

Protokół uwierzytelniający Otway–Reesa

Bartłomiej Garbacz

1 Ogólny opis protokołu

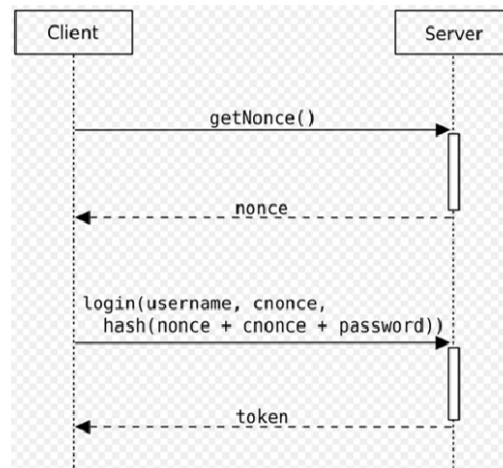
1.1 Opis algorytmu

Protokół służący do uwierzytelniania sieci komputerowej, przeznaczony do zabezpieczania ich i stosowany w kryptografii symetrycznej. Trent (zaufany sędzia) korzysta ze wspólnych kluczy z każdym użytkownikiem sieci. Pozwala to osobom komunikującym się na udowodnienie sobie tożsamości oraz zapobiega podsłuchiwanu.

1.2 Działanie algorytmu

Alice generuje wiadomość składającą się z liczby indeksowej C , jej nazwy P_1 , nazwy Boba P_2 i pewnej losowej wartości R_1 o długości 16B (nonce¹). Wszystko to szyfruje przy użyciu klucza dzielonego z Trentem. Gdzie liczbę indeksową możemy nazwać numerem sesji. Następnie przesyła tę wiadomość Bobowi wraz z liczbą indeksową, jej i jego nazwą:

$C, P_1, P_2, E_A(R_1, P_1, P_2, C)$



Rysunek 1: Działanie nonce

źródło:

www.researchgate.net/publication/338669692_Implementation_of_Otway-Rees_Protocol

¹nonce – jest dowolną liczbą, która może być używana tylko raz w komunikacji kryptograficznej. Często jest to losowy lub pseudolosowy numer wydany w protokole uwierzytelniania

Bob generuje wiadomość składającą się z nowej losowej wartości R_2 , liczby indeksowej C , nazwy Alice P_1 i nazwy Boba P_2 . Wszystko to szyfruje przy użyciu klucza dzielonego z Trentem przy użyciu algorytmu AES. Przesyła następnie tę wiadomość Trentowi wraz z zaszyfrowaną wiadomością Alice, liczbą indeksową, jej nazwą i jego nazwą:

$$C, P_1, P_2, E_A(R_1, C, P_1, P_2), E_B(R_2, C, P_1, P_2)$$

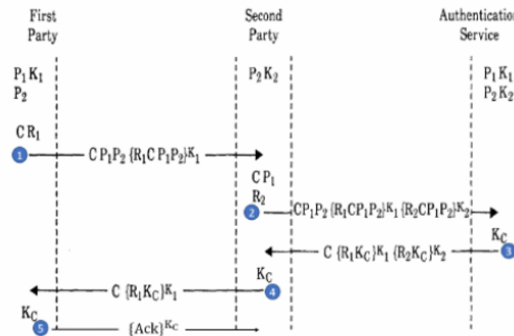
Trent generuje losowy klucz sesyjny K . Następnie tworzy dwie wiadomości. Jedna składa się z liczby losowej Alice R_1 i klucza sesyjnego, zaszyfrowanych przy użyciu klucza dzielonego z Alice. Druga z nich zawiera losową liczbę Boba R_2 i klucz sesyjny, zaszyfrowane przy użyciu klucza dzielonego z Bobem. Trent wysyła obie wiadomości Bobowi wraz z liczbą indeksową:

$$C, E_A(R_1, K), E_B(R_2, K)$$

Bob wysyła Alice wiadomość zaszyfrowaną za pomocą jej klucza wraz z liczbą indeksową:

$$C, E_A(R_1, K)$$

Jeżeli wszystkie liczby losowe są zgodne i liczba indeksowa nie zmieniła się w czasie przesyłania przez sieć, to Alice i Bob są nawzajem przekonani o tożsamości każdego z nich oraz mają teraz klucz tajny do wzajemnej komunikacji.



2

Rysunek 3: Schemat reprezentujący działanie protokołu

źródło:

www.researchgate.net/publication/338669692_Implementation_of_Otway-Rees_Protocol

1.3 Atak na protokół

Gdy Alice odbierze na końcu protokołu klucz K, to dzięki losowej liczbie R1 może mieć pewność, że klucz sesji jest prawdziwy i że został on w rzeczywistości wygenerowany przez serwer, poprzez sprawdzenie czy numer losowy otrzymany w ostatniej wiadomości jest taki sam, jak ten wygenerowany w pierwszej wiadomości. (Ta sama logika dla losowej wartości R2 wybranej przez Boba). Klucz sesji jest zawsze szyfrowany, a jedynymi stronami posiadającymi klucz są Alice, Bob i zaufany serwer. Dlatego haker nie może w żaden sposób odzyskać klucza sesji, jeśli algorytm nie ma luk bezpieczeństwa w implementacji protokołu. W ataku Man-In-the-Middle haker nie będzie w stanie zatem odzyskać klucza sesyjnego.

