Recently, the Narin Güran case has been in the news, and with the Turkish government requesting deleted WhatsApp records from Meta, many questions have arisen about encryption, privacy, and who can access your messages. Since I haven't seen many Turkish resources on these topics, I decided to write an article.

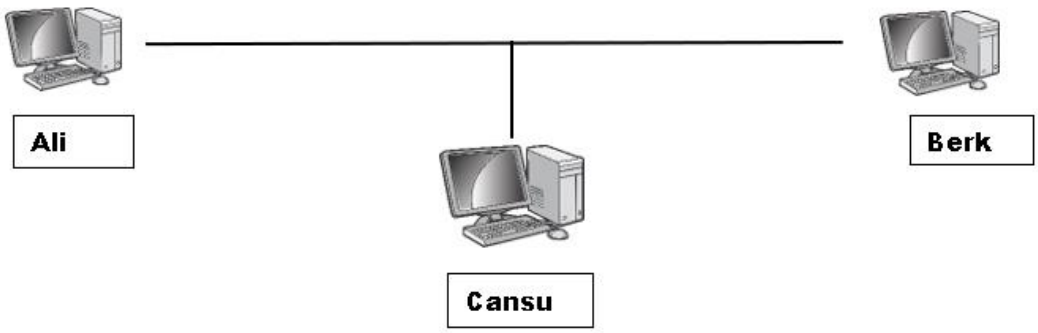
Today, I will explain how a government can wage a war against encryption standards for the purpose of mass surveillance. Even if some of you are aware of surveillance issues, you might not be fully aware of the technical mechanisms behind it. I hope that by the end of this article, you will have a clearer understanding of how these systems work. My goal is not to create paranoia about your privacy but to inform you about the conditions under which and by whom your privacy might be violated. Warning: This article will contain a lot of mathematics.

What is Encryption?

Encryption is the process of making readable data or messages unreadable for reasons of privacy and security. Our goal is that only those with the decryption key should be able to read this data. When a message is encrypted, an encryption algorithm is chosen and applied to the message with a selected encryption key. If the encrypted message is to be read, the decryption key and the reverse of the selected algorithm are applied, and the original message is obtained.

To make this concept clearer, let's use a simple example: Suppose Ali wants to encrypt the message "YARIN DOKUZDA ARKA KAPIYA GEL" and wants only his friend Berk to be able to read it, using the Caesar cipher algorithm. In Caesar cipher, each letter is shifted by n places in the alphabet. Ali whispers the value of n as 1 to Berk and gives him a paper with "ZBSIO EÖLÜAEB BSLB LBRİZB ĞFM" written on it. Berk can then read the original message by shifting each letter in the received message 1 step back using the value of n. In this scenario, our encryption and decryption key is 1.

You can guess that sending a message encrypted with this algorithm over the internet would cause significant problems, and decrypting the message would not take very long. If the message falls into the hands of someone eavesdropping on conversations, such as Cansu, they can try all 29 possible values for n (since the alphabet has 29 letters and we return to the beginning after Z) and manage to read the message. Another problem is the transmission of the decryption key. If Ali sends this message to Berk over the internet, he also needs to send the decryption key, which can be overheard by a third party on the same network or listening to the network traffic.



To ensure secure communication between two parties, we need to use more complex encryption methods. The RSA algorithm, discovered in 1978, solves both of these problems.

Before explaining this solution, let me briefly clarify what modulo (or simply mod) means: Modulo refers to the remainder of a division operation, so for example, in 7 / 3, the remainder is 1, and we can express this as 1 = 7 mod 3. As you might guess, since the remainder in a division operation can never be greater than the divisor, for any numbers a and n, the result of a mod n cannot be greater than n, so our solution set has specific limits: {0…n-1}.

When Ali wants to send a message to Berk, he first sends a request indicating his desire. Berk, to ensure secure communication, uses the RSA encryption method as follows: Berk chooses two very large prime (and secret) numbers, p and q. We can call their product n, so

$$p * q = n.$$

He then selects a public e value to be used as the encryption key (typically 65537, and it must be coprime with φ(n), but we'll discuss that shortly).

