Bezpieczny klient pocztowy

Secure Mail Client

Bezpieczny, prosty klient pocztowy z możliwością szyfrowania i podpisywania maili

Jacek Jabłoński

2015

2015

Założenia

Celem projektu jest projekt i implementacja bezpiecznego klienta pocztowego z możliwością szyfrowania wiadomości i bezpiecznego podpisu.

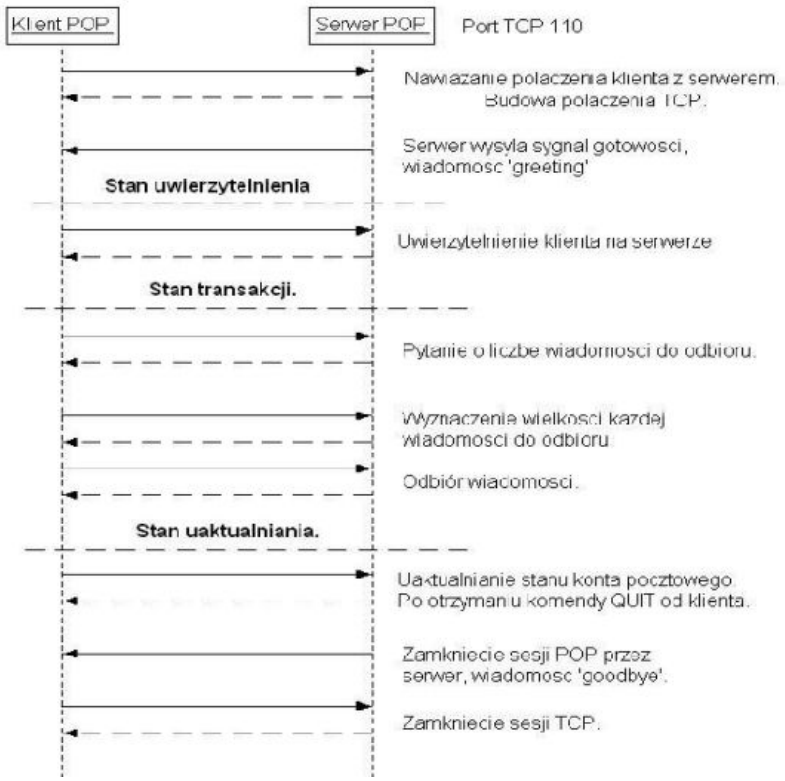
Zagadnienia teoretyczne

1. Protokół POP3

   POP3 (Post OFFICE Protocol version 3) , protokół internetowy warstwy aplikacji, który umożliwia odbiór poczty elektronicznej z serwera do lokalnego komputera poprzez połączenie TCP/IP. Protokół stworzony został dla użytkowników którzy nie są ciągle obecni w internecie. W przypadku, gdy połączenie z siecią jest chwilowe, poczta nie może przyjść do użytkownika protokołem SMTP. Na potrzeby rozwiązania tego problemu stworzono specjalny serwer, który przez SMTP odbiera przychodzącą pocztę i ustawia ją w kolejce. W momencie połączenia się użytkownika do sieci, korzystając z protokołu POP3 może pobrać oczekujące na niego listy do lokalnego komputera

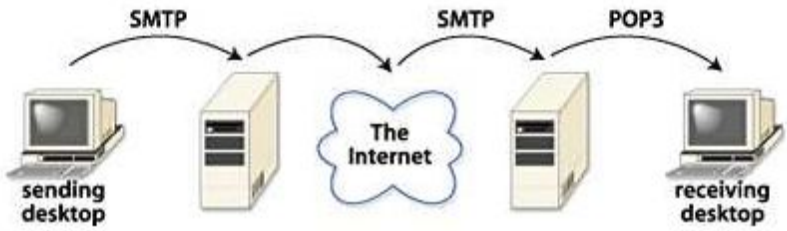
   Połączenie z serwerem przy pomocy protokołu POP3 odbywa się na porcie 110. Jeśli klient chce skorzystać z protokołu POP3 ustawia połączenie TCP z hostem serwera. Następnie serwer wysyła powitanie, i dalej kolejno klient i serwer POP3 prowadza wymianę komend dopóty, dopóki połączenie nie zostanie zerwane lub zakończone.

