

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 ИБ-621

Работу приняла \_\_\_\_\_ ассистент кафедры ПМиК  
Я.В. Петухова  
подпись

Защищена \_\_\_\_\_ Оценка \_\_\_\_\_

Новосибирск – 2019 г.

## Оглавление

1	Постановка задачи .....	3
2	Теоретические сведения .....	4
	ПРИЛОЖЕНИЕ.....	9
1.	Исходный код .....	9

## 1 Постановка задачи

В рамках расчётно-графического задания необходимо написать программу, 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 претендента, 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

## 2 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.

А доказывает В 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  $n = p * q$ , где  $p, q$  — простые и держатся в секрете
- Каждый претендент А выбирает  $s$  взаимно-простое с  $n$ , где  $s \in [1, n - 1]$ . Затем вычисляется  $V = s^2 \bmod n$ .  $V$  регистрируется Т в качестве открытого ключа А

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

- $A \Rightarrow B : x = r^2 \bmod n$
- $A \Leftarrow B : e \in 0,1$
- $A \Rightarrow B : y = r * s^e \bmod n$

### **Основные действия:**

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

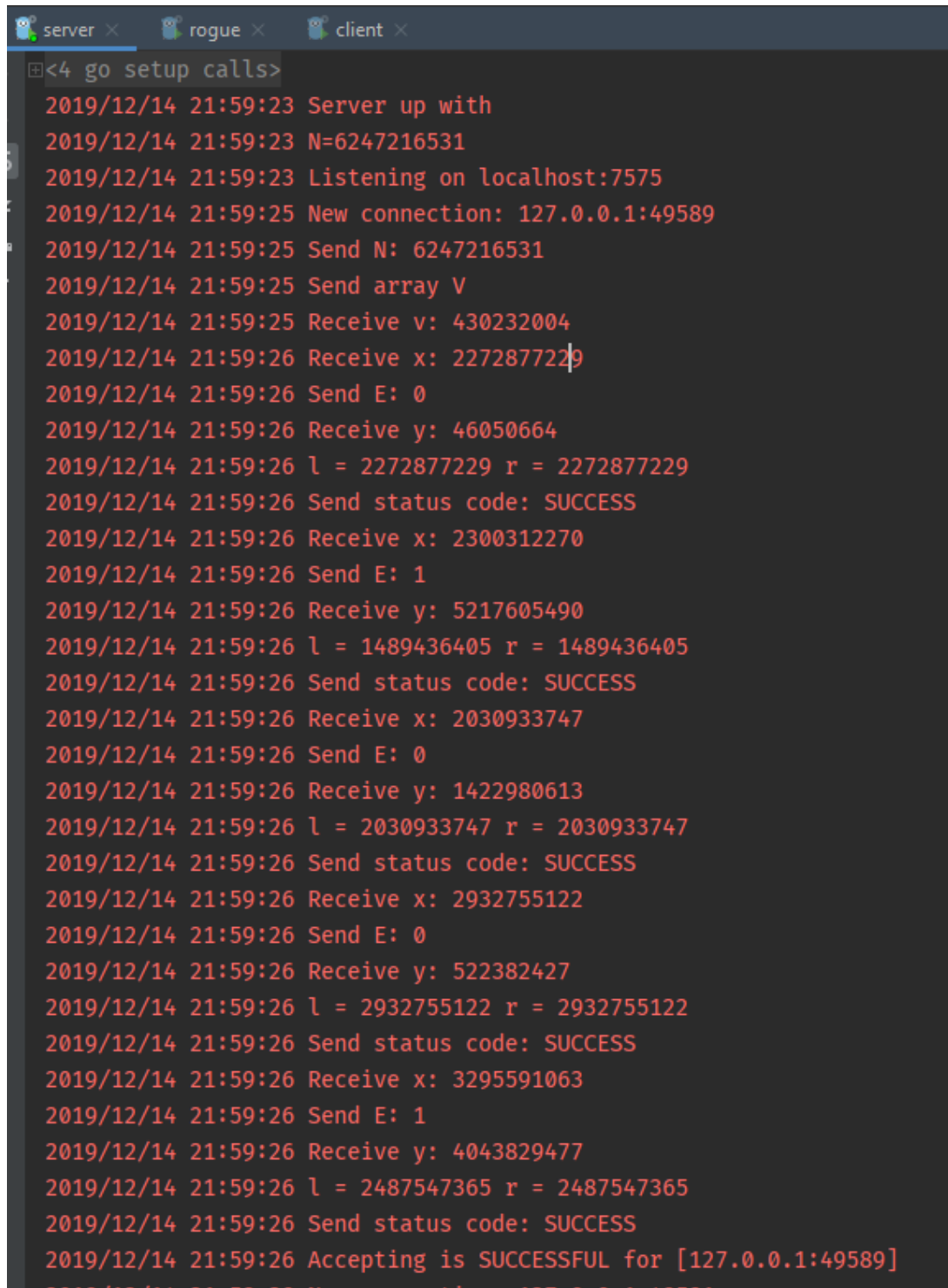
- А выбирает случайное число  $r$ , такое, что  $r \in [1, n - 1]$  и отправляет  $x = r^2 \bmod n$  стороне В (доказательство)
- В случайно выбирает бит  $e$  ( $e = 0$  или  $e = 1$ ) и отправляет его А (вызов)

