Cryptography

ORMC - Intermediate 2

Richemond Shin[1]

Oct 26, 2025

1 Caesar cipher

It is 50 BC. Julius Caesar is leading the armies of Rome to deal a finishing blow to the Gauls in the North. But to do so, he must contact his generals and send off his messengers to find them. Yet the barbarians have placed messenger spies among his ranks.

So to obfuscate his messages, he shifted all the letters in his messages by 3, turning "A" into "D", "B" into "E", "Z" into "C", and so on. He told only his generals the trick so only they were capable of translating it back. Some of you may already know this as a **Caesar Cipher**, though Caesar himself probably used something a little more complicated.

But before diving in, let's rattle off some important vocabulary:

- **Plaintext**: The original, readable message that is being encrypted.
- **Ciphertext**: The encrypted message, where each letter has been transformed according to some rule (e.g., shifting).
- **Key**: The rule used to encode (and decode) the message, often a number (like the shift in the Caesar cipher).
- **Encryption**: The process of converting plaintext into ciphertext.
- **Decryption**: The reverse process, turning ciphertext back into plaintext.
- **Substitution Cipher**: A cipher in which each letter of the plaintext is replaced by another letter.

Problem 1. Translate the following messages into ciphertext:

(a) "Hello"

(b) "Zoe shook my hand yesterday"

Problem 2. Translate the following messages into plaintext:

(a) "Wklv lv d yhub vhfuhw phvvdjh"

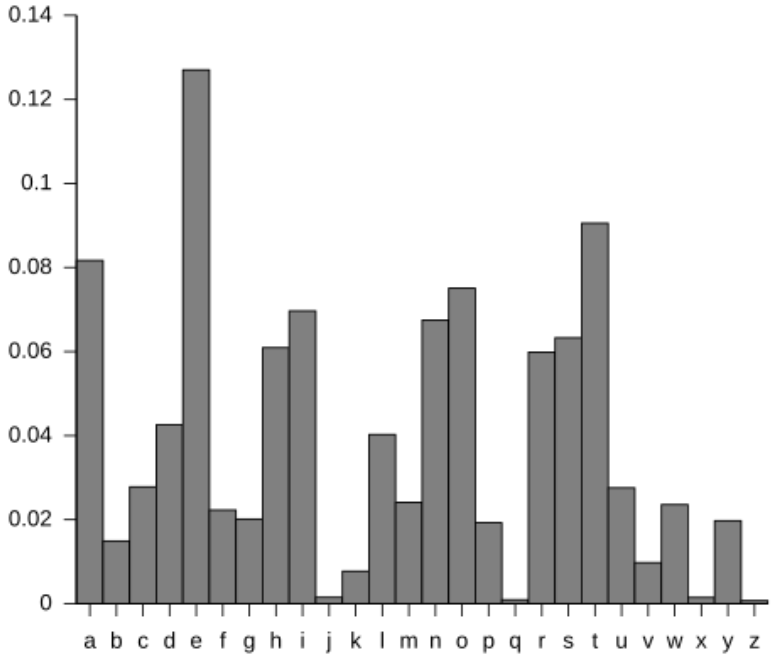
(b) "L vqruh orxgob"

Problem 3. You came across an encrypted message that you knew was Caesar cipher, though you don't know by how much each letter was shifted. How long would it take to decrypt in the worst case scenario"? Is the Caesar cipher even that secure?

[1]And a LOT of help from Wikipedia

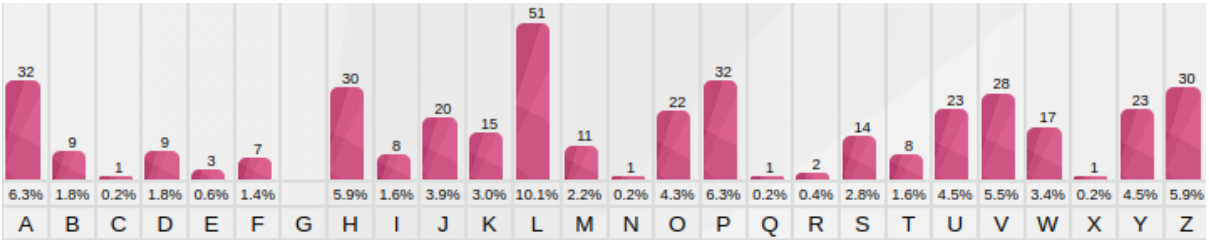
Problem 4. Consider that "E" is the most common letter in the alphabet. How can that help you in decrypting a Caesar cipher with an unknown shift?