   Sesja między klientem a serwerem POP podzielona jest na stany wdł. poniższego schematu



Rysunek 1 Sesja mi dzy klientem a serwerem POP

2. Protocol SMTP (Simple Mail Transfer Protocol)

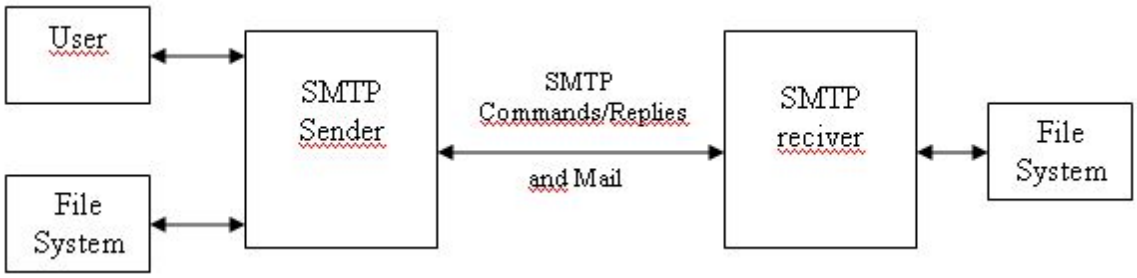


Rysunek 2 Przebieg standardowego przekazywania poczty

SMTP (ang. Simple Mail Transfer Protocol), to protokół komunikacyjny opisujący sposób przekazywania poczty elektronicznej w Internecie. Został on zaprojektowany na podstawie następującego modelu:

W pierwszym kroku nadawca SMTP ustanawia połączenie w dwie strony między nim, a odbiorcą SMTP. Odbiorca ten może być ostatecznym odbiorcą wiadomości albo pośrednim i przekazać ją dalej. Kiedy zostanie ustanowione połączenie SMTP nadawca SMTP wysyła komendę, dzięki której odbiorca SMTP będzie mógł go zweryfikować. Jeżeli odbiorca SMTP jest w stanie zaakceptować wiadomość pocztową od nadawcy SMTP, wysyła on do niego komendę OK. W następnym kroku nadawca SMTP wysyła komendę RCPT identyfikując

odbiorcę wiadomości. Jeśli odbiorca SMTP może zaakceptować wiadomość dla tego odbiorcy wiadomości, odpowiada komendą OK, jeśli nie, odrzuca tego odbiorcę. Nadawca i odbiorca SMTP mogą jednocześnie uzgadniać kilku odbiorców wiadomości. Kiedy odbiorca zostanie uzgodniony, nadawca SMTP wysyła treść wiadomości przerywaną specjalnymi sekwencjami. Jeśli odbiorca SMTP przetworzy otrzymaną wiadomość, odpowiada komendą OK.



Rysunek 3. Mechanizm dzia ania protoko u SMTP.

SMTP zapewnia bezpośredni mechanizm transmisji wiadomości między hostem nadawcy a hostem odbiorcy bądź, co jest najczęstszym przypadkiem, przy użyciu serwerów pośredniczących w dostarczeniu wiadomości. Aby serwery pośredniczące mogły odpowiednio przesyłać wiadomość w dalszą drogę, koperta wiadomości musi być wyposażona w adres końcowego serwera SMTP, jak również w adres odbiorcy wiadomości (*format adresu pocztowego: nazwaUzytkownika@nazwaDNSowaDomeny).* Zawierać ona musi ponad to adres zwrotny, który mówi kto jest nadawcą wiadomości. Może być on użyty do odesłania wiadomości do nadawcy, gdy będzie zawierać ona błędy.

Kroki wymiany wiadomości przy użyciu SMTP:

1. Wysłanie komendy MAIL z identyfikatorem nadawcy,
2. Wymiana informacji przy pomocy komend RCPT i przesłanie treści wiadomości przy użyciu komendy DATA,
3. Zakończenie wymiany wiadomości.

3. Algorithm AES (Advanced Encryption Standard)

Algorytm AES inaczej zwany również ***Rijndael ,*** jest blokowym *(128, 192, 256 bitowe bloki danych)* algorytmem szyfrowania z kluczem symetrycznym , pozwalającym na wykorzystanie klucza szyfrującego o długości 128, 192, 256 bitów.

W projekcie został on użyty do szyfrowania wiadomości, a ze względu na swoją symetryczność, która w tym modelu wymiany wiadomości jest niepożądana, wykorzystano też algorytm RSA.

