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 аутентификации**

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 протокол аутентификации Шнорра.

**2 Теоретические сведения**

Безопасность схемы проверки подлинности и подписи Клауса Шнорра опирается на трудность вычисления дискретных логарифмов. Для генерации пары ключей сначала выбираются два простых числа и так, чтобы было сомножителем . Затем выбирается , не равное 1, такое, что Все эти числа могут быть свободно опубликованы и использоваться группой пользователей.

Генерация ключей для схемы Шнорра происходит так же, как и генерация ключей для DSA, кроме того, что не существует никаких ограничений по размерам.

**Генерация ключей**

1. Выбирается простое число , которое по длине обычно равняется 1024 битам.
2. Выбирается другое простое число таким, чтобы оно было делителем числа . То есть должно выполняться Размер для числа принято выбирать равным 160 битам.
3. Выбирается число , отличное от 1, такое, что
4. Пегги выбирает случайное целое число меньшее .
5. Пегги вычисляет
6. Общедоступный ключ Пегги – (, секретный ключ Пегги –

**Протокол аутентификации**

1. *Предварительная обработка*. Алиса выбирает случайное число , меньшее , и вычисляет Эти вычисления являются предварительными и могут быть выполнены задолго до появления Боба.
2. *Инициирование*. Алиса посылает Бобу.
3. Боб выбирает случайное число из диапазона от 0 до и отправляет его Алисе.
4. Алиса вычисляет и посылает Бобу.
5. *Подтверждение*. Боб проверяет, что

Безопасность алгоритма зависит от параметра . Сложность вскрытия алгоритма примерно равна . Шнорр советует использовать около 72 битов для и Для решения задачи дискретного логарифма в этом случае требуется, по крайней мере, шагов известных алгоритмов.

**3 Практическая реализация**

**3.1 Описание программы**

Язык программной реализации – Common Lisp. Две основные функции программы представляют собой функцию генерации ключей и функцию, реализующую протокол проверки подлинности. Программа может работать в двух основных режимах в зависимости от битовой размерности чисел и : первый режим для битовых размерностей 1024 и 160, второй режим для битовых размерностей 768 и 120.

**3.2 Результаты тестирования программы**

Рассмотрим процесс выполнения программы для битовых длин чисел и , равные 1024 и 160. Для удобства представления числа представлены в шестнадцатеричной системе счисления и сокращены на скриншотах.