W przypadku gdy protokół posiada jednak luki w implementacji zatem haker może próbować ukraść klucz sesyjny

²Authentication Service(Przedstawiony na rysunku nr 3) = KDC

Przechwytywanie sesji

Przyjmijmy wiadomości :

Wiadomość 1 : Alice³ \rightarrow Bob⁴ :

$C, P_1, P_2, E(R_1, C, P_1, P_2)$

Wiadomość 2 : Bob \rightarrow Serwer :

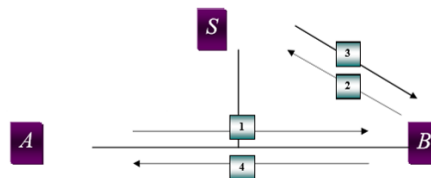
$C, P_1, P_2, E(R_1, C, P_1, P_2), E(R_2, C, P_1, P_2)$

Wiadomość 3 : Serwer \rightarrow Bob :

$C, E(R_1, K), E(R_2, K)$

Wiadomość 4 : Bob \rightarrow Alice :

$C, E(R_1, K)$



Rysunek 4: Diagram przesyłania wiadomości w protokole

źródło: www.giac.org/paper/gcih/81/

man-in-the-middle-attack-initiator-otway-rees-key-exchange-protocol/100561

³Oznaczenie A=Alice

⁴Oznaczenie B=Bob

Jeśli haker chce ukraść klucz sesji, musi być w stanie określić treść wiadomości 3 i 4. Może on również zaatakować serwer, lub inny zaufany podmiot, aby odzyskać klucze stałe lub podjąć próbę brutalnego ataku na zaszyfrowane wiadomości, które odzyskał z sieci komputerowej. Może to jednak być bardziej skomplikowane niż wykorzystanie luki w protokole. Haker może odzyskać klucz, po prostu manipulując informacjami. Musi podszywać się pod B. Kiedy A jest gotowy do rozpoczęcia sesji protokolarnej, haker rozpoczyna atak w następujący sposób:

Wiadomość 1: Alice \rightarrow C(B) :

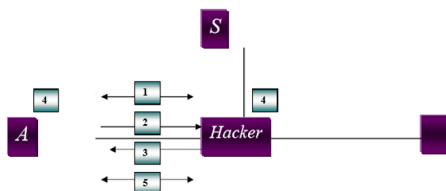
$C, P_1, P_2, E(R_1, C, P_1, P_2)$

Wiadomość 2: C(B) \rightarrow Alice :

$C, E(R_2, C, P_1, P_2)$

Kiedy A chce rozpocząć sesję z B, haker C(B) podszywa się pod B. Pobiera on pierwszą wiadomość, numer sesji i zaszyfrowaną część wiadomości, łączy wszystkie składniki i wysyła wynik do zleceniodawcy A. Strona ta pobiera wiadomość, weryfikuje numer sesji, odszyfrowuje zaszyfrowaną wiadomość swoim kluczem stałym, weryfikuje, czy otrzymał prawidłowo losowy numer, który wysłał w wiadomości 1, i stwierdza, że druga część odszyfrowanej wiadomości jest kluczem sesji. Dlatego też kluczem sesji dla tej sesji jest wiadomość C, P1, P2. Ponieważ wiadomość ta nie jest zakodowana przez sieć komputerową, haker może ją przechwycić i w ten sposób ukraść tajny klucz. Rzeczywiście, A nie zna klucza sesji przed otrzymaniem komunikatu 4.

Dlatego A zaakceptuje dowolny ciąg bitów o tej samej długości co klucz sesyjny i zaszyfrowany odpowiednią liczbą losową oraz kluczem kas. Aby przeprowadzić ten atak, haker musi tylko znać protokół i sposób jego zachowania. Nie musi wykonywać kroków 2 i 3 protokołu. W rzeczywistości nie ma możliwości, aby A wiedziało, że te dwa kroki w protokole nie zostały wykonane. W ten sposób haker nie musi znać kluczy stałych i nie musi w ogóle szyfrować ani odszyfrowywać żadnych informacji. Poprzez prostą manipulację informacjami, może on znaleźć klucz sesji protokołu i rozpocząć wymianę tajnych informacji z głównym A, bez żadnych podejrzeń, że A jest w trakcie wymiany tajnych informacji z hakerem.



Rysunek 5: Diagram przechwytywania sesji

źródło: www.giac.org/paper/gcih/81/

[man-in-the-middle-attack-initiator-otway-rees-key-exchange-protocol/100561](https://nvd.nist.gov/vuln/detail/CVE-2005-1637)