In the English language, letters are not equally distributed. Some letters, like E or A appear often, but other likes J, Q, X, or Z barely ever appear.



Problem 5. Use frequency analysis to determine the shift of the following ciphertext. Its letter distribution is also below for reference:

Pu jyfwavnyhwof, h Jhlzhy jpwoly, hszv ruvdu hz Jhlzhy'z jpwoly, aol zopma jpwoly, Jhlzhy'z jvkl, vy Jhlzhy zopma, pz vul vm aol zptwslza huk tvza dpklsf ruvdu lujyfwapvu aljoupxblz. Pa pz h afwl vm zbizapabapvu jpwoly pu dopjo lhjo slaaly pu aol wshpualea pz ylwshjlk if h slaaly zvtl mpelk ubtily vm wvzpapvuz kvdu aol hswohila. Mvy lehtwsl, dpao h slma zopma vm aoyll, K dvbsk il ylwshjlk if H, L dvbsk iljvtl I, huk zv vu. Aol tlaovk pz uhtlk hmaly Qbspbz Jhlzhy, dov bzlk pa pu opz wypchal jvyylzwvuklujl.



Problem 6. CHALLENGE: The following is a substitution cipher. Unlike a Caesar cipher where each letter is shifted a fixed amount, a letter can just be mapped to any other particular letter. *Hint: I is the most common single letter, XL is the most common two-letter chain, and XLI is the most common three-letter chain in the ciphertext. What could these be in the English language?*

LIVITCSWPIYVEWHEVSRIQMXLEYVEOIEWHRXEXIPFEMVEWHKVSTYLXZIX
LIKIIXPIJVSZEYPERRGERIMWQLMGLMXQERIWGPSRIHMXQEREKIETXMJTP
RGEVEKEITREWHEXXLEXXMZITWAWSQWXSWEXTVEPMRXRSJGSTVRIEYV
IEXCVMUIMWERGMIWXMJMGCSMWXSJOMIQXLIVIQIVIXQSVSTWHKPEGAR
CSXRWIEVSWIIBXVIZMXFSJXLIKEGAEWHEPSWYSWIWIEVXLISXLIVXLIRGE
PIRQIVIIBGIIHMWYPFLEVHEWHYPSRRFQMXLEPPXLIECCIEVEWGISJKTVW
MRLIHYSPHXLIQIMYLXSJXLIMWRIGXQEROIVFVIZEVAEKPIEWHXEAMWYEP
PXLMWYRMWXSGSWRMHIVEXMSWMGSTPHLEVHPFKPEZINTCMXIVJSVLMR
SCMWMSWVIRCIGXMWYMX

2 Vigenère Cipher

As we've established, a generic Caesar cipher is a poor method of encryption, even if the shift is unknown. It's susceptible to frequency analysis and is simply too easy to brute force. So why not make it harder to do so?

With a **Vigenère cipher**, we solve this issue. Rather than shifting each letter by a fixed amount, we instead vary the shifting by using a *keyword*.

For instance, if the plaintext is "ATTACK" and the keyword is "KEY":

- A is shifted by 10 (since K is the 10th letter of the alphabet, starting from 0)
- T is shifted by 4 (since E is 4th letter of the alphabet)
- T is shifted by 24 (since Y is the 24th letter of the alphabet)
- A is shifted by 10 (since the keyword is shorter than the plaintext, we restart the keyword)
- ...

As a result, we get our ciphertext of "KXYEKF". The advantage of this is that now you and your recipient only need to be aware of the keyword, while the ciphertext itself is significantly harder to decrypt.

Problem 7. Encrypt "Hello" if you know the keyword is "Key".

Problem 8. Decrypt "Rijvs" if you know the keyword is "Key".

Problem 9. Assume a Vigenère cipher over a l-letter alphabet. You intercept a ciphertext of length n, and the attacker knows the length of the key as k. Using a brute-force strategy (pick a key, decrypt, and check if the result is readable), how many distinct keys must an attacker try in the worst case to guarantee finding the correct key?

