ECDH: Securing your internet connection

Introduction

In a world where we rely on computer systems that communicate with each other, keeping this communication secure is very important. Your PC, phone and IoT devices use ECDH and other protocols to secure this communication. These devices use ECDH when using protocols and software such as TLS[1], OpenVPN and WireGuard.

What is ECDH?

ECDH is a protocol used by for example TLS to establish a key which will be used to encrypt and decrypt communications with a server. When you browse the internet, you do not want an attacker to be able to intercept the passwords that you use. By using encryption, an attacker can not read the passwords that you use on websites. To be able to use encryption, you will need to agree on a key that you and the server use for encryption and decryption. Now a problem arises, how do you agree on this key? When you generate a random key and send it to the server, an attacker listening on the network can intercept this key and thus decrypt the connection with the server. ECDH solves this problem. The abbreviation ECDH consists of two parts: Elliptic-curve and Diffie-Hellman. Diffie-Hellman is a method that can be used to securely exchange a key with a server, even if an attacker is listening on the network. ECDH uses this method with elliptic curves.

How does ECDH work?

Actually using ECDH consists of multiple steps. Let's say Alice and Bob want to generate a shared secret. We will explain ECDH using Alice and Bob as this is the conventional way to explain cryptography. When securing your connection to a website using TLS, Alice is a phone, Bob a server, and the shared secret the key used for encryption and decryption.

Step 1: Choose a curve

ECDH operations are done on an elliptic curve. The first step is to agree on this curve. Alice and Bob agree to use Curve25519[2]. ECDH with Curve25519 is called X25519.

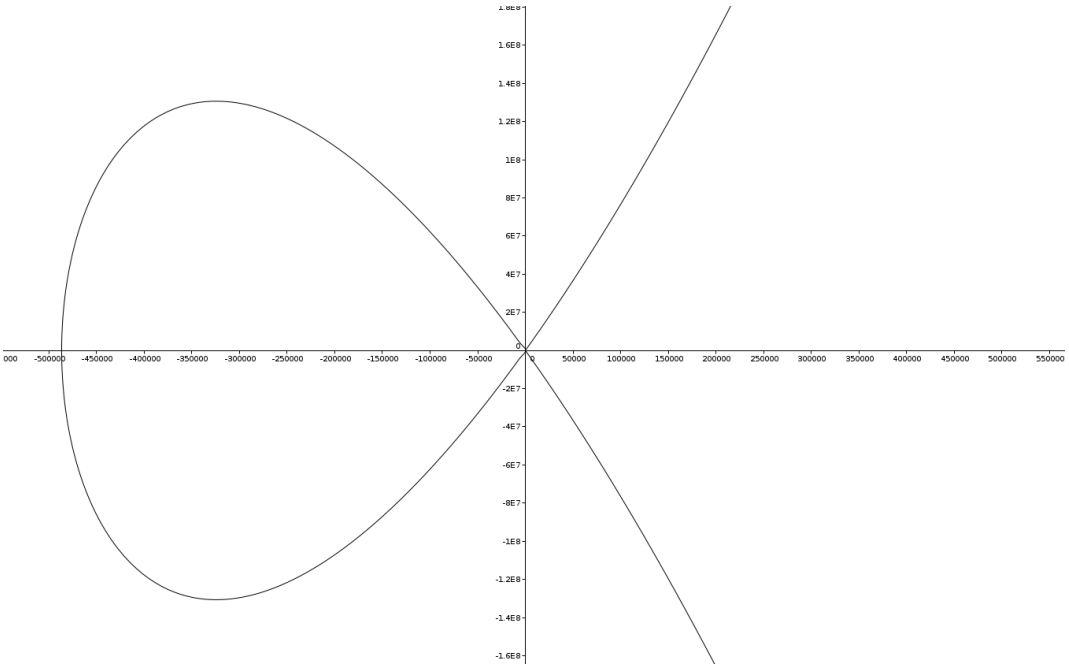
TLS 1.3 supports different curves with ECDH including Curve25519 and secp256r1. When using TLS, the client tells the server what curves it supports and the server will choose a curve.

Curve25519 is defined as $v^2 = u^3 + 486662\,u^2 + u$. v and u are used to indicate that points are on Curve25519. Curve25519 looks like:

---

1 In older versions of TLS, besides ECDH, another key exchange method could be used: RSA. Starting from TLS 1.3, only ECDH is used.

