

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.

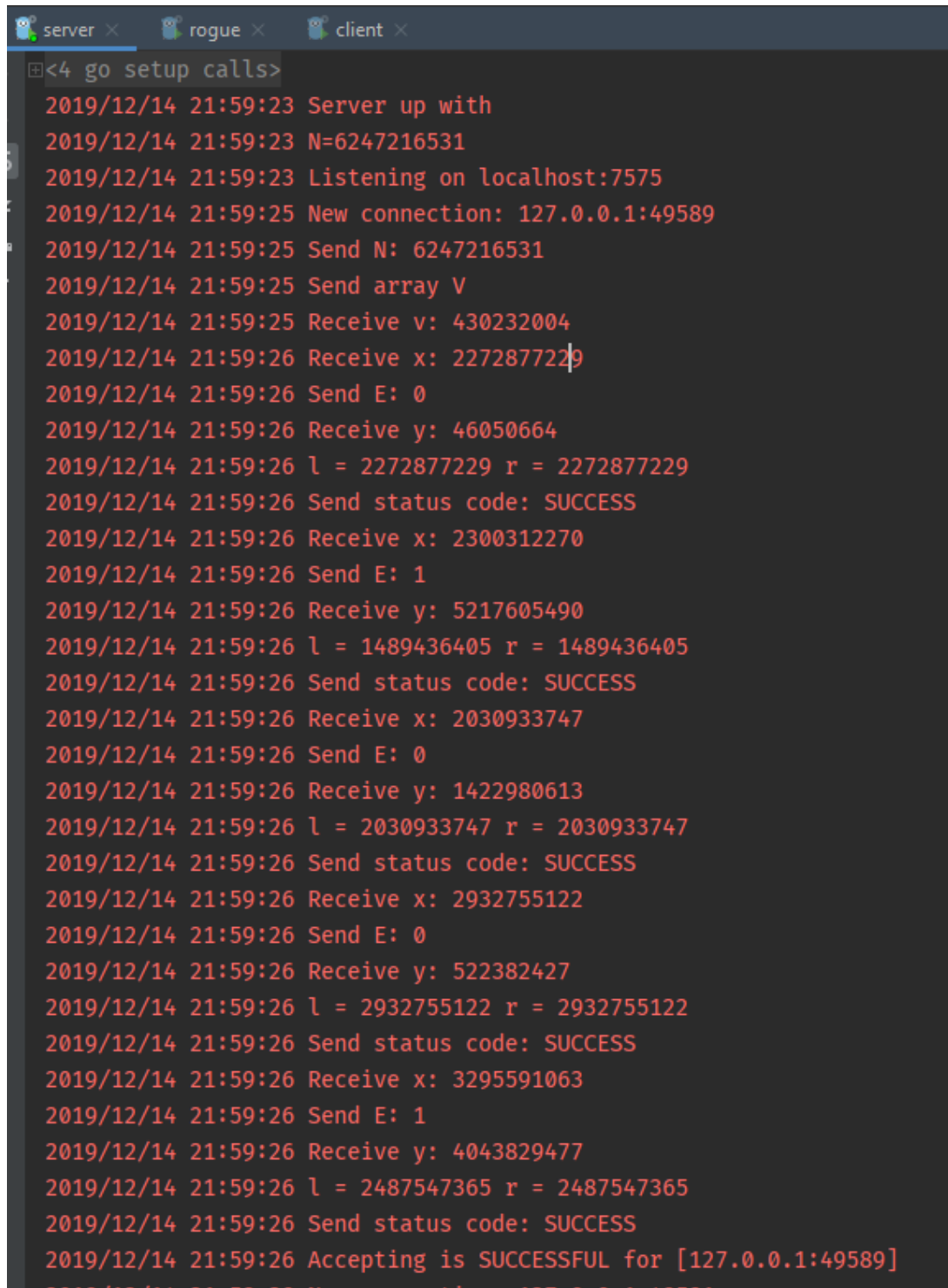
- А выбирает случайное число r , такое, что $r \in [1, n - 1]$ и отправляет $x = r^2 \bmod n$ стороне В (доказательство)
- В случайно выбирает бит e ($e = 0$ или $e = 1$) и отправляет его А (вызов)

- 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:



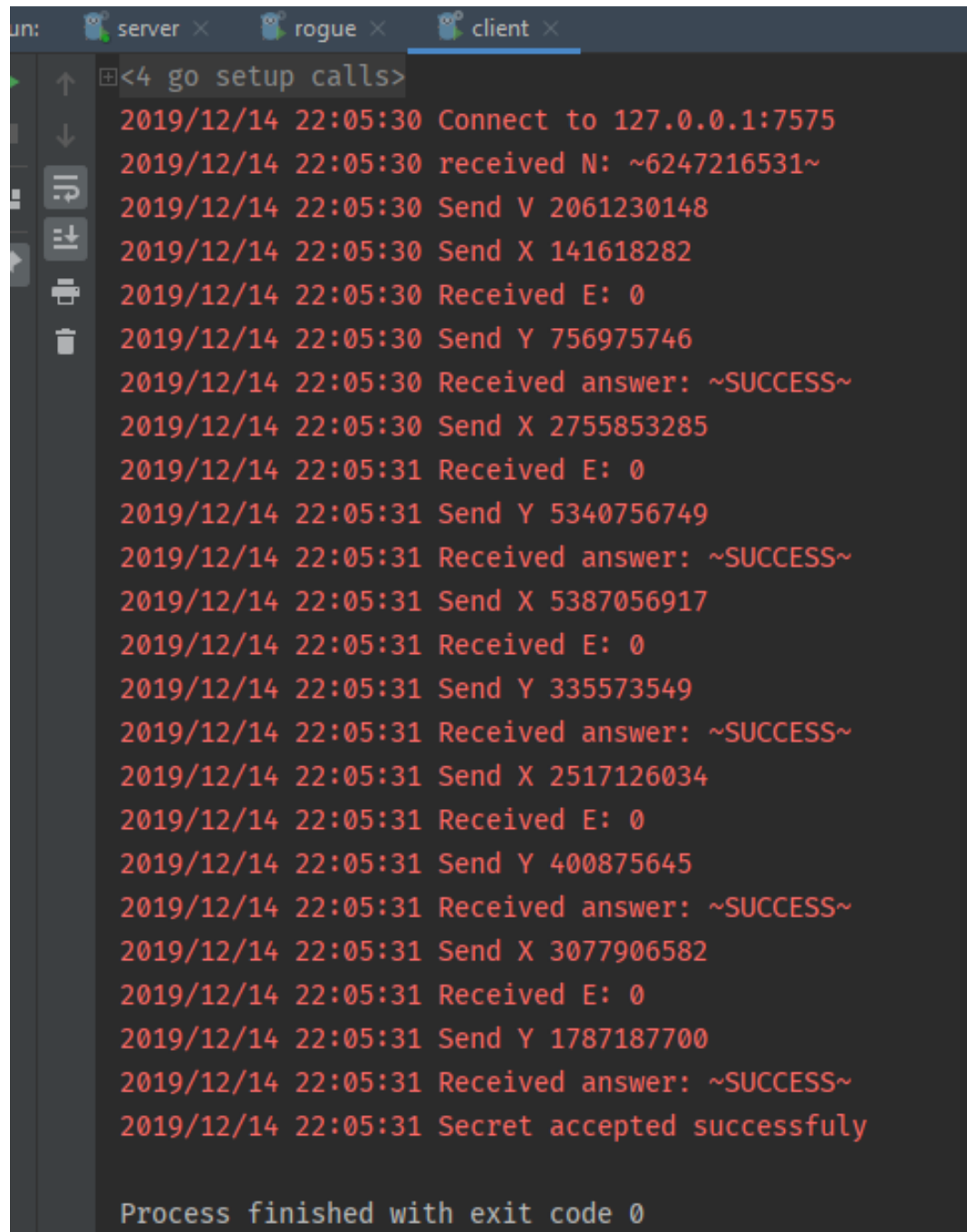
```