- А вычисляет  $y$  и отправляет его обратно к В. Если  $e = 0$ , то  $y = r$ , иначе  $y = r * s \bmod n$  (ответ)
- Если  $y = 0$ , то В отвергает доказательство или, другими словами, А не удалось доказать знание  $s$ . В противном случае, сторона В проверяет, действительно ли  $y^2 = x * v^e \bmod n$  и, если это так, то происходит переход к следующему раунду протокола

Выбор  $e$  из множества  $\{0,1\}$  предполагает, что если сторона А действительно знает секрет, то она всегда сможет правильно ответить, вне зависимости от выбранного  $e$ . Допустим, что А хочет обмануть В. В этом случае А, может отреагировать только на конкретное значение  $e$ . Например, если А знает, что получит  $e = 0$ , то А следует действовать строго по инструкции и В примет ответ. В случае, если А знает, что получит  $e = 1$ , то А выбирает случайное  $r$  и отправляет  $x = \frac{r^2}{v}$  на сторону В, в результате получаем нам нужное  $y = r$ . Проблема заключается в том, что А изначально не знает какое  $e$  он получит и поэтому не может со 100 % вероятностью выслать на сторону В нужные для обмана  $r$  и  $x$  ( $x = r^2$  при  $e = 0$  и  $x = \frac{r^2}{v}$  при  $e = 1$ ). Поэтому вероятность обмана в одном раунде составляет 50 %. Чтобы снизить вероятность жульничества (она равна  $\frac{1}{2^t}$ )  $t$  выбирают достаточно большим ( $t = 20, t = 40$ ). Таким образом, В удостоверяется в знании А тогда и только тогда, когда все  $t$  раундов прошли успешно.

## Пример работы программы

Вывод логов сервера за одну сессию при подключении клиента:



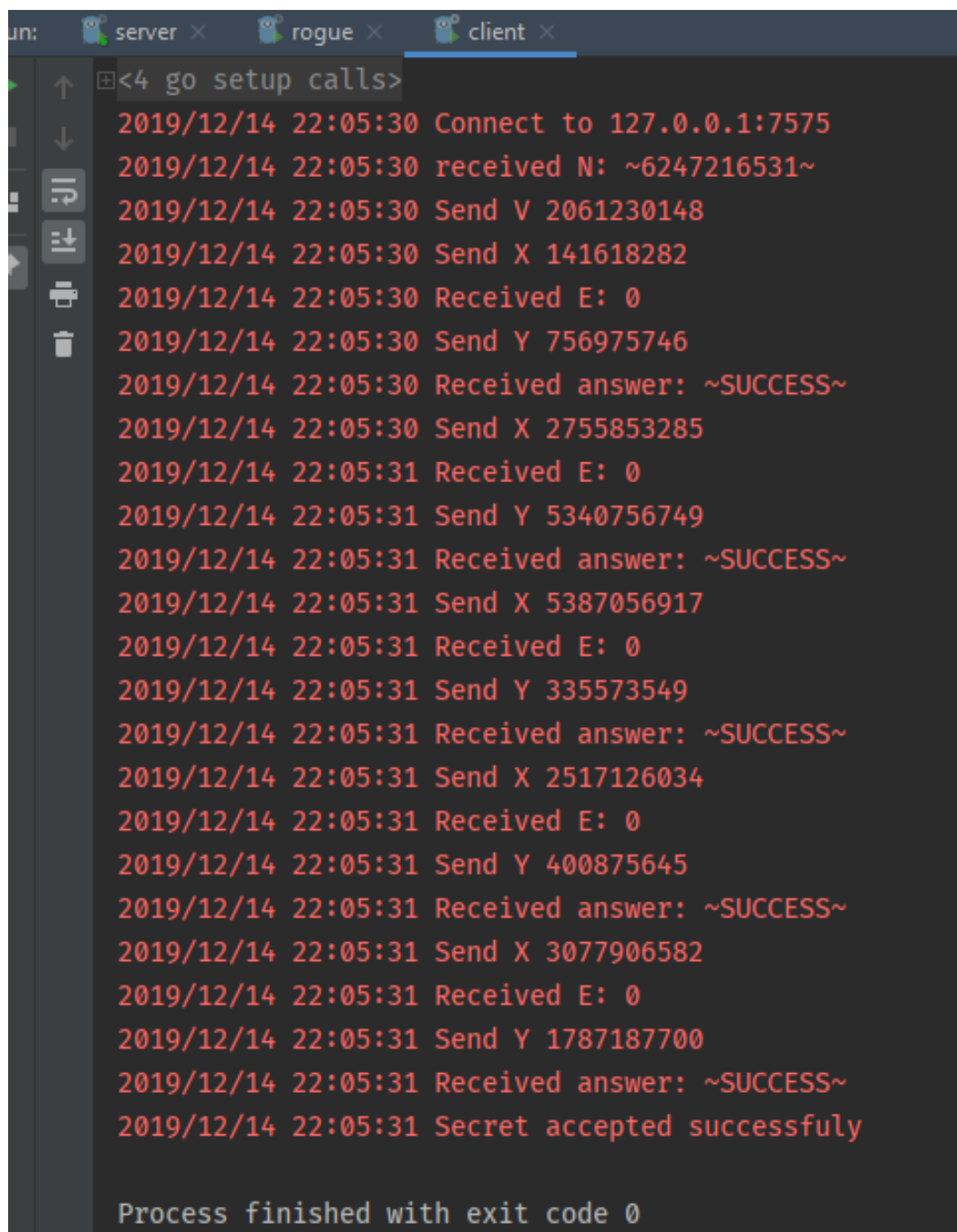
```
<4 go setup calls>
2019/12/14 21:59:23 Server up with
2019/12/14 21:59:23 N=6247216531
2019/12/14 21:59:23 Listening on localhost:7575
2019/12/14 21:59:25 New connection: 127.0.0.1:49589
2019/12/14 21:59:25 Send N: 6247216531
2019/12/14 21:59:25 Send array V
2019/12/14 21:59:25 Receive v: 430232004
2019/12/14 21:59:26 Receive x: 2272877229
2019/12/14 21:59:26 Send E: 0
2019/12/14 21:59:26 Receive y: 46050664
2019/12/14 21:59:26 l = 2272877229 r = 2272877229
2019/12/14 21:59:26 Send status code: SUCCESS
2019/12/14 21:59:26 Receive x: 2300312270
2019/12/14 21:59:26 Send E: 1
2019/12/14 21:59:26 Receive y: 5217605490
2019/12/14 21:59:26 l = 1489436405 r = 1489436405
2019/12/14 21:59:26 Send status code: SUCCESS
2019/12/14 21:59:26 Receive x: 2030933747
2019/12/14 21:59:26 Send E: 0
2019/12/14 21:59:26 Receive y: 1422980613
2019/12/14 21:59:26 l = 2030933747 r = 2030933747
2019/12/14 21:59:26 Send status code: SUCCESS
2019/12/14 21:59:26 Receive x: 2932755122
2019/12/14 21:59:26 Send E: 0
2019/12/14 21:59:26 Receive y: 522382427
2019/12/14 21:59:26 l = 2932755122 r = 2932755122
2019/12/14 21:59:26 Send status code: SUCCESS
2019/12/14 21:59:26 Receive x: 3295591063
2019/12/14 21:59:26 Send E: 1
2019/12/14 21:59:26 Receive y: 4043829477
2019/12/14 21:59:26 l = 2487547365 r = 2487547365
2019/12/14 21:59:26 Send status code: SUCCESS
2019/12/14 21:59:26 Accepting is SUCCESSFUL for [127.0.0.1:49589]
```

