Example 12 — The Tale of the Two Keys: Understanding SSH and ED25519

Scene 1 — The Birth of a Key Pair

Once upon a command line, a developer typed:

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

and with that, two keys were born.

- The **private key** (usually stored as `/.ssh/id_ed25519`) — a secret mathematical number you must never share.
- The **public key** (stored as `/.ssh/id_ed25519.pub`) — a mathematically related value that anyone may see.

These two keys are linked by a one-way mathematical bond: from the private key you can derive the public key, but from the public key alone, you cannot feasibly find the private key.

In mathematical terms: The private key is a random number k (an integer modulo a large prime number), and the public key is $K = k \cdot B$, where B is a special base point on an elliptic curve called *Curve25519*.

The multiplication $\cdot$ here isn't ordinary arithmetic — it's *elliptic curve point multiplication*. It's easy to go forward (compute K from k), but practically impossible to go backward (find k given K).

This one-way direction is what makes the system secure.

Scene 2 — The Meeting with GitHub

When you upload your `id_ed25519.pub` to GitHub, you're essentially saying:

> "GitHub, if anyone ever proves they know the private number k that goes with this public key K, please trust them — that's me."

GitHub doesn't need to store your private key or a password. It just saves your public key K and waits for you to prove your identity through math.

Scene 3 — The Silent Handshake

Later, when you try to connect:

```
$ git push origin main
```

your computer (the *client*) sends GitHub (the *server*) a friendly request: "Hey, I'd like to log in using my public key K."

GitHub checks if it recognizes that K from your account, and if so, it sends back a one-time random challenge — a random number r that must be signed.

Your computer uses your private key k to produce a mathematical signature σ that proves "I know k" without ever revealing k itself.

GitHub then uses your public key K to check the signature:

$$\text{Verify}(\sigma, r, K) \Rightarrow \text{True or False.}$$

If the math checks out, GitHub knows it's really you.

No passwords were typed. No secrets were sent. Only proof.

This is called **public-key authentication**, and the math behind it (in ED25519) is built on the secure foundation of elliptic curve cryptography.

Scene 4 — Under the Hood: ED25519's Elegant Math

At its heart, ED25519 uses a specific elliptic curve over a finite field $\mathbb{F}_p$ with $p = 2^{255} - 19$. The curve equation looks like this:

$$-x^2 + y^2 = 1 + dx^2y^2$$

where $d = -121665/121666$.

All computations are done *modulo* p.

Signing process (simplified):

1. Your private key k is used to generate a random value r.
2. Compute a public value $R = r \cdot B$.
3. Hash together $(R, K, \text{message})$ to get a challenge h.
4. Compute a response $s = r + h \cdot k \pmod{q}$.
5. Your signature is the pair (R, s).

Verification process:

$$sB \stackrel{?}{=} R + hK.$$

If that equality holds, the signature is valid — proving you knew the private key k that corresponds to the public key K, without ever revealing k itself.

Scene 5 — Trust, Speed, and Beauty

Why ED25519? Because it's:

- **Fast.** Operations on Curve25519 are designed for efficiency on modern CPUs.
- **Secure.** 255-bit keys are strong enough for decades of safety.
- **Simple.** The algorithm avoids messy mathematical corner cases that plagued older elliptic curves.
- **Deterministic.** The same private key always yields the same signature, eliminating random side-channel leaks.

In short: ED25519 is the elegant modern successor to RSA — leaner, faster, and just as trustworthy.

Scene 6 — What Really Happens When You Push to GitHub

When you type:

```
git push origin main
```

a full cryptographic handshake unfolds in milliseconds:

1. GitHub sends a challenge (random data).
2. Your SSH client signs that challenge using your private key.
3. GitHub verifies the signature with your public key.
4. The secure session is established — encrypted, authenticated, and trusted.

Every push, pull, or fetch after that uses the same session's encrypted channel. No passwords. No eavesdropping. Just math.

Epilogue — The Moral of the Story

SSH with ED25519 is not just about convenience — it's about **trust through proof, not secrecy.**

> Your private key whispers: "I know something only I could know." Your public key declares: "Check my math, and you'll believe me."

Together they make it possible for developers around the world to collaborate securely — building bridges of trust made entirely of numbers.

Optional Extension for the Curious

If you'd like to visualize what's really happening:

- Generate a key: `ssh-keygen -t ed25519`

- Print your public key: `cat  /.ssh/id_ed25519.pub`
- Run: `ssh -vT git@github.com`

You'll see each step of the cryptographic dialogue. It's math in motion — security as symphony.

Takeaway: When you use SSH with GitHub, you are not "logging in." You are *proving who you are with math.*