ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

**Расчётно-графическая работа**

по дисциплине “Защита информации”

на тему

**Доказательство с нулевым знанием**

Вариант №3

|  |  |
| --- | --- |
| Выполнил студент | Дьяченко Даниил Вадимович |
|  | Ф.И.О. |

|  |  |
| --- | --- |
| Группы | ИВ-621 |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работу приняла |  | ассистент кафедры ПМиК Я.В. Петухова |
|  | подпись |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Защищена |  | Оценка |  |
|  |  |  |  |

Новосибирск – 2019 г.

Оглавление

[1 Постановка задачи 3](#_Toc27002262)

[2 Теоретические сведения 4](#_Toc27002263)

[ПРИЛОЖЕНИЕ 9](#_Toc27002264)

[1. Исходный код 9](#_Toc27002265)

1. Постановка задачи

В рамках расчётно-графического задания необходимо написать прог-рамму, реализующую протокол доказательства с нулевым знанием Фиата-Шамира.

При установлении подлинности пароля Алиса должна передать свой секрет (пароль) верификатору; это может привести к перехвату информации Евой. Кроме того, нечестный верификатор может показать пароль другим или использовать его, чтобы исполнить роль претендента.

При установлении подлинности объекта методом вызова-ответа секрет претендента не передают верификатору. Претендент применяет некоторую функцию для обработки вызова, которая передана верификатором, но при этом включает свой секрет. В некоторых методах "вызова-ответа" верификатор фактически знает секрет претендента, при этом он может неправильно использоваться нечестной верификацией. В других методах верификатор может извлечь некоторую информацию о секрете претендент а, выбирая заранее запланированное множество вызовов.

В установлении подлинности с нулевым разглашением претендент не раскрывает ничего, что могло бы создать угрозу конфиденциальности секрета. Претендент доказывает верификатору, что он знает секрет, не раскрывая и не показывая его. В таком случае взаимодействие разработано так, чтобы не привести к раскрытию или предположению о содержании секрета. После обмена сообщениями верификатор только знает, что претендент имеет или не имеет секрета - и ничего больше. В этой ситуации результат - да/нет. Это единственный бит информации

1. Теоретические сведения

Протокол Фиата — Шамира — это один из наиболее известных протоколов идентификации с нулевым разглашением (Zero-knowledge protocol). Протокол был предложен Амосом Фиатом (англ. Amos Fiat) и Ади Шамиром (англ. Adi Shamir)

Пусть А знает некоторый секрет s. Необходимо доказать знание этого секрета некоторой стороне В без разглашения какой-либо секретной информации. Стойкость протокола основывается на сложности извлечения квадратного корня по модулю достаточно большого составного числа n, факторизация которого неизвестна.

A доказывает B знание s в течение t раундов. Раунд называют также аккредитацией. Каждая аккредитация состоит из 3х этапов.

**Предварительные действия:**

* Доверенный центр Т выбирает и публикует модуль , где , — простые и держатся в секрете
* Каждый претендент A выбирает взаимно-простое с , где . Затем вычисляется . регистрируется Т в качестве открытого ключа А

**Передаваемые сообщения (этапы каждой аккредитации):**

**Основные действия:**

Следующие действия последовательно и независимо выполняются t раз. В считает знание доказанным, если все t раундов прошли успешно.

* А выбирает случайное число , такое, что и отсылает стороне B (доказательство)
* B случайно выбирает бит ( или ) и отсылает его A (вызов)
* А вычисляет и отправляет его обратно к B. Если , то , иначе (ответ)
* Если , то B отвергает доказательство или, другими словами, А не удалось доказать знание . В противном случае, сторона B проверяет, действительно ли и, если это так, то происходит переход к следующему раунду протокола

Выбор е из множества предполагает, что если сторона А действительно знает секрет, то она всегда сможет правильно ответить, вне зависимости от выбранного . Допустим, что А хочет обмануть B. В этом случае А, может отреагировать только на конкретное значение . Например, если А знает, что получит , то А следует действовать строго по инструкции и В примет ответ. В случае, если А знает, что получит , то А выбирает случайное и отсылает на сторону В, в результате получаем нам нужное . Проблема заключается в том, что А изначально не знает какое он получит и поэтому не может со 100 % вероятностью выслать на сторону В нужные для обмана и ( при и при ). Поэтому вероятность обмана в одном раунде составляет 50 %. Чтобы снизить вероятность жульничества (она равна )) выбирают достаточно большим (). Таким образом, B удостоверяется в знании А тогда и только тогда, когда все раундов прошли успешно.