Problem 10. Consider problem 9 where the attacker does not know the length of the key. How many distinct keys must an attacker try in the worst case to guarantee finding the correct key?

By now it should be easy to tell that this method is both computationally intensive and nearly invulnerable to frequency analysis. If you have a long ciphertext with an equivalently lengthed keyword, it'd be nearly impossible to crack.

However, this method originated in the 16th century. So for practical reasons it was unreasonable to have a keyword be as long as the message itself because it's cumbersome to write it all out and was also prone to typos. As a result, the length of keyword is often significantly shorter than the message itself. But this shortcut leaves it open to attacks.

Problem 11. Consider the following example of a Vigenère ciphertext with a repeated keyword:

Plaintext	`CRYPTOISSHORTFORCRYPTOGRAPHY`
Keyword	`ABCDABCDABCDABCDABCDABCDABCD`
Ciphertext	`CSASTPKVSIQUTGQUCSASTPIUAQJB`

Do you see any repeated characters in the ciphertext? What can you determine about the length of the keyword?

In a Vigenère cipher, if a repeated sequence of letters appears in the ciphertext, the distance (in characters) between the repetitions must be divisible by the length of the keyword. This result provides a practical way to estimate the possible length of the keyword.

Problem 12. Prove that the distance between repeated ciphertext sequences in a Vigenère cipher must be divisible by the length of the keyword.

Problem 13. What are the possible keyword lengths of the following ciphertext:
VHVSSPQUCEMRVBVBBBVHVSURQGIBDUGRNICJQUCERVUAXSSR

Given a long enough plaintext and a short enough keyword, there is bound to be repeated characters in the ciphertext. And through this fault, we can reverse engineer the keyword length in question.

Problem 14. Why is knowing the keyword length so important, and how can you combine it with frequency analysis like in Problem 5?

3 Modular Arithmetic

If you've been paying attention, you may have noticed that we haven't done a ton of math yet. What a shame! Let's fix that right now.

In the previous sections, we explored methods of encryption that manipulate letters through shifting or substitution. But as ciphers became more complex, so too did their mathematical foundations.

As such, it's only necessary that we learn about **modular arithmetic**, a system of arithmetic for integers, where numbers "wrap around" upon reaching a certain value, the **modulus**. Put another way, try thinking about it as an operation that returns the remainder of division. To illustrate, suppose we are working *modulo 12* (that is, with modulus 12). Then:

$$27 \equiv 15 \equiv 3 \pmod{12}$$

Problem 15. Compute the following modulo operations:

(a) $14 + 9 \pmod{12}$
(b) $7 \times 5 \pmod{12}$
(c) $20 - 15 \pmod{7}$

Problem 16. If $a \equiv b \pmod{n}$, show that $(a+c) \equiv (b+c) \pmod{n}$ and $(a \times c) \equiv (b \times c) \pmod{n}$.

The **modular inverse** is the number which, when multiplied by a given number, yields 1 under modulo n:

$$a \times a^{-1} \equiv 1 \pmod{n}$$

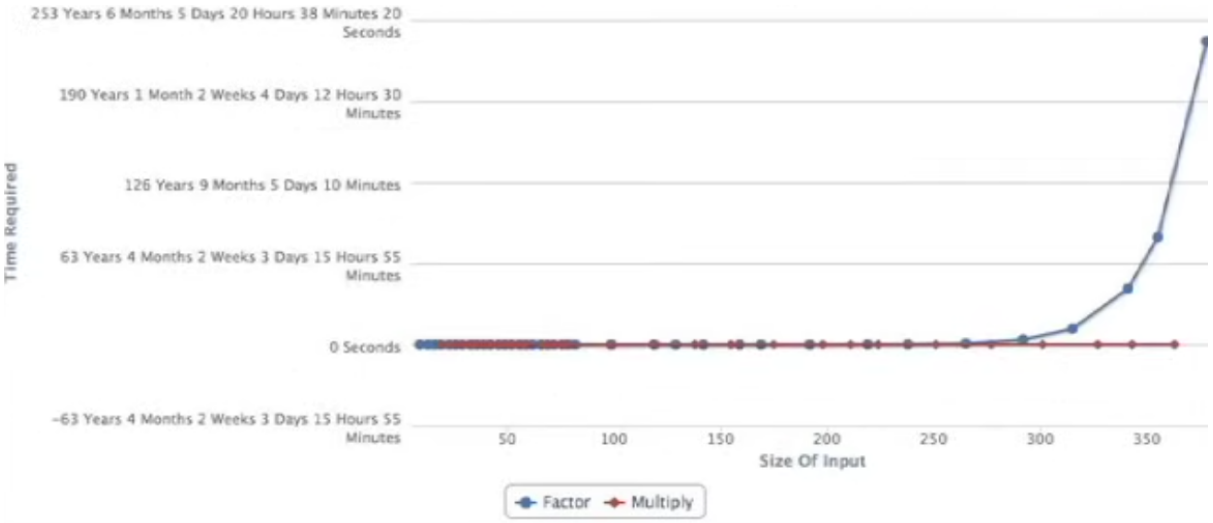
For example, under modulo 7, the modular inverse of 3 is 5, since $3 \times 5 = 15 \equiv 1 \pmod{7}$. Modular inverses only exist if a and n are **coprime** (they share no common divisors other than 1).

