort! Addressing this defect and understanding why Turing’s code works at all requires a bit more number theory.
3.4 Cancellation Modulo a Prime

An immediate question about Turing’s code is whether there could be two different messages with the same encoding. For example, perhaps the messages “Fight on!” and “Surrender!” actually encrypt to the same thing. This would be a disaster, because the receiver could not possibly determine which of the two was actually sent! The following lemma rules out this unpleasant possibility.

Lemma 6. Suppose $p$ is a prime and $k$ is not a multiple of $p$. If

$$ak \equiv bk \pmod{p}$$

then:

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

Proof. If $ak \equiv bk \pmod{p}$, then $p \mid (ak - bk)$ by the definition of congruence, and so $p \mid k(a - b)$. Therefore, $p$ divides either $k$ or $a - b$ by Theorem 2. The former case is ruled out by assumption, so $p \mid (a - b)$, which means $a \equiv b \pmod{p}$. □

To understand the relevance of this lemma to Turing’s code, regard $a$ and $b$ as two messages. Their encryptions are the same only if:

$$(ak \text{ rem } p) = (bk \text{ rem } p)$$

or, equivalently:

$$ak \equiv bk \pmod{p}$$

But then the lemma implies that $a \equiv b \pmod{p}$. Since the messages $a$ and $b$ are drawn from the set $\{1, 2, \ldots, p - 1\}$, this means that $a = b$. In short, two messages encrypt to the same thing only if they are themselves identical.

In the bigger picture, Lemma 6 says that the encryption operation in Turing’s code permutes the space of messages. This is stated more precisely in the following corollary.

Corollary 7. Suppose $p$ is a prime and $k$ is not a multiple of $p$. Then the sequence:

$$(0 \cdot k) \text{ rem } p, \ (1 \cdot k) \text{ rem } p, \ (2 \cdot k) \text{ rem } p, \ \ldots, \ ((p - 1) \cdot k) \text{ rem } p$$

is a permutation of the sequence:

$$0, \ 1, \ 2, \ \ldots, \ (p - 1)$$

This remains true if the first term is deleted from each sequence.

Proof. The first sequence contains $p$ numbers, which are all in the range 0 to $p - 1$ by the definition of remainder. By Lemma 6, no two of these are congruent modulo $p$ and thus no two are equal. Therefore, the first sequence must be all of the numbers from 0 to $p - 1$ in some order. The claim remains true if the first terms are deleted, because both sequences begin with 0. □
For example, suppose \( p = 5 \) and \( k = 3 \). Then the sequence:

\[
\begin{align*}
(0 \cdot 3) \mod 5 &= 0, \\
(1 \cdot 3) \mod 5 &= 3, \\
(2 \cdot 3) \mod 5 &= 1, \\
(3 \cdot 3) \mod 5 &= 4, \\
(4 \cdot 3) \mod 5 &= 2
\end{align*}
\]

is a permutation of 0, 1, 2, 3, 4 and the last four terms are a permutation of 1, 2, 3, 4. As long as the Nazis don’t know the secret key \( k \), they don’t know how the message space is permuted by the process of encryption and thus can’t read encoded messages.

Lemma 6 also has a broader significance: it identifies one situation in which we can safely cancel a multiplicative term in a congruence. For example, if we know that:

\[
8x \equiv 8y \pmod{37}
\]

then we can safely conclude that:

\[
x \equiv y \pmod{37}
\]

because 37 is prime, and 8 is not a multiple of 37. We’ll come back to this issue later and prove a more general theorem about cancellation.

### 3.5 Multiplicative Inverses

The real numbers have a nice quality that the integers lack. Namely, every nonzero real number \( r \) has a multiplicative inverse \( r^{-1} \) such that \( r \cdot r^{-1} = 1 \). For example, the multiplicative inverse of \(-3\) is \(-1/3\). Multiplicative inverses provide a basis for division: \( a/b \) can be defined as \( a \cdot b^{-1} \). In contrast, most integers do not have multiplicative inverses within the set of integers. For example, no integer can be multiplied by 5 to give 1. As a result, if we want to divide integers, we are forced to muck about with remainders.