Пример работы программы

Вывод логов сервера за одну сессию при подключении клиента:

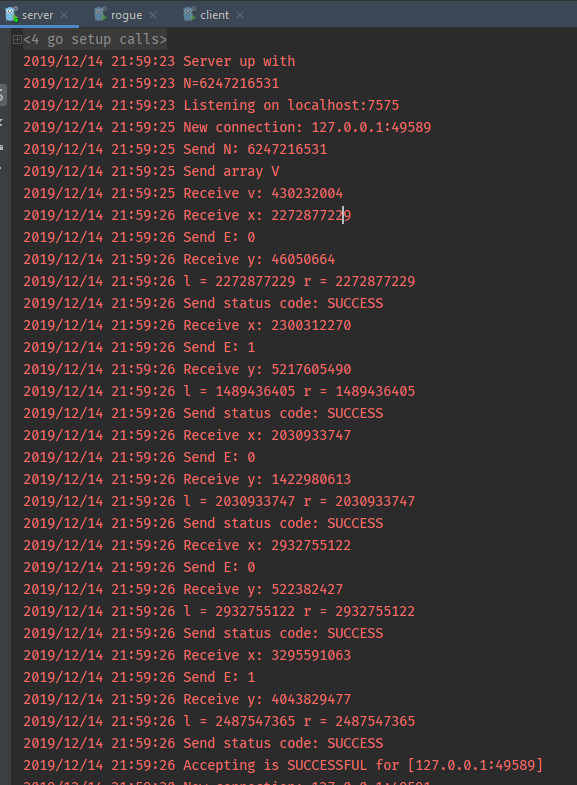


Рисунок 1 - Пример вывода программы

В логах можно увидеть как создается открытый ключ N, создается соединение с пользователем и начинается сессия из пяти раундом путем передачи открытых ключей N и V, приеме пользовательских открытых ключей V и X, передачи случайного Е, приеме сгенерированного пользователем Y, проверке правильности принятого Y и отправка кода статуса ответа (либо SUCCESSFUL, либо ERROR)

Дальше вывод логов пользователя при той же сессии, что была показана выше:

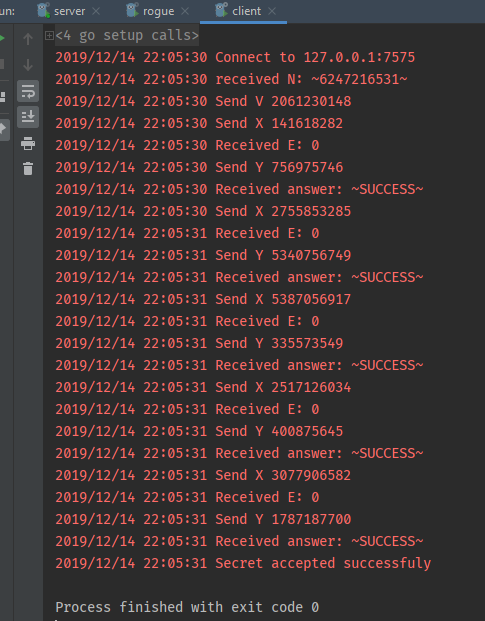
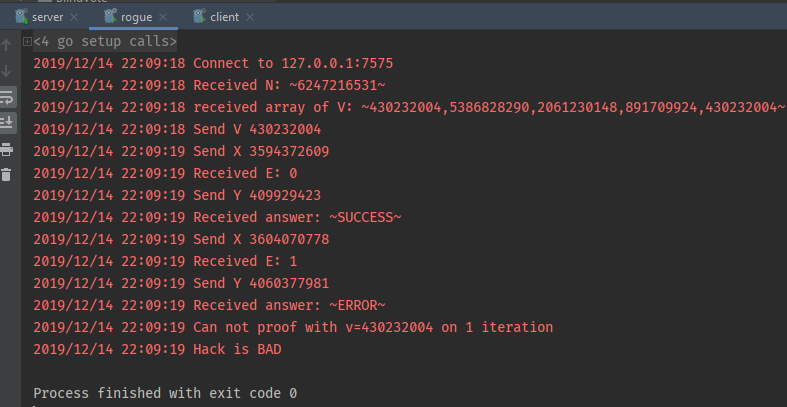


