Jędrzej Wydra

How Safe Are Your Spicy Messages? A Simple Guide to Encryption

The other day, while texting a nice girl and asking “What are you wearing?”, I started wondering — what actually makes our conversation private? How is it that no one else can read our messages (thank goodness)? After all, popular messengers like WhatsApp or Facebook Messenger work over the Internet, which means our messages travel across a global network. So how is it that only the sender and recipient can access them? The answer is simple: encryption keeps things private. But how exactly does it work?

The first thing that comes to mind when I think of encryption is simply shifting letters around. Let’s say that instead of writing “A” I’ll write “B,” instead of “B” I’ll write “C,” and so on, with “Z” wrapping back around to “A.” In that case, the word “otter” becomes “puufs.” Looks pretty good — but is it actually secure? Notice that if I wanted to send such a message to someone, I’d also have to let them know how to read it. Which means I’d need to send them the key in a private and secure way. But then… if I can already send something securely, why not just send the message itself instead of bothering with encryption? Clearly, we need a different solution.

That’s where asymmetric encryption comes in — a technique that allows the sender and the recipient to agree on a shared encryption key, without ever revealing it directly. One well-known method is the Diffie-Hellman protocol. It’s based on generating what's called a pair of public and private keys, which together allow both parties to encrypt and decrypt messages. The key idea is that neither side has all the information on their own — and that’s what makes it secure.

Let’s imagine that Person A and Person B agree on a shared prime number — say, 23. Then, each secretly picks a private number: A chooses 4, and B chooses 5. Next, A multiplies 23 * 4 = 92, and B does 23 * 5 = 115. These numbers will serve as their public keys, which they exchange openly.

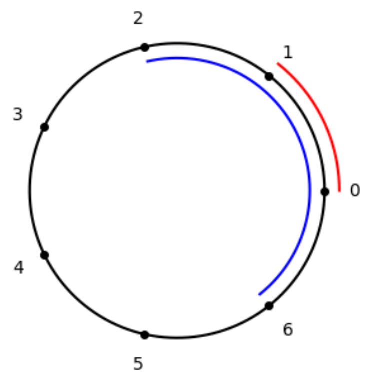
At this point, Person A knows two things: (1) B’s public key is 115, and (2) A’s own private key is 4. Likewise, Person B knows: (1) A’s public key is 92, and (2) B’s private key is 5. Both people keep their private keys secret. With just this information, both can arrive at the same encryption key: 115 * 4 = 460, which is exactly the same as 92 * 5. Now, if they agree (and they can do so publicly) that letters will be converted into numbers based on their alphabetical order — A becomes 1, B is 2, C is 3, and so on — they can start encrypting messages without needing a secure channel. Here’s how it works:

The word “otter” is converted into the number sequence “15-20-20-5-18”. To encrypt it, Person B adds 460 to each number, resulting in “475-480-480-465-478”, and sends this sequence to A. When A receives the encrypted message, they simply subtract 460 from each number to recover the original code: “15-20-20-5-18”, which clearly spells “otter”. It seems like a clever system. The numbers 92 (A’s public key) and 115 (B’s public key) are publicly known, but the private keys required to compute the shared encryption key (which is 460) are kept secret — known only to the people involved.

There’s just one small problem. A and B publicly agreed on the number 23 at the start. If someone divides the public keys by this number, they can figure out the secret private keys. That means the encryption still isn’t secure. But what if the math involved wasn’t so simple? As it turns out, it doesn’t have to be.

One of the fundamental concepts in abstract algebra is a group. A group is a structure made up of a set and an operation defined on that set. This operation must be associative — that is, when combining three elements, you can move the parentheses without changing the result: (a + b) + c = a + (b + c). There must also be an identity element (an element that doesn’t change others when combined — for example, zero for addition or one for multiplication), and every element must have an inverse (something that brings you back to the identity — like 2 + (–2) = 0, or 2 * ½ = 1). The real numbers with addition form a group. So do the real numbers with multiplication. But in those groups, the operations are relatively simple.

