МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Схемы ЭЦП

ОТЧЁТ
ПО ДИСЦИПЛИНЕ
«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Бородина Артёма Горовича

Преподаватель
аспирант _____ Р. А. Фарахутдинов
подпись, дата

Саратов 2023

1 Постановка задачи

Необходимо реализовать схему подписи Фиата-Шамира.

2 Теоретические сведения

Превращение схемы идентификации Фиата-Шамира в схему подписи – это, по сути, вопрос, замены Виктора (проверяющей стороны) на хэш-функцию. Главным преимуществом схемы цифровой подписи Фиата-Шамира по сравнению с RSA является её скорость: для схемы Фиата-Шамира требуется всего лишь от 1 до 4 процентов модульных умножений, используемых в RSA.

Смысл переменных – такой же, как и в схеме идентификации. Выбирается n – произведение двух больших простых чисел. Генерируется открытый ключ $v_1, v_2, \dots, v_k$, и закрытый ключ $s_1, s_2, \dots, s_k$, где $s_i \equiv \sqrt{v_i^{-1}}\ (mod\ n)$.

Схема подписи Фиата-Шамира.

1. Алиса выбирает t случайных целых чисел в диапазоне от 1 до n – $r_1, r_2, \dots, r_t$ – и вычисляет $x_1, x_2, \dots, x_t$, такие, что $x_i = r_i^2\ (mod\ n)$.
2. Алиса хэширует конкатенацию сообщения и строки x_i, создавая битовый поток: $H(m, x_1, x_2, \dots, x_t)$. Она использует первые $k * t$ битов этой строки в качестве значений b_{ij}, где i пробегает от 1 до t, а j от 1 до k.
3. Алиса вычисляет $y_1, y_2, \dots, y_t$, где $y_i = r_i * \left(s_1^{b_{i1}} * s_2^{b_{i2}} * \dots * s_k^{b_{ik}}\right)\ (mod\ n)$ (для каждого i она перемножает вместе значения s_i, в зависимости от случайных значений b_{ij}. Если $b_{ij} = 1$, то s_i участвует в вычислениях, если $b_{ij} = 0$, то нет).
4. Алиса посылает Бобу m, все биты b_{ij}, и все значения y_i. У Боба уже есть открытый ключ Алисы: $v_1, v_2, \dots, v_k$.

5. Боб вычисляет $z_1, z_2, \dots, z_t$, где $z_i = y^2 * \left(v_1^{bi1} * v_2^{b_{i2}} * \dots * v_k^{b_{ik}}\right) (mod\ n)$ (Боб выполняет умножение в зависимости от значений b_{ij}. Также z_i должно быть равно x_i).
6. Боб проверяет, что первые $k * t$ битов $H(m, z_1, z_2, \dots, z_t)$ – это значения b_{ij}, которые прислала ему Алиса.

Как и в схеме идентификации безопасность схемы подписи пропорциональна $\frac{1}{2^{kt}}$. Она также зависит от сложности разложения n на множители.

3 Практическая реализация

3.1 Описание программы

Язык программной реализации – Common Lisp. Программа реализует четыре основные функции – функцию определения используемых параметров, функцию выпуска смарт-карты, функцию подписания сообщения и функцию проверки подписи. Каждая функция разбита на этапы, описываемые в процессе выполнения. Для удобства все значения вычислений представлены в шестнадцатеричной системе счисления.

3.2 Результаты тестирования программы

Рассмотрим процесс выполнения программы для введённой битовой длины l модуля простого числа, равной 128.

```
FSDS> (setup)

[0] -- Перед выпуском смарт-карт центр выбирает значение модуля n и псевдослучайную
       функцию f, отображающую строку произвольной длины в целое число из диапазона [0; n).

Введите битовую длину l модуля (l = 2^m, l > 16) (по умолчанию 1024): 128

    p = 0xAFA526F7A739F3D3;
    q = 0x884620D0228009BD;
    n = 0x5D7FDA5FAB48E9EEBC9844836EDB6DC7.
```

Рисунок 1 – Определение чисел p и q на этапе определения параметров

```
Введите номер хеш-функции, используемой в псевдослучайном отображении f (по умолчанию 7 -- SHA256):

    [1]: BLAKE2/256; [ 2]: GROESTL/256; [ 3]:       JH/256; [ 4]: KECCAK/256;
    [5]:        MD5; [ 6]:  RIPEMD-160; [ 7]:       SHA256; [ 8]:     SHA384;
    [9]:     SHA512; [10]:   SKEIN1024; [11]: STREEBOG/256; [12]:      TIGER.

Ваш выбор:

Псевдослучайной функцией f назначена функция: SHA256(string) (mod n).
```