Remarkably, this defect of the integers vanishes when we work modulo a prime number \( p \). In this setting, most integers do have multiplicative inverses! For example, if we are working modulo 11, then the multiplicative inverse of 5 is 9, because:

\[
5 \cdot 9 \equiv 1 \pmod{11}
\]

The only exceptions are multiples of the modulus \( p \), which lack inverses in much the same way as 0 lacks an inverse in the real numbers. The following corollary makes this observation precise.

**Corollary 8.** Let \( p \) be a prime. If \( k \) is not a multiple of \( p \), then there exists an integer \( k^{-1} \in \{1, 2, \ldots, p - 1\} \) such that:

\[
k \cdot k^{-1} \equiv 1 \pmod{p}
\]
Proof. Corollary 7 says that the expression \((m \cdot k \text{ rem } p)\) takes on all values in the set \(\{1, 2, \ldots, p-1\}\) as \(m\) ranges over all values in the same set. Thus, in particular, \((m \cdot k \text{ rem } p) = 1\) for some \(m\), which means \(m \cdot k \equiv 1 \pmod{p}\). Let \(k^{-1} = m\).

The existence of multiplicative inverses has far-ranging consequences. Many theorems that hold for the real numbers (from linear algebra, say) have direct analogues that hold for the integers modulo a prime.

Multiplicative inverses also have practical significance in the context of Turing’s code. Since we encode by multiplying the message \(m\) by the secret key \(k\), we can decode by multiplying by the encoded message \(m'\) by the inverse \(k^{-1}\). Let’s justify this formally:

\[
m' \cdot k^{-1} \text{ rem } p \equiv m' \cdot k^{-1} \pmod{p} \quad \text{by part 2 of Lemma 5}
\]
\[
\equiv (mk \text{ rem } p) \cdot k^{-1} \pmod{p} \quad \text{by definition of } m'
\]
\[
\equiv mkk^{-1} \pmod{p} \quad \text{by parts 1 and 2 of Lemma 5}
\]
\[
\equiv m \pmod{p} \quad \text{by definition of } k^{-1}
\]

Therefore, if the receiver can compute the multiplicative inverse of the secret key \(k\) modulo \(p\), then he can decrypt with a single multiplication rather than an exhaustive search! The only remaining problem is finding the multiplicative inverse \(k^{-1}\) in the first place. Fermat’s Theorem provides a way.

### 3.6 Fermat’s Theorem

We can now prove a classic result known as Fermat’s Theorem, which is much easier than his famous Last Theorem— and also more useful.

**Theorem 9 (Fermat’s Theorem).** Suppose \(p\) is a prime and \(k\) is not a multiple of \(p\). Then:

\[k^{p-1} \equiv 1 \pmod{p}\]

**Proof.**

\[
1 \cdot 2 \cdot 3 \cdots (p-1) \equiv (k \text{ rem } p) \cdot (2k \text{ rem } p) \cdot (3k \text{ rem } p) \cdots ((p-1)k \text{ rem } p) \pmod{p}
\]
\[
\equiv k \cdot 2k \cdot 3k \cdots (p-1)k \pmod{p}
\]
\[
\equiv (p-1)! \cdot k^{p-1} \pmod{p}
\]

The expressions on the first line are actually equal, by Corollary 7, so they are certainly congruent modulo \(p\). The second step uses part (3) of Lemma 5. In the third step, we rearrange terms in the product.

