공개키 암호의 구현

Part 2.Ep 4: RSA 구현
YouTube: https://youtu.be/-2Dm9wC9SFc
Git: https://github.com/minpie/CryptoCraftLab-minpie_public

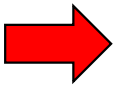
한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

발표 계획 목록

RSA C언어 구현

발표 계획: 24.07.19ver

- Part 1. 대칭키 암호 단일블록 C언어 구현
  - Ep1. AES
  - Ep2. DES
- Part 2. 64비트 이상 키 길이의 공개키 암호 C언어 구현
  - Ep3. GMP 라이브러리
  - Today  Ep4. RSA 구현
  - Ep5. Rabin 구현
  - Ep6. Elgamal 구현
  - Ep7. ECDSA 구현
- Part 3. AES-운영모드 with 병렬컴퓨팅
  - Ep8. OpenMPI 라이브러리
  - Ep9. OpenMPI-AES
  - Ep10. CUDA C
  - Ep11. CUDA-AES

RSA C언어 구현 - 개요

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R.L. Rivest, A. Shamir, and L. Adleman*

- 기본적인 수준으로 RSA를 구현.
- 기반문제: 인수분해 문제
- 권장 키 길이: 2048비트

RSA C언어 구현 – 전체 흐름

```
137  int main(void)
138  {
139      // start:
140      mpz_t p, q, n, e, d;
141      mpz_t plain, cipher, plain2;
142      mpz_inits(p, q, n, e, d, NULL);
143      mpz_inits(plain, cipher, plain2, NULL);
144
145      // input:
146      mpz_set_si(p, 1);
147      mpz_set_si(q, 1);
148      mpz_set_si(plain, 1);
149
150      // get n, e, d
151
152      mpz_mul(n, p, q); // n = p = q;
153      KeyGeneration(e, d, p, q);
154
155      gmp_printf("Public Key (n, e)=(%Zd, %Zd)\n", n, e);
156      gmp_printf("Private Key (d)=(%Zd)\n", d);
157
158      Encryption(cipher, plain, n, e);
159      Decryption(plain2, cipher, n, d);
160
161      gmp_printf("Encryption: P=%Zd, C=%Zd\n", plain, cipher);
162      gmp_printf("Decryption: C=%Zd, P=%Zd\n", cipher, plain2);
163
164      // end:
165      mpz_clears(p, q, n, e, d, NULL);
166      mpz_clears(plain, cipher, plain2, NULL);
167      return 0;
168  }
```

- 키 생성-암호화-복호화 과정을 위한 main() 코드

RSA C언어 구현 – KeyGeneration()

```
84  void KeyGeneration(mpz_t e, mpz_t d, mpz_t p, mpz_t q)
85  {
86      mpz_t n, phi_n;
87      mpz_t i;
88      mpz_t tmp1, tmp2;
89      mpz_inits(n, phi_n, NULL);
90      mpz_inits(i, NULL);
91      mpz_set_si(i, 2);
92      mpz_inits(tmp1, tmp2, NULL);
93
94      // get n , phi(n)
95      mpz_mul(n, p, q);           // n = p * q;
96      mpz_sub_ui(tmp1, p, 1);     // tmp1 = p - 1
97      mpz_sub_ui(tmp2, q, 1);     // tmp2 = q - 1
98      mpz_mul(phi_n, tmp1, tmp2); // phi_n = tmp1 * tmp2 = (p - 1) * (q - 1)
99      printf("Done: Get n, phi_n\n");
100
101     // get e
102     mpz_set_si(tmp1, 0);
103     mpz_set_si(tmp2, 0);
104     while (mpz_cmp(phi_n, i))
105     {
106         // loop 조건: i < phi_n
107         GetGCD(tmp1, phi_n, i); // tmp1 = gcd(phi_n, i)
108         if (!mpz_cmp_si(tmp1, 1))
109         {
110             // gcd(phi_n, i) == 1
111             mpz_set(e, i); // e = i
112         }
113         mpz_add_ui(i, i, 1); // i++
114     }
115     printf("Done: Get e\n");
```

```
117     // get d
118     GetModularMultiplicativeInverse(d, phi_n, e);
119     printf("Done: Get d\n");
120
121     // end:
122     mpz_clears(n, phi_n, NULL);
123     mpz_clears(i, NULL);
124     mpz_clears(tmp1, tmp2, NULL);
125 }
```

