첨부파일 개요

- `half_half.sage`: 영수의 개인키 복구 공격 코드

1 목표

본 문제의 목표는 제시된 ECDSA 서명 생성 알고리즘의 취약점을 파악하여 영수의 비밀키를 복원하는 것이다.

2 취약점

서명 알고리즘의 취약점을 찾기 위해, 제공된 철수의 데이터를 분석한다.

$$k = s^{-1}(h + rd) \mod n$$

위 식을 사용해 k 값을 계산한다. 철수의 데이터 중 개인키 d, 메시지 해시값 h, nonce k 값은 다음과 같다.

```
d = 0xbde07e98f0437a531c014a1fe6fd69c2cfb6c3657072696e7432303233383431
h = 0x9a2e62818ad55aeb8ac319820b2d595660b9af57c0c7123bd6c6dfde2d9a1753
k = 0x9a2e62818ad55aeb8ac319820b2d5956bde07e98f0437a531c014a1fe6fd69c2
```

위 데이터에서 다음 식이 성립함을 볼 수 있다. 아래첨자 m는 상위 128비트, 아래첨자 l 는 하위 128비트를 의미하며, 이후 수식에서도 동일한 표기를 사용한다.

$$k = 2^{128}h_m + d_m$$

일반적인 ECDSA에서 nonce k를 무작위 선택하는 것과 달리, 문제에 제시된 알고리즘은 위 등식이 성립하는 k 값을 사용한다. 이 취약점은 아래에서 설명할 공격으로 이어진다.

3 공격

$$\begin{aligned}
k &= s^{-1}(h + rd) \mod n \\
2^{128}h_m + d_m &= s^{-1}(h + r(2^{128}d_m + d_l)) \mod n \\
2^{128}sh_m + sd_m &= h + 2^{128}rd_m + rd_l \mod n \\
(2^{128}r - s)d_m + rd_l + (h - 2^{128}sh_m) &= 0 \mod n \\
\underbrace{(2^{128} - sr^{-1})}_{A} d_m + d_l + \underbrace{r^{-1}(h - 2^{128}sh_m)}_{b} &= 0 \mod n
\end{aligned}$$

미지수 d의 상, 하위 128비트를 기준으로 식을 변형하면 위와 같이 전개된다. 위 식을 행렬로 나타낸 뒤, Lenstra–Lenstra–Lovász lattice basis reduction algorithm (LLL reduction)을 적용하여 d를 복구하는 공격 기법을 아래에 설명한다.

$$\begin{aligned}
Ad_m + d_l + b &= 0 \mod n \\
Ad_m + d_l + b &= cn \\
d_l &= cn - Ad_m - b
\end{aligned}$$

$$v_1 = (n, 0, 0), v_2 = (A, -1, 0), v_3 = (-b, 0, X)$$
$$(n, 0, 0) \cdot c + (A, -1, 0) \cdot (-d_m) + (-b, 0, X) \cdot 1 = (cn - Ad_m - b, d_m, X) = (d_l, d_m, X)$$

d_m, d_l을 복구할 수 있도록 (v_1, v_2, v_3)으로 Lattice를 구성한다. X는 LLL reduction 이후에도 값이 유지될 수 있도록 충분히 큰 수를 사용한다. 본 공격 시나리오에서는 $d_m, d_l < 2^{128}$ 이기 때문에 2^{128} 이상의 수를 사용하는 것이 안정적이지만, 실험에서는 $X = 2^{121}$ 으로 LLL reduction을 수행해도 키 복구를 성공할 수 있었다.

위 Lattice를 행렬 형태로 나타내면 아래와 같다.

$$M = \begin{bmatrix} n & 0 & 0 \\ A & -1 & 0 \\ -b & 0 & X \end{bmatrix}$$

행렬 M에 대해 LLL reduction을 적용하여 얻은 벡터 중 마지막 원소가 X인 벡터를 v라고 하자. v가 X값을 그대로 유지하면서 가장 작은 길이가 되기 위해서는 $v_1^2 + v_2^2$ 값이 최소가 되어야 하고, 행렬의 구성에 의해 $v_1 = d_l, v_2 = d_m$을 만족하게 된다.

4 결과

위 공격 기법을 적용하여 영수의 비밀키를 복구한 결과는 다음과 같다. (전체 공격 코드: `half_half.sage`)

```
[+] private key found
d = 0x1d08a31305e240e0add3df2958063ad63160930d17c13af08f72038f13e02078
h = 0x568b4901cc2dac4a13af161bbf6b2087c94d8b8223755fd121ec6aa0519ecee2
k = 0x568b4901cc2dac4a13af161bbf6b20871d08a31305e240e0add3df2958063ad6
```

제시한 취약점 ($k = 2^{128}h_m + d_m$) 이 복구된 영수의 데이터에서도 드러남을 확인할 수 있다.

5 참고 문헌

[1] The curious case of the half-half Bitcoin ECDSA nonces, https://eprint.iacr.org/2023/841