2 Implementacja

2.1 Uwrzytelnienie

```
def sesion_number(self):
    C = random.randint(1, 9)
    return C

def Generate_K(self):
    GENERATED_R = generateprime(16)
    return GENERATED_R

def encrypt(data, key):
    cipher = AES.new(bin(key)[2:]) #Creating AES object with binary key
    length = 16 - (len(data) % 16) # data padding
    data += chr(length) * length # -||-
    ctxt = cipher.encrypt(base64.b32encode(data)) # encryption
    return ctxt

def decrypt(data, key):
    cipher = AES.new(bin(key)[2:]) #Creating AES object with binary key
    data = base64.b32decode(cipher.decrypt(data)) # deencryption
    data = data[:-ord(data[-1])] # padding removal
    return data

def authentication(A, B):
    #Generating C for Users
    S.C = Server.sesion_number(S)
    A.C = S.C
    B.C = S.C
    # Step 0 Alice is preparing to send msg
    A.K = Server.Generate_K(S)
    Server.GetUser_K(S, A)
    A.ctxt = encrypt(str(A.R) + str(A.C) + str(A.P) + str(B.P), A.K)

    # Step1
    # A->B: C,P1,P2,Ea(Ra,C,P1,P2)
    B.data1 = A.ctxt #Bob has received Ea
    B.data3 = A.P #Bob has received P1
    B.data4 = B.P #Bob has received P2
    B.K = Server.Generate_K(S)
    Server.GetUser_K(S, B)
    B.ctxt = encrypt(str(B.R) + str(B.C) + str(B.P) + str(B.data3), B.K)

    # Step2
    # B->S: C,P1,P2,Ea,Eb(Rb,C,A,B)
```

```

S.data1 = A.C #Server has received C
S.data2 = A.ctxt #Server has received Ea
S.data3 = B.ctxt #Server has received Eb
S.data4 = decrypt(S.data2, A.K)
S.data5 = decrypt(S.data3, B.K)

if (S.C != S.data1): return False

S.SecretKey = Server.Generate_K(S)

del_lenght = len(str(A.C) + str(A.P) + str(B.P))
S.data6 = S.data4[:len(S.data4) - del_lenght]
S.MsgForUser1 = encrypt(str(A.R) + str(S.SecretKey), A.K)

del_lenght = len(str(B.C) + str(B.P) + str(B.data3))
S.data7 = S.data5[:len(S.data5) - del_lenght]
S.MsgForUser2 = encrypt(str(B.R) + str(S.SecretKey), B.K)

# Step3
# S->B: C,Ea(Ra,K),Eb(Rb,K)
B.data1 = S.data1 #Bob has received C
if (B.data1 != B.C): return False
B.data2 = S.MsgForUser1 # #Bob has received Ea
B.data3 = S.MsgForUser2 # #Bob has received Eb
B.data4 = decrypt(B.data3, B.K)
del_lenght_B = len(str(B.R))
B.RR = int(B.data4[:del_lenght_B])
B.SecretKey = int(B.data4[del_lenght_B:])
if (B.R != B.RR):
    False
#else: print("Nonce is corretly. User B has received the key")

# Step4
# B->A: C,Ea(Ra,K)
A.data1 = B.data1 #Alice has received C
if (A.data1 != A.C): return False
A.data2 = B.data2 #Alice has received Ea
A.data3 = decrypt(A.data2, A.K)
del_lenght_A = len(str(A.R))
A.RR = int(A.data3[:del_lenght_A])
A.SecretKey = int(A.data3[del_lenght_A:])
if (A.R != A.RR):

```

```

        return False
    #else: print("Nonce is corretly. User A has received the key")
    #print("authorization was successful")

```

2.2 Opis części kodu źródłowego przedstawiającego uwierzytelnienie

W punkcie 2.1 zostały przedstawione najważniejsze funkcje

sesion number(self) Jest to metoda klasy Server która rozdaje użytkownikom liczbę indeksową

Generate K(self) Jest to metoda klasy Server która generuje 16 bitową liczbę pierwszą

encrypt(data, key) Jest to funkcja szyfrująca wiadomość w trybie AES 16 bit

decrypt(data,key) Jest to funkcja deszyfrująca wiadomość w trybie AES 16 bit

authentication(A,B) Jest to funkcja uwierzytelniająca pomiędzy dwoma użytkownikami i tren-tem(Serwer), poniżej przedstawiam jej działanie.

Na początku sesji przez metodę **sesion number(self)** jest generowana przez Trenta(Serwer) liczba indeksowa która zostanie przydzielona każdemu użytkownikowi

"Krok 0"

Alice otrzymuje od Serwera klucz a następnie za pomocą tego klucza tworzy zaszyfrowaną wiadomość Ea składającą się z jej nonce, liczby indeksowej, jej nazwy i nazwy Boba(W tym przypadku jest to liczba w postaci ID użytkownika). Oczywiście za pomocą funkcji **encrypt(data, key)**

Krok 1

Alice przesyła dane do Boba. Bob otrzymuje je do atrybutów które służą jako buffor danych (B.data1,B.data2,B.data3). Następnie Bob wykonuje tę samą czynność jak poprzednio. Czyli otrzymuje klucz od Serwera, dzięki któremu zaszyfruje swoją wiadomość Eb składającą się z jego nonce,liczby indeksowej,jego nazwy, nazwy Alice za pomocą swojego klucza.

Krok 2

Bob wysyła dane na Serwer(Trent), po czym je odbiera do swoich atrybutów które służą jako buffor danych(S.data1, S.data2, S.data3). Adekwatnie jest to nonce, zaszyfrowana Ea wiadomość od Alice przekazana przez Boba i zaszyfrowana wiadomość Eb od Boba .

Dla atrybutów S.data4 i S.data5 pomocą funkcji **decrypt(data,key)** zostanie do nich przesłane zdeszyfrowane wiadomości obu użytkowników.