Рисунок 2 - Пример логов пользователя

В логах пользователя можно увидеть соединение с сервером, приеме открытых ключей N и V, передаче сгенерированного открытого ключа V, на основе закрытого ключа, который не показан в логах по понятным причинам, отправке открытого ключа X, сгенерированного на основе случайного большого числа R, приеме параметра E, отправке ответного ключа Y и получение кода статуса ответа (в данном случае это SUCCESSFUL). В случае, если хотя бы в одном из раундом сервер ответит кодом ERROR, авторизация не проходит и клиент отключается.

Дальше вывод логов мошенника при той же сессии, что была показана выше:

Рисунок 3 – Пример логов мошенника

В данных логах можно увидеть, как мошенник подключается к серверу, получает открытые ключи N и V, получает параметр Е и пытается подобрать ключи Y и X. Как видно при Е равным 0 у мошенника получается обмануть сервер, так как по алгоритму Фиата-Шамира в данном случае клиенту необходимо передать в ответе ключ , следовательно, так как ключ V известен мошеннику, то и обмануть сервер получается со 100% вероятностью. Но в следующем раунде мошеннику не везет и в качестве Е выпадет 1, и в этот раз сервер уже не удается обмануть.

ПРИЛОЖЕНИЕ

1. Исходный код
2. client/main.go
4. package main
5. import "cryptocrouse/src/go/FiatShamirProtocol/client/clnt"
6. func main() {
7. client := clnt.Client{}
8. client.ConnectToServer()
9. client.StartProof()
10. }
12. client/client.go
13. package clnt
14. import (
15. "bufio"
16. "cryptocrouse/src/go/FiatShamirProtocol"
17. "cryptocrouse/src/go/Fingerprints"
18. "fmt"
19. "log"
20. "math/big"
21. "net"
22. "os"
23. "strconv"
24. "strings"
25. "time"
26. )
28. var (
29. MIN\_P = big.NewInt(0).Exp(big.NewInt(2), big.NewInt(16), nil)
30. MAX\_P = big.NewInt(0).Exp(big.NewInt(2), big.NewInt(32), nil)
31. )
32. type Client struct {
33. conn net.Conn
34. reader \*bufio.Reader
35. writer \*bufio.Writer
36. data \*ClientData
37. }
38. type ClientData struct {
39. S \*big.Int
40. V \*big.Int
41. N \*big.Int
42. E int
43. Y \*big.Int
44. R \*big.Int
45. X \*big.Int
46. }
47. func (c \*Client) ConnectToServer() {
48. arguments := os.Args
49. if len(arguments) == 1 {
50. fmt.Println("Please provide host:port.")
51. return
52. }
53. connect := arguments[1]
54. conn, err := net.Dial("tcp", connect)
55. if err != nil {
56. fmt.Println(err)
57. return
58. }
59. log.Printf("Connect to %s\n", connect)
60. c.conn = conn
61. c.data = &ClientData{}
62. c.setupConnections()
63. }
64. func (c\* Client) setupConnections() {
65. c.reader = bufio.NewReader(c.conn)
66. c.writer = bufio.NewWriter(c.conn)
67. }
68. func (c \*Client) StartProof() {
69. for i := 0; i < 10; i++ {
70. answerCode := c.round()
71. if answerCode == false {
72. log.Fatalf("Can not proof on %d iteration\n", i)
73. return
74. }
75. }
76. }
77. func (c \*Client) round() bool {
78. c.receiveN()
79. c.generateS()
80. c.computeV()
81. c.sendV()
82. c.generateR()
83. c.computeX()
84. c.sendX()
85. c.receiveE()
86. c.computeY()
87. c.sendY()
88. return c.getAnswer()
89. }
90. func (c \*Client) receiveN() {