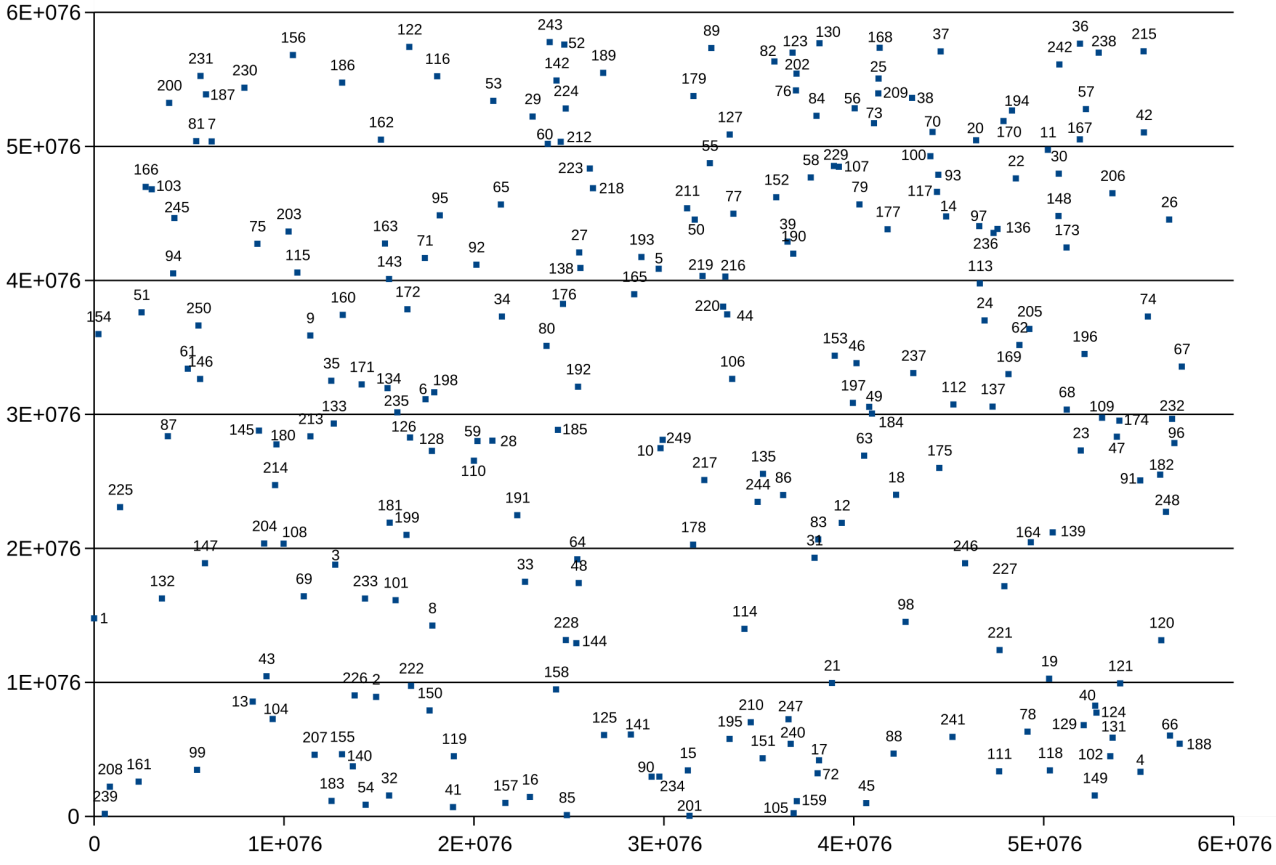
2 Why Curve25519? Because the author was researching WireGuard and WireGuard uses ECDH with Curve25519.



The picture of the curve looks nice, but the full curve is not used in calculations. Some mathematical magic is applied to the curve:

- All points used are calculated by applying scalar multiplication to a base point.
- The curve is used over a prime field.
- Only 1/8 of all points are used. (A subgroup)

You do not have to understand any of this mathematical magic, but this magic has big implications. The first 250 points of our curve now look like:



Our curve now looks like total randomness. This unpredictability is an important part of what makes X25519 secure.