Рисунок 2 – Выбор псевдослучайной функции на этапе определения параметров

```
FSDS> (issue-a-smartcard)

Введите имя пользователя (I), для которого выпускается смарт-карта (по умолчанию Alice):

Введите количество значений v_j (9 < j < 51, по умолчанию 20):

[1] -- Вычислить значения v_j = f(I, j) для небольших значений j.

    v_ 1 = 0x4956FD415C1865C06F17D2B67A06D8F5;
    v_ 2 = 0x2C7415C800809E0D1F79D345A9FDAF1F;
    v_ 3 = 0x1917E766B8CF5C49E6371E54BBFAB0A9;
    v_ 4 = 0x368BE6360EDEC72EE147B72A959C149B;
    v_ 5 = 0x3D96961C0B7D2ECB385ADD965C17AF6C;
    v_ 6 = 0x3AA58504B3EFDD5333A03BD45059758B;
    v_ 7 = 0x2F637BBA3BF8A2CE7566DABD2F1B0897;
    v_ 8 = 0x331D2F446930E1845E20F7EAE2F9BCD6;
    v_ 9 = 0x1ED26F958AB4610E9EF7B5BBAE390668;
    v_10 = 0x3FBAFEBD4389FCBE98C8EDE40E1CDBCD;
    v_11 = 0x2262D31C901D3FE1997D8932E3CC3001;
    v_12 = 0xE5EEA95288139BEAD3BA85C6A19D2A0;
    v_13 = 0x417E0192461FFEED9DE72D33160F3C7E;
    v_14 = 0x49C6F1DD512DE22F58E8F0ADCF1871B6;
    v_15 = 0xEAACB10F94189D1BFB65E74FFA0C205;
    v_16 = 0x3AE2F8513CF21E3DC3B2155BB61C289;
    v_17 = 0x3A56BC1C35A384A1332DE059E4A3592D;
    v_18 = 0xA27C7900F4FA7C8779EE6DCF9A86B3;
    v_19 = 0x4F26D971787C9BED6F1A76FA740E33E0;
    v_20 = 0x3BF6384722ECDFECE32641D781EAFABA;
```

Рисунок 3 – Генерация значений v_j на этапе выпуска смарт-карты

```
[2] -- Выбрать k разных значений j, для которых v_j является квадратичным вычетом по
       модулю n, и вычислить наименьший квадратный корень s_j элемента (v_j)^-1 (mod n):

    s_ 2 = 0x1FDD26C17B646B123DB1FE26DD792635;
    s_ 4 = 0x2290CF554DB39E1D7EB10B33D61EE9A6;
    s_ 7 = 0xE4761A415DF663116EC9357D6989235;
    s_ 8 = 0x9D73A4CA7C267FB5E1B29D7DABE159C;
    s_13 = 0x11B3A2234EDAC12A29162988B43406C2;
    s_18 = 0x23164737395EDB5F0D93F41A9B60DE5F;


[3] -- Выпустить смарт-карту, содержащую I, k значений s_j и их индексы.
```

Рисунок 4 – Определение значений s_j на этапе выпуска смарт-карты

Будем подписывать сообщение со следующим содержимым:

```
~ cat message
this is my message
new line
october the 19-th
```

Рисунок 5 – Содержимое подписываемого сообщения

```
FSDS> (sign-message)

Введите количество раундов подписи t (t > 4, по умолчанию 5):

Введите имя файла, в котором содержится сообщение (по умолчанию message):

[1] -- Алиса выбирает случайные r_1, ..., r_t из [0, n) и вычисляет x_i = (r_i)^2 (mod n):

    r_ 1 = 0x5311AE4B7399FFA1F0E77C63F6B9DD5E;
    x_ 1 = 0x3E5930661BB9C0F764B1D47A195DA11A;

    r_ 2 = 0x3BBDBA23608C5DB5FE8BEEAA71ACEEF9;
    x_ 2 = 0x1FF2441D54664EAF8CA025E42FBA06F3;

    r_ 3 = 0x3AD97892B5CF058618B356FAF157EEF;
    x_ 3 = 0x3FC05F98B1467B0B5D31365FFD7C4DFC;

    r_ 4 = 0x54C56EF7EAA88F02203BCAA5ADA7DB17;
    x_ 4 = 0x4348FC563484AD3C26EE607A9198C53C;

    r_ 5 = 0x102C0FFEFFE4BF1DACDF03758233687A;
    x_ 5 = 0x48158F4AFB6F1233C4798B6B7827FB79;
```

Рисунок 5 – Выполнение шага 1 этапа генерации подписи

```
[2] -- Алиса вычисляет f(m, x_1, ..., x_t) и использует первые max(js) * t битов в качестве значений e_{ij}.

    f(m, x_1, ..., x_t) = 0x35983C23912A8F9CBF8E9A286CD7D839;
    Первые max(js) * t битов: 0b11010110011000001111000010001110010001001010100011111001110010111111100011101001101000010.


[3] -- Алиса вычисляет y_i = r_i П_{e_ij = 1} s_j (mod n) для i = 1, ..., t и
       отправляет I, m, строку с e_ij и элементы y_i Бобу.

    Вычисленные значения y_i:

    y_ 1 = 0x47C35722324D89C5FA8CB3E0B6A5263C;
    y_ 2 = 0x556FFF28E9F26245EE3AEA780149394A;
    y_ 3 = 0x356E96A3BC100E7E318E7BA005C8806;
    y_ 4 = 0x3199DF1104CBE294584FE662FCAFA594;
    y_ 5 = 0x142EDECC50D08CB70E265947D989B1A7;
```