Geometrically speaking, real numbers form an infinite line — but it turns out that operations on a circular structure are much more complicated.



Let’s take a circle and place a finite number of points on it — say, seven points, as shown in the image above. Now, let’s define addition as moving from one point to another in a counterclockwise direction by a certain number of steps. So, for example, 6 + 3 = 2, because starting at 6 and moving three steps counterclockwise brings us to 2. In this structure, we can also define multiplication as repeated addition. For instance, 2 * 4 means 2 * 2 * 2 * 2 = 2 + 6 = 1. This is what we call modular multiplication, because the result is the remainder after dividing the usual product by the number of points on the circle. For example, the usual product 2 * 4 = 8; divide 8 by 7, and you get a remainder of 1 — that’s the result on the circle. Similarly, 5 * 4 = 20; divide 20 by 7, and you get a remainder of 6, so on the circle, 5 * 4 = 6. In general, to make it clear we’re using modular arithmetic, we write 5 * 4 = 6 (mod 7). But for simplicity, we'll skip the "mod" notation from now on.

Just a heads-up: zero must be excluded in order for multiplication on the circle to form a group.

Now, if A and B run the Diffie-Hellman protocol on a circular structure like this, they end up with a more secure form of encryption. Let's say they're working with a six-point circle. They choose a number from 1 to 6. But they shouldn't just pick one at random — they need to choose what's called a generator of the group. That's a number which, when raised to successive powers, produces all the elements of the group. In other words:

- $1^1 = 1$, $1^2 = 1$, $1^3 = 1$, $1^4 = 1$, $1^5 = 1$, $1^6 = 1$, $1^7 = 1$;

- $2^1 = 2$, $2^2 = 4$, $2^3 = 1$, $2^4 = 2$, $2^5 = 4$, $2^6 = 1$, $2^7 = 2$;

- $3^1 = 3$, $3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 5$, $3^6 = 1$, $3^7 = 3$;

- $4^1 = 4$, $4^2 = 2$, $4^3 = 1$, $4^4 = 4$, $4^5 = 2$, $4^6 = 1$, $4^7 = 4$;

- $5^1 = 5$, $5^2 = 4$, $5^3 = 6$, $5^4 = 2$, $5^5 = 3$, $1^6 = 1$, $5^7 = 5$;

- $6^1 = 6$, $6^2 = 1$, $6^3 = 6$, $6^4 = 1$, $6^5 = 6$, $6^6 = 1$, $6^7 = 6$.

It's easy to see that 3 and 5 are generators. Let's say A and B choose 3. Each of them now picks a private key: A chooses 2, and B chooses 3. Then they compute their public keys by raising the chosen generator to their private exponents. So A's public key is $3^2 = 2$, and B's public key is $3^3 = 6$. To find the shared encryption key, each person secretly raises the other's public key to the power of their own private key. A computes $6^2 = 1$, and B computes $2^3 = 1$. Just like with real numbers and addition, they end up with the same encryption key — even though neither side knows everything.

But why is this more secure than using real numbers with addition? Well, the following information is publicly known: A and B are operating in a six-element group under multiplication — that is, they're doing arithmetic modulo 7. It's also known that they agreed on the number 3 as their generator; A's public key is 2; and B's public key is 6.

The security of a finite group lies in the fact that, unlike with real numbers, there's no simple way to compute the private key. In other words, there's no efficient way to solve the equation, where the generator raised to the power of one person's private key equals the other person's public key, which. In our example, takes the form:

$$3^a = 2$$

or

$$3^b = 6$$

where a and b are the private keys of Person A and Person B, respectively.