Szyfr AES może operować na bloku o zmiennej długości, używając kluczy zmiennej długości. Oficjalna specyfikacja dopuszcza użycie bloków 128-, 192- lub 256-cio bitowych,

szyfrowanych kluczami 128-, 192- lub 256-cio bitowymi. Dopuszczalne są wszystkie z 9 kombinacji.

Rijndael jest ***"iterowanym szyfrem blokowym"***, co oznacza, że blok wejściowy oraz klucz przechodzą wielokrotne **RUNDY** transformacji, zanim wyprodukują wynik. Po każdej rundzie, powstaje szyfr pośredni, zwany ***STANEM (State)***.

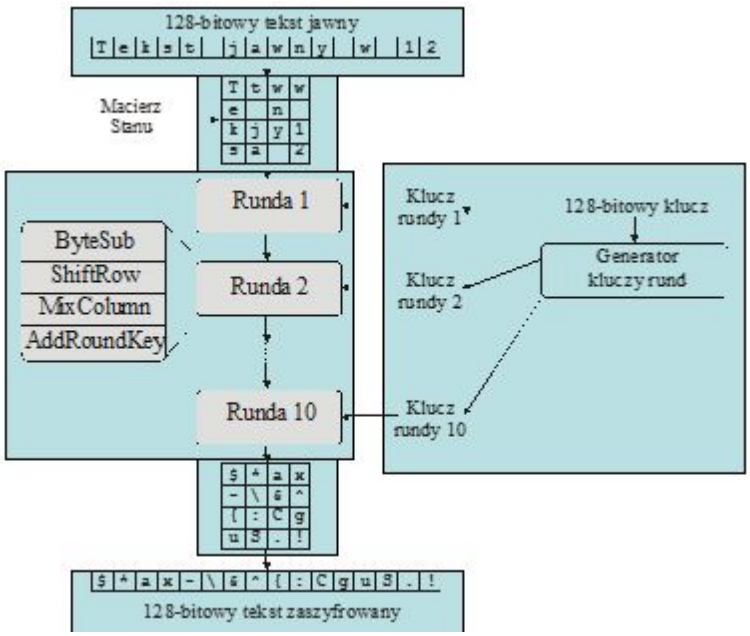
Liczba rund (przebiegów transformacji) w szyfrze Rijndael jest określona w następujący sposób:

Liczba Rund (Nr)		Wielkość bloku		
		128 bitów Ns = 4	192 bity Ns = 6	256 bitów Ns = 8
Wielkość klucza	128 bitów Nk = 4	10	12	14
	192 bity Nk = 6	12	12	14
	256 bitów Nk = 8	14	14	14

Gdzie:

- Ns = wielkość bloku w bitach / 32
- Nk = wielkość klucza w bitach / 32
- Nr - liczba rund

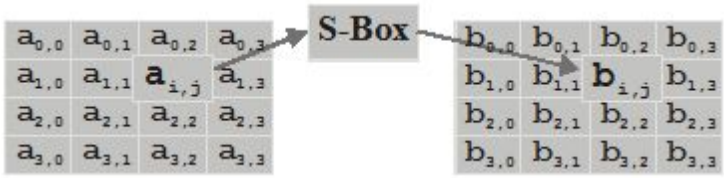
Ogólny opis działania



Szyfr AES/Rijndael składa się z trzech operacyjnych faz:

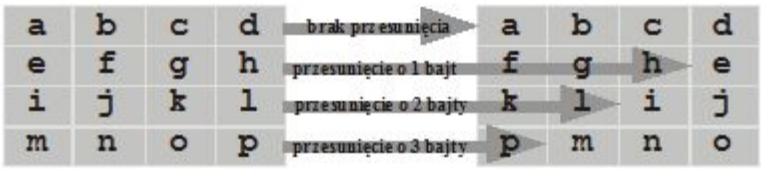
1) *Transformacja klucza "AddRoundKey"*
2) *Nr-1 rund sk adaj cych si z:*

- **Transformacja SubBytes**
  Przy transformacji **ByteSub** operuje się na poszczególnych elementach macierzy Stanu,które sa pojedynczymi bajtami. Każdy bajt przechodzi przez transformację **S-Boxem** i jest wpisywana w to samo miejsce. W fazie tej wykonuje się jedynie operacje na bajtach.



