# BC-2.2.2.4 Elliptic Curve Digital Signature Algorithm (ECDSA)! 1/2

## introducing the public-private key pair generator – the ECDSA

This is the elliptic curve digital signature algorithm

![Image ECDSA]( https://raw.githubusercontent.com/koiosonline/literature-images/main/blockchain-level2/bc-2-2-2-4-the-crypto-flower-leaf-1-elliptic-curve-1-2-image1.png)

([[source](https://github.com/bitcoinbook/bitcoinbook)](<https://github.com/bitcoinbook/bitcoinbook)>)

You don't often see this line in a graph. Other than an extraordinary design, we use this formula to \*\*create public-private key pairs (!)\*\*. Please remember that public-private key pairs are nothing more than numbers running through a cryptographic formula during this session. This time not the SHA-256 formula, but the ECDSA formula.

Before you continue, watch these two introductory videos about the role these key pairs play. They are important.

[[Video 1](%20%5bVideo%201) asymmetric encryption]( https://www.youtube.com/watch?v=AQDCe585Lnc)

[[Video 2](https://www.youtube.com/watch?v=67uW07QDHxE) public and private key pairs explained]( <https://www.youtube.com/watch?v=67uW07QDHxE>)

Still don't get its role or importance? [[Be my guest](https://www.youtube.com/results?search_query=public+private+key+encryption+explained)😊]( <https://www.youtube.com/results?search_query=public+private+key+encryption+explained>)

## Don't lose your keys!

Okay, the latter wasn't fair. We'll go easy on you and explain the most important concepts here. Let's start. As you know, blockchain is a digital ledger, so everything is recorded in numbers (bits!). Otherwise, your hardware can't run it. So this ECDSA is like mentioned, once again, nothing more than numbers (significant ones, though!). Also, once again, like with hashing and PoW, the number that comes out of this formula is presented in a humanly readable hash to make it readable for us. \*\*Just as the hash is used with PoW, the hash in the ECDSA is used to change a huuuuuuuuuuuge number into a readable output (which is still alphanumeric). So still talking zero's and one's in its very core here and still made readable by hashing the bit number.\*\*

\*\*In its most simple form: a private key is like your password to your email address ( to get access to your public key, you need your private key like you need your password to use your mail address.\*\* You can also compare it with a physical key (private) and physical safe (public), where you keep the key for yourself (hence the name private) but share the safe with everybody to receive deposits (thus the name public).

>💡 Just like you use your mail account to communicate on the internet, you use your public key to communicate on the blockchain. Communicating is sending value. No private key means no communicating. You lost your private key = lost access to your account!

## What is a private key

Up for level 2. To be honest: it is not precisely like a password and mail or like a key and a safe. The difference, namely, is that the private key and your public key are the same "thing". As you might have guessed, this "thing" is once again……a number, of course!

>💡 Your private key is a number between 0 and 2^156. You can create one by flipping a coin 256 times and write down 0 for heads and 1 for tails.

>💡 Do note: the number you select must be completely random. Sometimes people's key was guessed because the software they used was not random.

Your private key is a number between 0 and 2^156. So the private key is a number based on a 256-bit sequence, so a number between 0 and 10^70. This number is insanely high. Other than the previous comparison with atoms, another example: it is like picking a sand grain from this world, make a new world from that sand grain, and pick once more one sand grain from this second world). Chances of you guessing somebody's random number between zero and 10^70 are equal of us being hit by a comet within next two seconds. One. Two. Luckily, we are still alive 😉

Okay, so far, so good. A private key is just a number between 0 and 10^70, and there are insanely amount of options to pick. But how do we derive the public key from that private key? This number is your starting point. Instead of typing data infield, we now use this particular number as we did in the Anders example.

We use the number as input in a formula called the ECDSA. The concept is still the same as the hash formula, input goes in, we calculate, and output comes out. Easy to calculate, (nearly!!) impossible to reverse. It is currently still impossible based on current technology and estimated technology within the upcoming few decades. This ECDSA is one of the Achilles heels of bitcoin because if you can reverse this code, this formula, and if you can calculate the input = you calculate the random number = you have calculated the password of the public key. This means that funds can be accessed. Luckily we are nowhere near that far, and even if that happens, there are other ways to generate key pairs. That would mean an adjustment in the protocol, however, and that would mean a fork. It would harm the network, possibly shut it down for a certain period if it completely takes everybody blind-sided. Still, it wouldn't kill the concept of bitcoin or other decentralized ledgers.

Equally important: if some entity uses such a power, more important risks, like having access to nuclear lunch codes, lay dormant. Many important infrastructures like nuclear launching codes rely on this formula. I like [<this>](https://www.youtube.com/watch?v=wlzJyp3Qm7s&vl=en ) explanation from Andreas Antonopoulos, but to be honest: who knows what the future holds? It seems naïve to think that something like Bitcoin is bullet-proof. Still, it does seem tremendously better protected than other ledgers out there (which keep getting hacked as well).

## The basics

But let's get back to the basics of ECDSA. They are similar to the hash function. So random input in, calculation, the fixed outcome comes out. It's easy to calculate in one direction. It is nearly impossible to reverse it. So, we use the private key to prove ownership of the public key (which holds the funds).