Рисунок 6 – Выполнение шагов 2 и 3 этапа генерации подписи

```
FSDS> (verify-signature)

Введите имя файла, в котором содержится сообщение, подпись которого проверяем (по умолчанию message):

[1] -- Боб вычисляет v_j = f(i, j) для полученных значений j.

    v_ 1 = 0x2C7415C800809E0D1F79D345A9FDAF1F;
    v_ 2 = 0x368BE6360EDEC72EE147B72A959C149B;
    v_ 3 = 0x2F637BBA3BF8A2CE7566DABD2F1B0897;
    v_ 4 = 0x331D2F446930E1845E20F7EAE2F9BCD6;
    v_ 5 = 0x417E0192461FFEED9DE72D33160F3C7E;
    v_ 6 = 0xA27C7900F4FA7C8779EE6DCF9A86B3;
```

Рисунок 7 – Выполнение шага 1 этапа проверки подписи

```
[2] -- Боб вычисляет z_i = (y_i)^2 П_{e_ij = 1} v_j (mod n) для i = 1, ..., t.

    z_ 1 = 0x3E5930661BB9C0F764B1D47A195DA11A;
    z_ 2 = 0x1FF2441D54664EAF8CA025E42FBA06F3;
    z_ 3 = 0x3FC05F98B1467B0B5D31365FFD7C4DFC;
    z_ 4 = 0x4348FC563484AD3C26EE607A9198C53C;
    z_ 5 = 0x48158F4AFB6F1233C4798B6B7827FB79;
```

Рисунок 8 – Выполнение шага 2 этапа проверки подписи

```
[3] -- Боб убеждается, что первые max(js) * t битов f(m, z_1, ..., z_t) совпадают с e_{ij}.

   Первые max(js) * t битов:

   f(m, z_1, ..., z_t): 0b11010110011000001111000010001110010001001010101000111110011100101111111000111010011010010;
              e_{ij}: 0b11010110011000001111000010001110010001001010101000111110011100101111111000111010011010010.

Подпись сообщения m подтверждена.
```