Problem 17. Find the modular inverse of 4 modulo 11.

4 Primes, Factorization, and Euler's Totient

A **prime number** is an integer greater than 1 whose only positive divisors are 1 and itself. Primes are the "atoms" of the integer world: every integer greater than 1 can be uniquely factored into primes. This property is called the **Fundamental Theorem of Arithmetic**.

An interesting fact about factorization is that current algorithms to reduce large numbers to their prime factors are incredibly inefficient. Multiplication, which could be thought as the reverse of factorization, is on the orders of magnitude faster than factorization. To build intuition, play with some numbers! Or you can just look at the graph...



With that aside, we introduce **Euler's totient function**, denoted $\phi(n)$, which counts the number of integers between 1 and n that are coprime with n. Formally:

$$\phi(n) = |\{k \in \mathbb{Z} \mid 1 \leq k \leq n, \gcd(k, n) = 1\}|$$

Problem 18. Compute $\phi(6)$ and $\phi(8)$.

Problem 19. For any prime number p, what is $\phi(p)$?

Problem 20. Show that if $n = p \cdot q$ for distinct primes p and q, then

$$\phi(n) = (p-1)(q-1)$$

Problem 21. Compute $\phi(21)$.

More generally, if n has the prime factorization

$$n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k},$$

then Euler's totient function can be computed as:

$$\phi(n) = n\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\cdots\left(1 - \frac{1}{p_k}\right).$$

This formula follows from the fact that each prime factor p_i eliminates $\frac{1}{p_i}$ of the integers up to n from being coprime with n.

Problem 22. CHALLENGE: Prove the theorem for Euler's totient function. *First show the formula for a prime power p^r. Then use the fact that ϕ is multiplicative on coprime arguments ($\phi(ab) = \phi(a)\phi(b)$ when $\gcd(a, b) = 1$).*

Euler's totient function interacts interestingly with modular exponentiation. Specifically, Euler's theorem states that if $\gcd(a, n) = 1$, then

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

Problem 23. CHALLENGE: Prove the above statement.

5 Public Key Encryption and the RSA Algorithm

Until now, the ciphers we learned about have all been *symmetric*: both sender and receiver use the same secret key. In modern terms, this approach is called **symmetric key encryption**. It remains the backbone of most real-world data encryption, particularly where speed and efficiency are essential. But it carries a major flaw: how can two parties securely exchange a key if they have never met?

Consider a scenario with Alice and Bob. Alice knows that Bob wants to send her a message. So Alice sends Bob an open lock and a key which is only capable of locking (it's incapable of unlocking). Alice meanwhile keeps an unlocking-only key for herself. With the lock and locking-only key in hand, Bob secures the message and sends it to Alice. Once Alice receives the message, she can unlock and view it with the key she retained.

When Alice and Bob send the locks and messages to each other, there's a third party, Charlie, who can see what gets sent. Every day and night, he is wishing for Alice's and Bob's downfall. Since he sees what gets sent, he knows of the lock, locking-only key, and sealed message. How is it possible that Charlie cannot fabricate the unlocking key from the locking key? How is it even possible to split up the key to only be capable of locking and unlocking? Is it really so hard to reverse engineer the unlocking key?