In this process, n and e are public keys and there's no issue with their values being known to everyone. When Ali sends the message, he converts it into a numerical format, m (for example, by writing the ASCII value of each character), and sends the result to Berk by performing the following operation:

$$c = m^e \bmod n.$$

For example: Let p = 61 and q = 53. In this case, p * q = n = 3233. Let's choose e as 17 and the numerical value of the message to be 65. The resulting equation is:

$$c = 65^{17} \bmod 3233$$

Using a calculator, we find that c=2790

The message's original (numerical) value is m, and c is the encrypted form. This is the value sent over the internet. Therefore, Cansu, who intercepts the transmitted data, only sees c. So, how can Berk, who receives c, read the message in its original form? To decrypt it, Berk needs a secret key d that only he possesses.

Berk calculates ϕ(n), where

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

The reason for calculating this peculiar value is that we use Euler's theorem to generate our secret key. According to Euler's theorem, if a and n are coprime (the likelihood of this not being the case is very low since n is the product of two large prime numbers), i.e., gcd(a,n)=1, then

$a^{\phi(n)} = 1 \bmod n$

With this premise, if we return to our equation, our secret key d must satisfy:

$1=e*d \bmod \phi(n)$

The number d that satisfies this equation is found using the extended Euclidean algorithm. Although I won't go into the details of how it's computed, the key point is that it is not a difficult value to compute. Now that we have calculated all the variables, we can use d to decrypt c:

$c^d \bmod n$

Substituting the initial equation:

$(m^e)^d \bmod n$

$= m^{e*d} \bmod n$

Given that

$e*d \bmod \phi(n) = 1$

(and remembering that modular operations refer to the remainder of the division), for some integer k we can express

$e*d = k*\phi(n) + 1$

In that case we can write:

$m^{k*\phi(n) + 1} \bmod n$

$= (m^{\phi(n)})^k * m \bmod n$

Finally, inserting Euler's theorem into our equation:

$(1)^k * m \bmod n$

$= m \bmod n$

And we have obtained the original message mmm.

Continuing with our earlier numerical example:

$\phi(n) = 60 * 52 = 3120$

$1 = d * 17 \mod 3120$

$d = 2753$ (using the extended euclidian algorithm)

$2790^{2753} \mod 3233$

$= 65$

This algorithm is based on the mathematical difficulty of finding sufficiently large values for the prime factors p and q of n, which is known as the factorization problem. Anyone who knows p and q can also compute the secret key d. An important question then becomes how p and q are selected. If these numbers are chosen according to a specific pattern and are not random, someone who discovers this pattern could predict future values of p and q, making RSA encryption ineffective.

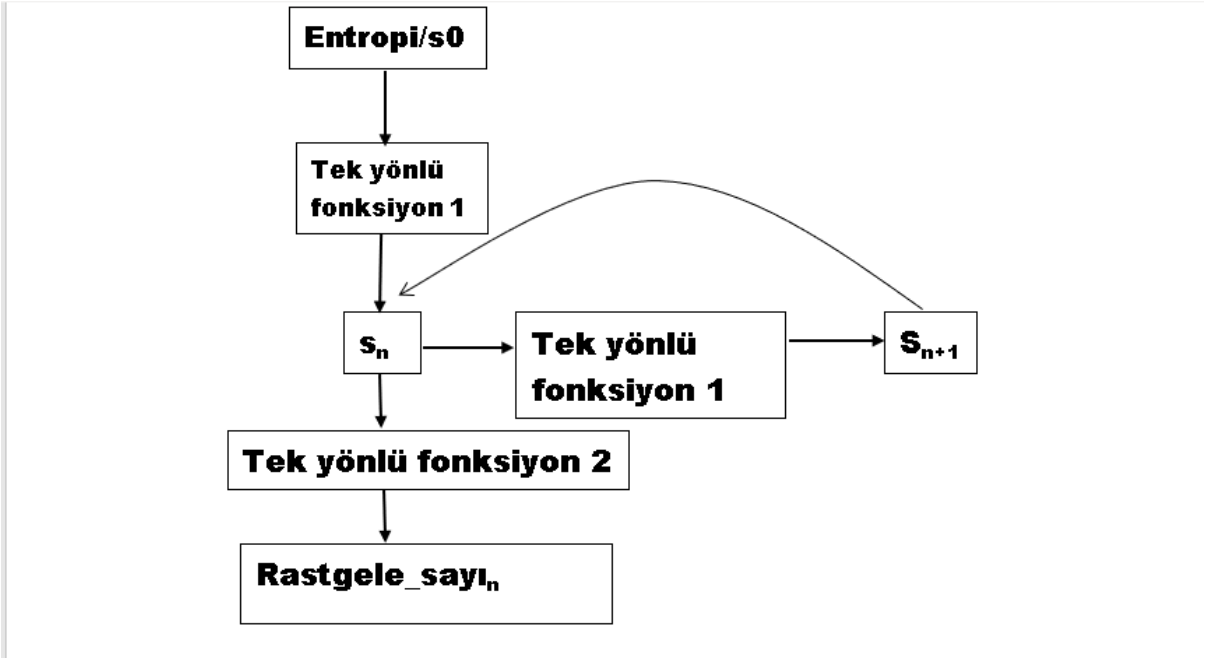
The programs needed to generate these numbers are called CSPRNGs, or Cryptographically Secure Pseudo-Random Number Generators.