Рисунок 9 – Выполнение шага 3 этапа проверки подписи

Листинг программы

```
(defpackage #:aux
  (:use :cl))

(in-package #:aux)


(defmacro while (condition &body body)
  `(loop while ,condition
         do (progn ,@body)))


(defun write-to-file (data filename)
  (with-open-file (out filename :direction :output :if-exists :supersede
                                :if-does-not-exist :create)
    (dolist (param data)
      (format out "~a~%" param))))


(defun read-parse (filename &optional (at 0))
  (parse-integer (uiop:read-file-line filename :at at)))


(defun n-elts (elt n)
  (if (> n 1)
      (list n elt)
      elt))


(defun compr (elt n lst)
  (if (null lst)
      (list (n-elts elt n))
      (let ((next (car lst)))
        (if (eql next elt)
            (compr elt (1+ n) (cdr lst))
            (cons (n-elts elt n) (compr next 1 (cdr lst)))))))


(defun compress (x)
  (if (consp x)
      (compr (car x) 1 (cdr x))
      x))


(defun is-pow-of-2? (num)
  (zerop (logand num (1- num))))
```

```
(defun mod-expt (base power modulo)
  (setq base (mod base modulo))
  (do ((product 1)) ((zerop power) product)
    (do () ((oddp power))
      (setq base (mod (* base base) modulo)
            power (ash power -1)))
    (setq product (mod (* product base) modulo)
          power (1- power))))


(defun miller-rabin (n &optional (k 10))
  (when (or (= 2 n) (= 3 n)) (return-from miller-rabin t))
  (when (or (< n 2) (= 0 (logand n 1))) (return-from miller-rabin))
  (let* ((n-pred (1- n)) (bound (- n-pred 2)) (t-val n-pred) (s 0) (round 0)
(x))
    (while (= 0 (logand t-val 1)) (setq s (1+ s) t-val (ash t-val -1)))
    (do () (nil)
      (tagbody next-iteration
         (when (= k round) (return-from miller-rabin t))
         (setq x (mod-expt (+ 2 (random bound)) t-val n))
         (when (or (= 1 x) (= n-pred x))
           (incf round) (go next-iteration))
         (do ((iter 0 (1+ iter))) ((= iter (1- s)) (return-from miller-rabin))
           (setq x (mod (* x x) n))
           (when (= 1 x) (return-from miller-rabin))
           (when (= n-pred x)
             (incf round) (go next-iteration)))))))


(defparameter *base-primes*
  (remove-if-not #'(lambda (prime?) (miller-rabin prime? 12))
                 (loop for prime? from (1+ (ash 1 15)) to (1- (ash 1 16)) by 2
                       collect prime?)))


(defun ext-gcd (a b)
  (let ((s 0) (old-s 1) (r b) (old-r a)
        (quotient) (bezout-t))
    (while (not (zerop r))
      (setq quotient (floor old-r r))
      (psetq old-r r r (- old-r (* quotient r))
             old-s s s (- old-s (* quotient s))))
    (if (zerop b) (setq bezout-t 0)
        (setq bezout-t (floor (- old-r (* old-s a)) b)))
    (list old-r old-s bezout-t)))


(defun rho-pollard-machinerie (n x-0 &optional (c 1) (rounds 1000))
  (when (miller-rabin n) (return-from rho-pollard-machinerie 'PRIME))
```

```
  (let ((mapping (lambda (x) (mod (+ c (* x x)) n)))
        (a x-0) (b x-0) (round 0) (q))
    (tagbody map
       (incf round)
       (when (> round rounds) (return-from rho-pollard-machinerie 'GEN-NEW))
       (setq a (funcall mapping a)
             b (funcall mapping (funcall mapping b))
             q (gcd (- a b) n))
       (cond ((< 1 q n) (return-from rho-pollard-machinerie
                          (list q (miller-rabin q))))
             ((= n q) (return-from rho-pollard-machinerie))
             (t (go map))))))


(defun rho-pollard-wrapper (n x-0)
  (let ((c 1) (head) (factor) (factors))
    (while (zerop (logand n 1))
      (setq factors (cons 2 factors) n (ash n -1)))
    (setq x-0 (mod x-0 n))
    (while (/= 1 n)
      (setq factor (rho-pollard-machinerie n x-0 c))
      (cond ((eql 'PRIME factor) (setq factors (cons n factors) n 1))
            ((eql 'GEN-NEW factor) (return))
            ((cadr  factor)  (setq  factors  (cons  (setq  head  (car  factor))
factors)
                                 n (/ n head)))
            ((null factor) (while (= (- n 2) (setq c (1+ (random (1- n)))))))
            (t (setq n (/ n (setq head (car factor)))
                     factors (append factors
                                     (rho-pollard-wrapper    head    (random
head)))))))
    factors))


(defun rho-pollard (n x-0)
  (let* ((factors (rho-pollard-wrapper n x-0)))
    (when (null factors) (return-from rho-pollard))
    (when (= n (apply #'* factors))
      (compress (sort (rho-pollard-wrapper n x-0) #'<)))))


(defun get-bit-len ()
  (let ((bit-len))
    (tagbody try-again
       (setq bit-len (read))
       (when (and (integerp bit-len) (is-pow-of-2? bit-len) (> bit-len 16))
         (return-from get-bit-len bit-len))
       (format t "Некорректный ввод! Попробуйте снова: ")