91. \_, err := c.writer.WriteString(FiatShamirProtocol.COMMAND\_GET\_N + "\n")
92. if err != nil {
93. log.Fatal(err)
94. }
95. err = c.writer.Flush()
96. if err != nil {
97. log.Fatal(err)
98. }
99. time.Sleep(50 \* time.Millisecond)
100. msg, err := c.reader.ReadString('\n')
101. if err != nil {
102. log.Fatal(err)
103. }
104. msg = strings.TrimSuffix(msg, "\n")
105. log.Printf("received N: ~%s~\n", msg)
106. var flag bool
107. c.data.N, flag = big.NewInt(0).SetString(msg, 10)
108. if flag == false {
109. log.Fatal("Received N is bad")
110. }
111. }
112. func (c \*Client) generateS() {
113. for {
114. c.data.S = Fingerprints.GetBigRandomWithLimit(c.data.N)
115. if c.data.S.Cmp(big.NewInt(1)) == 0 {
116. continue
117. }
118. GCD := big.NewInt(0).GCD(
119. nil,
120. nil,
121. c.data.S,
122. c.data.N)
123. if GCD.Cmp(big.NewInt(1)) == 0 {
124. break
125. }
126. }
127. }
128. func (c \*Client) computeV() {
129. c.data.V = big.NewInt(0).Exp(c.data.S, big.NewInt(2), c.data.N)
130. }
131. func (c \*Client) receiveE() {
132. \_, \_ = c.writer.WriteString(FiatShamirProtocol.COMMAND\_GET\_E + "\n")
133. \_ = c.writer.Flush()
134. time.Sleep(50 \* time.Millisecond)
135. msg, \_ := c.reader.ReadString('\n')
136. msg = strings.TrimSuffix(msg, "\n")
137. log.Printf("received E: %s\n", msg)
138. c.data.E, \_ = strconv.Atoi(msg)
139. }
140. func (c \*Client) computeY() {
141. switch c.data.E {
142. case 0:
143. c.data.Y = c.data.R
144. case 1:
145. c.data.Y = big.NewInt(0).Mod(
146. big.NewInt(0).Mul(
147. c.data.S,
148. c.data.R),
149. c.data.N)
150. }
151. }
152. func (c \*Client) generateR() {
153. for {
154. c.data.R = Fingerprints.GetBigRandomWithLimit(c.data.N)
155. if c.data.R.Cmp(big.NewInt(1)) > 0 && c.data.R.Cmp(c.data.N) < 0 {
156. break
157. }
158. }
159. }
160. func (c \*Client) computeX() {
161. c.data.X = big.NewInt(0).Exp(c.data.R, big.NewInt(2), c.data.N)
162. }
163. func (c \*Client) sendX() {
164. \_, \_ = c.writer.WriteString(FiatShamirProtocol.COMMAND\_GET\_X + "\n")
165. \_ = c.writer.Flush()
166. log.Println("Send " + FiatShamirProtocol.COMMAND\_GET\_X)
167. time.Sleep(50 \* time.Millisecond)
168. \_, \_ = c.writer.WriteString(c.data.X.Text(10) + "\n")
169. \_ = c.writer.Flush()
170. log.Printf("Send X %s\n", c.data.X.Text(10))
171. time.Sleep(50 \* time.Millisecond)
172. }
173. func (c \*Client) sendY() {
174. \_, \_ = c.writer.WriteString(FiatShamirProtocol.COMMAND\_GET\_Y + "\n")
175. \_ = c.writer.Flush()
176. log.Println("Send " + FiatShamirProtocol.COMMAND\_GET\_Y)
177. time.Sleep(50 \* time.Millisecond)
178. \_, \_ = c.writer.WriteString(c.data.Y.Text(10) + "\n")
179. \_ = c.writer.Flush()
180. log.Printf("Send Y %s\n", c.data.Y.Text(10))
181. time.Sleep(50 \* time.Millisecond)
182. }
183. func (c \*Client) sendV() {
184. \_, \_ = c.writer.WriteString(FiatShamirProtocol.COMMAND\_GET\_V + "\n")
185. \_ = c.writer.Flush()
186. log.Println("Send " + FiatShamirProtocol.COMMAND\_GET\_V)
187. time.Sleep(50 \* time.Millisecond)
188. \_, \_ = c.writer.WriteString(c.data.V.Text(10) + "\n")
189. \_ = c.writer.Flush()
190. log.Printf("Send V %s\n", c.data.V.Text(10))