<4 go setup calls>
2019/12/14 21:59:23 Server up with
2019/12/14 21:59:23 N=6247216531
2019/12/14 21:59:23 Listening on localhost:7575
2019/12/14 21:59:25 New connection: 127.0.0.1:49589
2019/12/14 21:59:25 Send N: 6247216531
2019/12/14 21:59:25 Send array V
2019/12/14 21:59:25 Receive v: 430232004
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

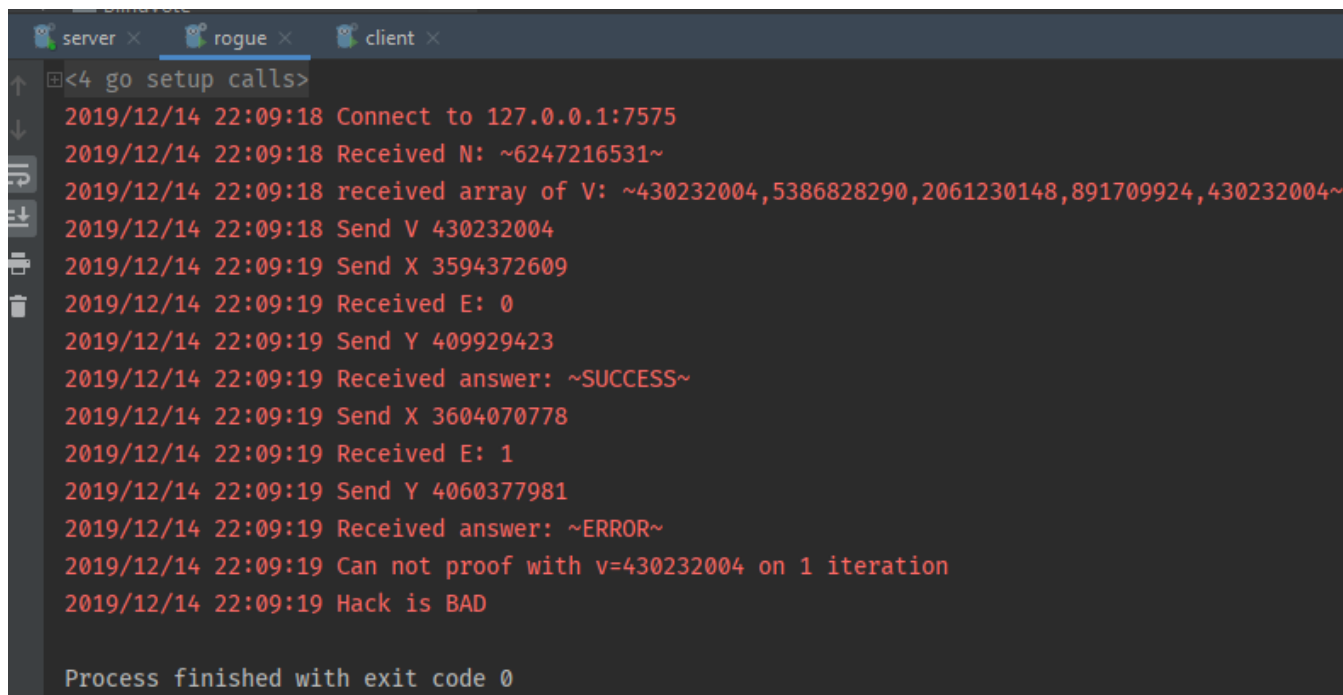
Process finished with exit code 0
```

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:



```
<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. package clnt
15.
16. import (
17.     "bufio"
18.     "cryptocrouse/src/go/FiatShamirProtocol"
19.     "cryptocrouse/src/go/Fingerprints"
20.     "fmt"
21.     "log"
22.     "math/big"
23.     "net"
24.     "os"
25.     "strconv"
26.     "strings"
27.     "time"
28. )
29.