Serwer sprawdza czy liczba indeksowa jest poprawna.

Następnie za pomocą zmiennej **del-lenght** sprawdzi długość złożenia liczby indeksowej,nazwy użytkownika Alice, nazwy użytkownika Boba a następnie do następnego atrybutu S.data6 który służy jako

buffor, wiadomość zostanie wycięta i pozostanie tylko nonce użytkownika. Następnie do atrybutu S.MsgForUser1 zostanie załadowana wiadomość z połączenia nonce użytkownika i klucza sesyjnego, która zostanie zaszyfrowana przez funkcję **encrypt(data, key)**.

Czynność ta zostanie wykonana dwa razy najpierw dla użytkownika Alice, następnie dla użytkownika Bob.

Krok 3

Trent(Serwer) wyśle wiadomość do Boba. Składającą się z liczby indeksowej, Zaszyfrowanej wiadomości Eb dla Boba i zaszyfrowanej wiadomości Ea dla Alice.

Adekwatnie atrybuty B.data1,B.data2,B.data3 zostaną nadpisane jako buffor danych.

Bob do atrybutu B.data4 zamieści zdeszyfrowaną wiadomość za pomocą swojego klucza.

Zmienna **del-lenght-B** sprawdzi długość nonce Boba , następnie do atrybutu B.RR zostanie załadowany nonce który powrócił, poprzez ucięcie wiadomości o długość **del-lenght-B** .

Dla atrybutu B.SecretKey, zostanie ucięty z drugiej strony. W ten sposób Bob otrzyma klucz sesji do wspólnej komunikacji z Alice. Dodatkowo od razu zostanie przekonwertowany z typu string na typ integer. Na koniec zostanie sprawdzony warunek czy nonce nie uległ zmianie, w celu weryfikacji poprawności danych.

Krok 4

Czynność jest powtarzana jak przy poprzednim kroku.

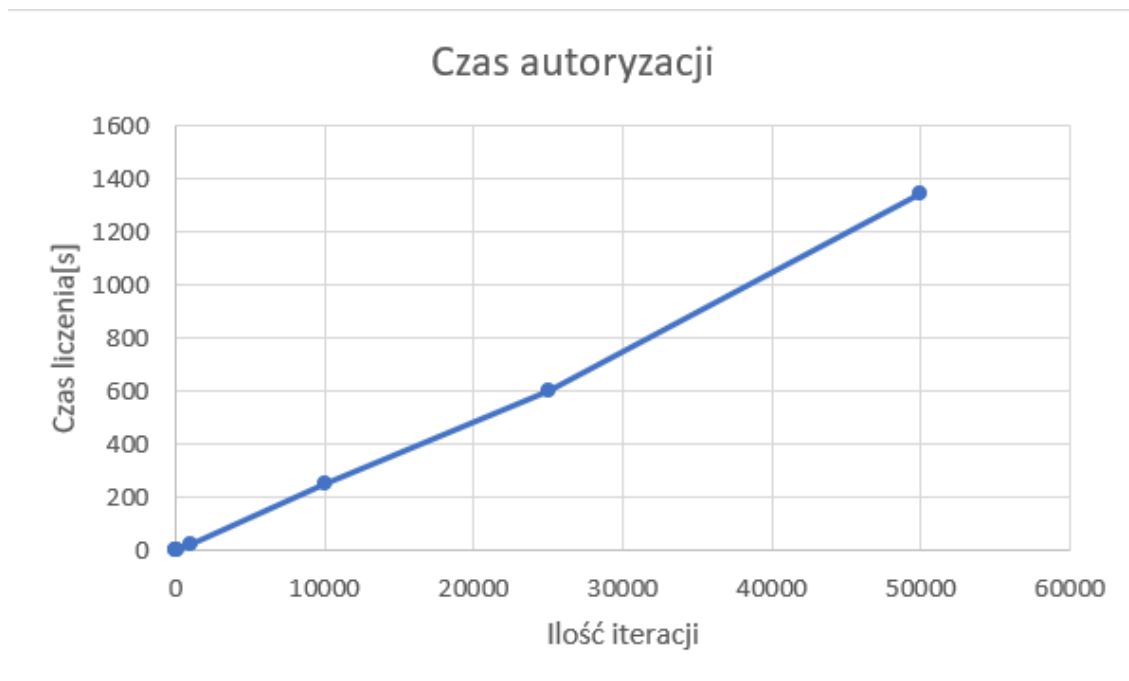
2.3 Uwrzytelnienie

Wszystkie próby weryfikacji przeszły z powodzeniem. Średni czas jednej weryfikacji wynosi 0,017[s], natomiast dla 50000 prób weryfikacji czas wynosi 1342[s]. Czyli około 22 minuty. Ocena wydajności w ten sposób oceniam na dobrą. Funkcja w trakcie wszystkich iteracji nie wyrzuca żadnych błędów.