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 E, 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:



```
un: server x rogue x client x
<4 go setup calls>
2019/12/14 22:05:30 Connect to 127.0.0.1:7575
2019/12/14 22:05:30 received N: ~6247216531~
2019/12/14 22:05:30 Send V 2061230148
2019/12/14 22:05:30 Send X 141618282
2019/12/14 22:05:30 Received E: 0
2019/12/14 22:05:30 Send Y 756975746
2019/12/14 22:05:30 Received answer: ~SUCCESS~
2019/12/14 22:05:30 Send X 2755853285
2019/12/14 22:05:31 Received E: 0
2019/12/14 22:05:31 Send Y 5340756749
2019/12/14 22:05:31 Received answer: ~SUCCESS~
2019/12/14 22:05:31 Send X 5387056917
2019/12/14 22:05:31 Received E: 0
2019/12/14 22:05:31 Send Y 335573549
2019/12/14 22:05:31 Received answer: ~SUCCESS~
2019/12/14 22:05:31 Send X 2517126034
2019/12/14 22:05:31 Received E: 0
2019/12/14 22:05:31 Send Y 400875645
2019/12/14 22:05:31 Received answer: ~SUCCESS~
2019/12/14 22:05:31 Send X 3077906582
2019/12/14 22:05:31 Received E: 0
2019/12/14 22:05:31 Send Y 1787187700
2019/12/14 22:05:31 Received answer: ~SUCCESS~
2019/12/14 22:05:31 Secret accepted successfully

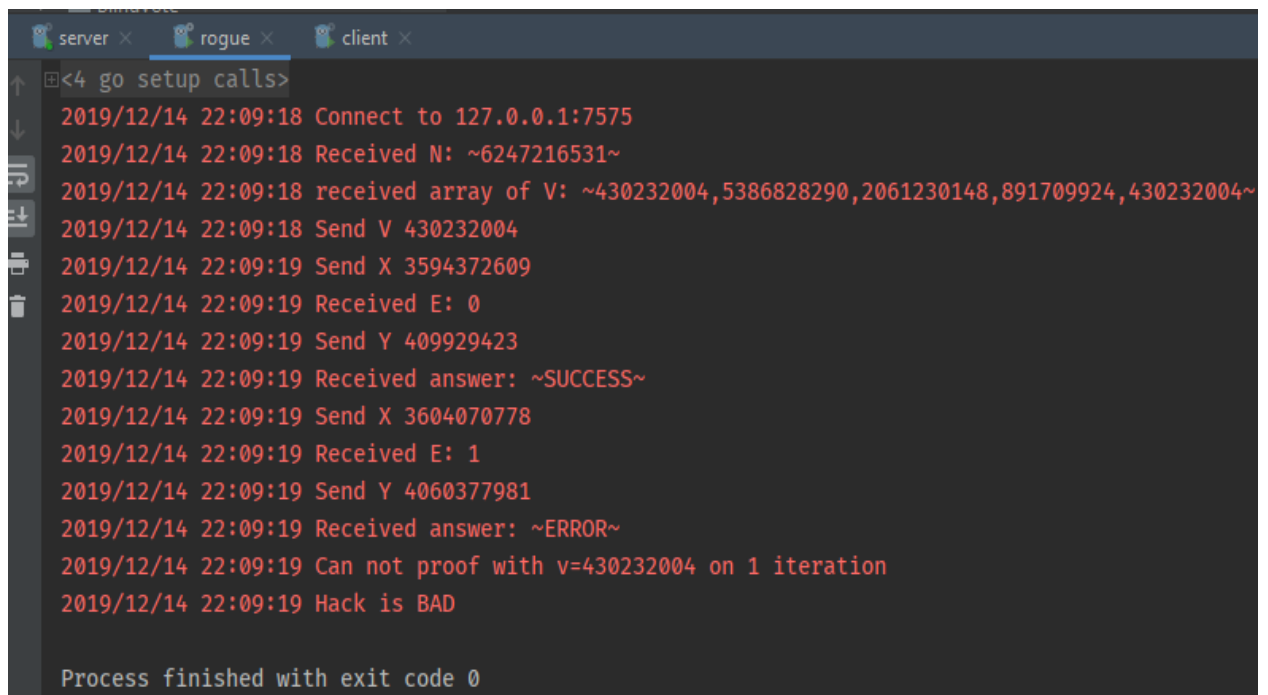
Process finished with exit code 0
```