       (go try-again))))
```

```
(defun find-g (p)
  (when (not (miller-rabin p)) (return-from find-g))
  (let ((phi (1- p)) (factors) (g?) (bound (- p 2)))
    (setq factors (rho-pollard phi (random p)))
    (when (null factors) (return-from find-g))
    (setq factors (mapcar #'(lambda (factor) (cond ((atom factor) factor)
                                                   (t     (cadr     factor))))
factors)
          factors (mapcar #'(lambda (factor) (floor phi factor)) factors))
    (tagbody try-again
       (setq g? (+ 2 (random bound)))
       (when (= 1 (mod-expt g? (car factors) p)) (go try-again))
       (when (remove-if-not #'(lambda (pow) (= 1 (mod-expt g? pow p)))
                            factors) (go try-again))) g?))


(defun generate-even (target-len)
  (apply #'+ (ash 1 (1- target-len))
         (mapcar #'(lambda (bit pow) (* bit (ash 1 pow)))
                 (append (loop for bit from 0 to (- target-len 3)
                               collect (random 2)) '(0))
                 (loop for pow from (- target-len 2) downto 0 collect pow))))


(defun generate-prime (target-len)
  (when (not (is-pow-of-2? target-len))
    (return-from generate-prime))
  (when (= 16 target-len)
    (return-from generate-prime (nth (random (length *base-primes*))
                                     *base-primes*)))
  (let ((prime) (s) (prime?) (req-len (- target-len 16)))
    (tagbody pick-prime
       (setq prime (nth (random (length *base-primes*)) *base-primes*))
       (when (not (miller-rabin prime)) (go pick-prime)))
    (tagbody try-again
       (setq s (generate-even req-len)
             prime? (1+ (* prime s)))
       (if (and (= 1 (mod-expt 2 (1- prime?) prime?))
                (/= 1 (mod-expt 2 s prime?))
                (zerop (logxor (length (write-to-string prime? :base 2))
                               target-len)))
           (return-from generate-prime prime?)
           (go try-again)))))


(defun gen-p&g (bit-len)
  (let ((p) (g))
```

```
    (tagbody gen-prime
       (setq p (generate-prime bit-len)
             g (find-g p))
       (when (null g) (go gen-prime)))
    (list p g)))


(defun compute-jacobi-machinerie (a b)
  (let ((r 1) (t-val) (c))
    (when (< a 0) (setq a (- a))
      (when (= 3 (mod b 4)) (setq r (- r))))
    (tagbody eliminate-evenness
       (setq t-val 0)
       (while (zerop (logand a 1))
         (setq t-val (1+ t-val) a (ash a -1)))
       (when (= 1 (logand t-val 1))
         (when (or (= 3 (mod b 8)) (= 5 (mod b 8)))
           (setq r (- r))))
       (when (and (= 3 (mod a 4)) (= 3 (mod b 4)))
         (setq r (- r)))
       (setq c a a (mod b c) b c)
       (if (not (zerop a))
           (go eliminate-evenness)
           (return-from compute-jacobi-machinerie r)))))


(defun compute-jacobi (a b)
  (when (and (integerp a) (integerp b) (= 1 (logand b 1)) (> b 1))
    (if (= 1 (gcd a b))
        (compute-jacobi-machinerie a b) 0)))


(defun is-square-residue (a p)
  (= 1 (mod-expt a (ash (1- p) -1) p)))


(defun compute-legendre-machinerie (a p)
  (cond ((zerop (mod a p)) 0)
        ((is-square-residue a p) 1)
        (t -1)))


(defun compute-legendre (a p)
  (when (and (integerp a) (miller-rabin p))
    (compute-legendre-machinerie a p)))


(defun find-k-i (a-i q p)
  (do ((k 0 (1+ k))) ((= 1 (mod-expt a-i (* (expt 2 k) q) p)) k)))
```

```
(defun get-inv (a p)
  (cadr (ext-gcd a p)))


(defun seq-sqrt-Zp (a p)
  (when (/= 1 (compute-legendre a p)) (return-from seq-sqrt-Zp))
  (let ((b) (k-i -1) (k-is) (r-i) (m 0) (q (1- p))
        (a-prev a) (a-cur a) (pow))
    (while (zerop (logand q 1)) (setq m (1+ m) q (ash q -1)))
    (while (/= -1 (compute-legendre (setq b (random p)) p)))
    (while (not (zerop k-i))
      (setq  k-i    (find-k-i a-cur q p) k-is (cons k-i k-is))
      (psetq a-cur  (mod (* a-cur (mod-expt b (ash 1 (- m k-i)) p)) p)
             a-prev a-cur))
    (setq k-is (cdr k-is) r-i (mod-expt a-prev (ash (1+ q) -1) p))
    (do ((i (length k-is) (1- i))) ((= 0 i) r-i)
      (setq pow (ash 1 (- m (car k-is) 1))
            r-i (mod (* r-i (get-inv (mod-expt b pow p) p)) p)
            k-is (cdr k-is)))))


(defun sqrt-Zn (a p q n)
  (let ((r) (s) (x) (y) (rdq) (scp))
    (when (= 1 (compute-jacobi a p))
      (setq r (seq-sqrt-Zp a p)))
    (when (= 1 (compute-jacobi a q))
      (setq s (seq-sqrt-Zp a q)))
    (when (or (null r) (null s))
      (return-from sqrt-Zn))
    (destructuring-bind (c d) (cdr (ext-gcd p q))
      (setq rdq (mod (* r d q) n) scp (mod (* s c p) n)
            x (mod (+ rdq scp) n)
            y (mod (- rdq scp) n))
      (list x (mod (- x) n) y (mod (- y) n)))))
```