Ilość iteracji	Czas Liczenia[s]
1	0,017
10	0,243
100	1,991
1000	21,78
10000	255,9
25000	599,9
50000	1342

Zestawienie ilości powtórzeń do czasu ich trwania

Wykres na podstawie danych



Rysunek 6 Wykres proporcjonalny do zestawienia ilości powtórzeń do czasu ich trwania

3 Weryfikacja formalna

3.1 Kod źródłowy

```
attacker[active]

principal Server[]
principal Alice[]
principal Bob[]

//Krok1
// A← Alice , B← Bob

principal Alice[
    //Alice zna C,P1,P2
    knows private C,P1,P2
    //Alice zna Sw j nonce i Sw j klucz
    knows private R1,K1
    //CONCAT- dodanie do siebie tych warto ci
    xa = CONCAT(R1,C,P1,P2)

    //Szyfrowanie przy pomocy AES
    A_ctxt = ENC(K1,xa)

]
//A wysy a do B – C,P1,P2M,A_ctxt
Alice -> Bob:C,P1,P2,A_ctxt

//krok2

principal Bob[

    // Bob otrzyma C,P1,P2
    // Zna Sw j nonce i klucz
    knows private R2,K2

    xb = CONCAT(R2,C,P1,P2) // -//-

    B_ctxt = ENC(K2,xb) // -//-

]
```

```
// Bob wysyła C,P1,P2, i dwie zaszyfrowane wiadomości od A i B
```

```
Bob → Server: C,P1,P2, A_ctxt, B_ctxt
```

```
principal Server[
    // serwer zna ich wszystkie wartości
    knows private C,P1,P2,K1,K2
    //Serwer generuje wspólny klucz do komunikacji dla A i B
    generates KC
    //Server deszyfruje wiadomość od A
    A_ptxt = DEC(K1, A_ctxt)
    //Serwer rozdziela wcześniej po czon przez CONCAT wiadomość
    RC_1,C_C,P1_1,P2_2 = SPLIT(A_ptxt)
    // RC_1= R1, C_C = C P1_1 = P1, P2_2 = P2

    // serwer czy nonce i klucz wygenerowany dla A i B
    xac=CONCAT(RC_1,KC)
    // Wiad. dla Alice
    CA_ctxt=ENC(K1,xac)

    // Powtórzenie tej samej czynności dla B
    B_ptxt=DEC(K2, B_ctxt)
    RC_2,C_CC,P1_11,P2_22 = SPLIT(B_ptxt)
    xbc=CONCAT(RC_2,KC)
    CB_ctxt=ENC(K2,xbc)//Dla Boba
]
```

```
// Serwer wysyła do B zaszyfrowane wiadomości w których znajdują się
// Klucze i nonce dla A i B
Server → Bob: CA_ctxt, CB_ctxt
```

```
principal Bob[
    //Bob deszyfruje Wiadomość od serwera za pomocą swojego klucza
    CB_ptxt=DEC(K2, CB_ctxt)

    // Rozdziela z wiadomości swój nonce i klucz wygenerowany przez
    // do komunikacji z Alice
    RC_2_final,KCB=SPLIT(CB_ctxt)
```

]

Bob → Alice: CA_ctxt // Alice nie chce przyjąć bo już ją zna

principal Alice[

CA_ptxt=DEC(K1, CA_ctxt) // -//-

RC_1_final, KCA=SPLIT(CA_ctxt) // -//-

]

queries[

authentication? Server → Bob: CB_ctxt

authentication? Bob → Alice: CA_ctxt

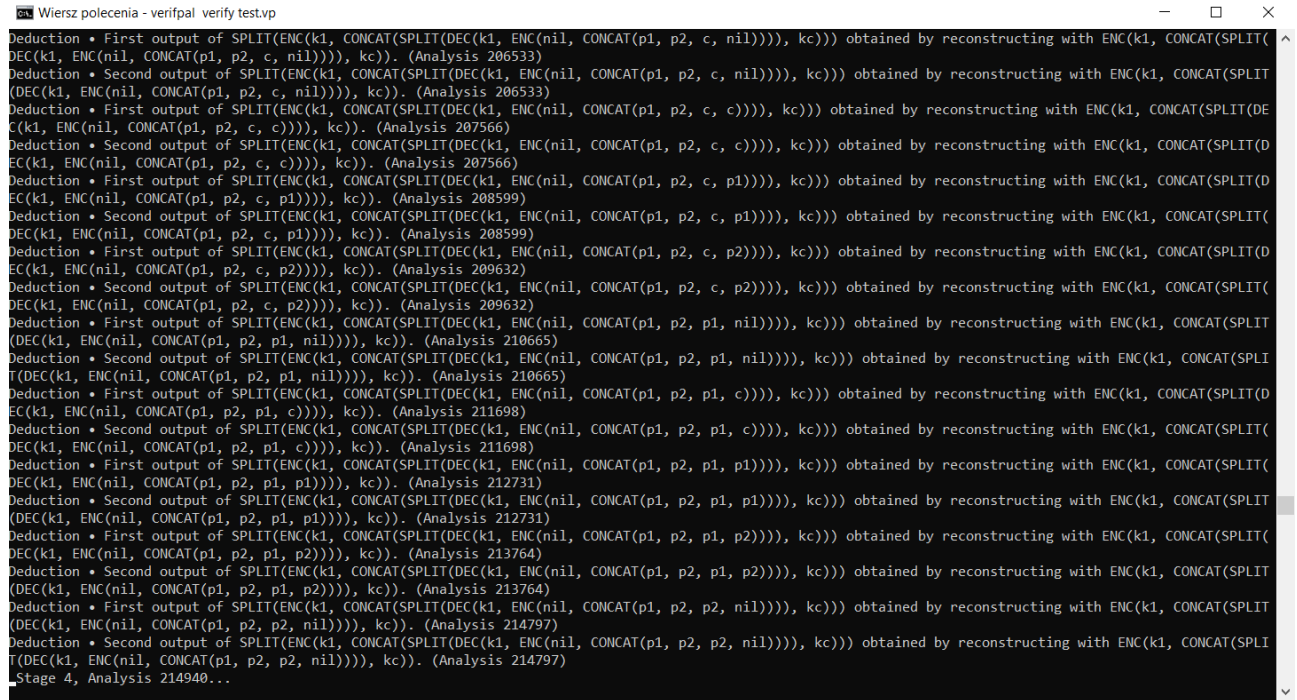
confidentiality? KCB

confidentiality? KCA

]