Because software has deterministic properties (the same program always produces the same result with the same inputs), it is not possible for them to generate "truly" random numbers. They can only produce "pseudo-random" results that appear random.

CSPRNGs work as follows: First, an initial value s_0 is chosen for the variable s that represents the "internal state" of the program. This value is typically obtained from a high-entropy source, such as timings between mouse and keyboard clicks or hard disk activity. Keeping the internal states secret is crucial for the security of the encryption system.

The remainder of the process requires one-way functions. One-way functions are functions for which it is very easy to compute the output from the input, but very difficult to determine the input from the output (e.g., hash functions). These functions are accessible to everyone.

When we want to obtain a random number, s_0 is modified by a one-way function (let's call this function TYF1), producing a new "internal state" s_1. s_1 is then input into another one-way function (e.g., a hash function, which we will call TYF2) to obtain the desired random value. To get a new random number, we input s_1 into TYF1 to obtain s_2, and applying TYF2 to s_2 gives us a second random number.

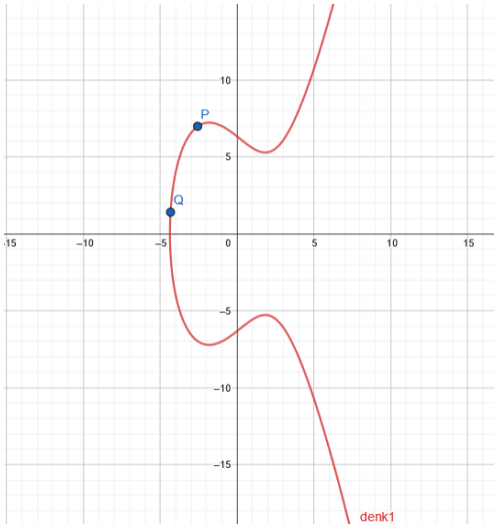


The fact that both functions are one-way is an important detail. If we can discover the internal state s_n that generates the rastgele_sayı$_n$ we can calculate future internal states as well. To prevent the calculation of past internal states, a one-way function must also be used when generating new internal states.

Now we come to the main issue. The CSPRNG used in RSA's cryptography library was Dual_EC_DRBG, or Dual Elliptic Curve Deterministic Random Bit Generator, and this is where the problem began.

Dual_EC_DRBG was one of the four standard CSPRNGs approved by NIST from 2006 until its withdrawal in 2014. NIST, or the National Institute of Standards and Technology, is a U.S. government agency that provides guidance and standards for information systems security.

To understand the problem with the standard they used, let's first understand why elliptic curves are used in cryptography.