30. var (
31.     MIN_P = big.NewInt(0).Exp(big.NewInt(2), big.NewInt(16), nil)
32.     MAX_P = big.NewInt(0).Exp(big.NewInt(2), big.NewInt(32), nil)
33. )
34.
35. type Client struct {
36.     conn net.Conn
37.     reader *bufio.Reader
38.     writer *bufio.Writer
39.     data *ClientData
40. }
41.
42. type ClientData struct {
43.     S *big.Int
44.     V *big.Int
45.     N *big.Int
46.     E int
47.     Y *big.Int
48.     R *big.Int
49.     X *big.Int
50. }
51.
52. func (c *Client) ConnectToServer() {
53.     arguments := os.Args
54.     if len(arguments) == 1 {
55.         fmt.Println("Please provide host:port.")
56.         return
57.     }
58.
59.     connect := arguments[1]
60.     conn, err := net.Dial("tcp", connect)
61.     if err != nil {
62.         fmt.Println(err)
63.         return
64.     }
65.     log.Printf("Connect to %s\n", connect)
66.     c.conn = conn
67.
68.     c.data = &ClientData{}
69.     c.setupConnections()
70. }
71.
```

```

72. func (c* Client) setupConnections() {
73.     c.reader = bufio.NewReader(c.conn)
74.     c.writer = bufio.NewWriter(c.conn)
75. }
76.
77. func (c *Client) StartProof() {
78.     for i := 0; i < 10; i++ {
79.         answerCode := c.round()
80.         if answerCode == false {
81.             log.Fatalf("Can not proof on %d iteration\n", i)
82.             return
83.         }
84.     }
85. }
86.
87. func (c *Client) round() bool {
88.     c.receiveN()
89.     c.generateS()
90.     c.computeV()
91.     c.sendV()
92.
93.     c.generateR()
94.     c.computeX()
95.     c.sendX()
96.
97.     c.receiveE()
98.     c.computeY()
99.     c.sendY()
100.    return c.getAnswer()
101.}
102.
103. func (c *Client) receiveN() {
104.    _, err := c.writer.WriteString(FiatShamirProtocol.COMMAND_GET_N + "\n")
105.    if err != nil {
106.        log.Fatal(err)
107.    }
108.    err = c.writer.Flush()
109.    if err != nil {
110.        log.Fatal(err)
111.    }
112.
113.    time.Sleep(50 * time.Millisecond)
114.
115.    msg, err := c.reader.ReadString('\n')
116.    if err != nil {
117.        log.Fatal(err)
118.    }
119.
120.    msg = strings.TrimSuffix(msg, "\n")
121.    log.Printf("received N: %s\n", msg)
122.
123.    var flag bool
124.    c.data.N, flag = big.NewInt(0).SetString(msg, 10)
125.    if flag == false {
126.        log.Fatal("Received N is bad")
127.    }
128.}
129.
130. func (c *Client) generateS() {
131.    for {
132.        c.data.S = Fingerprints.GetBigRandomWithLimit(c.data.N)
133.        if c.data.S.Cmp(big.NewInt(1)) == 0 {
134.            continue
135.        }
136.        GCD := big.NewInt(0).GCD(
137.            nil,
138.            nil,
139.            c.data.S,
140.            c.data.N)
141.        if GCD.Cmp(big.NewInt(1)) == 0 {
142.            break
143.        }
144.    }
145.}
146.
147. func (c *Client) computeV() {
148.    c.data.V = big.NewInt(0).Exp(c.data.S, big.NewInt(2), c.data.N)

```

```

149.}
150.
151.func (c *Client) receiveE() {
152.    _, _ = c.writer.WriteString(FiatShamirProtocol.COMMAND_GET_E + "\n")
153.    _ = c.writer.Flush()
154.
155.    time.Sleep(50 * time.Millisecond)
156.
157.    msg, _ := c.reader.ReadString('\n')
158.    msg = strings.TrimSuffix(msg, "\n")
159.    log.Printf("received E: %s\n", msg)
160.