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 раундов сервер ответит кодом ERROR, авторизация не проходит и клиент отключается.

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



```
server x  rogue x  client x
<4 go setup calls>
2019/12/14 22:09:18 Connect to 127.0.0.1:7575
2019/12/14 22:09:18 Received N: ~6247216531~
2019/12/14 22:09:18 received array of V: ~430232004,5386828290,2061230148,891709924,430232004~
2019/12/14 22:09:18 Send V 430232004
2019/12/14 22:09:19 Send X 3594372609
2019/12/14 22:09:19 Received E: 0
2019/12/14 22:09:19 Send Y 409929423
2019/12/14 22:09:19 Received answer: ~SUCCESS~
2019/12/14 22:09:19 Send X 3604070778
2019/12/14 22:09:19 Received E: 1
2019/12/14 22:09:19 Send Y 4060377981
2019/12/14 22:09:19 Received answer: ~ERROR~
2019/12/14 22:09:19 Can not proof with v=430232004 on 1 iteration
2019/12/14 22:09:19 Hack is BAD

Process finished with exit code 0
```

Рисунок 3 – Пример логов мошенника

В данных логах можно увидеть, как мошенник подключается к серверу, получает открытые ключи  $N$  и  $V$ , получает параметр  $E$  и пытается подобрать ключи  $Y$  и  $X$ . Как видно при  $E$  равным 0 у мошенника получается обмануть сервер, так как по алгоритму Фиата-Шамира в данном случае клиенту необходимо передать в ответе ключ  $y = v^2$ , 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  $E$  выпадет 1, и в этот раз сервер уже не удастся обмануть.



## ПРИЛОЖЕНИЕ

### 1. Исходный код

```
1. client/main.go
2.
3. package main
4.
5. import "cryptocrouse/src/go/FiatShamirProtocol/client/clnt"
6.
7. func main() {
8.     client := clnt.Client{}
9.     client.ConnectToServer()
10.    client.StartProof()
11. }
12.
13. client/client.go
14.
15. type Client struct {
16.     conn net.Conn
17.     reader *bufio.Reader
18.     writer *bufio.Writer
19.     data *ClientData
20. }
21.
22. type ClientData struct {
23.     S *big.Int
24.     V *big.Int
25.     N *big.Int
26.     E int
27.     Y *big.Int
28.     R *big.Int
29.     X *big.Int
30.     arrayV []*big.Int
31. }
32.
33. func (c *Client) ConnectToServer() {
34.     arguments := os.Args
35.     if len(arguments) == 1 {
36.         fmt.Println("Please provide host:port.")
37.         return
38.     }
39.
40.     connect := arguments[1]
41.     conn, err := net.Dial("tcp", connect)
42.     if err != nil {
43.         fmt.Println(err)
44.         return
45.     }
46.     log.Printf("Connect to %s\n", connect)
47.     c.conn = conn
48.
49.     c.data = &ClientData{}
50.     c.setupConnections()
51. }
52.
53. func (c* Client) setupConnections() {
54.     c.reader = bufio.NewReader(c.conn)
55.     c.writer = bufio.NewWriter(c.conn)
56. }
57.
58. func (c *Client) StartProof() {
59.     c.receiveN()
60.     c.receiveV()
61.     c.generateS()
62.     c.computeV()
63.     c.sendV()
64.
65.     for i := 0; i < 5; i++ {
66.         answerCode := c.round()
67.         if answerCode == false {
68.             log.Fatalf("Can not proof on %d iteration\n", i)
69.             return
70.         }
71.     }
```

```

72.     log.Printf("Secret accepted successfully")
73.     c.sendEnd()
74. }
75.
76. func (c *Client) round() bool {
77.     c.generateR()
78.     c.computeX()
79.     c.sendX()
80.
81.     c.receiveE()
82.
83.     c.computeY()
84.     c.sendY()
85.
86.     return c.getAnswer()
87. }
88.
89. func (c *Client) receiveN() {
90.     time.Sleep(50 * time.Millisecond)
91.
92.     msg, err := c.reader.ReadString('\n')
93.     if err != nil {
94.         log.Fatal(err)
95.     }
96.
97.     msg = strings.TrimSuffix(msg, "\n")
98.     log.Printf("received N: %s~\n", msg)
99.
100.    var flag bool
101.    c.data.N, flag = big.NewInt(0).SetString(msg, 10)
102.    if flag == false {
103.        log.Fatal("Received N is bad")
104.    }
105.}
106.
107.func (c *Client) generateS() {
108.    for {
109.        c.data.S = Fingerprints.GetBigRandomWithLimit(c.data.N)
110.        if c.data.S.Cmp(big.NewInt(1)) == 0 {
111.            continue
112.        }
113.        GCD := big.NewInt(0).GCD(
114.            nil,
115.            nil,
116.            c.data.S,
117.            c.data.N)
118.        if GCD.Cmp(big.NewInt(1)) == 0 {
119.            break
120.        }
121.    }
122.}
123.
124.func (c *Client) computeV() {
125.    c.data.V = big.NewInt(0).Exp(c.data.S, big.NewInt(2), c.data.N)
126.}
127.
128.func (c *Client) receiveE() {
129.    time.Sleep(50 * time.Millisecond)
130.
131.    msg, _ := c.reader.ReadString('\n')
132.    msg = strings.TrimSuffix(msg, "\n")
133.    log.Printf("Received E: %s\n", msg)
134.
135.    c.data.E, _ = strconv.Atoi(msg)
136.}
137.
138.func (c *Client) computeY() {
139.    switch c.data.E {
140.    case 0:
141.        c.data.Y = big.NewInt(0).Mod(c.data.R, c.data.N)
142.    case 1:
143.        c.data.Y = big.NewInt(0).Mod(
144.            big.NewInt(0).Mul(
145.                c.data.S,
146.                c.data.R),
147.            c.data.N)
148.    }