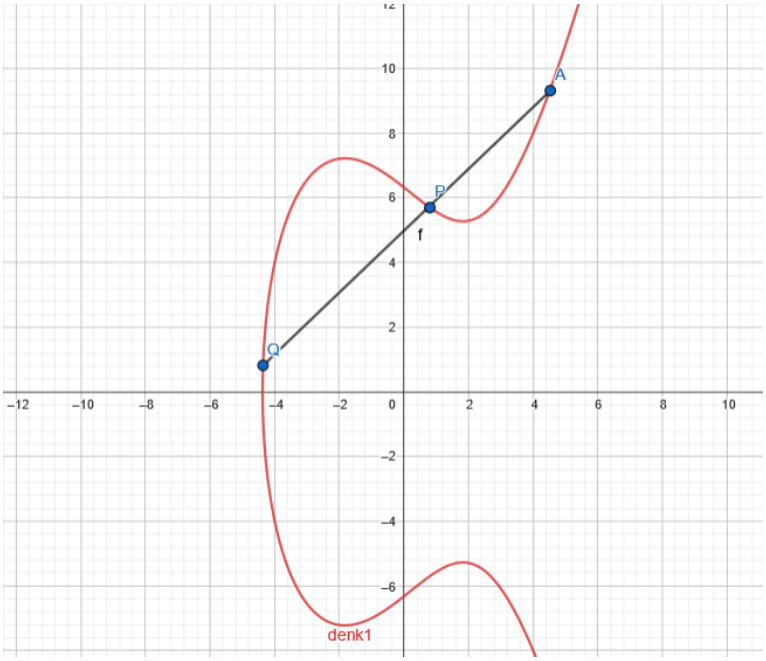


Elliptic curves are curves like the one you see above, and they have equations in the form $y^2 = x^3 + ax + b$. As you might guess, the points on the curve are those that satisfy this equation with their x and y coordinates.

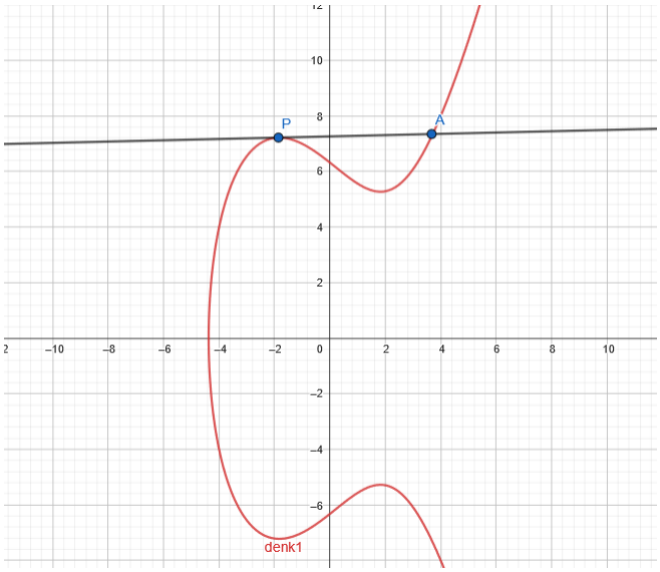
The property that makes elliptic curves useful for cryptography is that finding the value of k such that Q=k*P for some points Q and P on the curve is very difficult. I will explain what this means.

In elliptic curves, the "addition" of two different points is not performed with simple arithmetic operations like 2+2=4 that we learned in elementary school. For example, consider the elliptic curve $y^2 = x^3 + -4x + 1$. The point (0,1) satisfies this equation and is a valid point on the curve. Similarly, (2,1) is also a valid point, but the vector sum of these two points, (2,2), does not satisfy the curve's equation.

In elliptic curves, the operation we call "addition" actually involves finding the third point where the line passing through the two chosen points intersects the elliptic curve. This is represented as P+Q=A

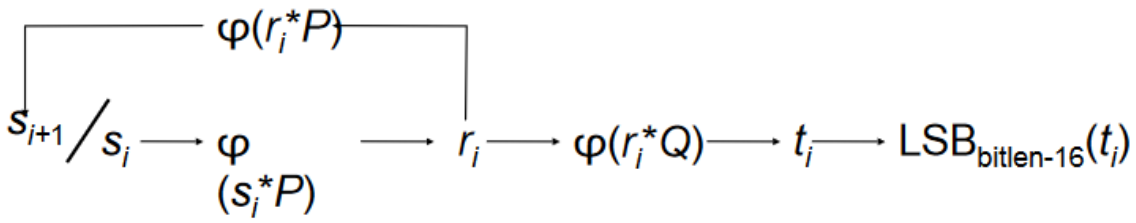


When adding a point to itself, we draw a tangent to the curve at that point and find the second intersection of the tangent with the curve. This is represented as P+P=A