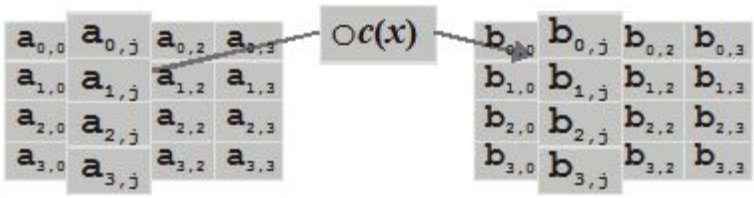
Rysunek 4 Operacje na bajtach

- **Transformacja ShiftRows**
  Transformacja ta przesuwa **cyklicznie** kolejne wiersze macierzy o odpowiednią liczbę pozycji. Wartość przesunięcia zależy od wielkości bloku i klucza, w przedstawionym przypadku nie przesuwa się danych pierwszego wiersza, drugi przesuwa się o 1 kolumnę, trzeci przesuwa się o 2 kolumny, a czwarty o 3 kolumny. Ponieważ takie przesunięcie sprowadza się jedynie do zmiany uporządkowania danych w pamięci, jest ono łatwe w realizacji dla każdego procesora



Rysunek 5 Zobrazowana transformacja ShiftRows

- **Transformacja MixColumns**
  Transformacja MixColumn używa pewnej funkcji matematycznej do wykonania transformacji całej kolumny macierzy stanu. Działa ona w taki sposób, jakby wszystkie 4 bajty kolumny reprezentowały współczynniki pewnego wielomianu o czterech członach.
  Operacja ta polega na przemnożeniu wielomianu utworzonego z bajtów kolumny, W praktyce, operacja mnożenia wielomianów została zaimplementowana jako mnożenie macierzy A przez macierz C



Rysunek 6 Transformacja MixColumns

- **Transformacja AddRoundKey**
  Klucz szyfrujący może być 128-, 192- lub 256-cio bitowy. Klucz szyfrujący używany jest wewnątrz algorytmu do otrzymania odrębnego klucza w każdej rundzie procesu szyfrowania. Klucze takie są zwane kluczami rundy i mają długość taką jak długość bloku danych. Operacja AddRoundKey polega jedynie na wykonaniu operacji XOR w każdej rundzie pomiędzy całym blokiem a kluczem rundy.

3) Runda finałowa składa się z:

- Transformacja SubBytes
- Transformacja ShiftRows
- Transformacja AddRoundKey

Operacje MixColumn i ShiftRow zapewniają silną dyfuzję (zamiana jednego bitu stanu wpływa na wszystkie bity stanu w małej liczbie rund)
Operacje ByteSub i dodawanie klucza rundy zapewniają silną konfuzję (zgubienie zależności – na podstawie rezultatu jednej rundy nie można wywnioskować macierzy stanu na początku rundy)
Aby przeprowadzić kryptoanalizę różnicową, między poszczególnymi rundami muszą istnieć przewidywalne różnice. Udowodniono, że odpowiednie prawdopodobieństwa wykorzystywane przy kryptoanalizie DES-a w przypadku Rijndaela nie są wystarczające do przeprowadzenia skutecznego ataku

4. Algorytm RSA

Algorytm RSA to szyfr blokowy, w którym tekst jawny i tekst zaszyfrowany są liczbami całkowitymi od *0* do *n-1* dla pewnego *n* . Tekst jawny jest szyfrowany blokami, z których każdy ma wartość binarną mniejszą niż *n.* RSA umożliwia utworzenie dwóch powiązanych kluczy: publicznego oraz prywatnego, a następnie wykorzystywanie ich w celu ochrony treści przekazywanych wiadomości. Klucz publiczny jest powszechnie znany i każdy może za jego pomocą zaszyfrować dowolną wiadomość. Natomiast jedynie posiadacz klucza prywatnego może odszyfrować otrzymane szyfrogramy. RSA jest dość silnym algorytmem, jednak w przypadku przechwycenia wielu wiadomości istnieją metody skutecznej analizy celem ustalenia klucza prywatnego z wykorzystaniem tekstu jawnego. Z tego powodu w projekcie użyto RSA do zaszyfrowania tylko klucza AES, a sama wiadomość jest szyfrowana

trudniejszym do złamania AESem, odpornym zarówno na analizę różnicową jak i inne rodzaje ataków, w tym nieopłacalny obliczeniowo brute-force.