```

```

149.}
150.
151.func (c *Client) generateR() {
152.    for {
153.        c.data.R = Fingerprints.GetBigRandomWithLimit(c.data.N)
154.        if c.data.R.Cmp(big.NewInt(1)) > 0 && c.data.R.Cmp(c.data.N) < 0 {
155.            break
156.        }
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.
164.func (c *Client) sendX() {
165.    time.Sleep(50 * time.Millisecond)
166.    _, _ = c.writer.WriteString(c.data.X.Text(10) + "\n")
167.    _ = c.writer.Flush()
168.    log.Printf("Send X %s\n", c.data.X.Text(10))
169.}
170.
171.func (c *Client) sendY() {
172.    time.Sleep(50 * time.Millisecond)
173.    _, _ = c.writer.WriteString(c.data.Y.Text(10) + "\n")
174.    _ = c.writer.Flush()
175.    log.Printf("Send Y %s\n", c.data.Y.Text(10))
176.}
177.
178.func (c *Client) sendV() {
179.    time.Sleep(50 * time.Millisecond)
180.    _, _ = c.writer.WriteString(c.data.V.Text(10) + "\n")
181.    _ = c.writer.Flush()
182.    log.Printf("Send V %s\n", c.data.V.Text(10))
183.}
184.
185.func (c *Client) getAnswer() bool {
186.    msg, err := c.reader.ReadString('\n')
187.    if err != nil {
188.        log.Fatal(err)
189.    }
190.
191.    msg = strings.TrimSuffix(msg, "\n")
192.    log.Printf("Received answer: ~%s~\n", msg)
193.
194.    switch msg {
195.    case FiatShamirProtocol.COMMAND_ANSWER_CODE_SUCCESS:
196.        return true
197.    case FiatShamirProtocol.COMMAND_ANSWER_CODE_ERROR:
198.        return false
199.    default:
200.        return false
201.    }
202.}
203.
204.func (c *Client) sendEnd() {
205.    _, _ = c.writer.WriteString(FiatShamirProtocol.COMMAND_END + "\n")
206.    _ = c.writer.Flush()
207.}
208.
209.func (c *Client) receiveV() {
210.    time.Sleep(50 * time.Millisecond)
211.
212.    _, err := c.reader.ReadString('\n')
213.    if err != nil {
214.        log.Fatal(err)
215.    }
216.}
217.
218.server/main.go
219.
220.package main
221.
222.import "cryptocrouse/src/go/FiatShamirProtocol/server/srvr"
223.
224.func main() {
225.    server := srvr.ServerInit()