161.    c.data.E, _ = strconv.Atoi(msg)
162.}
163.
164.func (c *Client) computeY() {
165.    switch c.data.E {
166.    case 0:
167.        c.data.Y = c.data.R
168.    case 1:
169.        c.data.Y = big.NewInt(0).Mod(
170.            big.NewInt(0).Mul(
171.                c.data.S,
172.                c.data.R),
173.            c.data.N)
174.    }
175.}
176.
177.func (c *Client) generateR() {
178.    for {
179.        c.data.R = Fingerprints.GetBigRandomWithLimit(c.data.N)
180.        if c.data.R.Cmp(big.NewInt(1)) > 0 && c.data.R.Cmp(c.data.N) < 0 {
181.            break
182.        }
183.    }
184.}
185.
186.func (c *Client) computeX() {
187.    c.data.X = big.NewInt(0).Exp(c.data.R, big.NewInt(2), c.data.N)
188.}
189.
190.func (c *Client) sendX() {
191.    _, _ = c.writer.WriteString(FiatShamirProtocol.COMMAND_GET_X + "\n")
192.    _ = c.writer.Flush()
193.    log.Println("Send " + FiatShamirProtocol.COMMAND_GET_X)
194.
195.    time.Sleep(50 * time.Millisecond)
196.
197.    _, _ = c.writer.WriteString(c.data.X.Text(10) + "\n")
198.    _ = c.writer.Flush()
199.    log.Printf("Send X %s\n", c.data.X.Text(10))
200.
201.    time.Sleep(50 * time.Millisecond)
202.}
203.
204.func (c *Client) sendY() {
205.    _, _ = c.writer.WriteString(FiatShamirProtocol.COMMAND_GET_Y + "\n")
206.    _ = c.writer.Flush()
207.    log.Println("Send " + FiatShamirProtocol.COMMAND_GET_Y)
208.
209.    time.Sleep(50 * time.Millisecond)
210.
211.    _, _ = c.writer.WriteString(c.data.Y.Text(10) + "\n")
212.    _ = c.writer.Flush()
213.    log.Printf("Send Y %s\n", c.data.Y.Text(10))
214.
215.    time.Sleep(50 * time.Millisecond)
216.}
217.
218.func (c *Client) sendV() {
219.    _, _ = c.writer.WriteString(FiatShamirProtocol.COMMAND_GET_V + "\n")
220.    _ = c.writer.Flush()
221.    log.Println("Send " + FiatShamirProtocol.COMMAND_GET_V)
222.
223.    time.Sleep(50 * time.Millisecond)
224.
225.    _, _ = c.writer.WriteString(c.data.V.Text(10) + "\n")

```

```

226.     _ = c.writer.Flush()
227.     log.Printf("Send V %s\n", c.data.V.Text(10))
228.
229.     time.Sleep(50 * time.Millisecond)
230.}
231.
232.func (c *Client) getAnswer() bool {
233.    log.Println("Wait answer")
234.    msg, err := c.reader.ReadString('\n')
235.    if err != nil {
236.        log.Fatal(err)
237.    }
238.
239.    msg = strings.TrimSuffix(msg, "\n")
240.    log.Printf("received answer: ~%s~\n", msg)
241.
242.    switch msg {
243.    case FiatShamirProtocol.COMMAND_ANSWER_CODE_SUCCESS:
244.        log.Println("Round ok")
245.        return true
246.    case FiatShamirProtocol.COMMAND_ANSWER_CODE_ERROR:
247.        log.Println("Round bad")
248.        return false
249.    default:
250.        log.Println("Round fi")
251.        return false
252.    }
253.}
254.
255.server/main.go
256.
257.package main
258.
259.import "cryptocrouse/src/go/FiatShamirProtocol/server/srvr"
260.
261.func main() {
262.    server := srvr.ServerInit()
263.    server.Run()
264.}
265.
266.server/server.go
267.
268.package srvr
269.