```
(defpackage :crypt
  (:use :common-lisp)
  (:export :f :list-all-digests
           :hash-setter *hash*))


(in-package :crypt)


(defvar *hash*)

(defvar *digests* '(:blake2/256 :groestl/256 :jh/256       :keccak/256
                    :md5        :ripemd-160  :sha256       :sha384
                    :sha512     :skein1024   :streebog/256 :tiger))


(defun hash-file-setter (filename)
  (let ((hash-of-choice))
    (setq hash-of-choice (read-from-string (uiop:read-file-line filename)))
    (if (member hash-of-choice *digests*)
        (setf *hash* hash-of-choice)
        (return-from hash-file-setter))))


(defun hash-setter (c)
  (let ((hash-of-choice))
    (setq hash-of-choice (nth (1- c) *digests*))
    (aux::write-to-file (list (write-to-string hash-of-choice))
                        "hash-of-choice")
    (setf *hash* hash-of-choice) hash-of-choice))


(defun list-all-digests ()
  (format t
"~%~4t[1]: BLAKE2/256; [ 2]: GROESTL/256; [ 3]:      JH/256; [ 4]: KECCAK/256;
    [5]:        MD5; [ 6]:  RIPEMD-160; [ 7]:       SHA256; [ 8]:     SHA384;
    [9]:          SHA512;  [10]:      SKEIN1024;  [11]:  STREEBOG/256;  [12]:
TIGER.~%"))


(defun hash (str)
  (ironclad:byte-array-to-hex-string
   (ironclad:digest-sequence
    *hash*
    (ironclad:ascii-string-to-byte-array str))))


(defun f (str modulo)
  (let ((digest (hash str)))
```

```
(mod (ironclad:octets-to-integer
      (ironclad:ascii-string-to-byte-array digest)) modulo)))
```

```
(defpackage :fsds-aux
  (:use :cl)
  (:export :get-bit-len :get-c
           :form-I :get-j
           :get-t :get-message
           :compute-ys :read-parse-l))


(in-package :fsds-aux)


(defun get-bit-len ()
  (let ((bit-len))
    (tagbody try-again
       (setq bit-len (read-line))
       (when (zerop (length bit-len)) (return-from get-bit-len 1024))
       (setq bit-len (parse-integer bit-len :junk-allowed t))
       (when (or (null bit-len) (not (and (zerop (logand bit-len (1- bit-len)))
                                          (> bit-len 16))))
         (format t "~%Некорректное значение длины l модуля! Попробуйте ввести
l снова: ")
         (go try-again))) bit-len))


(defun get-c ()
  (let ((c))
    (tagbody try-again
       (setq c (read-line))
       (when (zerop (length c)) (return-from get-c 7))
       (setq c (parse-integer c :junk-allowed t))
       (when (not (member c '(1 2 3 4 5 6 7 8 9 10 11 12)))
         (format t "~%Некорректный номер хеш-функции! Попробуйте ввести номер
снова: ")
         (go try-again))) c))


(defun form-I ()
  (let ((name))
    (format t "~%Введите имя пользователя (I), для которого выпускается смарт-
карта (по умолчанию Alice): ")
    (setq name (read-line))
    (when (zerop (length name))
      (setq name "Alice"))
    (aux::write-to-file (list name) "I") name))


(defun get-j ()
  (let ((j))
```

```
    (format t "~%Введите количество значений v_j (9 < j < 51, по умолчанию 20):
")
    (tagbody try-again
       (setq j (read-line))
       (when (zerop (length j)) (return-from get-j 20))
       (setq j (parse-integer j :junk-allowed t))
       (when (or (null j) (not (< 9 j 51)))
         (format t "~%Некорректное значение j! Попробуйте ввести его снова: ")
         (go try-again))) j))


(defun get-t ()
  (let ((t-val))
    (format t "~%Введите количество раундов подписи t (t > 4, по умолчанию 5):
")
    (tagbody try-again
       (setq t-val (read-line))
       (when (zerop (length t-val))
         (aux::write-to-file (list 5) "num-rounds")
         (return-from get-t 5))
       (setq t-val (parse-integer t-val :junk-allowed t))
       (when (or (null t-val) (> 5 t-val))
         (format t "~%Некорректное значение количества раундов подписи!
Попробуйте ввести t снова: ")
         (go try-again)))
    (aux::write-to-file (list t-val) "num-rounds") t-val))


(defun get-message ()
  (let ((filename))
    (format t "~%Введите имя файла, в котором содержится сообщение (по
умолчанию message): ")
    (tagbody try-again
       (setq filename (read-line))
       (when (zerop (length filename)) (setq filename "message"))
       (when (not (uiop:file-exists-p filename))
         (format t "~%Файла с заданным именем не существует! Попробуйте ввести
имя файла с сообщением снова: ")
         (go try-again)))
    (uiop:read-file-lines filename)))


(defun compute-ys (rs sjs first-bits indices n)
  (let* ((t-val (length rs)) (ys) (cur-bits) (prod 1)
         (j (floor (length first-bits) t-val)))
    (do ((k 0 (1+ k))) ((= t-val k) (reverse ys))
      (setq cur-bits (subseq first-bits (* j k) (* j (1+ k))))
      (do ((i 0 (1+ i))) ((= (length indices) i))
        (when (not (zerop (digit-char-p (char cur-bits (1- (nth i indices))))))