```

```

226.     server.Run()
227.}
228.
229.server/server.go
230.
231.package srvr
232.
233.const (
234.     CONN_HOST = "localhost"
235.     CONN_PORT = "7575"
236.     CONN_TYPE = "tcp"
237.)
238.
239.var (
240.     MIN_P = big.NewInt(0).Exp(big.NewInt(2), big.NewInt(16), nil)
241.     MAX_P = big.NewInt(0).Exp(big.NewInt(2), big.NewInt(32), nil)
242.)
243.
244.import (
245.     "bufio"
246.     "cryptocrouse/src/go/FiatShamirProtocol"
247.     "cryptocrouse/src/go/Fingerprints"
248.     "log"
249.     "math/big"
250.     "net"
251.     "strconv"
252.     "strings"
253.     "time"
254.)
255.
256.type Server struct {
257.     data ServerData
258.}
259.
260.type ServerData struct {
261.     p *big.Int
262.     q *big.Int
263.     N *big.Int
264.     V []*big.Int
265.}
266.
267.func ServerInit() *Server {
268.     data := ServerData{
269.         p: big.NewInt(0),
270.         q: big.NewInt(0),
271.         N: big.NewInt(0),
272.         V: make([]*big.Int, 0),
273.     }
274.     return &Server{
275.         data: data,
276.     }
277.}
278.
279.func (s *Server) Run() {
280.     s.serverPrepare()
281.     log.Printf("Server up with\n")
282.     log.Printf("N=%s\n", s.data.N.Text(10))
283.     s.serverListen()
284.}
285.
286.func (s *Server) serverPrepare() {
287.     s.data.generateP()
288.     s.data.computeN()
289.}
290.
291.func (s *Server) serverListen() {
292.     l, err := net.Listen(CONN_TYPE, CONN_HOST+ ":" +CONN_PORT)
293.     if err != nil {
294.         log.Fatalf("Error listening:", err.Error())
295.     }
296.     defer l.Close()
297.
298.     log.Println("Listening on " + CONN_HOST + ":" + CONN_PORT)
299.
300.     for {
301.         conn, err := l.Accept()
302.         if err != nil {

```

```

303.             log.Fatalf("Error accepting: %s\n", err.Error())
304.         }
305.
306.         log.Printf("New connection: %s\n", conn.RemoteAddr())
307.
308.         go s.startRound(conn)
309.     }
310.}
311.
312.func (s *Server) startRound(conn net.Conn) {
313.    r := bufio.NewReader(conn)
314.    w := bufio.NewWriter(conn)
315.
316.    var x *big.Int
317.    var y *big.Int
318.
319.    s.sendN(w)
320.    s.sendV(w)
321.    v := s.receiveV(r)
322.    if v != nil {
323.        s.data.V = append(s.data.V, v)
324.    }
325.
326.    for t := 0; t < 5; t++ {
327.        x = s.receiveX(r)
328.        e := generateE()
329.        s.sendE(w, e)
330.        y = s.receiveY(r)
331.        if x == nil || v == nil || y == nil {
332.            return
333.        }
334.        statusCode := s.computeY(y, x, v, w, e)
335.        s.sendAnswerCode(w, statusCode)
336.        if statusCode == FiatShamirProtocol.COMMAND_ANSWER_CODE_ERROR {
337.            log.Printf("Accepting is BAD for [%d]\n", conn.RemoteAddr())
338.            return
339.        }
340.    }
341.    log.Printf("Accepting is SUCCESSFUL for [%v]\n", conn.RemoteAddr())
342.}
343.
344.func (s *Server) sendN(w *bufio.Writer) {
345.    log.Printf("Send N: %s\n", s.data.N.Text(10))
346.    _, _ = w.WriteString(s.data.N.Text(10) + "\n")
347.    _ = w.Flush()
348.}
349.
350.func (s *Server) receiveX(r *bufio.Reader) *big.Int {
351.    msg, _ := r.ReadString('\n')
352.    msg = strings.TrimSuffix(msg, "\n")
353.    x, _ := big.NewInt(0).SetString(msg, 10)
354.    log.Printf("Receive x: %s\n", msg)
355.    return x
356.}
357.
358.func (s *Server) receiveV(r *bufio.Reader) *big.Int {
359.    msg, _ := r.ReadString('\n')
360.    msg = strings.TrimSuffix(msg, "\n")
361.    v, _ := big.NewInt(0).SetString(msg, 10)
362.    log.Printf("Receive v: %s\n", msg)
363.    return v
364.}
365.
366.func (s *Server) sendE(w *bufio.Writer, e int) {
367.    _, _ = w.WriteString(strconv.Itoa(e) + "\n")
368.    _ = w.Flush()
369.    log.Println("Send E: " + strconv.Itoa(e))
370.}
371.
372.func (s *Server) receiveY(r *bufio.Reader) *big.Int {
373.    msg, _ := r.ReadString('\n')
374.    msg = strings.TrimSuffix(msg, "\n")
375.    y, _ := big.NewInt(0).SetString(msg, 10)
376.    log.Printf("Receive y: %s\n", msg)
377.    return y
378.}
379.