Now \((p-1)!\) can not be a multiple of \(p\) by Theorem 2, since \(p\) is a prime and does not divide any of \(1, 2, \ldots, (p-1)\). Therefore, we can cancel \((p-1)!\) from the first expression and the last by Lemma 6, which proves the claim. □
3.7 Finding Inverses with Fermat’s Theorem

Fermat’s Theorem suggests an efficient procedure for finding the multiplicative inverse of a number modulo a large prime, which is just what we need for fast decryption in Turing’s code. Suppose that $p$ is a prime and $k$ is not a multiple of $p$. Then, by Fermat’s Theorem, we know that:

$$k^{p-2} \cdot k \equiv 1 \pmod{p}$$

Therefore, $k^{p-2}$ must be a multiplicative inverse of $k$. For example, suppose that we want the multiplicative inverse of 6 modulo 17. Then we need to compute $6^{15} \pmod{17}$, which we can do by successive squaring. All the congruences below hold modulo 17.

\[
\begin{align*}
6^2 &\equiv 36 \equiv 2 \\
6^4 &\equiv (6^2)^2 \equiv 2^2 \equiv 4 \\
6^8 &\equiv (6^4)^2 \equiv 4^2 \equiv 16 \\
6^{15} &\equiv 6^8 \cdot 6^4 \cdot 6^2 \cdot 6 \equiv 16 \cdot 4 \cdot 2 \cdot 6 \equiv 3 \\
\end{align*}
\]

Therefore, $6^{15} \pmod{17} = 3$. Sure enough, 3 is the multiplicative inverse of 6 modulo 17, since:

$$3 \cdot 6 \equiv 1 \pmod{17}$$

In general, if we were working modulo a prime $p$, finding a multiplicative inverse by trying every value between 1 and $p - 1$ would require about $p$ operations. However, the approach above requires only about $\log p$ operations, which is far better when $p$ is large. (We’ll see another way to find inverses later.)

3.8 Breaking Turing’s Code— Again

German weather reports were not encrypted with the highly-secure Enigma system. After all, so what if the Allies learned that there was rain off the south coast of Iceland? But, amazingly, this practice provided the British with a critical edge in the Atlantic naval battle during 1941.

The problem was that some of those weather reports had originally been transmitted from U-boats out in the Atlantic. Thus, the British obtained both unencrypted reports and the same reports encrypted with Enigma. By comparing the two, the British were able to determine which key the Germans were using that day and could read all other Enigma-encoded traffic. Today, this would be called a known-plaintext attack.

Let’s see how a known-plaintext attack would work against Turing’s code. Suppose that the Nazis know both $m$ and $m'$ where:

$$m' \equiv mk \pmod{p}$$
Now they can compute:

\[ m^{p-2} \cdot m' \equiv m^{p-2} \cdot mk \pmod{p} \]
\[ \equiv m^{p-1} \cdot k \pmod{p} \]
\[ \equiv k \pmod{p} \]

The last step relies on Fermat’s Theorem. Now the Nazis have the secret key \( k \) and can decrypt any message!

This is a huge vulnerability, so Turing’s code has no practical value. Fortunately, Turing got better at cryptography after devising this code; his subsequent cracking of Enigma surely saved thousands of lives, if not the whole of Britain.

A few years after the war, Turing’s home was robbed. Detectives soon determined that a former homosexual lover of Turing’s had conspired in the robbery. So they arrested him; that is, they arrested Alan Turing. Because, at that time, homosexuality was a crime in Britain, punishable by up to two years in prison. Turing was sentenced to a humiliating hormonal “treatment” for his homosexuality: he was given estrogen injections. He began to develop breasts.

Three years later, Alan Turing, the founder of computer science, was dead. His mother explained what happened in a biography of her own son. Despite her repeated warnings, Turing carried out chemistry experiments in his own home. Apparently, her worst fear was realized: by working with potassium cyanide while eating an apple, he poisoned himself.

However, Turing remained a puzzle to the very end. His mother was a devoutly religious woman who considered suicide a sin. And, other biographers have pointed out, Turing had previously discussed committing suicide by eating a poisoned apple. Evidently, Alan Turing, who founded computer science and saved his country, took his own life in the end, and in just such a way that his mother could believe it was an accident.