Generowanie klucza

Sposób generacji powiązanych kluczy RSA:

1. Wybrać dwie różne liczby pierwsze, zwykle oznaczane jako **p** oraz **q**. Liczby powinny być wybrane losowo i składać się z podobnej liczby bitów.

2. Obliczyć:
   $\mathbf{n = p \cdot q}$
   Liczba **n** jest wykorzystywana jako moduł dla kluczy prywatnego i publicznego. Długość **n** jest długością klucza RSA.

3. Obliczyć wartość funkcji Eulera dla **n**:
   $\boldsymbol{\phi(n) = \phi(p) \cdot \phi(q) = (p - 1) \cdot (q - 1)}$

4. Wybrać liczbę całkowitą **e**, która jest większa od 1 i mniejsza niż poprzednio wyliczona wartość $\boldsymbol{\phi(n)}$. Liczby **e** oraz $\boldsymbol{\phi(n)}$ powinny być względnie pierwsze. Liczba **e** jest używana jako wykładnik klucza publicznego.

5. Wyznaczyć liczbę **d** taką, że:
   $\boldsymbol{d \cdot e = 1 \pmod{\phi(n)}}$
   Liczba **d** jest używana jako wykładnik klucza prywatnego.

Klucz publiczny składa się z modułu **n** i publicznego wykładnika **e**, natomiast klucz prywatny składa się z tego samego modułu **n** i prywatnego wykładnika **d**. Wszystkie liczby powiązane z kluczem prywatnym powinny być trzymane w tajemnicy: obie liczby **n** i **d** oraz trzy kolejne: **p**, **q** i $\boldsymbol{\phi(n)}$, które mogą być użyte do wyliczenia **d**.

Wiele użytkowników może współdzielić **e,** przy czym **e** nie powinno być długie ze względu na zwiększającą się złożoność obliczeniową, ale nie powinno być zbyt krótkie aby nie osłabić zabezpieczeń. Każdy użytkownik powinien mieć przypisaną własną liczbę **n** (która bierze się z iloczynu dwóch liczb pierwszych).

Szyfrowanie

Szyfrowanie odbywa się za pomocą klucza publicznego **(n, e)**. Wiadomość musi zostać podzielona na części, a następnie każda część powinna zostać zmieniona na liczbę (która musi być większa od 0 i mniejsza niż **n**). W praktyce dzieli się wiadomość na fragmenty, z których każdy składa się z określonej ilości bitów.

Następnie każda liczba wchodząca w skład wiadomości jest podnoszona modulo **n** do potęgi równej **e**:

$$c_i = m_i^e \pmod n$$

Algorytmu RSA można użyć wielokrotnie (przy zastosowaniu różnych kluczy) do zaszyfrowania danej wiadomości, a następnie odszyfrować ją w dowolnej kolejności. Otrzymany wynik będzie zawsze taki sam, bez względu na kolejność operacji. Nie należy szyfrować w ten sposób wiadomości więcej niż dwa razy, ponieważ ujawniają się wtedy podatności na ataki oparte na *chi skim twierdzeniu o resztach*.

Deszyfrowanie

Deszyfrowanie odbywa się za pomocą klucza prywatnego **(n, d)**. Szyfrogram składa się z kolejnych liczb, mniejszych niż **n**. Każda liczba wchodząca w skład szyfrogramu jest podnoszona modulo **n** do potęgi równej **d**:

$$m_i = c_i^d \pmod n$$

Otrzymywane liczby tekstu jawnego należy połączyć w odpowiedniej kolejności, aby utworzyły oryginalną wiadomość.

Uwierzytelnianie wiadomości

RSA może być zastosowane do podpisywania wiadomości. Wykorzystano to w projekcie. Nadawca musi wyliczyć wartość skrótu wiadomości (MD5), a następnie podnieść ją do potęgi **d** (modulo **n**), a więc wykonać te same operacje jak podczas zwykłego dekodowania szyfrogramów. Zakodowana w ten sposób wartość skrótu powinna zostać załączona do wysyłanej wiadomości.

Odbiorca wiadomości powinien podnieść zakodowaną wartość skrótu do potęgi **e** (modulo **n**), a następnie porównać wynik z wartością skrótu wyliczoną przez siebie. Jeżeli obie wartości są takie same, to można mieć pewność, że nikt nie zmodyfikował treści wiadomości. W tym projekcie weryfikacja jest dokonywana natychmiastowo po odszyfrowaniu wiadomości, a o jej wyniku odbiorca jest informowany stosownym komunikatem.