```

```

380.func (s *Server) computeY(y *big.Int, x *big.Int, v *big.Int, w *bufio.Writer, e int) string {
381.    if y.Cmp(big.NewInt(0)) == 0 {
382.        _, _ = w.WriteString(FiatShamirProtocol.COMMAND_ANSWER_CODE_ERROR)
383.        _ = w.Flush()
384.        return FiatShamirProtocol.COMMAND_ANSWER_CODE_ERROR
385.    }
386.
387.    l := big.NewInt(0).Exp(y, big.NewInt(2), s.data.N)
388.    var r *big.Int
389.
390.    switch e {
391.    case 0:
392.        r = big.NewInt(0).Mod(x, s.data.N)
393.    case 1:
394.        r = big.NewInt(0).Mod(
395.            big.NewInt(0).Mul(
396.                x,
397.                v),
398.            s.data.N)
399.    }
400.
401.    log.Printf("l = %s r = %s\n", l.Text(10), r.Text(10))
402.
403.    code := ""
404.
405.    if l.Cmp(r) == 0 {
406.        code = FiatShamirProtocol.COMMAND_ANSWER_CODE_SUCCESS
407.    } else {
408.        code = FiatShamirProtocol.COMMAND_ANSWER_CODE_ERROR
409.    }
410.
411.    return code
412.}
413.
414.func (s *Server) sendAnswerCode(w *bufio.Writer, statusCode string) {
415.    _, err := w.WriteString(statusCode + "\n")
416.    if err != nil {
417.        log.Fatal(err)
418.    }
419.    err = w.Flush()
420.    if err != nil {
421.        log.Fatal(err)
422.    }
423.    log.Println("Send status code: " + statusCode)
424.    time.Sleep(50 * time.Millisecond)
425.}
426.
427.func (s *Server) sendV(w *bufio.Writer) {
428.    vString := ""
429.    for _, v := range s.data.V {
430.        vString += v.Text(10) + ","
431.    }
432.
433.    vString = strings.TrimSuffix(vString, ",")
434.    _, _ = w.WriteString(vString + "\n")
435.    _ = w.Flush()
436.    log.Printf("Send array V %s\n", vString)
437.}
438.
439.func generateE() int {
440.    rand := Fingerprints.GetBigRandom()
441.    answer, _ := strconv.Atoi(big.NewInt(0).Mod(rand, big.NewInt(2)).Text(10))
442.    return answer
443.}
444.
445.func (data *ServerData) generateQ() {
446.    data.q = Fingerprints.GenerateBigPrimeNumberWithLimit(MIN_P)
447.}
448.
449.func (data *ServerData) generateP() {
450.    data.p = big.NewInt(0)
451.
452.    for {
453.        data.generateQ()
454.        data.p.Add(
455.            big.NewInt(0).Mul(
456.                big.NewInt(2),

```

```

457.             data.q),
458.             big.NewInt(1))
459.         if Fingerprints.IsPrimeRef(data.p) {
460.             if data.p.Cmp(MIN_P) > 0 && data.p.Cmp(MAX_P) < 0 {
461.                 break
462.             }
463.         }
464.     }
465.}
466.
467.func (data *ServerData) computeN() {
468.    data.N = big.NewInt(0).Mul(data.p, data.q)
469.}
470.
471.package rg
472.
473.import (
474.    "bufio"
475.    "cryptocrouse/src/go/FiatShamirProtocol"
476.    "cryptocrouse/src/go/Fingerprints"
477.    "fmt"
478.    "log"
479.    "math/big"
480.    "net"
481.    "os"
482.    "strconv"
483.    "strings"
484.    "time"
485.)
486.
487.type Rogue struct {
488.    conn net.Conn
489.    reader *bufio.Reader
490.    writer *bufio.Writer
491.    data *RogueData
492.}
493.
494.type RogueData struct {
495.    singleV *big.Int
496.    V []*big.Int
497.    N *big.Int
498.    R *big.Int
499.    X *big.Int
500.    E int
501.    Y *big.Int
502.    S *big.Int
503.}
504.
505.func InitRogue() *Rogue {
506.    data := &RogueData{
507.        V: make([]*big.Int, 0),
508.        N: big.NewInt(0),
509.    }
510.    return &Rogue{
511.        data: data,
512.    }
513.}
514.
515.func (r *Rogue) ConnectToServer() {
516.    arguments := os.Args
517.    if len(arguments) == 1 {
518.        fmt.Println("Please provide host:port.")
519.        return
520.    }
521.
522.    connect := arguments[1]
523.    conn, err := net.Dial("tcp", connect)
524.    if err != nil {
525.        fmt.Println(err)
526.        return
527.    }
528.    log.Printf("Connect to %s\n", connect)
529.    r.conn = conn
530.
531.    r.data = &RogueData{}
532.    r.setupConnections()
533.}

```

```

534.
535.func (r *Rogue) setupConnections() {
536.    r.reader = bufio.NewReader(r.conn)
537.    r.writer = bufio.NewWriter(r.conn)
538.}
539.
540.func (r *Rogue) TryToAcceptSecret() {
541.    r.receiveOpenKeys()
542.
543.    r.generateS()
544.
545.    var v *big.Int
546.    for i := 0; i < len(r.data.V); i++ {
547.        v = r.data.V[i]
548.        if v != nil {
549.            r.data.singleV = v
550.            break
551.        }
552.    }
553.
554.    flag := r.hackSecret(v)
555.    if flag {
556.        log.Println("Hack is SUCCESSFUL")
557.        return
558.    }
559.    log.Println("Hack is BAD")
560.}
561.
562.func (r *Rogue) receiveOpenKeys() {
563.    r.receiveN()
564.    r.receiveV()
565.}
566.
567.func (r *Rogue) receiveN() {
568.    time.Sleep(50 * time.Millisecond)
569.
570.    msg, err := r.reader.ReadString('\n')
571.    if err != nil {
572.        log.Fatal(err)
573.    }
574.
575.    msg = strings.TrimSuffix(msg, "\n")
576.    log.Printf("Received N: ~%s~\n", msg)
577.
578.    var flag bool
579.    r.data.N, flag = big.NewInt(0).SetString(msg, 10)
580.    if flag == false {
581.        log.Fatal("Received N is bad")
582.    }
583.}
584.
585.func (r *Rogue) receiveV() {
586.    time.Sleep(50 * time.Millisecond)
587.
588.    msg, err := r.reader.ReadString('\n')
589.    if err != nil {
590.        log.Fatal(err)
591.    }
592.
593.    msg = strings.TrimSuffix(msg, "\n")
594.    log.Printf("received array of V: ~%s~\n", msg)
595.
596.    vString := strings.Split(msg, ",")
597.    for _, vstr := range vString {
598.        var flag bool
599.        v, flag := big.NewInt(0).SetString(vstr, 10)
600.        if flag == false {
601.            log.Fatal("Received N is bad")
602.        }
603.        r.data.V = append(r.data.V, v)
604.    }
605.}
606.
607.func (r *Rogue) hackSecret(v *big.Int) bool {
608.    r.sendV()
609.    for i := 0; i < 5; i++ {
610.        if !r.round() {