There are complex arithmetic formulas used to obtain the coordinates of point A from the coordinates of points P and Q, but since this is not relevant to our topic, I will not discuss it in this article. The key concept to remember here is that even if we know the original point P and the resulting point Q, which satisfies the equation P*k=Q (meaning the operation of P+P performed k times), it is difficult to determine the value of k. This is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP). This allows us to use elliptic curves as a one-way function.

Dual_EC_DRBG works as follows: Before starting the process, points P and Q are selected. As mentioned at the beginning of the article, an entropy source is first found to determine s_0. The operation s_0*P is performed on the curve (i.e., the P+P operation is performed s_0 times) to obtain point r_1. The x-coordinate of point r_1 is then used as the new "internal state" value, s_1. This is our TYF1. When we want to generate a new internal state, we apply the s_1*P operation to obtain point r_1. To get the random number, the x-coordinate of r_1 is multiplied by point Q, i.e., s_1*Q is computed to get t_i, which is our TYF2 (with the first 16 bits of the result ignored), and our calculations end here.



An important question we need to ask is: How were the numbers P and Q selected? After NIST published the standards, they did not provide answers to these questions. This was suspicious because, in a presentation by Microsoft researchers Dan Shumow and Niels Ferguson in 2007, it was warned that the person selecting P and Q could potentially create a backdoor that would invalidate the encryption.

If there exists a number d such that P=Q*d (whether this relationship exists or not can only be known by the person who determined P and Q due to ECDLP), then this d would constitute a backdoor.

Let's consider multiplying the obtained t_1 by d. We can write this as

t_1*d,

which, using our formula, can also be written as r_1 * Q instead of t_1

Thus:

[r_1 * Q]*d = t_1*d Using the commutative property of multiplication:

r_1*[Q*d] = t_1*d

Finally, if we substitute the suspected relationship for [Q*d]

r_1*P = t_1*d

This result, as shown in the graph above, would provide the value of the next internal state s_2, meaning we have successfully infiltrated the encryption, allowing us to predict all future random numbers.

Bruce Schneier, in a November 2007 post, expressed that he could not understand why NIST recommended this standard in their guidelines, stating, “It doesn’t make sense as a backdoor:

It’s both obvious and quite blatant. It doesn’t make sense from an engineering standpoint either: It’s so slow that no one would use it voluntarily.”

In September 2013, documents leaked by Edward Snowden revealed that the NSA was working to weaken encryption standards and insert backdoors as part of its signals intelligence collection project and "war on encryption." This decryption program was named "Bullrun."

In December 2013, Reuters reported that RSA Security had received $10 million from the NSA in exchange for using Dual_EC_DRBG in their BSAFE library for random number generation.

RSA denied these claims, stating: "RSA has not entered into any agreements or participated in any projects with the intention of weakening our products or inserting 'backdoors' into them." However, they advised their customers against using elliptic curve algorithms.

When NIST researcher John Kelsey asked where the P and Q values came from, he was told that they could be randomly selected but that the NSA should not discuss this matter.

Richard George, who was the Technical Director at NSA’s Information Assurance Directorate from 2003 until his retirement in 2011, said in a May 2014 conference speech at Infiltrate: “We were going to use elliptic curves. I said, if you put this in your standards, nobody else will use it because it looks ugly and is really slow. There’s no good reason for anyone to use it. But I can. So they added it, and I said as long as we use these parameters, we can use them, and everyone else can add any parameters they want.”

Does this mean that encryption is now useless? Not exactly. As you remember, Dual_EC_DRBG was just one of four CSPRNG standards, and despite repeated warnings about this vulnerability, it continued to be used as a standard. The takeaway should be the awareness of using the correct types of encryption rather than the invalidation of encryption as a whole.

Lastly, an important note I want to make is that backdoor methods are one of the privacy violation techniques used by governments or companies but not the only one.