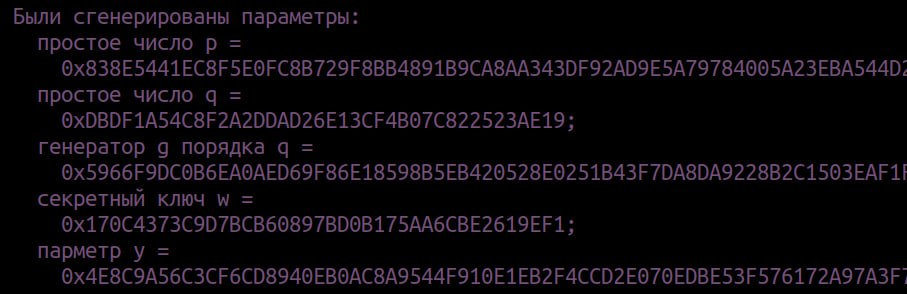


Рисунок 1 – Результат выполнения предварительного этапа генерации ключей



Рисунок 2 – Начало выполнения протокола аутентификации

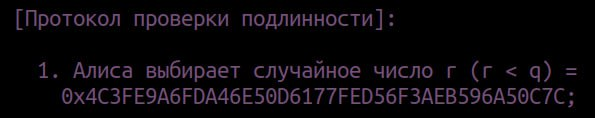


Рисунок 3 – Выполнение шага 1 протокола

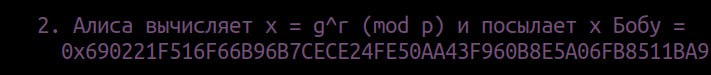


Рисунок 4 – Выполнение шага 2 протокола

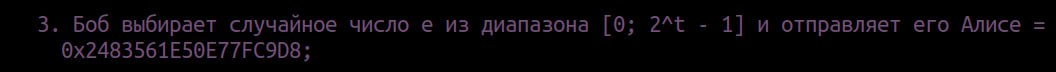


Рисунок 5 – Выполнение шага 3 протокола

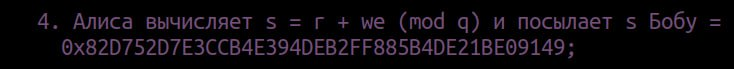


Рисунок 6 – Выполнение шага 4 протокола

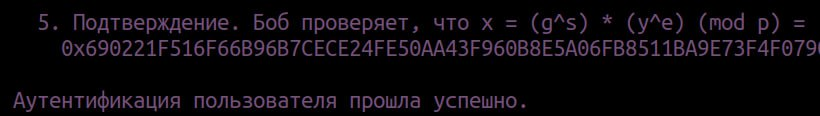


Рисунок 5 – Выполнение шага 5 протокола

**Листинг программы**

(defpackage #:macro

(:use :cl))

(in-package #:macro)

(defmacro while (condition &body body)

`(loop while ,condition

do (progn ,@body)))

(defpackage #:aux

(:use :cl))

(in-package #:aux)

(defun mod-expt (base power divisor)

(setq base (mod base divisor))

(do ((product 1))

((zerop power) product)

(do () ((oddp power))

(setq base (mod (\* base base) divisor)

power (ash power -1)))

(setq product (mod (\* product base) divisor)

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))

(macro::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)))))))

(defpackage #:schnorr

(:use :cl))

(in-package #:schnorr)

(defvar p-bit-length)

(defvar q-bit-length)

(defvar t-param 72)

; Функции для компактного отображения списка множителей

(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))

; Алгоритм разложения числа n rho-методом Полларда

(defun rho-pollard-machinerie (n x-0 &optional (c 1) (rounds 1000))

(when (aux::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 (aux::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))

(macro::while (zerop (logand n 1))

(setq factors (cons 2 factors) n (ash n -1)))

(setq x-0 (mod x-0 n))

(macro::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) (macro::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) #'<)))))

; Генерация ключей для схемы Шнорра

(defun from-bin-to-dec (binary)

(apply #'+ (mapcar #'(lambda (bit pow) (\* bit (expt 2 pow)))

binary

(loop for idx from (1- (length binary))

downto 0 collect idx))))

(defun get-n-bit-num (bit-length &optional (bit 1))

(when (and (/= 0 bit) (/= 1 bit)) (return-from get-n-bit-num))

(cons 1 (append (loop for bit from 2 to (1- bit-length) collect (random 2))

`(,bit))))

(defun gen-p (q)

(let\* ((bit-dif (- p-bit-length q-bit-length)) (bin (get-n-bit-num bit-dif 0))

(p (1+ (\* q (from-bin-to-dec bin)))))

(macro::while (or (/= p-bit-length (length (write-to-string p :base 2)))

(not (aux::miller-rabin p)))

(setq bin (get-n-bit-num bit-dif 0)

p (1+ (\* q (from-bin-to-dec bin))))) p))

(defun gen-q ()

(let\* ((bin) (q))

(macro::while (not (aux::miller-rabin (setq bin (get-n-bit-num q-bit-length)

q (from-bin-to-dec bin))))) q))

(defun is-primitive? (g p factors)

(let ((phi (1- p)))

(when (= 1 (aux::mod-expt g phi p))

(dolist (factor factors t)

(when (= 1 (aux::mod-expt g (/ phi factor) p))

(return-from is-primitive?))))))

(defun gen-g (p q)

(let\* ((phi (1- p)) (phi-div (/ phi q)) (g)

(factors (rho-pollard phi-div (random phi-div))))

(when (null factors) (return-from gen-g))

(setq factors (mapcar #'(lambda (factor) (cond ((atom factor) factor)

(t (cadr factor)))) factors))

(psetf (nth 0 factors) (nth 1 factors)

(nth 1 factors) (nth 0 factors))

(macro::while (not (is-primitive? (setq g (+ 2 (random (- p 2)))) p factors)))

(setq g (aux::mod-expt g phi-div p))))

(defun write-to-file (list-to-write &optional (filename "public\_key"))

(with-open-file (out filename :direction :output :if-exists :supersede

:if-does-not-exist :create)

(dolist (point list-to-write)

(format out "~A~%" point))))

(defun read-from-file (filename &optional (len-form 1))

(handler-case (mapcar #'parse-integer

(uiop:read-file-lines filename))

(error (err)

(format t "В ходе исполнения программы была обнаружена ошибка:~%~a~%" err)

(values (make-list len-form)))))

(defun print-status (status-code args)

(cond ((eql 'GENERATED status-code)