191. time.Sleep(50 \* time.Millisecond)
192. }
193. func (c \*Client) getAnswer() bool {
194. log.Println("Wait answer")
195. msg, err := c.reader.ReadString('\n')
196. if err != nil {
197. log.Fatal(err)
198. }
199. msg = strings.TrimSuffix(msg, "\n")
200. log.Printf("received answer: ~%s~\n", msg)
201. switch msg {
202. case FiatShamirProtocol.COMMAND\_ANSWER\_CODE\_SUCCESS:
203. log.Println("Round ok")
204. return true
205. case FiatShamirProtocol.COMMAND\_ANSWER\_CODE\_ERROR:
206. log.Println("Round bad")
207. return false
208. default:
209. log.Println("Round fi")
210. return false
211. }
212. }
214. server/main.go
216. package main
217. import "cryptocrouse/src/go/FiatShamirProtocol/server/srvr"
218. func main() {
219. server := srvr.ServerInit()
220. server.Run()
221. }
223. server/server.go
225. package srvr
226. const (
227. CONN\_HOST = "localhost"
228. CONN\_PORT = "7575"
229. CONN\_TYPE = "tcp"
230. )
231. var (
232. MIN\_P = big.NewInt(0).Exp(big.NewInt(2), big.NewInt(16), nil)
233. MAX\_P = big.NewInt(0).Exp(big.NewInt(2), big.NewInt(32), nil)
234. )
236. import (
237. "bufio"
238. "cryptocrouse/src/go/FiatShamirProtocol"
239. "cryptocrouse/src/go/Fingerprints"
240. "log"
241. "math/big"
242. "net"
243. "strconv"
244. "strings"
245. "time"
246. )
247. type Server struct {
248. data ServerData
249. }
250. type ServerData struct {
251. p \*big.Int
252. q \*big.Int
253. N \*big.Int
254. }
255. func ServerInit() \*Server {
256. return &Server{}
257. }
258. func (s \*Server) Run() {
259. s.serverPrepare()
260. log.Printf("Server up with\n")
261. log.Printf("N=%s\n", s.data.N.Text(10))
262. s.serverListen()
263. }
264. func (s \*Server) serverPrepare() {
265. s.data.generateP()
266. s.data.computeN()
267. }
268. func (s \*Server) serverListen() {
269. l, err := net.Listen(CONN\_TYPE, CONN\_HOST+ ":" +CONN\_PORT)
270. if err != nil {
271. log.Fatalf("Error listening:", err.Error())
272. }
273. defer l.Close()
274. log.Println("Listening on " + CONN\_HOST + ":" + CONN\_PORT)
275. for {
276. conn, err := l.Accept()
277. if err != nil {
278. log.Fatalf("Error accepting: %s\n", err.Error())
279. }
280. log.Printf("New connection: %s\n", conn.RemoteAddr())
281. go s.startRound(conn)
282. }
283. }
284. func (s \*Server) startRound(conn net.Conn) {
285. r := bufio.NewReader(conn)
286. w := bufio.NewWriter(conn)
287. scanr := bufio.NewScanner(r)
288. var x \*big.Int
289. var y \*big.Int
290. var v \*big.Int
291. e := generateE()
292. for {
293. scanned := scanr.Scan()
294. if !scanned {
295. if err := scanr.Err(); err != nil {
296. log.Printf("%v(%v)\n", err, conn.RemoteAddr())
297. return
298. }
299. break
300. }
301. msg := scanr.Text()
302. msg = strings.TrimSuffix(msg, "\n")
303. log.Printf("Reveived [%s]: ~%s~\n", conn.RemoteAddr(), msg)
304. switch msg {
305. case FiatShamirProtocol.COMMAND\_GET\_N:
306. s.sendN(w)
307. case FiatShamirProtocol.COMMAND\_GET\_X:
308. x = s.receiveX(r)
309. case FiatShamirProtocol.COMMAND\_GET\_Y:
310. y = s.receiveY(r)
311. statusCode := s.computeY(y, x, v, w, e)
312. s.sendAnswerCode(w, statusCode)