270.const (
271.    CONN_HOST = "localhost"
272.    CONN_PORT = "7575"
273.    CONN_TYPE = "tcp"
274.)
275.
276.var (
277.    MIN_P = big.NewInt(0).Exp(big.NewInt(2), big.NewInt(16), nil)
278.    MAX_P = big.NewInt(0).Exp(big.NewInt(2), big.NewInt(32), nil)
279.)
280.
281.import (
282.    "bufio"
283.    "cryptocrouse/src/go/FiatShamirProtocol"
284.    "cryptocrouse/src/go/Fingerprints"
285.    "log"
286.    "math/big"
287.    "net"
288.    "strconv"
289.    "strings"
290.    "time"
291.)
292.
293.type Server struct {
294.    data ServerData
295.}
296.
297.type ServerData struct {
298.    p *big.Int
299.    q *big.Int
300.    N *big.Int
301.}
302.

```

```

303.func ServerInit() *Server {
304.    return &Server{}
305.}
306.
307.func (s *Server) Run() {
308.    s.serverPrepare()
309.    log.Printf("Server up with\n")
310.    log.Printf("N=%s\n", s.data.N.Text(10))
311.    s.serverListen()
312.}
313.
314.func (s *Server) serverPrepare() {
315.    s.data.generateP()
316.    s.data.computeN()
317.}
318.
319.func (s *Server) serverListen() {
320.    l, err := net.Listen(CONN_TYPE, CONN_HOST+ ":" +CONN_PORT)
321.    if err != nil {
322.        log.Fatalf("Error listening:", err.Error())
323.    }
324.    defer l.Close()
325.
326.    log.Println("Listening on " + CONN_HOST + ":" + CONN_PORT)
327.
328.    for {
329.        conn, err := l.Accept()
330.        if err != nil {
331.            log.Fatalf("Error accepting: %s\n", err.Error())
332.        }
333.
334.        log.Printf("New connection: %s\n", conn.RemoteAddr())
335.
336.        go s.startRound(conn)
337.    }
338.}
339.
340.func (s *Server) startRound(conn net.Conn) {
341.    r := bufio.NewReader(conn)
342.    w := bufio.NewWriter(conn)
343.    scanr := bufio.NewScanner(r)
344.
345.    var x *big.Int
346.    var y *big.Int
347.    var v *big.Int
348.    e := generateE()
349.
350.    for {
351.        scanned := scanr.Scan()
352.        if !scanned {
353.            if err := scanr.Err(); err != nil {
354.                log.Printf("%v(%v)\n", err, conn.RemoteAddr())
355.                return
356.            }
357.            break
358.        }
359.        msg := scanr.Text()
360.        msg = strings.TrimSuffix(msg, "\n")
361.        log.Printf("Reveived [%s]: ~%s~\n", conn.RemoteAddr(), msg)
362.
363.        switch msg {
364.        case FiatShamirProtocol.COMMAND_GET_N:
365.            s.sendN(w)
366.        case FiatShamirProtocol.COMMAND_GET_X:
367.            x = s.receiveX(r)
368.        case FiatShamirProtocol.COMMAND_GET_Y:
369.            y = s.receiveY(r)
370.            statusCode := s.computeY(y, x, v, w, e)
371.            s.sendAnswerCode(w, statusCode)
372.        case FiatShamirProtocol.COMMAND_GET_V:
373.            v = s.receiveV(r)
374.        case FiatShamirProtocol.COMMAND_GET_E:
375.            s.sendE(w, e)
376.        }
377.    }
378.}
379.

```

```

380.func (s *Server) sendN(w *bufio.Writer) {
381.    log.Printf("Send N: %s\n", s.data.N.Text(10))
382.    _, _ = w.WriteString(s.data.N.Text(10) + "\n")
383.    _ = w.Flush()
384.}
385.