With real numbers and multiplication, you can just take the logarithm of both sides to solve the equation. But in a finite group, that's not possible. The only way to solve it is by trying every possible value. In a small group like the one in our example, that's easy. But if we were using a group with, say, 10,006 elements (in that case we say the group has order 10,006), solving the equation becomes practically impossible without a computer. And in a group of order 100,000,000,000,061, it could take about 1,000 years to find the solution — even with a computer. That number, by the way, is a 47-bit number (you'd need 47 bits to represent it in binary). In real-world encryption, we use numbers that are 2048 bits long. So trying to compute the private key based on public information is basically hopeless.

At this point, it also becomes clear why A and B should have chosen a generator of the group rather than just any number. If their chosen number isn't a generator, then the equation used to compute the private key might have more than one solution. That means there could be additional keys that would also decrypt the message — which lowers the level of security.

Is this kind of encryption perfectly secure? Not quite. Sure, calculating the private key is practically impossible — but someone could still analyze the encrypted message and look for patterns in the frequency of certain numbers. For example, if I know that the most common letter in English is "e", I might be able to partially decrypt a long enough message. So if the number 3522 shows up very frequently in a captured message, there's a good chance it represents the letter "e". I could then replace all instances of 3522 with "e" and start guessing the rest. By repeating this with the next most frequent number and mapping it to the next common letter, I could eventually reconstruct the message — even without knowing the private key.

But there's a fix for that too: probabilistic encryption. One such method is the ElGamal algorithm. Probabilistic encryption adds a random element to the process, so that even the same message (or the same character) gets encrypted differently each time.

Let's encrypt the word "otter" again. First, we need to generate the public and private keys, and then convert the word into a sequence of numbers. We'll use the same method as before.

Let's say the finite group has order 10,006, and the agreed-upon generator is 536. Person A chooses 345 as their private key, and Person B chooses 567. Then, A's public key — which contains all the publicly shared information — will be:

10,007-536-8,336.

The first number defines the modulo operation, the second is the group's generator, and the third is the actual public key.

Similarly, Person B’s public key — containing all the necessary information — is:

10,007-536-4,746.

The word “otter” is converted into the sequence 15-20-20-5-18.

Now, when Person B wants to encrypt a message for Person A, they choose a random number k before encrypting each character — this will be the temporary key. The process goes as follows:

1. Let’s say the randomly chosen number is k = 3. The generator is raised to the power of k times Person B’s private key: $(536^3)^{567}$ = 4,337. Then, Person A’s public key is raised to the same power: $(8{,}336^3)^{567}$ = 182. This resulting encryption key is multiplied by the number representing the character to be encrypted: 15 * 182 = 2,730. So the encrypted form of the first character is the pair: (4,337; 2,730).
2. The same process is repeated for the remaining characters. Let’s assume the following temporary keys were randomly selected: 3, 5, 2, 1. This results in the encrypted message:

(4,337; 2,730)-(4,337; 3,640)-(1,549; 5,043)-(8,766; 598)-(4,746; 3,073)