(destructuring-bind (p q g w y) args

(format t "Были сгенерированы параметры:

простое число p =~% 0x~x;

простое число q =~% 0x~x;

генератор g порядка q =~% 0x~x;

секретный ключ w =~% 0x~x;

парметр y =~% 0x~x."

p q g w y)))

((eql 'EXTRACTED status-code)

(destructuring-bind (file-pub file-priv p q g y w) args

(format t "Из файлов ~s и ~s были извлечены параметры:

простое число p =~% 0x~x;

простое число q =~% 0x~x;

генератор g порядка q =~% 0x~x;

секретный ключ w =~% 0x~x;

параметр y =~% 0x~x.~%~%"

file-pub file-priv p q g y w)))

((eql 'AUTH status-code)

(destructuring-bind (r x e s x?) args

(format t "[Протокол проверки подлинности]:

1. Алиса выбирает случайное число r (r < q) =~% 0x~x;

2. Алиса вычисляет x = g^r (mod p) и посылает x Бобу = ~% 0x~x;

3. Боб выбирает случайное число e из диапазона [0; 2^t - 1] и отправляет его Алисе =~% 0x~x;

4. Алиса вычисляет s = r + we (mod q) и посылает s Бобу =~% 0x~x;

5. Подтверждение. Боб проверяет, что x = (g^s) \* (y^e) (mod p) =~% 0x~x.~%~%"

r x e s x?)))

(t (return-from print-status))))

(defun gen-g\* (p q)

(let ((pow (/ (1- p) q)) (bound (- p 3)) (g))

(macro::while (= 1 (setq g (aux::mod-expt (+ 2 (random bound)) pow p))))

g))

(defun opter ()

(let ((choice) (opt))

(format t "Режимы работы:

[1] -- Битовая длина p и q = 768 и 120, генерация g разложением;

[2] -- Битовая длина p и q = 1024 и 160, генерация g разложением;

[3] -- Битовая длина p и q = 768 и 120, генерация g по dss;

[4] -- Битовая длина p и q = 1024 и 160, генерация g по dss;

[0] -- Выход.~%")

(format t "Ваш выбор: ")

(macro::while (not (member (setq choice (read)) '(1 2 3 4 0)))

(format t "Некорректный ввод! Попробуйте снова: "))

(cond ((= 1 choice) (setf p-bit-length 768 q-bit-length 120 opt 'F-FAC))

((= 2 choice) (setf p-bit-length 1024 q-bit-length 160 opt 'S-FAC))

((= 3 choice) (setq p-bit-length 768 q-bit-length 120 opt 'F-DSS))

((= 4 choice) (setq p-bit-length 1024 q-bit-length 160 opt 'S-DSS))

(t (return-from opter))) opt))

(defun gen-g-caller (p q opt)

(cond ((or (eql opt 'F-FAC) (eql opt 'S-FAC)) (gen-g p q))

((or (eql opt 'F-DSS) (eql opt 'S-DSS)) (gen-g\* p q))

(t (return-from gen-g-caller))))

(defun generate-keys ()

(let\* ((q) (p) (g) (w) (y) (gen-g-opt))

(setq gen-g-opt (opter))

(when (null gen-g-opt) (return-from generate-keys))

(time (macro::while (null (setq q (gen-q)

p (gen-p q)

g (gen-g-caller p q gen-g-opt)))))

(setq w (1+ (random (1- q)))

y (aux::mod-expt g (- q w) p))

(print-status 'GENERATED (list p q g w y))

(write-to-file (list p q g y))

(write-to-file (list w) "private\_key")))

(defun authenticate (&optional (file-pub "public\_key") (file-priv "private\_key"))

(let ((w) (r) (x) (e) (s) (x?))

(destructuring-bind (p q g y)

(read-from-file file-pub 4)

(when (some #'not (list p q g y)) (return-from authenticate))

(setq w (car (read-from-file file-priv)))

(when (not w) (return-from authenticate))

(setq x (aux::mod-expt g (setq r (+ 2 (random (- q 2)))) p)

e (random (expt 2 t-param))

s (mod (+ r (\* w e)) q)

x? (mod (\* (aux::mod-expt g s p) (aux::mod-expt y e p)) p))

(print-status 'EXTRACTED (list file-pub file-priv p q g y w))

(print-status 'AUTH (list r x e s x?))

(if (= x? x)

(format t "Аутентификация пользователя прошла успешно.")

(format t "Аутентификация пользователя не удалась.")))))