2024 암호분석경진대회 - 비밀분쇄기팀 4번 문제 풀이

복원한 영수의 개인키.

`1D 08 A3 13 05 E2 40 E0 AD D3 DF 29 58 06 3A D6 31 60 93 0D 17 C1 3A F0 8F 72 03 8F 13 E0 20 78`

1 전자서명 프로그램의 취약점 파악

Table 1: 철수의 ECDSA 전자서명 정보

철수		값
메시지 m		`cryptoanalysiscontest`
해시 e		`9A 2E 62 81 8A D5 5A EB 8A C3 19 82 0B 2D 59 56 60 B9 AF 57 C0 C7 12 3B D6 C6 DF DE 2D 9A 17 53`
서명	r	`EB 71 F2 4C E4 4A A9 9D 89 1B BA 76 23 41 43 55 E6 3B F9 2A 74 D7 53 F7 CB AA B7 83 1A 35 79 08`
	s	`80 60 D4 0B C3 BF 41 F5 D8 45 E3 EF 6A E2 27 00 47 A1 E2 A3 E6 C0 57 BF C5 77 D7 D8 84 08 9D 47`
개인키 d		`BD E0 7E 98 F0 43 7A 53 1C 01 4A 1F E6 FD 69 C2 CF B6 C3 65 70 72 69 6E 74 32 30 32 33 38 34 31`
k		`9A 2E 62 81 8A D5 5A EB 8A C3 19 82 0B 2D 59 56 BD E0 7E 98 F0 43 7A 53 1C 01 4A 1F E6 FD 69 C2`

철수의 개인키를 알고 있는 상황에서는 다음과 같이 철수가 메시지를 서명할 때 사용한 정수 k 의 값을 계산할 수 있다.

$$\begin{aligned} s &= k^{-1}(e + rd) \mod n \\ \rightarrow k &= s^{-1}(e + rd) \mod n \end{aligned}$$

이를 통해 계산한 k 값이 Table 1 에 나타나 있다. k 값을 관찰해보면, 상위 16바이트는 메시지의 SHA-256 해시값인 e 의 상위 16바이트와 일치하고, 하위 16바이트는 개인키 d 의 상위 16바이트와 일치하는 것을 확인할 수 있다. 즉, 서명 프로그램의 취약점은 k 를 **랜덤으로 생성하지 않고** $k = e_H \,||\, d_H$ **로 사용하는 것**이다. 여기서 e_H, d_H 는 각각 e, d 의 상위 16바이트, e_L, d_L 은 각각 e, d 의 하위 16바이트이다.

2 취약점을 이용한 개인키 복원

Table 2: 영수의 ECDSA 전자서명 정보

영수		값
메시지 m		`helloecdsa`
해시 e		`56 8B 49 01 CC 2D AC 4A 13 AF 16 1B BF 6B 20 87 C9 4D 8B 82 23 75 5F D1 21 EC 6A A0 51 9E CE E2`
서명	r	`DA 78 66 63 21 09 E7 7F 0D 3C 5B DD 49 03 1E 6D 9A 94 0F CB 7D 29 EA 2F 85 8B 99 1D 1F 17 CE F8`
	s	`A4 A7 00 AC 4F 18 63 4A C8 45 73 9E 03 42 CD 83 35 BF 6E 06 06 CA 9D C9 D6 68 AB F9 ED 81 2E 6D`
k		`56 8B 49 01 CC 2D AC 4A 13 AF 16 1B BF 6B 20 87 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??`

발견한 취약점을 바탕으로 영수가 서명을 생성할 때 사용한 k 의 상위 16바이트를 바로 알아낼 수 있다. 그러나 Table 2 에 보이는 것처럼 하위 16바이트 (d_H) 는 여전히 모르는 상태이다. 이를 알아내기 위해 $d = 2^{128}d_H + d_L$ 및 $e = 2^{128}e_H + e_L$ 로 나타내고 s 를 계산하는 식을 풀어보면 다음과 같다.

$$\begin{aligned} & s = k^{-1}(e + rd) \mod n \\ \rightarrow\ & ks = e + rd \mod n \\ \rightarrow\ & (2^{128}e_H + d_H)s = 2^{128}e_H + e_L + r(2^{128}d_H + d_L) \mod n \\ \rightarrow\ & (s - 2^{128}r)d_H = rd_L + 2^{128}e_H + e_L - 2^{128}e_H s \mod n \\ \rightarrow\ & d_H = (s - 2^{128}r)^{-1}rd_L + (s - 2^{128}r)^{-1}(2^{128}e_H + e_L - 2^{128}e_H s) \mod n \end{aligned}$$

여기서 이미 알고 있는 값들을 $\alpha = (s - 2^{128}r)^{-1}r,\ \beta = (s - 2^{128}r)^{-1}(2^{128}e_H + e_L - 2^{128}e_H s)$ 로 치환하면, 다음과 같은 형태로 문제가 변환된다.

$$d_H = \alpha d_L + \beta \mod n$$
$$0 \le d_L, d_H < 2^{128}$$

α, β 값을 실제로 계산하면 다음과 같다.

