1. **Giới thiệu bài toán**

- **Bối cảnh**

Elliptic Curve Diffie-Hellman (ECDH) là giao thức phổ biến dùng để trao đổi khóa bí mật giữa hai bên thông qua điểm sinh G trên đường cong elliptic. Một bước quan trọng trong ECDH là nhân khóa riêng d với public key đối phương P, tính Q = d*P

Tuy nhiên, trong nhiều triển khai thực tế, điểm PPP không được kiểm tra xem có thực sự thuộc về đường cong chuẩn đã chọn hay không.

- **Vấn đề**

Nếu attacker gửi một điểm P không nằm trên đường cong hợp lệ E nhưng có cùng tham số a và p, thì phép nhân d*P vẫn được tính hợp lệ về mặt kỹ thuật, vì công thức nhân điểm không sử dụng hệ số b của đường cong.

Attacker có thể lợi dụng điều này để:

  - Gửi điểm thuộc đường cong giả E′ với order nhỏ.
  - Nhận lại ciphertext từ server sử dụng shared key Q.
  - Brute-force key → suy ra dmod n, với n là order của P.
  - Lặp lại nhiều lần với các order khác nhau.
  - Dùng định lý số dư Trung Hoa để khôi phục full private key d.

2. **Giới thiệu bài thực hành**

- **Mục đích**
  - Xác định cách attacker tạo được các điểm $P_i \in E'_i$ với order nhỏ.
  - Gửi các điểm này đến server đang dùng ECDH không kiểm tra điểm.
  - Từ các ciphertext nhận về, brute-force được các đồng dư: $d \equiv r_1 \bmod n_1$, $d \equiv r_2 \bmod n_2$, …
  - Khôi phục d bằng định lý số dư Trung Hoa (CRT).
- **Ý nghĩa thực tiễn**
  - Invalid Curve Attack đã xảy ra trong thực tế: OpenSSL, Smartcard, HSM…
  - Cho thấy rằng dù thuật toán mật mã có mạnh, nhưng nếu triển khai sai → vẫn bị khai thác.
  - Đây là ví dụ điển hình cho tấn công kiểu *implementation attack* – rất quan trọng trong pentest, audit.
- **Yêu cầu đối với sinh viên**
  - Có hiểu biết về ECC, ECDH, CRT
  - Có kỹ năng lập trình python và sagemath, làm việc với socket, xử lý dữ liệu dạng byte

3. Nội dung bài thực hành

- Tải bài thực hành:

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

- Khởi động bài lab: Vào terminal gõ:

  rebuild ecdh_invalid_curve_attack

- Khi khởi động màn hình sẽ xuất hiện hai terminal của server và attacker
- Trong server có sẵn server.py, trong attacker có sẵn file attacker.py
- Sinh viên chạy

  python3 server.py

  Khi này Server sẽ:

  - Lắng nghe trên cổng 9999
  - Gửi tham số đường cong (p, a, G) cho attacker khi có kết nối
  - Chấp nhận nhiều public key gửi đến từ attacker
  - Với mỗi điểm A, server sẽ tính shared_point = d * A (không kiểm tra A hợp lệ)
  - Mã hóa thông điệp bằng AES dùng x, y của shared_point, và trả ciphertext về cho attacker
- Ở phía bên attacker, sinh viên cần chỉnh sửa attacker.py sao cho:
  - Attacker sẽ kết nối tới client
  - Sinh các điểm có bậc n nhỏ trên đường cong E’ với b’ != b
  - Gửi lần lượt các điểm này tới server
  - Nhận lại thông điệp được mã hóa
  - Giải mã và tìm ra d mod n
  - Sau khi tìm đủ thì sử dụng định lý số dư Trung Hoa để tìm ra d
- Hướng dẫn code:

  Dùng

  nano attacker.py

  để chỉnh sửa

- Đầu tiên, cần kiểm tra ip của server bằng ifconfig, sau đó sửa tham số host và port của hàm main để kết nối tới server host="<server_ip>", port=9999
- Hàm decrypt_data(shared_point, ciphertext)
  - Đọc hàm encrypt_data của bên Server để viết hàm decrypt giải mã
  - Đầu tiên cần xử lý x, y khi điểm là vô cùng. Có thể sử dụng cú pháp shared_point.is_zero(). Nếu điểm là vô cùng, ta sẽ đặt x,y = 0, còn không thì lấy x,y của điểm như thường
  - Chuyển x, y thành chuỗi byte để làm khóa và vector khởi tạo bằng long_to_bytes(int(x)) rồi đệm cho đủ 16 byte với rjust(16, b"\x00")
  - Tạo một đối tượng AES: cipher = AES.new(key, AES.MODE_CBC, iv)
  - Dùng cipher.decrypt(ciphertext) rồi unpad(...): decrypted = cipher.decrypt(ciphertext) return unpad(decrypted, 16)
- Hàm brute_force_encrypted_message(A, ciphertext, max_order)
  - Ta cần lặp qua tất cả giá trị i từ 1 → max_order
  - Tính shared_point = i * A
  - Với mỗi shared_point dùng hàm decrypt_data(shared_point, ciphertext) để giải mã
  - Dùng .decode() để kiểm tra xem plaintext có hợp lệ không
  - Nếu thành công, trả về i
  - Lưu ý để tránh lỗi thì nên để quá trình decrypt ở trong try except
- Hàm find_curves_with_small_subgroup(p, a, max_order)
  - Đầu tiên hãy thử sinh ra một điểm từ một đường cong mới đã
  - Đặt b = 1
  - Tạo đường cong mới bằng E = EllipticCurve(GF(p), [a, b])
  - Tạo điểm ngẫu nhiên bằng R = E.random_point()
  - Lấy bậc của R với n = R.order()
  - Điểm này có thể không có bậc nhỏ, tuy nhiên ta có thể lấy điểm bậc nhỏ bằng cách lấy R*(n/f) khi này ta sẽ có điểm mới P có bậc f
  - Lấy các ước số nguyên tố của R bằng for f, e in n.factor():
  - Lấy điểm P mới có bậc f bằng: P = (n // f)*R

- Trả về P f b với yield(f , P , b)
- Giờ ta đã có một điểm P với bậc nhỏ, ta sẽ muốn thêm, vậy nên ta cần dùng vòng lặp
- Trước hết thì quay lại đầu hàm, ta sẽ khởi tạo một set: orders_found = set(). Chúng ta sẽ không muốn có hai điểm P cùng một bậc n, vậy nên ta sẽ lưu các bậc tìm được vào đây
- Ta sẽ để b = 0 và lặp với While True: b+=1

  if b == p: break

- bằng cách này ta sẽ lặp qua các giá trị của b từ 1->p-1 để thử mọi đường cong có thể, tuy nhiên cũng cần kiểm tra điều kiện (4*a^3 + 27*b^2) % p == 0 để tránh sinh ra đường cong kỳ dị
- Cứ tiếp tục như cũ đến bước R = E.random_point(), mỗi lần sẽ sinh ra một điểm ngẫu nhiên R, vậy thì ta sẽ muốn thử nhiều R nhất có thể những cũng không quá nhiều để khiến ta gặp lại điểm cũ quá nhiều. Vậy thì ta sẽ cho lặp for _ in range(100): R = E.random_point(), với cách này ta sẽ sinh ra 100 điểm R ngẫu nhiên, 100 là một con số khá ổn cho việc này, có thể tăng giảm tùy thích
- Gần xong rồi, giờ với mỗi ước số nguyên tố f của n, ta sẽ kiểm tra xem nó có trong orders_found chưa, nếu có thì thêm vào, và nếu f > max_order thì sẽ bỏ qua. max_order là một biến để đảm bảo rằng bậc của điểm sinh giả không quá lớn để khi brute force không bị mất nhiều thời gian

- Sau khi chỉnh sửa xong, chạy bằng lệnh:

  sage attacker.py

- Nếu mọi thứ hoạt động đúng, khóa riêng d của server sẽ được tìm ra
- Kết thúc bài lab:

  stoplab ecdh_invalid_curve_attack

- 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 ecdh_invalid_curve_attack