Siła RSA

W przypadku użycia małego wykładnika **e** (przykładowo równego 4) do zaszyfrowania małej wartości **m**(mniejszej niż $n^{1/e}$), otrzymana liczba szyfrogramu będzie miała wartość mniejszą niż wynosi wartość modułu **n**. Umożliwia to wyznaczenie wartości **m** przy użyciu zwykłych operacji arytmetycznych, co jest czynnością nieskomplikowaną i szybką.

Algorytm RSA jest deterministyczny, wobec tego szyfr ten jest podatny na ataki z wybranym tekstem jawnym. Można zaszyfrować wiele wiadomości za pomocą znanego klucza publicznego, więc napastnik może odgadnąć zawartość wcześniej przechwyconych

zaszyfrowanych wiadomości, przez porównywanie ich z wiadomościami utworzonymi przez siebie.

RSA jest generalnie uważany za bezpieczny system zabezpieczeń i jest powszechnie stosowany w przeróżnych aplikacjach, protokołach i rodzajach komunikacji.

Wstęp do zagadnień realizacji API i mechanizmów szyfrowania oraz instrukcja użytkowania

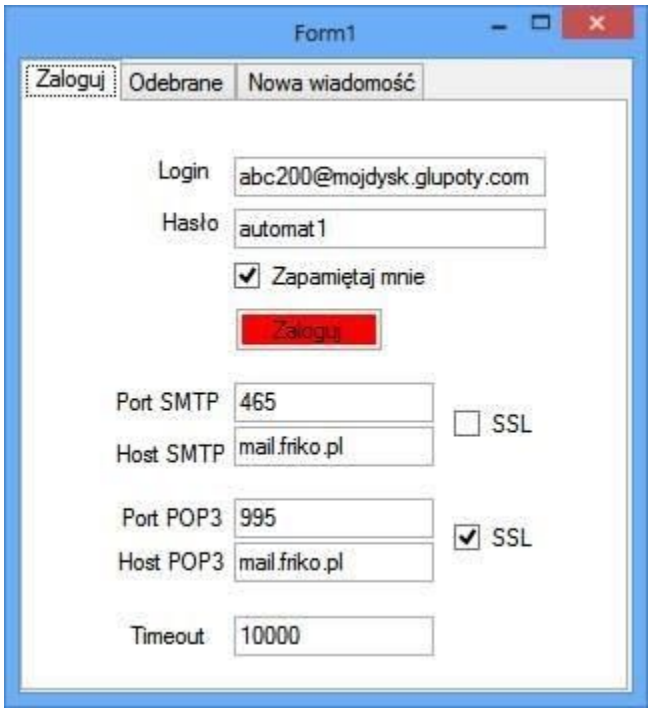
Projekt został zrealizowany z użyciem środowiska Visual Studio i języka C#. Wykorzystuje natywne elementy języka (i środowiska .NET) w celu połączenia z serwerem pocztowym i odbierania oraz wysyłania wiadomości (POP3 i SMTP). Konfiguracja klienta odbywa się z użyciem interfejsu okienkowego, co znacznie upraszcza proces konfiguracji.

Przy każdym uruchomieniu programu pojawia się okno dialogowe z prośbą o wskazanie istniejącego pliku z kluczami. Klucz publiczny jest używany do szyfrowania wiadomości, a prywatny – do odszyfrowania. Jeśli użytkownik nie wskaże pliku z kluczami, zamykając okno przyciskiem anuluj, to wyświetli się inny dialog – tym razem taki, który pozwala utworzyć nowy plik, zawierający klucze wygenerowane losowo przez program. Takie podejście pozwala zarówno rozpocząć komunikację z użyciem nowego zestawu kluczy, jak i kontynuować wymianę wiadomości używając klucza udostępnionego przez nadawcę. Zamknięcie obu okien dialogowych spowoduje wyjście z programu (ponieważ szyfrowanie bez klucza zapisanego na dysku spowodowałoby brak możliwości odszyfrowania wiadomości; klucze są usuwane z ramu w momencie zamknięcia programu).

Uwaga
W momencie zamknięcia programu użyte klucze są usuwane z dysku ze względów bezpieczeństwa. Należy je przenieść w bezpieczne miejsce przed zamknięciem programu.