$\alpha =$ `37 97 4E 35 A9 F0 CC C7 A0 36 9E 55 DB 72 8E 9B 7C 44 26 91 4B 22 6C 53 37 6F 5E C4 0C 10 27 94`
$\beta =$ `49 7C 82 06 49 46 2C 2D 3F 79 4D 6D 57 73 5C 9E F4 4F 72 D2 B8 44 25 0F F9 8F 64 0C 7D 0B AE C6`

$2^{128} < \alpha, \beta < n$ 임을 확인하고, 풀어야 하는 문제를 한번 더 변환하면 다음과 같다.

$$\text{Find minimum } d_L \text{ such that,}$$
$$n - \beta \le \alpha d_L \le 2^{128} - \beta + n$$

찾아낸 d_L 이 $0 \le d_L < 2^{128}$ 를 만족하면 d_L, d_H 를 모두 찾아낸 것이다. 조건을 만족하는 d_L의 값을 구하기 위해 확장 유클리드 알고리즘을 변형해서 사용한다. 파이썬으로 구현한 알고리즘은 다음과 같다.

```python
# Get smallest non-negative integer s.t L <= Ax mod P <= R
# Constraint 1: 0 < A < P and 0 <= L <= R < P
# Constraint 2: L != 0 and (L - 1) // g < R // g, where g = gcd(A, P)
def get_smallest(P, A, L, R):
    if L == 0:
        return 0
    if 2 * A > P:
        L, R = R, L
        A = P - A
        L = P - L
        R = P - R
    t = (L + A - 1) // A
    if t * A <= R:
        return t
    y = get_smallest(A, A - P % A, L % A, R % A)
    return (L + P * y + A - 1) // A
```

위 코드에 다음 값들을 대입해 d_L 의 값을 계산할 수 있다.

$$P = n$$
$$A = \alpha,$$
$$L = n - \beta$$
$$R = 2^{128} - \beta + n$$

결과로 얻은 d_L 및 d_H 값은 다음과 같고, 두 값 모두 $[0, 2^{128})$ 범위에 포함되므로 성공적으로 키를 복원했음을 알 수 있다.

$d_H =$ `1D 08 A3 13 05 E2 40 E0 AD D3 DF 29 58 06 3A D6`
$d_L =$ `31 60 93 0D 17 C1 3A F0 8F 72 03 8F 13 E0 20 78`

최종적인 영수의 개인키는 다음과 같다.

`1D 08 A3 13 05 E2 40 E0 AD D3 DF 29 58 06 3A D6 31 60 93 0D 17 C1 3A F0 8F 72 03 8F 13 E0 20 78`

3 소스 코드 설명

타원곡선 라이브러리로 tinyec 를 사용했다. 다음과 명령으로 설치 가능하다.

```
pip install tinyec
```

취약점 분석 코드 `find_weakness.py` 에 전자서명 프로그램의 취약점을 파악하는 코드가 구현되어 있다. 실행 방법 및 결과는 다음과 같다.

```
$ python find_weakness.py
ECDSA info of Cheolsoo
hex(d)='0xbde07e98f0437a531c014a1fe6fd69c2cfb6c3657072696e7432303233383431'
hex(e)='0x9a2e62818ad55aeb8ac319820b2d595660b9af57c0c7123bd6c6dfde2d9a1753'
hex(k)='0x9a2e62818ad55aeb8ac319820b2d5956bde07e98f0437a531c014a1fe6fd69c2'
```

취약점을 이용한 영수의 개인키 복원 코드 `find_secret_key.py` 에 취약점을 이용해 영수의 개인키를 복원하는 코드가 구현되어 있다. 실행 방법 및 결과는 다음과 같다.

```
$ python find_secret_key.py
hex(eH)='0x568b4901cc2dac4a13af161bbf6b2087'
hex(eL)='0xc94d8b8223755fd121ec6aa0519ecee2'
hex(alpha)='0x37974e35a9f0ccc7a0369e55db728e9b7c4426914b226c53376f5ec40c102794'
hex(beta)='0x497c820649462c2d3f794d6d57735c9ef44f72d2b844250ff98f640c7d0baec6'
hex(dH)='0x1d08a31305e240e0add3df2958063ad6'
hex(dL)='0x3160930d17c13af08f72038f13e02078'
Secret key d of Yeongsoo: 0x1d08a31305e240e0add3df2958063ad63160930d17c13af08f72038f13e02078
```

복원한 영수의 개인키 검증 코드 `validate.py` 에 복원한 영수의 개인키를 이용해 전자서명을 다시 수행해 키를 검증하는 코드가 구현되어 있다. 실행 방법 및 결과는 다음과 같다.

```
$ python validate.py
Correct secret key: 0x1d08a31305e240e0add3df2958063ad63160930d17c13af08f72038f13e02078
```