386.func (s *Server) receiveX(r *bufio.Reader) *big.Int {
387.    msg, _ := r.ReadString('\n')
388.    msg = strings.TrimSuffix(msg, "\n")
389.    x, _ := big.NewInt(0).SetString(msg, 10)
390.    log.Printf("Receive x: %s\n", msg)
391.    return x
392.}
393.
394.func (s *Server) receiveV(r *bufio.Reader) *big.Int {
395.    msg, _ := r.ReadString('\n')
396.    msg = strings.TrimSuffix(msg, "\n")
397.    v, _ := big.NewInt(0).SetString(msg, 10)
398.    log.Printf("Receive v: %s\n", msg)
399.    return v
400.}
401.
402.func (s *Server) sendE(w *bufio.Writer, e int) {
403.    _, _ = w.WriteString(strconv.Itoa(e) + "\n")
404.    _ = w.Flush()
405.    log.Println("Send E: " + strconv.Itoa(e))
406.}
407.
408.func (s *Server) receiveY(r *bufio.Reader) *big.Int {
409.    msg, _ := r.ReadString('\n')
410.    msg = strings.TrimSuffix(msg, "\n")
411.    y, _ := big.NewInt(0).SetString(msg, 10)
412.    log.Printf("Receive y: %s\n", msg)
413.    return y
414.}
415.
416.func (s *Server) computeY(y *big.Int, x *big.Int, v *big.Int, w *bufio.Writer, e int) string {
417.    if y.Cmp(big.NewInt(0)) == 0 {
418.        _, _ = w.WriteString(FiatShamirProtocol.COMMAND_ANSWER_CODE_ERROR)
419.        _ = w.Flush()
420.        return FiatShamirProtocol.COMMAND_ANSWER_CODE_ERROR
421.    }
422.
423.    l := big.NewInt(0).Exp(y, big.NewInt(2), s.data.N)
424.    var r *big.Int
425.
426.    switch e {
427.    case 0:
428.        r = x
429.    case 1:
430.        r = big.NewInt(0).Mod(
431.            big.NewInt(0).Mul(
432.                x,
433.                v),
434.            s.data.N)
435.    }
436.
437.    log.Printf("l = %s r = %s\n", l.Text(10), r.Text(10))
438.
439.    code := ""
440.
441.    if l.Cmp(r) == 0 {
442.        code = FiatShamirProtocol.COMMAND_ANSWER_CODE_SUCCESS
443.    } else {
444.        code = FiatShamirProtocol.COMMAND_ANSWER_CODE_ERROR
445.    }
446.
447.    return code
448.}
449.
450.func (s *Server) sendAnswerCode(w *bufio.Writer, statusCode string) {
451.    _, err := w.WriteString(statusCode + "\n")
452.    if err != nil {
453.        log.Fatal(err)
454.    }
455.    err = w.Flush()
456.    if err != nil {

```

```

457.         log.Fatal(err)
458.     }
459.     log.Println("Send status code: " + statusCode)
460.     time.Sleep(50 * time.Millisecond)
461. }
462.
463. func generateE() int {
464.     rand := Fingerprints.GetBigRandom()
465.     answer, _ := strconv.Atoi(big.NewInt(0).Mod(rand, big.NewInt(2)).Text(10))
466.     return answer
467. }
468.
469. func (data *ServerData) generateQ() {
470.     data.q = Fingerprints.GenerateBigPrimeNumberWithLimit(MIN_P)
471. }
472.
473. func (data *ServerData) generateP() {
474.     data.p = big.NewInt(0)
475.
476.     for {
477.         data.generateQ()
478.         data.p.Add(
479.             big.NewInt(0).Mul(
480.                 big.NewInt(2),
481.                 data.q),
482.             big.NewInt(1))
483.         if Fingerprints.IsPrimeRef(data.p) {
484.             if data.p.Cmp(MIN_P) > 0 && data.p.Cmp(MAX_P) < 0 {
485.                 break
486.             }
487.         }
488.     }
489. }
490.
491. func (data *ServerData) computeN() {
492.     data.N = big.NewInt(0).Mul(data.p, data.q)
493. }

```