```

```
          (setq prod (mod (* prod (nth i sjs)) n))))
      (setq ys (cons (mod (* (nth k rs) prod) n) ys)
            prod 1))))


(defun read-parse-l (filename)
  (mapcar #'parse-integer (uiop:read-file-lines filename)))
```

```
(defpackage :fsds
  (:use :cl))


(in-package :fsds)


(defun stop () (read-line))


(defun setup ()
  (let ((len) (p) (q) (n) (c))
    (format t "~%[0] -- Перед выпуском смарт-карт центр выбирает значение модуля n и псевдослучайную
       функцию f, отображающую строку произвольной длины в целое число из диапазона [0; n).")
    (format t "~2%Введите битовую длину l модуля (l = 2^m, l > 16) (по умолчанию 1024): ")
    (setq len (fsds-aux:get-bit-len)
          p (aux::generate-prime (ash len -1)) q (aux::generate-prime (ash len -1))
          n (* p q))
    (aux::write-to-file (list p q) "factors")
    (aux::write-to-file (list n) "modulo")
    (format t "~%~4tp = 0x~x;~%~4tq = 0x~x;~%~4tn = 0x~x.~%" p q n)
    (format t "~%Введите номер хеш-функции, используемой в псевдослучайном отображении f (по умолчанию 7 -- SHA256):~%")
    (crypt:list-all-digests) (format t "~%Ваш выбор: ")
    (setq c (fsds-aux:get-c))
    (format t "~%Псевдослучайной функцией f назначена функция: ~a(string) (mod n)."
            (crypt:hash-setter c)) t))


(defun step-1-issue ()
  (let ((I (fsds-aux:form-I)) (j (fsds-aux:get-j)) (vjs)
        (n (aux::read-parse "modulo")))
    (format t "~%[1] -- Вычислить значения v_j = f(I, j) для небольших значений j.")
    (setq vjs (mapcar #'(lambda (k)
                          (crypt:f (concatenate 'string I (write-to-string k)) n))
                      (loop for k from 1 to j collect k)))
    (aux::write-to-file vjs "vjs-issue") (terpri)
    (do ((k 0 (1+ k))) ((= (length vjs) k))
      (format t "~%~4tv_~2d = 0x~x;" (1+ k) (nth k vjs)))
    (terpri) t))
```

```
(defun step-2-issue ()
  (format t "~%[2] -- Выбрать k разных значений j, для которых v_j является
квадратичным вычетом по
       модулю n, и вычислить наименьший квадратный корень s_j элемента (v_j)^-
1 (mod n): ") (terpri)
  (let* ((vjs (mapcar #'parse-integer (uiop:read-file-lines "vjs-issue")))
         (j (length vjs)) (n (aux::read-parse "modulo"))
         (p (aux::read-parse "factors")) (q (aux::read-parse "factors" 1))
         (sjs) (val) (roots) (indices))
    (do ((k 0 (1+ k))) ((= j k))
      (setq val (nth k vjs))
      (when (= 1 (aux::compute-jacobi val n))
        (setq val (mod (cadr (aux::ext-gcd val n)) n)
              roots (aux::sqrt-Zn val p q n))
        (when roots
          (setq sjs (cons (apply #'min roots) sjs)
                indices (cons (1+ k) indices)))))
    (if indices
      (progn (setq sjs (reverse sjs) indices (reverse indices))
             (aux::write-to-file sjs "sjs")
             (aux::write-to-file indices "indices"))
      (progn (format t "~%Ни один из v_j не подошёл! Повторная генерация
модуля...")
             (return-from step-2-issue)))
    (do ((k 0 (1+ k))) ((= (length sjs) k))
      (format t "~%~4ts_~2d = 0x~x;" (nth k indices) (nth k sjs))) (terpri)
t))


(defun step-3-issue ()
  (format t "~%[3] -- Выпустить смарт-карту, содержащую I, k значений s_j и их
индексы.")
  (uiop:run-program "mkdir smartcard/")
  (uiop:run-program "mv I sjs indices smartcard/") t)


(defun issue-a-smartcard ()
             (step-1-issue)                                    (stop)
  (when (not (step-2-issue)) (return-from issue-a-smartcard)) (stop)
             (step-3-issue) t)


(defun step-1-sign (rounds)
  (let ((n (aux::read-parse "modulo")) (r) (rs) (x) (xs))
    (format t "~%[1] -- Алиса выбирает случайные r_1, ..., r_t из [0, n) и
вычисляет x_i = (r_i)^2 (mod n):~%")
    (do ((i 0 (1+ i))) ((= rounds i))
      (setq r (random n)
            rs (cons r rs)
```

```
            x (mod (* r r) n)
            xs (cons x xs))
      (format t "~%~4tr_~2d = 0x~x;~%~4tx_~2d = 0x~x;~%" (1+ i) r (1+ i) x))
(terpri)
    (aux::write-to-file (reverse rs) "rs")
    (aux::write-to-file (reverse xs) "xs") t))


(defun step-2-sign (message)
  (let* ((n (aux::read-parse "modulo")) (xs (uiop:read-file-lines "xs"))
(image)
         (max-j    (parse-integer    (car    (last    (uiop:read-file-lines
"smartcard/indices")))))
         (first-bits) (rounds (aux::read-parse "num-rounds"))
         (image-bin-len))
    (format t "[2] -- Алиса вычисляет f(m, x_1, ..., x_t) и использует первые
max(js) * t битов в качестве значений e_{ij}.")
    (setq image (reduce #'(lambda (f s) (concatenate 'string f s)) (append
message xs))
          image (crypt:f image n)
          image-bin-len (length (write-to-string image :base 2)))
