CSCI-762 Assignment 7

Tom Arnold <tca4384@rit.edu>

1. (8.6) ElGamal signature variant

a. Describe signature verification

Let's work from the δ equation backwards, the reverse of what is done in the book in 8.3.

$$
\begin{aligned}
&\delta \equiv (x - k\gamma) \times a^{-1} \pmod{p - 1}\\
&\delta a \equiv (x - k\gamma) \pmod{p - 1}\\
&\delta a + k\gamma \equiv x \pmod{p - 1}\\
&\alpha^{\delta a + k\gamma} \equiv \alpha^{x} \pmod{p}\\
&\alpha^{x} \equiv \alpha^{\delta a} + \alpha^{k\gamma} \pmod{p}\\
\\
&ver(x,(\gamma,\delta)) = \alpha^{x} \equiv \beta^{\delta}\gamma^{\gamma} \pmod{p}
\end{aligned}
$$

Let's check our work using the example from the book.

$$
\begin{aligned}
p &= 467\\
\alpha &= 2\\
a &= 127\\
\beta &= 132\\
x &= 100\\
k &= 213\\
\gamma &= 29\\
\\
\delta &= (x - k\gamma) \times a^{-1} \bmod 466\\
&= (100 - 213\times29) \times 455 \bmod 466\\
&= (100 - 119) \ldots\\
&= 447 \times 455 \bmod 466\\
&= 209
\end{aligned}
$$

Now using the equation we derived we can verify the signature:

$$
\begin{aligned}
&\alpha^{x} \equiv \beta^{\delta}\gamma^{\gamma} \pmod{p} \\
&2^{100} \equiv 132^{209} \times 29^{29} \pmod{467} \\
&189 \equiv 189 \checkmark
\end{aligned}
$$

b. Describe computational advantage of scheme

One computational advantage of this scheme is that a^{-1} can be precomputed for faster signing; this is not true for k^{-1} since a new `k` must be chosen for each message.

c. Compare security with original

ElGamal is broken if the nonce is reused; however with this scheme it is even more broken.

Let's start with two messages signed with the same key and (incorrectly) the same nonce.

$$
\begin{aligned}
&(x1,(\gamma 1,\delta 1)) \\
&(x2,(\gamma 2,\delta 2))
\end{aligned}
$$

`k` is constant because it was reused, α is part of the public key so it's also constant, therefore `γ` is constant because $\gamma = \alpha^{k}$.

$$
\begin{aligned}
&\alpha^{x1} \equiv \beta^{\delta 1}\gamma^{\gamma} \pmod{p} \\
&\alpha^{x2} \equiv \beta^{\delta 2}\gamma^{\gamma} \pmod{p}
\end{aligned}
$$

therefore

$$
\begin{aligned}
&\alpha^{x1-x2} \equiv \beta^{\delta 1-\delta 2} \pmod{p} \\
&\alpha^{x1-x2} \equiv (\alpha^{a})^{\delta 1-\delta 2} \pmod{p} \qquad \text{[by definition]} \\
&x1 - x2 \equiv a \times (\delta 1 - \delta 2) \pmod{p - 1}
\end{aligned}
$$

If `gcd(δ1 - δ2, p - 1) = 1` then we can solve for `a`.

```
a = (x1 - x2) × (δ1 - δ2)⁻¹ (mod p - 1)
```

Here's a quick example of this using numbers from one of the examples in the book.

```
p = 467
α = 2
β = 132

a = 127 [secret]
k = 217 [secret, accidentally constant]

γ = 2²¹⁷ mod 467 = 464 [constant because k constant]

x1 = 100 [first message]
x2 = 137 [second message]

δ1 = (100 - 217 × 464) × 455 mod 466 = 184
δ2 = (137 - 217 × 464) × 455 mod 466 = 243
```

Now solve for private key.

```
a = (100 - 137) × (184 - 243)⁻¹ mod 466
  = 127 ✓
```

2. (8.7) DSA

```
q = 101
p = 7879
α = 170
a = 75 [secret]
β = 4567

SHA3-224(x) = 52
k = 49
```

Let's compute the signature:

```
γ = (170^49 mod 7879) mod 101
  = 59
δ = (52 + 75 × 59) × 33
  = 79
```

Now let's verify the signature:

```
e1 = 52 × 78 mod 101
   = 16
e2 = 59 × 78 mod 101
   = 57
59 = (170^16 × 4567^57 mod 7879) mod 101
   = 59 ✓
```

3. (8.10) DSA & ECDSA Forgeries

Suppose x0 is a bitstring such that SHA3-224(x0) is 0.

a. DSA Forgery

```
δ = γ [hint]
γ = (αᵏ mod p) mod q
δ = aγ × k⁻¹ mod q [because SHA(x) = 0]
```

Observe that if `k` is chosen such that it is coprime with `p` and `q` then `γ = 0` and thus `δ = k⁻¹`. ✓

b. ECDSA Forgery

```
k × A = (u,v)
r = u mod q
s = k⁻¹ × m × r mod q [because SHA(x) = 0]
```

Observe that if `r = 0` then `s = 0`. ✓

4. (8.14) ECDSA

Let `E: y² ≡ x³ + x + 26 mod 127`, and note that `#E = 131` which is prime. Suppose ECDSA is implemented with `A = (2,6)` and `m = 54`.

a.

We'll reuse some Maxima functions from a previous assignment to compute `B = mA`.

```maxima
/* Perform point addition on a curve with points P, Q, coefficient A, and modulus N.
*/
pt_add(p, q, a, n) := block([x1,y1,x2,y2,x3,y3,slope],
  if q = inf then return (p),
  if p = inf then return (q),
  x1: p[1],
  y1: p[2],
  x2: q[1],
  y2: q[2],

  if x1 # x2 then (
    /* Case 1 */
    slope: mod(mod(y2 - y1, n) * inv_mod(x2 - x1, n), n),
    x3: mod(mod(slope^2 - x1, n) - x2, n),
    y3: mod(mod(slope * mod(x1 - x3, n), n) - y1, n),
    [x3, y3]
  ) else if y1 = -y2 then
    /* Case 2 */
    inf
  else (
    /* Case 3 */
    slope: mod((3*x1^2 + a) * inv_mod(2 * y1, n), n),
    x3: mod(slope^2 - 2*x1, n),
    y3: mod(slope * (x1 - x3) - y1, n),
    [x3, y3]
  )
)$

/* Compute Ith multiple of P for coefficient A and modulus N. */
pt_exp(p, i, a, n) := block([q],
    /* Start with Q = P, then compute Q' = P + Q and so on... */
    q: p,
    for j: 1 step 1 unless j >= i do (
       q: pt_add(p, q, a, n)
    ),
    q
)$
```

```maxima
pt_exp([2,6], 54, 1, 127);
```

```
[24,44]
```

b.

Next we’ll compute the signature, where `SHA(x) = 10` and `k = 75`.

```
[u,v] = 75[2,6]
      = [88,55]
r = 88 mod 131
  = 88
s = 7 × (10 + 54 × 88) mod 131
  = 60
```

c.

Finally we’ll verify the signature from the previous step.

```
w = 107
i = 107 × 10 mod 131
  = 22
j = 107 × 88 mod 131
  = 115
[u,v] = 22[2,6] + 115[24,44]
      = [91,18] + [18,62]
      = [88,55]
u mod q = r
88 mod 131 = 88 ✓
```

Last updated 2020-04-18 23:17:09 -0400