```



```

611.             log.Printf("Can not proof with v=%s on %d iteration\n", v.Text(10), i)
612.             return false
613.         }
614.     }
615.     log.Printf("Secret accepted successfully")
616.     r.sendEnd()
617.     return true
618.}
619.
620.func (r *Rogue) round() bool {
621.    r.generateR()
622.    r.computeX()
623.    r.sendX()
624.
625.    r.receiveE()
626.    r.computeY()
627.    r.sendY()
628.    return r.getAnswer()
629.}
630.
631.func (r *Rogue) generateR() {
632.    for {
633.        r.data.R = Fingerprints.GetBigRandomWithLimit(r.data.N)
634.        if r.data.R.Cmp(big.NewInt(1)) > 0 && r.data.R.Cmp(r.data.N) < 0 {
635.            break
636.        }
637.    }
638.}
639.
640.func (r *Rogue) computeX() {
641.    r.data.X = big.NewInt(0).Exp(r.data.R, big.NewInt(2), r.data.N)
642.}
643.
644.func (r *Rogue) sendX() {
645.    time.Sleep(50 * time.Millisecond)
646.
647.    _, _ = r.writer.WriteString(r.data.X.Text(10) + "\n")
648.    _ = r.writer.Flush()
649.    log.Printf("Send X %s\n", r.data.X.Text(10))
650.
651.    time.Sleep(50 * time.Millisecond)
652.}
653.
654.func (r *Rogue) receiveE() {
655.    time.Sleep(50 * time.Millisecond)
656.
657.    msg, _ := r.reader.ReadString('\n')
658.    msg = strings.TrimSuffix(msg, "\n")
659.    log.Printf("Received E: %s\n", msg)
660.
661.    r.data.E, _ = strconv.Atoi(msg)
662.}
663.
664.func (r *Rogue) computeY() {
665.    switch r.data.E {
666.    case 0:
667.        r.data.Y = r.data.R
668.    case 1:
669.        r.data.Y = big.NewInt(0).Mod(
670.            big.NewInt(0).Mul(
671.                r.data.S,
672.                r.data.R),
673.            r.data.N)
674.    }
675.}
676.
677.func (r *Rogue) generateS() {
678.    for {
679.        r.data.S = Fingerprints.GetBigRandomWithLimit(r.data.N)
680.        if r.data.S.Cmp(big.NewInt(1)) == 0 {
681.            continue
682.        }
683.        GCD := big.NewInt(0).GCD(
684.            nil,
685.            nil,
686.            r.data.S,
687.            r.data.N)

```

```

688.         if GCD.Cmp(big.NewInt(1)) == 0 {
689.             break
690.         }
691.     }
692.}
693.
694.func (r *Rogue) sendY() {
695.    time.Sleep(50 * time.Millisecond)
696.
697.    _, _ = r.writer.WriteString(r.data.Y.Text(10) + "\n")
698.    _ = r.writer.Flush()
699.    log.Printf("Send Y %s\n", r.data.Y.Text(10))
700.
701.    time.Sleep(50 * time.Millisecond)
702.}
703.
704.func (r *Rogue) getAnswer() bool {
705.    msg, err := r.reader.ReadString('\n')
706.    if err != nil {
707.        log.Fatal(err)
708.    }
709.
710.    msg = strings.TrimSuffix(msg, "\n")
711.    log.Printf("Received answer: ~%s~\n", msg)
712.
713.    switch msg {
714.    case FiatShamirProtocol.COMMAND_ANSWER_CODE_SUCCESS:
715.        return true
716.    case FiatShamirProtocol.COMMAND_ANSWER_CODE_ERROR:
717.        return false
718.    default:
719.        return false
720.    }
721.}
722.
723.func (r *Rogue) sendEnd() {
724.    _, _ = r.writer.WriteString(FiatShamirProtocol.COMMAND_END + "\n")
725.    _ = r.writer.Flush()
726.}
727.
728.func (r *Rogue) computeV() {
729.    r.data.singleV = big.NewInt(0).Exp(r.data.S, big.NewInt(2), r.data.N)
730.}
731.
732.func (r *Rogue) sendV() {
733.    time.Sleep(50 * time.Millisecond)
734.    _, _ = r.writer.WriteString(r.data.singleV.Text(10) + "\n")
735.    _ = r.writer.Flush()
736.    log.Printf("Send V %s\n", r.data.singleV.Text(10))
737.}

```