    (when (< image-bin-len (* max-j rounds))
      (format t "~2%В образе меньше, чем max(js) * t битов! Попробуйте
подписывать по другому модулю,
выбрать другую хеш-функцию или заново сгенерировать модуль.")
      (return-from step-2-sign))
    (format t "~2%~4tf(m, x_1, ..., x_t) = 0x~x;~%~4tПервые max(js) * t битов:
0b~a."
            image (setq first-bits (subseq (write-to-string image :base 2) 0
(* max-j rounds))))
    (terpri) (aux::write-to-file (list first-bits) "first-bits") t))


(defun step-3-sign ()
  (let* ((sjs (fsds-aux:read-parse-l "smartcard/sjs"))
         (indices (fsds-aux:read-parse-l "smartcard/indices"))
         (rs (fsds-aux:read-parse-l "rs")) (ys)
         (first-bits (uiop:read-file-line "first-bits")) (n (aux::read-parse
"modulo")))
    (format t "~%[3] -- Алиса вычисляет y_i = r_i П_{e_ij = 1} s_j (mod n) для
i = 1, ..., t и
       отправляет I, m, строку с e_ij и элементы y_i Бобу.")
    (setq ys (fsds-aux:compute-ys rs sjs first-bits indices n))
    (format t "~2%~4tВычисленные значения y_i:~%")
    (do ((i 0 (1+ i))) ((= (length ys) i))
      (format t "~%~4ty_~2d = 0x~x;" (1+ i) (nth i ys))) (terpri)
    (aux::write-to-file ys "ys") t))
```

```
(defun sign-message ()
  (let ((rounds (fsds-aux:get-t)) (message (fsds-aux:get-message)))
              (step-1-sign  rounds)                              (stop)
    (when (not (step-2-sign message)) (return-from sign-message)) (stop)
              (step-3-sign        ) t))


(defun step-1-verify ()
  (format t "~%[1] -- Боб вычисляет v_j = f(i, j) для полученных значений j.")
  (let ((indices (uiop:read-file-lines "smartcard/indices"))
        (I (uiop:read-file-line "smartcard/I"))
        (n (aux::read-parse "modulo")) (vjs))
    (setq vjs (mapcar #'(lambda (j)
                          (crypt:f (concatenate 'string I j) n))
                      indices))
    (aux::write-to-file vjs "vjs-verify") (terpri)
    (do ((i 0 (1+ i))) ((= (length indices) i))
      (format t "~%~4tv_~2d = 0x~x;" (1+ i) (nth i vjs))) (terpri) t))


(defun step-2-verify ()
  (format t "~%[2] -- Боб вычисляет z_i = (y_i)^2 П_{e_ij = 1} v_j (mod n) для
i = 1, ..., t.~%")
  (let* ((ys (fsds-aux:read-parse-l "ys"))
         (vjs (fsds-aux:read-parse-l "vjs-verify"))
         (indices (fsds-aux:read-parse-l "smartcard/indices"))
         (n (aux::read-parse "modulo")) (t-val (length ys)) (zis)
         (first-bits (uiop:read-file-line "first-bits")) (prod 1)
         (step (floor (length first-bits) t-val)) (cur-bits) (y))
    (do ((k 0 (1+ k))) ((= t-val k))
      (setq cur-bits (subseq first-bits (* k step) (* (1+ k) step)))
      (do ((j 0 (1+ j))) ((= (length vjs) j))
        (when (not (zerop (digit-char-p (char cur-bits (1- (nth j indices))))))
          (setq prod (mod (* prod (nth j vjs)) n))))
      (setq y (nth k ys) zis (cons (mod (* y y prod) n) zis) prod 1))
    (setq zis (reverse zis))
    (do ((j 0 (1+ j))) ((= (length zis) j))
      (format t "~%~4tz_~2d = 0x~x;" (1+ j) (nth j zis)))
    (aux::write-to-file zis "zis") (terpri) t))


(defun step-3-verify (message)
  (format t "~%[3] -- Боб убеждается, что первые max(js) * t битов f(m, z_1,
..., z_t) совпадают с e_{ij}.")
  (let    ((max-j    (parse-integer    (car    (last    (uiop:read-file-lines
"smartcard/indices")))))
        (t-val (aux::read-parse "num-rounds")) (n (aux::read-parse "modulo"))
        (first-bits (uiop:read-file-line "first-bits"))
        (zis (uiop:read-file-lines "zis")) (digest) (digest-bits))
```

```
    (setq digest (crypt:f (reduce #'(lambda (f s) (concatenate 'string f s))
                                  (append message zis)) n)
          digest-bits (subseq (write-to-string digest :base 2) 0 (* max-j t-
val)))
    (format t "~2%~4tПервые max(js) * t битов:
~%~4tf(m, z_1, ..., z_t): 0b~a;
~4t             e_{ij}: 0b~a." digest-bits first-bits)
    (format t (if (string-equal first-bits digest-bits)
                  "~2%Подпись сообщения m подтверждена."
                  "~2%Подпись сообщения m некорректна.")) t))


(defun verify-signature ()
  (format t "~%Введите имя файла, в котором содержится сообщение, подпись
которого проверяем (по умолчанию message): ")
  (let ((filename) (message))
    (tagbody try-again
       (setq filename (read-line))
       (when (zerop (length filename))
         (setq filename "message"))
       (when (not (uiop:file-exists-p filename))
         (format t "Файла с указанным именем не существует! Попробуйте ввести
имя файла снова: ")
         (go try-again)))
    (setq message (uiop:read-file-lines filename))
    (step-1-verify        ) (stop)
    (step-2-verify        ) (stop)
    (step-3-verify message) t))
```