Forging Certificates

by Fabio Plunser & Cedric Sillaber

Case 1: Nonce Reuse Attack

```
nonce = random.randint(2, sig_scheme.q - 1)

cert1 = gen_certificate("lucacampa", "https://project3topic1.com", "01/01/2026")
cert1 = sign_certificate_v1(sig_scheme, cert1, nonce)

cert2 = gen_certificate("lucacampa", "https://project2topic2.com", "01/01/2027")
cert2 = sign_certificate_v1(sig_scheme, cert2, nonce)
```

Problem: Nonce k is reused → $r_1 = r_2 = r$

Case 1: Mathematical Attack

Given: Two signatures $(r, s_1), (r, s_2)$ with same nonce k

Setup:

- $s_1 = k^{-1}(H(m_1) + x_a \cdot r) \mod q$
- $s_2 = k^{-1}(H(m_2) + x_a \cdot r) \mod q$

Eliminate k:

- $k = (H(m_1) + x_a \cdot r)/s_1 = (H(m_2) + x_a \cdot r)/s_2$

Solve for x_a:

$x_a = \frac{s_2 H(m_1) - s_1 H(m_2)}{r(s_1 - s_2)} \mod q$

Case 2: Predictable Nonces

Vulnerability: $k_2 = 3k_1 + 5 \mod q$

Attack Strategy:

1. Extract relationship between nonces
2. Set up system of equations using both signatures
3. Eliminate both k_1 and k_2 to solve for x_a

Result:

$$x_a = \frac{s_1 H(m_2) - 5 s_2 s_1 - 3 s_2 H(m_1)}{3 s_2 r_1 - r_2 s_1} \mod q$$

Case 3: Reversible Hash Function

Vulnerability: Hash function uses AES-ECB with known IV

Attack Strategy:

1. **Choose random u_1, u_2**
2. **Compute valid signature**:
   - $r' = (g^{u_1} \cdot y^{u_2} \mod p) \mod q$
   - $s' = (r' \cdot u_2^{-1}) \mod q$
3. **Find matching message**:
   - target_hash = $u_1 \cdot s' \mod q$
   - Decrypt with known IV to get forged message

Key insight: We control the hash, not the message!

Case 3: Implementation

```
while forged_message is None:
    u1 = random.randint(1, q - 1)
    u2 = random.randint(1, q - 1)

    # Compute signature components
    r_new = (pow(g, u1, p) * pow(y, u2, p) % p) % q
    s_new = (r_new * pow(u2, -1, q)) % q

    # Find message that produces this signature
    target_hash = (u1 * s_new) % q
    target_hash_bytes = long_to_bytes(target_hash, AES.block_size)

    try:
        # most of the retries happen here
        forged_message = AES.new(IV, AES.MODE_ECB).decrypt(target_hash_bytes)
        forged_message = unpad(forged_message, AES.block_size)
    except ValueError:
        continue
```