313. case FiatShamirProtocol.COMMAND\_GET\_V:
314. v = s.receiveV(r)
315. case FiatShamirProtocol.COMMAND\_GET\_E:
316. s.sendE(w, e)
317. }
318. }
319. }
320. func (s \*Server) sendN(w \*bufio.Writer) {
321. log.Printf("Send N: %s\n", s.data.N.Text(10))
322. \_, \_ = w.WriteString(s.data.N.Text(10) + "\n")
323. \_ = w.Flush()
324. }
325. func (s \*Server) receiveX(r \*bufio.Reader) \*big.Int {
326. msg, \_ := r.ReadString('\n')
327. msg = strings.TrimSuffix(msg, "\n")
328. x, \_ := big.NewInt(0).SetString(msg, 10)
329. log.Printf("Receive x: %s\n", msg)
330. return x
331. }
332. func (s \*Server) receiveV(r \*bufio.Reader) \*big.Int {
333. msg, \_ := r.ReadString('\n')
334. msg = strings.TrimSuffix(msg, "\n")
335. v, \_ := big.NewInt(0).SetString(msg, 10)
336. log.Printf("Receive v: %s\n", msg)
337. return v
338. }
339. func (s \*Server) sendE(w \*bufio.Writer, e int) {
340. \_, \_ = w.WriteString(strconv.Itoa(e) + "\n")
341. \_ = w.Flush()
342. log.Println("Send E: " + strconv.Itoa(e))
343. }
344. func (s \*Server) receiveY(r \*bufio.Reader) \*big.Int {
345. msg, \_ := r.ReadString('\n')
346. msg = strings.TrimSuffix(msg, "\n")
347. y, \_ := big.NewInt(0).SetString(msg, 10)
348. log.Printf("Receive y: %s\n", msg)
349. return y
350. }
351. func (s \*Server) computeY(y \*big.Int, x \*big.Int, v \*big.Int, w \*bufio.Writer, e int) string {
352. if y.Cmp(big.NewInt(0)) == 0 {
353. \_, \_ = w.WriteString(FiatShamirProtocol.COMMAND\_ANSWER\_CODE\_ERROR)
354. \_ = w.Flush()
355. return FiatShamirProtocol.COMMAND\_ANSWER\_CODE\_ERROR
356. }
357. l := big.NewInt(0).Exp(y, big.NewInt(2), s.data.N)
358. var r \*big.Int
359. switch e {
360. case 0:
361. r = x
362. case 1:
363. r = big.NewInt(0).Mod(
364. big.NewInt(0).Mul(
365. x,
366. v),
367. s.data.N)
368. }
369. log.Printf("l = %s r = %s\n", l.Text(10), r.Text(10))
370. code := ""
371. if l.Cmp(r) == 0 {
372. code = FiatShamirProtocol.COMMAND\_ANSWER\_CODE\_SUCCESS
373. } else {
374. code = FiatShamirProtocol.COMMAND\_ANSWER\_CODE\_ERROR
375. }
376. return code
377. }
378. func (s \*Server) sendAnswerCode(w \*bufio.Writer, statusCode string) {
379. \_, err := w.WriteString(statusCode + "\n")
380. if err != nil {
381. log.Fatal(err)
382. }
383. err = w.Flush()
384. if err != nil {
385. log.Fatal(err)
386. }
387. log.Println("Send status code: " + statusCode)
388. time.Sleep(50 \* time.Millisecond)
389. }
390. func generateE() int {
391. rand := Fingerprints.GetBigRandom()
392. answer, \_ := strconv.Atoi(big.NewInt(0).Mod(rand, big.NewInt(2)).Text(10))
393. return answer
394. }
395. func (data \*ServerData) generateQ() {
396. data.q = Fingerprints.GenerateBigPrimeNumberWithLimit(MIN\_P)
397. }
398. func (data \*ServerData) generateP() {
399. data.p = big.NewInt(0)
400. for {
401. data.generateQ()
402. data.p.Add(
403. big.NewInt(0).Mul(
404. big.NewInt(2),
405. data.q),
406. big.NewInt(1))
407. if Fingerprints.IsPrimeRef(data.p) {
408. if data.p.Cmp(MIN\_P) > 0 && data.p.Cmp(MAX\_P) < 0 {
409. break
410. }
411. }
412. }
413. }
414. func (data \*ServerData) computeN() {
415. data.N = big.NewInt(0).Mul(data.p, data.q)
416. }