3.2 Analiza protokołu

Pierwsza weryfikacja mojego protokołu nie przeszła pomyślnie z powodu tego że oprogramowanie weryfikujące jest w wersji beta i program w którymś momencie się zapętlą, przez co tworzy się nieskończona liczba analiz. Ta próba weryfikacji trwała około 5-6 godzin i przez ten czas wytworzył ponad 200 000 analiz. Co uniemożliwiło mi sprawdzenie luk w protokole.



```
Wiersz polecenia - verifpal verify test.vp
Deduction • First output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, c, nil))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(
DEC(k1, ENC(nil, CONCAT(p1, p2, c, nil))))), kc)). (Analysis 206533)
Deduction • Second output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, c, nil))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(
DEC(k1, ENC(nil, CONCAT(p1, p2, c, nil))))), kc)). (Analysis 206533)
Deduction • First output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, c, c))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, c, c))))), kc)). (Analysis 207566)
Deduction • Second output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, c, c))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, c, c))))), kc)). (Analysis 207566)
Deduction • First output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, c, p1))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, c, p1))))), kc)). (Analysis 208599)
Deduction • Second output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, c, p1))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, c, p1))))), kc)). (Analysis 208599)
Deduction • First output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, c, p2))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, c, p2))))), kc)). (Analysis 209632)
Deduction • Second output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, c, p2))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, c, p2))))), kc)). (Analysis 209632)
Deduction • First output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, p1, nil))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, p1, nil))))), kc)). (Analysis 210665)
Deduction • Second output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, p1, nil))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, p1, nil))))), kc)). (Analysis 210665)
Deduction • First output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, p1, c))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, p1, c))))), kc)). (Analysis 211698)
Deduction • Second output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, p1, c))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, p1, c))))), kc)). (Analysis 211698)
Deduction • First output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, p1, p1))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, p1, p1))))), kc)). (Analysis 212731)
Deduction • Second output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, p1, p1))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, p1, p1))))), kc)). (Analysis 212731)
Deduction • First output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, p1, p2))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, p1, p2))))), kc)). (Analysis 213764)
Deduction • Second output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, p1, p2))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, p1, p2))))), kc)). (Analysis 213764)
Deduction • First output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, p2, nil))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, p2, nil))))), kc)). (Analysis 214797)
Deduction • Second output of SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(nil, CONCAT(p1, p2, p2, nil))))), kc))) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC
(k1, ENC(nil, CONCAT(p1, p2, p2, nil))))), kc)). (Analysis 214797)
Stage 4, Analysis 214940...
```