What we see in the plot is the multiplication of a base point times a number. The base point is the point where x is 9, this is the point on the left side in the plot marked with 1. We will identify this base point as G. When we multiply G with 2, we get the point marked 2 in the plot. When we multiply G with 42, we get the point marked 42 in the plot. The multiplication used is not regular multiplication, but elliptic curve scalar multiplication. How elliptic curve scalar multiplication works exactly is not described here, but the result of the first 250 scalar multiplications are visible in the plot.

Step 2: Generate a private-public key pair

The next step for Alice and Bob is to both generate a private key and a corresponding public key. Bob and Alice both generate a random number. This random number is the private key. To prevent certain attacks, some bits of this number are changed, but for simplicity we will omit this in our example. Let's say Alice randomly generates the number 13 and Bob generates the number 17. These numbers are used to generate the public key. This public key is calculated using multiplication with the base point G. These calculations are plotted in the plot. We can search the plot for the number 13 and can see that the public key of Alice is (8,351E+075; 8,569E+075). The public key of Bob, corresponding to the number 17, is (3,817E+076; 4,186E+075).

Step 3: Share the public key

Bob sends his public key to Alice and Alice sends her public key to Bob.

Step 4: Calculate the shared secret

The next step is to calculate the shared secret. Alice multiplies her private key (13) with the public key point of Bob, which results in a new point, the shared secret. Bob multiplies his private key (17) with the public key point of Alice and derives the same shared secret.

Why do Alice and Bob derive the same shared secret, even though they use different parameters? The public key of Bob is $17*G$. When Alice multiplies Bob's public key with her private key: $13*(17*G)$, she calculates $13*17*G = 221*G$. When Bob multiplies Alice's public key with his private key: $17*(13*G)$, he calculates $17*13*G = 221*G$.

As we can see in the plot, $221*G$ corresponds to the point (4,768E+076; 1,241+076). This point is the shared secret calculated by both Bob and Alice. In practice the bits of the X coordinate are used as the shared secret, because the Y coordinate can be retrieved using both the X coordinate and the curve, and thus does not add any randomness.

Why can an attacker not derive the shared secret?

An attacker listening on the network can see the public keys, because these keys are send over the network. The private keys are not send over the network and can thus not be seen by the attacker. An attacker has the public key of Alice (8,351E+075; 8,569E+075) and the public key of Bob (3,817E+076; 4,186E+075). To derive the shared secret, we need to know at least one private key. The challenge for the attacker is now to derive a private key from a public key point in a reasonable

time. No efficient algorithm has yet been discovered to calculate the private key from such a public key point[3]. There does exist an efficient algorithm however to calculate the public key when knowing the private key (the double-and-add algorithm). This difference makes the algorithm secure. Knowing the private key we can easily derive the public key. Knowing the public key, we can not easily derive the private key.

You might think that an attacker can try all possible private keys and make a list or plot of the corresponding public keys. Because there are so many possible public keys (more than 2^{249}), it is not feasible to calculate all of them in a short time.

Challenge for the reader

Alice and Bob perform another key exchange. The shared secret is (5,528E+076, 5,710E+076). What are the private keys of Alice and Bob?

Want to have your answer checked? E-mail it to nickaquina@gmail.com. Compliments, feedback and other comments about this article can also be sent to this e-mail address.

Sources

1. Rescorla, E. (2018, August). rfc8446. IETF | Internet Engineering Task Force. Retrieved 17 December 2021, from https://datatracker.ietf.org/doc/html/rfc8446
2. Langley, A., Hamburg, M., & Turner, S. (2016, January). rfc7748. IETF | Internet Engineering Task Force. Retrieved 17 December 2021, from https://datatracker.ietf.org/doc/html/rfc77483
3. Wikipedia contributors. (2021, 15 november). *Elliptic-curve Diffie–Hellman*. Wikipedia. Geraadpleegd op 17 januari 2022, van https://en.wikipedia.org/wiki/Elliptic-curve_Diffie%E2%80%93Hellman
4. Wikipedia contributors. (2021b, november 29). *Elliptic curve point multiplication*. Wikipedia. Geraadpleegd op 17 januari 2022, van https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication

3 This is actually not true, there exists an algorithm to break elliptic curve cryptography: Shor's algorithm. Breaking elliptic curve cryptography using Shor's algorithm requires however a quantum computer which are not easy to come by.