As it turns out, reversing the key is exceptionally difficult for the RSA algorithm we are about to explore. And this difficulty is precisely what makes it so secure. The core idea is simple: multiplying two large primes to form a composite number is computationally easy, but factoring that number back into its prime components is extraordinarily hard. Let's explore how it works.

How RSA Works

Let's first take a look at how Bob encrypts his message with the lock and **encryption key** e (the locking-only key). Say that Bob condenses his message to an integer, m. He gets his encrypted message c by doing the following:

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

Right now, don't worry about the exact derivations of n and e. Just note that they are the lock and encryption key that Alice sends. We call (n, e) the **public key** since Charlie is capable of seeing it when Alice sends it to Bob to encrypt his message.

Problem 24. So Bob sends his encrypted message c to Alice. How can Alice decrypt the message? *Hint: consider inverses*

For every encryption key e that Alice creates, she must also generate a corresponding decryption key d. However, the encryption key e is public, so Charlie can see it. The challenge is for Alice to compute d efficiently while making it computationally infeasible for anyone else to do so. The pair (n, d) forms the **private key**, known only to Alice.

Problem 25. Earlier, we discussed a process that is fast in one direction but extremely difficult to reverse. Identify this process and explain how Alice can rely on it to keep d secure while publishing e.

Alice chooses two large prime numbers on the magnitude of hundreds of digits. Call them p and q. She multiplies them together to get $n = pq$. This number n is what's used in the public and private keys.

Since n is a part of the public key, Charlie knows what it is. But he doesn't know its components, p and q. And recall that finding the factorization of large numbers is really, really hard. So for all intents and purposes, Charlie, will never know what p and q are.

But as Alice knows the prime factorization of n is p and q, she wants to somehow implement factorization into her encryption/decryption keys to strengthen them.

Problem 26. Connect the ϕ function with modular exponentiation to derive a formula for d.

Problem 27. Show that if Alice knows the prime factors p and q of $n = pq$, then she can compute $\phi(n)$ and from it compute the private exponent d given the public exponent e. *Hint: consider inverses!*

Problem 28. Explain why obtaining the private key (n, d) from the public key (n, e) and a ciphertext c is essentially equivalent to factoring n into its prime components. Why does this equivalence form the basis of RSA's security?

Recap

So to summarize:

1. Choose two large prime numbers. Multiply them together to get $n = pq$.
2. Compute Euler's totient $\phi(n) = (p-1)(q-1)$.
3. Choose a public exponent e such that it's coprime to $\phi(n)$.
4. Compute the private exponent d as the modular inverse of e modulo $\phi(n)$:

$$e \times d \equiv 1 \pmod{\phi(n)}.$$

To encrypt a message m, convert it into a number smaller than n and compute:

$$c \equiv m^e \pmod{n}.$$

To decrypt, compute:

$$m \equiv c^d \pmod{n}.$$

Problem 29. Say you are Alice, and Bob wants to send you a message. What do you send to him? Suppose $p = 23$ and $q = 41$. And for convenience, we set the encryption key to be $e = 7$.

Problem 30. Bob sent back the message $c = 545$. What's his original message?

In practice, p and q are not small primes like 23 or 41. They are hundreds or even thousands of bits long. Modern RSA commonly uses 2048-bit or 4096-bit keys, making the modulus n so large that factoring it into its prime components is computationally infeasible. But let's try a simple example.

Problem 31. You are Charlie. You saw that Alice sent Bob the public key (55, 3), and Bob sent back his encrypted message, $c = 14$. Is it possible for you to decrypt Bob's original message. How hard is it and what is the original message?

Factoring is really hard. Classical factoring records have reached only into the low hundreds of bits. The largest general-purpose semiprimes factored publicly are on the order of 800-900 bits, with modern factoring projects required hundreds to thousands of CPU-core years.

By contrast, choosing primes of several thousand bits each (for example, primes of roughly 2 000 bits) produces an n many orders of magnitude larger than current records and is infeasible to factor with today's classical hardware and algorithms. This is why practical RSA keys use large prime factors (2048 bits total modulus or larger, and many deployments move to 3072–4096-bit moduli) to stay well beyond known factoring capability. Estimates and algorithmic advances can shift safety margins, so recommended sizes are periodically updated.

Organizations have sponsored cooperative-computing prizes to encourage large-computation projects (including prime searches). For instance the EFF's Cooperative Computing Awards offer such incentives.