Rysunek 7 Analiza protokołu wykonana przez oprogramowanie Verifpal

Nie odrzucając problemu i nie poprzestając na drugiej próbie pozwoliłem sobie na skorzystanie z innego komputera w celu sprawdzenia czy oprogramowanie Verifpal nie koliduje z moim system i te myślenie okazało się trafne. Okazuje się że analiza protokołu może na jednym komputerze działać a na drugim nie. Powód jest uzasadniony ponieważ oprogramowanie Verifpal jest w wersji beta. Pomyślna i zakończona Analiza pokazuje nam że protokół zaimplementowany przeze mnie jest podatny na ataki. Atakujący może potrafić przejąć klucze wygenerowane przez serwer dla Alice i Boba. Wersja napisanego protokołu nie jest bezpieczna. Wyniki zostały zamieszczone na następnej stronie

```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL
>[!Warning] 0.11.10 - https://verifpal.com
[!Warning] Verifpal is experimental software.
Verifpal - Parsing model: 'veg.vp'
Verifpal - Verification initiated for 'veg.vp' at 11:54:35 PM.
Info - Attacker is configured as active.
Info - Running at phase 0.
Deduction - SPLIT(ca_ctxt) obtained by reconstructing with ch_ctxt.
Deduction - SPLIT(ch_ctxt) obtained by reconstructing with ch_ctxt.
Deduction - SPLIT(ca_ctxt) obtained by reconstructing with ca_ctxt.
Deduction - SPLIT(ca_ctxt) obtained by reconstructing with ca_ctxt.
Deduction - SPLIT(ENC(k2, CONCAT(r2, kc))) obtained by reconstructing with ENC(k2, CONCAT(r2, kc)). (Analysis 1)
Deduction - SPLIT(ENC(k2, CONCAT(r2, kc))) obtained by reconstructing with ENC(k2, CONCAT(r2, kc)). (Analysis 1)
Deduction - SPLIT(ENC(k1, CONCAT(r1, kc))) obtained by reconstructing with ENC(k1, CONCAT(r1, kc)). (Analysis 1)
Deduction - SPLIT(ENC(k1, CONCAT(r1, kc))) obtained by reconstructing with ENC(k1, CONCAT(r1, kc)). (Analysis 1)
Result - confidentiality? kcb:
kcb (SPLIT(ENC(k2, CONCAT(r2, kc)))) is obtained by Attacker.
(Analysis 2)
Result - confidentiality? kca:
kca (SPLIT(ENC(k1, CONCAT(r1, kc)))) is obtained by Attacker.
(Analysis 2)
Deduction - SPLIT(ENC(k2, CONCAT(SPLIT(DEC(k2, ENC(k1, CONCAT(r1, c, p1, p2))))), kc)) obtained by reconstructing with ENC(k2, CONCAT(SPLIT(DEC(k2, ENC(k1, CONCAT(r1, c, p1, p2))))), kc)). (Analysis 6)
Deduction - SPLIT(ENC(k2, CONCAT(SPLIT(DEC(k2, ENC(k1, CONCAT(r1, c, p1, p2))))), kc)) obtained by reconstructing with ENC(k2, CONCAT(SPLIT(DEC(k2, ENC(k1, CONCAT(r1, c, p1, p2))))), kc)). (Analysis 7)
Deduction - SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)). (Analysis 7)
Deduction - SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)). (Analysis 7)
Analysis - Constructed skeleton CONCAT(SPLIT(DEC(nil, ENC(nil, CONCAT(nil, nil, nil, nil))), nil). (Analysis 8)
Analysis - Constructed skeleton SPLIT(DEC(nil, ENC(nil, CONCAT(nil, nil, nil, nil))). (Analysis 8)
Analysis - Constructed skeleton DEC(nil, ENC(nil, CONCAT(nil, nil, nil, nil))). (Analysis 8)
Deduction - DEC(nil, CONCAT(nil, nil, nil, nil)) obtained by decomposing DEC(nil, ENC(nil, CONCAT(nil, nil, nil, nil))) with nil. (Analysis 8)
Result - authentication? Server -> Bob: ch_ctxt: When the following values are controlled by Attacker:
c = nil (originally c)
p1 = nil (originally p1)
p2 = nil (originally p2)
ch_ctxt = ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc) (originally ENC(k2, xhc))
ch_ctxt (ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)), sent by Attacker and not by Server, is successfully used in DEC(k2, ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) within Bob's state.
(Analysis 9)
Result - authentication? Bob -> Alice: ca_ctxt: When the following values are controlled by Attacker:
ca_ctxt = ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc) (originally ENC(k1, xac))
ca_ctxt (ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)), sent by Attacker and not by Bob, is successfully used in DEC(k1, ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) within Alice's state.
(Analysis 9)
Result - authentication? Server -> Bob: ch_ctxt: When the following values are controlled by Attacker:
c = nil (originally c)
p1 = nil (originally p1)
p2 = nil (originally p2)
ch_ctxt = ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc) (originally ENC(k2, xhc))
ch_ctxt (ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)), sent by Attacker and not by Server, is successfully used in DEC(k2, ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) within Bob's state.
Result - authentication? Bob -> Alice: ca_ctxt: When the following values are controlled by Attacker:
ca_ctxt = ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc) (originally ENC(k1, xac))
ca_ctxt (ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)), sent by Attacker and not by Bob, is successfully used in DEC(k1, ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) within Alice's state.
```