For the next part, we suggest you what the more [[visualized video](https://www.youtube.com/watch?v=z7AiQ1HQqVE&t=163s)]( https://www.youtube.com/watch?v=z7AiQ1HQqVE&t=163s ) first on Koios.

The goal is to prove ownership of something without revealing it. Therefore we use a public key to mask our private key. With the public key, we interact. With the private key, we gain access to the public key. The private key encrypted via ECDSA leads to the public key.

It works as follows (simplified 2.0): do you remember the old television or computer, that if it went in standby mode, you screen saver popped up and would show you an icon moving across the screen. You are bouncing from one corner to another, never leaving the screen (nor hitting a perfect corner of the screen, despite you waiting for quite some time, the good old days of true entertainment). The ECDSA is similar: instead of an icon that bounces, we use a random number between 10^70 (which is your private key). Let's ignore all previous warnings of selecting a completely random number and select the number "10" (variable 1). We grab this number, stretch is similar to a slingshot, and shoot it from a certain angle (variable 2) in the curve (like shooting an icon in the screen), where it starts bouncing against every corner of the curve (sort of a sideways bowl instead of tv-screen, so sometimes the number escapes from the bowl and we shoot it back). Every time it hits one of the curves, the number calculates (pie is involved in this, for example, times pie or divide it by pie, etc.). Therefore changing the input number to another number. Eventually, after a certain amount of bounces determined by the pace you shot it (variable 3), it has to stop somewhere and arrives on one of the sides of the curve. Where the icon kept on bouncing on your screen, the random number 10 stops after a certain amount and pops up somewhere totally different and is changed in number. So, for example, ten shots with a 45-degree angle and 1,000,000 bounces will result in the number 222,222. This number 222,222 is now your public key!

You see: still, the same number is only made unrecognizable by shooting it through the ECDSA from a certain angle at a certain pace, only you know and changing the number whenever it hits one of the curves. Thus, although the pace and angle are known, you only decide the random number with most pairing.

Fun fact for later: by using the same number but different degrees to shoot from and different amounts of bounces, you will result in other output numbers. Let's say that if you started with 10, 45 degrees, but this time only 100,000 bounces, you might have ended up at 333,333, for example. This way, you can generate multiple public keys with the same private key. Because of the vast amounts of possibilities, you picking the same input number is like being hit by a comet within two seconds. And that is how the ECDSA transforms your private number into a public number. Easy to recalculate, statistically (currently) nearly impossible to reverse.

Because your private key is now transformed and cloaked, thanks to the ECDSA, we can freely publish the altered version. As we will explain in level 3, there are additional and different hashes for extra security (to fortify the one-way direction). It looks [like [this](like%20%5bthis)](https://www.bitcoinnotbombs.com/wp-content/uploads/2014/01/address.png) in Bitcoin

![Image]( https://www.bitcoinnotbombs.com/wp-content/uploads/2014/01/address.png)

You will find multiple great clips in the further reading, explaining public and private key pairing. And how they exactly play a role in "signing" documents. In short: where we add a nonce to calculate the hash of a block, we add your private key number to a text to calculate the new hash. Because your private key is a number that consists of 256 random bits, the text changes and, therefore, the hash changes, only you can reproduce this text because only you know the added 256-bit number. Others can try, but once again: there are 2^256 possibilities. Even if somebody would succeed, would that be worth the energy and money? Let this all sink in before you watch the following videos. We HIGHLY recommend watching them at least twice until you fully understand the concepts of public-private key pairing and how they work. I have to re-watch it myself sometimes as well, therefore the challenge: try to explain these signing concepts to somebody else (ELI5).

[Video <https://www.youtube.com/watch?v=YEBfamv-_do&feature=emb_logo>)

[Anders Public / Private keys signing – How do you sign transactions?](<https://www.youtube.com/watch?v=xIDL_akeras>)

>💡 Many of the clips are of the utmost important to frame your basic understanding of cryptography. Grow your skills, and watch them as many times as needed. In a digital era, cryptography is getting more essential by the day!

## Further readings

\* [Mastering Bitcoin](<https://github.com/bitcoinbook/bitcoinbook> )

\* [Private public key explained (high overview 1) ](<https://www.youtube.com/watch?v=AQDCe585Lnc>)

\* [Private public key explained (high overview 2) ](<https://www.youtube.com/watch?v=67uW07QDHxE> )

\* [Quantum computing & ECDSA](<https://www.youtube.com/watch?v=wlzJyp3Qm7s&vl=en>

\* [Koios ECDSA: <https://www.youtube.com/watch?v=z7AiQ1HQqVE&t=163s>)

\* [Overview from private to public key](<https://www.bitcoinnotbombs.com/wp-content/uploads/2014/01/address.png>)

\* [Public key cryptography (2nd part if you are in a rush today) ](<https://www.youtube.com/watch?v=YEBfamv-_do&feature=emb_logo> )

\* [Anders Public / Private keys signing](<https://www.youtube.com/watch?v=xIDL_akeras> )