Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Лабораторная работа №4

Ассиметричная криптография. Алгоритм Эль-Гамаля

Выполнил:
студент гр. 653503
Лисковец Б. Н.

Проверил:
В. С. Артемьев

Минск, 2019

Введение

Схема Эль-Гамаля (Elgamal) — криптосистема с открытым ключом, основанная на трудности вычисления дискретных логарифмов в конечном поле. Криптосистема включает в себя алгоритм шифрования и алгоритм цифровой подписи. Схема Эль-Гамаля лежит в основе бывших стандартов электронной цифровой подписи в США (DSA) и России (ГОСТ Р 34.10-94).

Схема была предложена Тахером Эль-Гамалем в 1985 году. Эль-Гамаль разработал один из вариантов алгоритма Диффи-Хеллмана. Он усовершенствовал систему Диффи-Хеллмана и получил два алгоритма, которые использовались для шифрования и для обеспечения аутентификации. В отличие от RSA алгоритм Эль-Гамаля не был запатентован и, поэтому, стал более дешевой альтернативой, так как не требовалась оплата взносов за лицензию. Считается, что алгоритм попадает под действие патента Диффи-Хеллмана.

Задание: Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма Эль-Гамаля

Теоретическая часть

Генерация ключей

1. Генерируется случайное простое число p.
2. Выбирается целое число g — первообразный корень p.
3. Выбирается случайное целое число x такое, что $1 < x < p - 1$.
4. Вычисляется $y = g^x \mathrm{mod} p$.
5. Открытым ключом является y, закрытым ключом — число x.

Шифрование

Сообщение M должно быть меньше числа p. Сообщение шифруется следующим образом:

1. Выбирается сессионный ключ — случайное целое число k такое, что $1 < k < p - 1$.
2. Вычисляются числа $a = g^k \mathrm{mod} p$ и $b = y^k M \mathrm{mod} p$.
3. Пара чисел (a, b) является шифротекстом.

Нетрудно увидеть, что длина шифротекста в схеме Эль-Гамаля длиннее исходного сообщения M вдвое.

Расшифрование

Зная закрытый ключ x, исходное сообщение можно вычислить из шифротекста (a, b) по формуле:

$$M = b(a^x)^{-1} \mathrm{mod} p. \tag{1}$$

При этом нетрудно проверить, что

$$(a^x)^{-1} = g^{-kx} \mathrm{mod} p \tag{2}$$

и поэтому

$$b(a^x)^{-1} = (y^k M)g^{-xk} \equiv (g^{xk} M)g^{-xk} \equiv M \pmod{p} \tag{3}$$

Для практических вычислений больше подходит следующая формула:

$$M = b(a^x)^{-1} = ba^{(p-1-x)} \pmod{p} \tag{4}$$

Практическая часть

Для работы с программой простое число, необходимое для генерации ключей, а также задаётся сообщение для шифрования/расшифрования.

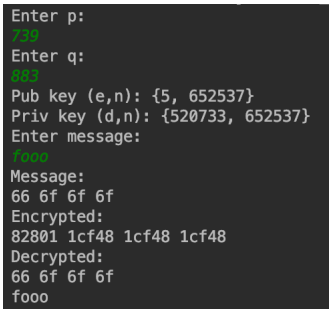

Рис. 1: Демонстрация работы программы

Код программы

```
u64 gcd(u64 a, u64 b) {
  if (a < b) {
    u64 t = a;
    a = b;
    b = t;
  }

  while (true) {
    u64 r = a % b;
    if (r == 0)
      return b;
    a = b;
    b = r;
  }
}

bool is_prime(u64 n) {
  u64 root = static_cast<u64>(floor(sqrt(static_cast<long double>(n))));

  for (u64 i = 2; i <= root; ++i) {
    if (n % i == 0)
      return false;
  }

  return true;
}

u64 powmod(u64 a, u64 b, u64 p) {
  u64 res = 1;
  while (b)
    if (b & 1)
      res = u64(res * a % p), --b;
    else
      a = u64(a * a % p), b >>= 1;
  return res;
}

u64 primitive_root(u64 p) {
  std::vector<u64> fact;
  u64 phi = p - 1, n = phi;
  for (u64 i = 2; i * i <= n; ++i)
    if (n % i == 0) {
      fact.push_back(i);
      while (n % i == 0)
        n /= i;
    }
  if (n > 1)
    fact.push_back(n);
```

```
  for (u64 res = 2; res <= p; ++res) {
    bool ok = true;
    for (std::size_t i = 0; i < fact.size() && ok; ++i)
      ok &= powmod(res, phi / fact[i], p) != 1;
    if (ok)
      return res;
  }
  return -1;
}

u64 get_rand_coprime(u64 p) {
  assert(p - 2 > 1);

  // u64 x = p - 2;
  // while (x > 1) {
  // if (gcd(p - 1, x) == 1) {
  // return x;
  // }
  // x--;
  // }

  while (true) {
    u64 x = 1 + u64(std::rand() / double(p - 2));
    if (gcd(p - 1, x) == 1) {
      return x;
    }
  }

  return -1;
}

std::pair<u64, std::vector<u64>> encrypt(const std::vector<u64> &msg, u64 g,
                                         u64 p, u64 y) {
  std::pair<u64, std::vector<u64>> enc(0, msg.size());

  u64 k = get_rand_coprime(p);
  enc.first = powmod(g, k, p);

  for (std::size_t i = 0; i < msg.size(); ++i) {
    enc.second[i] = (msg[i] * powmod(y, k, p)) % p;
  }
  return enc;
}

std::vector<u64> decrypt(u64 gamma, const std::vector<u64> &enc, u64 p, u64 x
  std::vector<u64> dec(enc.size());
  for (std::size_t i = 0; i < enc.size(); ++i) {
    dec[i] = (enc[i] * powmod(gamma, p - 1 - x, p)) % p;
  }
  return dec;
}
```

```
int main() {
  std::srand(std::time(nullptr));

  u64 p = prompt_prime("p");
  // u64 p = 65537;
  u64 g = primitive_root(p);
  u64 x = get_rand_coprime(p);
  u64 y = powmod(g, x, p);

  std::cout << "Priv key: " << x << std::endl;
  std::cout << "Pub key (g,p,y): {" << g << ", " << p << ", " << y << "}"
            << std::endl;

  auto msg = prompt_msg();
  std::cout << "Message:" << std::endl;

  auto enc = encrypt(msg, g, p, y);
  std::cout << "Encrypted:" << std::endl;
  print_vector_uint(enc.second);

  auto dec = decrypt(enc.first, enc.second, p, x);
  std::cout << "Decrypted:" << std::endl;
  print_vector_uint(dec);
  std::cout << std::string(dec.begin(), dec.end()) << std::endl;

  return 0;
}
```