Rysunek 8 Pierwsza część analizy protokołu wykonanej przez oprogramowanie Verifpal

```
DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL
Deduction - SPLIT(ENC(k2, CONCAT(r2, kc))) obtained by reconstructing with ENC(k2, CONCAT(r2, kc)). (Analysis 1)
Deduction - SPLIT(ENC(k2, CONCAT(r2, kc))) obtained by reconstructing with ENC(k2, CONCAT(r2, kc)). (Analysis 1)
Deduction - SPLIT(ENC(k1, CONCAT(r1, kc))) obtained by reconstructing with ENC(k1, CONCAT(r1, kc)). (Analysis 1)
Deduction - SPLIT(ENC(k1, CONCAT(r1, kc))) obtained by reconstructing with ENC(k1, CONCAT(r1, kc)). (Analysis 1)
Result - confidentiality? kcb:
kcb (SPLIT(ENC(k2, CONCAT(r2, kc)))) is obtained by Attacker.
(Analysis 2)
Result - confidentiality? kca:
kca (SPLIT(ENC(k1, CONCAT(r1, kc)))) is obtained by Attacker.
(Analysis 2)
Deduction - SPLIT(ENC(k2, CONCAT(SPLIT(DEC(k2, ENC(k1, CONCAT(r1, c, p1, p2))))), kc)) obtained by reconstructing with ENC(k2, CONCAT(SPLIT(DEC(k2, ENC(k1, CONCAT(r1, c, p1, p2))))), kc)). (Analysis 6)
Deduction - SPLIT(ENC(k2, CONCAT(SPLIT(DEC(k2, ENC(k1, CONCAT(r1, c, p1, p2))))), kc)) obtained by reconstructing with ENC(k2, CONCAT(SPLIT(DEC(k2, ENC(k1, CONCAT(r1, c, p1, p2))))), kc)). (Analysis 7)
Deduction - SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)). (Analysis 7)
Deduction - SPLIT(ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) obtained by reconstructing with ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)). (Analysis 7)
Analysis - Constructed skeleton CONCAT(SPLIT(DEC(nil, ENC(nil, CONCAT(nil, nil, nil, nil))), nil). (Analysis 8)
Analysis - Constructed skeleton SPLIT(DEC(nil, ENC(nil, CONCAT(nil, nil, nil, nil))). (Analysis 8)
Analysis - Constructed skeleton DEC(nil, ENC(nil, CONCAT(nil, nil, nil, nil))). (Analysis 8)
Deduction - DEC(nil, CONCAT(nil, nil, nil, nil)) obtained by decomposing DEC(nil, ENC(nil, CONCAT(nil, nil, nil, nil))) with nil. (Analysis 8)
Result - authentication? Server -> Bob: ch_ctxt: When the following values are controlled by Attacker:
c = nil (originally c)
p1 = nil (originally p1)
p2 = nil (originally p2)
ch_ctxt = ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc) (originally ENC(k2, xhc))
ch_ctxt (ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)), sent by Attacker and not by Server, is successfully used in DEC(k2, ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) within Bob's state.
(Analysis 9)
Result - authentication? Bob -> Alice: ca_ctxt: When the following values are controlled by Attacker:
ca_ctxt = ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc) (originally ENC(k1, xac))
ca_ctxt (ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)), sent by Attacker and not by Bob, is successfully used in DEC(k1, ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) within Alice's state.
(Analysis 9)
Result - authentication? Server -> Bob: ch_ctxt: When the following values are controlled by Attacker:
c = nil (originally c)
p1 = nil (originally p1)
p2 = nil (originally p2)
ch_ctxt = ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc) (originally ENC(k2, xhc))
ch_ctxt (ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)), sent by Attacker and not by Server, is successfully used in DEC(k2, ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) within Bob's state.
Result - authentication? Bob -> Alice: ca_ctxt: When the following values are controlled by Attacker:
ca_ctxt = ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc) (originally ENC(k1, xac))
ca_ctxt (ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)), sent by Attacker and not by Bob, is successfully used in DEC(k1, ENC(k1, CONCAT(SPLIT(DEC(k1, ENC(k2, CONCAT(r2, c, p1, p2))))), kc)) within Alice's state.
Result - confidentiality? kcb:
kcb (SPLIT(ENC(k2, CONCAT(r2, kc)))) is obtained by Attacker.
Result - confidentiality? kca:
kca (SPLIT(ENC(k1, CONCAT(r1, kc)))) is obtained by Attacker.
Verifpal - Verification completed for 'veg.vp' at 11:54:35 PM.
Verifpal - Thank you for using Verifpal.
on 11/04/2020 at 11:54:35 PM
```

Rysunek 9 Druga część analizy protokołu wykonanej przez oprogramowanie Verifpal

Środowisko Verifpal pokazało słabość w mojej implementacji protokołu przy składaniu zaszyfrowanych wiadomości przy każdej ze stron. Atakujący może zdekonstruować wiadomość, moim zdaniem jedynie za pomocą kryptoanalizy statystycznej. Jeśli wie w jakiej kolejności są ułożone dane, bądź jakiej one są długości. Wtedy protokół jest narażony na niepożądane odczytanie wiadomości.

4 Podsumowanie

Przedmiot ten i zrealizowana praca nauczyła mnie jak czytać protokoły, czego rezultatem jest rozumowanie i budowa protokołów, wystarczy teraz że spojrzę na schemat innego i wiem już jak działa. Planowanie implementacji protokołu nauczyło mnie też dokładniejszego research'u, ponieważ tworzenie protokołów jest niszą i jest mało informacji na ten temat. Po implementacji każdego protokołu ważna jest weryfikacja formalna i sprawdzenie w których miejscach nasz protokół jest najbardziej wrażliwy. W tym celu na zajęciach użyliśmy oprogramowania o nazwie Verifpal która pozwala sprawdzić bezpieczeństwo protokołu. Każdy protokół ma swoje wady i zalety oraz każdy jest podatny na jakiś atak, czasami nie warto modyfikować pewnych luk ze względu na to że przez naszą ingerencję może powstać ich nawet więcej. Przedmiot ten bardzo rozwija myślenie i przyda się w mojej przyszłej pracy. Przy okazji nauczyłem się korzystać pisać w Latex i rozszerzyłem swoje umiejętności w programowaniu w języku Python

Należy też pamiętać że oprogramowanie Verifpal jest w wersji Beta, czasem w oprogramowaniu może się wdać np błąd indeksacji listy, dobrym pomysłem jest wtedy skorzystanie z innego komputera. W moim przypadku rozwiązało to problem

Literatura

- [1] Mishra, Sourav. (2018). Implementation of Otway-Rees Protocol.
www.researchgate.net/publication/338669692_Implementation_of_Otway-Rees_Protocol?fbclid=IwAR0hWpbT-emNL6ySojX53WlwpMVbmhjKsILbq8v4l2myC1T6lmcPQtIbav0.
- [2] Man-in-the-middle attack against the initiator of OtwayRees Key Exchange Protocol
www.giac.org/paper/gcih/81/man-in-the-middle-attack-initiator-otway-rees-key-exchange-100561