1. **Giới thiệu bài toán**

Thuật toán chữ ký số ECDSA hoạt động dựa trên việc chọn ngẫu nhiên một số nguyên k. Ta sẽ tính

$$P = k*G$$

(G là điểm sinh đã chọn từ trước)

Sau khi có được P(x1, y1) ta sẽ có r = x1. Sau đó ta sẽ tính

$$s = k^{-1} (z + r*d)$$

(m là mẩu tin cần ký, z = hash(m) , d là khóa riêng)

Khi này một mẩu tin m sẽ được ký bởi hai giá trị là r và s. Có thể dễ dàng thấy, khi ký hai mẩu tin với cùng một k thì ta sẽ có hai giá trị r giống nhau, khi đó ta sẽ có (r, s1) và (r, s2).

Bây giờ thử tính s1 - s2 ta có:

$$s1 - s2 = k^{-1} (z1 + r*d) - k^{-1} (z2 + r*d) = k^{-1} (z1 + r*d - z2 - r*d) = k^{-1} (z1 - z2)$$

Từ phương trình trên ta có thể tìm ra k bằng cách tính

$$k = \frac{z1-z2}{s1-s2}$$

Hơn thế nữa, sau khi tìm thấy k, kẻ tấn công có thể tính khóa riêng từ một trong các chữ ký. Ta để ý rằng:

$$r^{-1}(k * s - z) = r^{-1}(k * k^{-1} (z + r * d) - z) = r^{-1}(z + r * d - z) = r^{-1} * r * d = d$$

Vậy là từ giờ phút này, kẻ tấn công đã có mọi thứ hắn cần để có thể ký bất kỳ mẩu tin nào mà hắn muốn, trên danh nghĩa của người mà kẻ tấn công đã đánh cắp khóa bí mật

2. **Giới thiệu bài thực hành**

- **Mục đích**
  - + Giúp sinh viên hiểu rõ cách hoạt động của thuật toán chữ ký số ECDSA
  - + Phân tích rủi ro bảo mật khi tái sử dụng cùng một giá trị k để ký nhiều thông điệp
  - + Viết mã khai thác để khôi phục khóa riêng tư từ hai chữ ký có cùng k
- **Yêu cầu đối với sinh viên**
  - + Nắm vững kiến thức cơ bản về thuật toán chữ ký số ECDSA, cách tính chữ ký (r, s) và vai trò của tham số k

+ Có khả năng sử dụng python để thao tác chuỗi, số nguyên lớn, viết script khai thác

3. Nội dung thực hành

- Tải bài thực hành:

imodule
https://github.com/Baorista/Labtainer/raw/refs/heads/main/imodule_ecdsa_reuse_k.tar

- Khởi động bài lab:
  Vào terminal, gõ:

rebuild ecdsa_reuse_k

- Sau khi khởi động xong, màn hình sẽ xuất hiện 1 terminal.
- Bài thực hành đã cho sẵn 2 mẩu tin và 2 file chữ ký được ký bởi cùng một khóa k, file pubkey.pem khóa công khai, một file secret.ct chứa mẩu tin bí mật được chuyển thành byte sau đó mã hóa bằng cách chuyển khóa d thành xâu rồi chuyển sang dạng byte, sau đó băm khóa này với SHA256, rồi cắt 16 byte đầu của giá trị băm nhận được để làm khóa cho AES-128 bit, sau đó dùng AES-ECB để mã hóa
- Sinh viên tự code file recover.py để thực hiện in ra khóa k, d và thông điệp bí mật:
- Gợi ý code recover.py:
  + Có thể sử dụng thư viện ecdsa để đọc khóa công khai và chữ ký, hashlib cho hàm băm, Crypto để giải mã AES
  + Đọc khóa công khai với VerifyingKey từ ecdsa
    ```
    with open("public_key.pem", "rb") as f:
        vk = VerifyingKey.from_pem(f.read())
    ```
  + Tiếp theo đọc chữ ký, và thông điệp
    ```
    with open("m1.txt", "rb") as f:
        m1 = f.read()
    with open("sig1.txt", "rb") as f:
        sig1 = f.read()
    ```
  + Sau khi có khóa công khai và chữ ký, có thể tách r, s bằng util.sigdecode_string từ ecdsa
    ```
    r1, s1 = util.sigdecode_string(sig1, vk.pubkey.order)
    ```
  + Ở bước này ta có thể so sánh r1 == r2, nếu trả về False thì có nghĩa là hai chữ ký sử dụng khác k
  + Lấy bậc sinh của đường cong rồi sau đó tính k theo công thức k = (z1 - z2)/(s1 - s2) mod n
    ```
    n = vk.curve.order
    ```

k = ((z1 - z2) * pow(s1 - s2, -1, n)) % n

+ Sau khi có k tiếp tục tính d theo công thức d = (s*k - z1)/r mod n

r = r1 # r1 = r2 nên lấy r1 là được

s = s1

d = ((s * k - z1) * pow(r, -1, n)) % n

+ Sau khi có d rồi ta có thể tiến hành giải mã mẩu tin bí mật, dầu tiên đọc mẩu tin

with open("secret.ct", "rb") as f:

ciphertext = f.read()

+ Tạo khóa theo mô tả của đề bài

key = sha256(str(d).encode()).digest()[:16]

+ Sử dụng AES được import từ Crypto.Cipher để giải mã

cipher = AES.new(key, AES.MODE_ECB)

plaintext = cipher.decrypt(ciphertext).rstrip(b' ')

plaintext = plaintext.decode()

Có thể tham khảo đoạn code sau:

```
from ecdsa import VerifyingKey, util
from hashlib import sha1, sha256
from Crypto.Cipher import AES

try:
    with open("public_key.pem", "rb") as f:
        vk = VerifyingKey.from_pem(f.read())

    with open("m1.txt", "rb") as f:
        m1 = f.read()
    with open("m2.txt", "rb") as f:
        m2 = f.read()
    with open("sig1.txt", "rb") as f:
        sig1 = f.read()
    with open("sig2.txt", "rb") as f:
        sig2 = f.read()

    r1, s1 = util.sigdecode_string(sig1, vk.pubkey.order)
    r2, s2 = util.sigdecode_string(sig2, vk.pubkey.order)

    if r1 != r2:
        raise Exception("Signatures do not reuse the same k (r values differ)")
```

```
    z1 = int.from_bytes(sha1(m1).digest(), 'big')
    z2 = int.from_bytes(sha1(m2).digest(), 'big')

    n = vk.curve.order
    k = ((z1 - z2) * pow(s1 - s2, -1, n)) % n

    r = r1
    s = s1

    d = ((s * k - z1) * pow(r, -1, n)) % n

    print(f"Recovered k = {k}")
    print(f"Recovered d = {d}")

    # Giải mã secret.ct bằng d
    with open("secret.ct", "rb") as f:
        ciphertext = f.read()

    key = sha256(str(d).encode()).digest()[:16]  # dùng d làm khóa AES
    cipher = AES.new(key, AES.MODE_ECB)
    plaintext = cipher.decrypt(ciphertext).rstrip(b' ')

    print("Decrypted secret:", plaintext.decode())

except Exception as e:
    print("Recover failed:", e)
```

+ Kết thúc bài lab:

stoplab ecdsa_reuse_k

+ Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.

+ Khởi động lại bài lab: Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

labtainer -r ecdsa_reuse_k