- 공개키 및 개인키를 생성하는 함수

RSA C언어 구현 – GetGCD()

```
31  // a, b의 GCD(최대공약수) 구하는 함수
32  void GetGCD(mpz_t result, mpz_t a, mpz_t b)
33  {
34      // 유클리드 알고리즘 사용
35      mpz_t r, r1, r2, q, tmp1;
36      mpz_inits(r, r1, r2, q, tmp1, NULL);
37      mpz_set(r1, a);
38      mpz_set(r2, b);
39      while (mpz_cmp_si(r2, 0) > 0)
40      {
41          mpz_fdiv_q(q, r1, r2); // q = r1 / r2
42          mpz_mul(tmp1, q, r2);  // tmp1 = q * r2
43          mpz_sub(r, r1, tmp1);  // r = r1 - tmp1 = r1 - (q * r2)
44          mpz_set(r1, r2);       // r1 = r2
45          mpz_set(r2, r);        // r2 = r
46      }
47
48      mpz_set(result, r1); // result = r1
49      mpz_clears(r, r1, r2, q, tmp1, NULL);
50  }
```

- 최대공약수를 구하는 함수
- 유클리드 알고리즘을 사용

RSA C언어 구현 – GetModularMultiplicativeInverse()

```
52  // 모듈러 곱셈 역 구하기
53  void GetModularMultiplicativeInverse(mpz_t a_1, mpz_t n, mpz_t a)
54  {
55      // 확장 유클리드 알고리즘 사용
56      mpz_t q, r1, r2, r, t, t1, t2, tmp1, tmp2;
57      mpz_inits(q, r1, r2, r, t, t1, t2, tmp1, tmp2, NULL);
58      mpz_set(r1, n);    // r1 = n;
59      mpz_set(r2, a);    // r2 = a
60      mpz_set_si(t1, 0); // t1 = 0
61      mpz_set_si(t2, 1); // t2 = 1
62
63      while (mpz_cmp_si(r2, 0))
64      {
65          mpz_fdiv_q(q, r1, r2); // q = r1 / r2
66          mpz_mul(tmp1, q, r2);  // tmp1 = q * r2
67          mpz_sub(r, r1, tmp1);  // r = r1 - tmp1 = r1 - (q * r2)
68          mpz_set(r1, r2);       // r1 = r2
69          mpz_set(r2, r);        // r2 = r
70
71          mpz_mul(tmp2, q, t2); // tmp2 = q * t2
72          mpz_sub(t, t1, tmp2); // t = t1 - tmp2 = t1 - (q * t2)
73          mpz_set(t1, t2);      // t1 = t2
74          mpz_set(t2, t);       // t2 = t
75      }
76      mpz_set(a_1, t1);
77      if (mpz_sgn(a_1) < 0)
78      {
79          mpz_add(a_1, n, a_1);
80      }
81      mpz_clears(q, r1, r2, r, t, t1, t2, tmp1, tmp2, NULL);
82  }
```

- 모듈로 곱 역원을 구하는 함수
- 확장 유클리드 알고리즘 사용

RSA C언어 구현 – Encryption(), Decryption()

```
127  void Encryption(mpz_t cipher, mpz_t plain, mpz_t n, mpz_t e)
128  {
129      FastModuloExponentiation(cipher, plain, e, n);
130  }

132  void Decryption(mpz_t plain, mpz_t cipher, mpz_t n, mpz_t d)
133  {
134      FastModuloExponentiation(plain, cipher, d, n);
135  }
```

- 암호화 및 복호화 연산.

RSA C언어 구현 – FastModuloExponentation()

```
4   // 고속 모듈러 지수연산
5   void FastModuloExponentiation(mpz_t result, mpz_t a, mpz_t x, mpz_t n)
6   {
7       mp_bitcnt_t bits_x, i;
8       mpz_t tmp_a, y;
9       mpz_inits(tmp_a, y, NULL);
10      mpz_set(tmp_a, a);
11      mpz_set_ui(y, 1); // y = 1
12      i = 0;
13      bits_x = mpz_sizeinbase(x, 2);
14
15      while (bits_x > i)
16      {
17          if (mpz_tstbit(x, i))
18          {
19              // x의 i번째 비트가 1이면:
20              mpz_mul(y, tmp_a, y); // y = tmp_a * y
21              mpz_mod(y, y, n);     // y = y mod n
22          }
23          mpz_mul(tmp_a, tmp_a, tmp_a); // tmp_a = tmp_a * tmp_a
24          mpz_mod(tmp_a, tmp_a, n);     // tmp_a = tmp_a mod n
25          i = (mp_bitcnt_t)(i + 1);     // i++
26      }
27      mpz_set(result, y);
28      mpz_clears(tmp_a, y, NULL);
29  }
```

- 모듈로 거듭제곱 함수
- 제곱-곱 방법을 이용

RSA C언어 구현 – 어려웠던 부분

- GMP 라이브러리 설치
- KeyGeneration() – 복호화 지수 d 구하기
- 작동 검증

RSA C언어 구현 – 참고문헌

Q & A