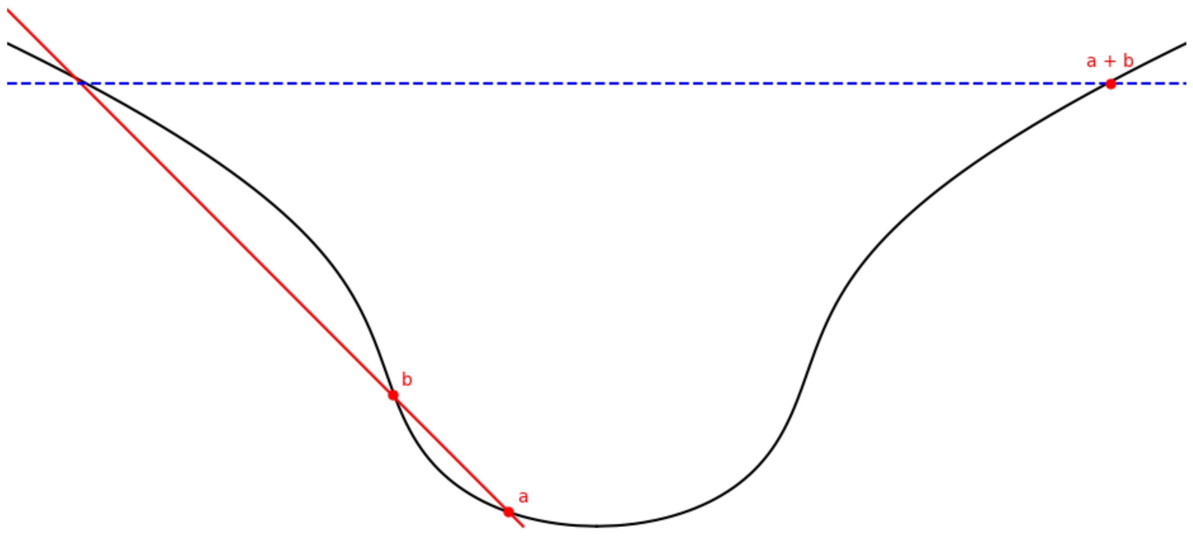
To decrypt the message, Person A takes the first number in each pair and raises it to the power of their private key. Then, they find the multiplicative inverse of that result. One way to do this is by using Fermat’s Little Theorem — raising the number to the power of p – 2, where p is the prime number used in the modulo operation (in this case, p = 10,007, so the exponent is 10,005). Then, Person A multiplies the second number in the pair by the inverse. In our case, that means multiplying the second number by the first one raised to the power of 345 * 10,005 (calculated as a real number for simplicity). This gives us back the original sequence: 15-20-20-5-18, which easily translates to the secret word “otter”.

It’s worth noting that in practice, the numbers used are much larger than 10,007, and letters are typically converted into numbers using standard methods — such as ASCII encoding.

The ElGamal algorithm also helps explain why the administrator of a service that requires a user-defined password can’t actually see that password. The user’s password is simply their private key.

The algorithm we've discussed offers a high level of security, but modern encryption methods go even further. They achieve even stronger protection by making the group operations more intricate. As we’ve seen, doing math on a circle is trickier than doing it on a straight line — and that added difficulty improves security. But algebra goes even further. It introduces structures that are harder to work with. Operations on so-called elliptic curves are more involved — and they offer even greater protection.

On an elliptic curve, the result of adding points a and b is the reflection of the third point where the line through a and b intersects the curve — just as shown in the image below.



At this point, it's worth commenting on the famous Riemann Hypothesis — a conjecture suggesting that prime numbers follow a certain logical pattern, making them easier to find. There's a popular rumor that proving the hypothesis would bring about the end of the digital world as we know it, because easily finding prime numbers would supposedly break all modern cryptographic systems. But I'd like to point out two problems with that fear.

First, being able to easily find prime numbers has no real impact on the ElGamal algorithm, because the prime number being used is already public. It's not the secrecy of that number that ensures security. So in that sense, at least, systems based on the ElGamal approach aren't threatened by the Riemann Hypothesis.

Second, the Riemann Hypothesis is still just that — a hypothesis. There's no proof that it's true, and no proof that it's false. So far, every attempt to test it has produced results that support its validity. If that weren't the case, we'd already have a counterexample — and that would be enough to disprove it. In that sense, if a hacker wanted to exploit the Riemann Hypothesis to break a cryptographic system, they could already try doing so by assuming it's true. And yet, as we can see, the world hasn't exactly collapsed. That suggests our current systems are, more or less, prepared for the possibility that the hypothesis might eventually be proven. Of course, such a proof would be a massive breakthrough — revolutionary, even. But I wouldn't expect it to bring about the end of the world as we know it.

So, can we safely send spicy messages? Absolutely — as long as we take good care of our private key, which, in most cases, simply means our password.