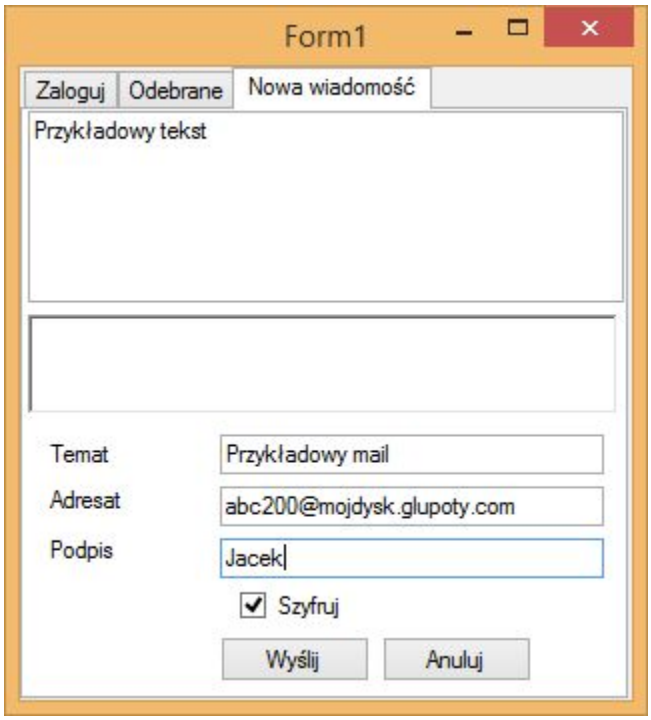
Po pomyślnym wygenerowaniu lub wczytaniu klucza należy podać dane autentyfikacji do skrzynki pocztowej, która będzie używana do komunikacji. Należy pamiętać o podaniu podstawowej konfiguracji portów oraz adresu serwera. **Następnie należy kliknąć czerwony przycisk Zaloguj.**

Forma główna programu została podzielona na dogodne zakładki. **Dopóki użytkownik nie zaloguje się, nie ma możliwości zmiany aktywnej zakładki.**

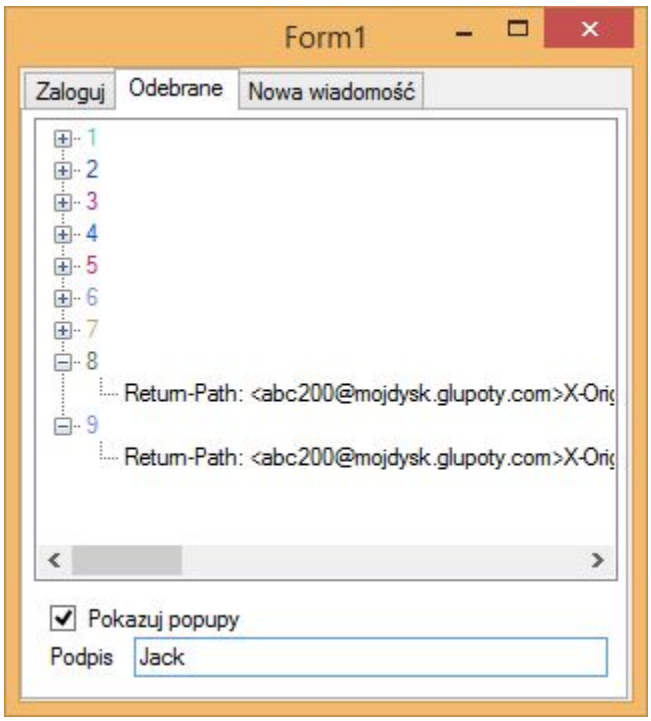


Rysunek 7. Interfejs u ytkownika

W drugiej zakładce [Rysunek 10] użytkownik ma możliwość otworzenia zawartości skrzynki odbiorczej za pośrednictwem protokołu POP3. **Aby podjąć próbę odszyfrowania wiadomości należy rozwinąć drzewo pod danym numerem wiadomości i kliknąć na wyświetlony tekst.** Ostatnia zakładka pozwala na utworzenie nowej wiadomości z opcją zaszyfrowania, z możliwością utworzenia bezpiecznego podpisu.



Rysunek 8 Okno tworzenia nowej wiadomo ci



Rysunek 9 Okno skrzynki odbiorczej z mo liwo ci podania podpisu do weryfikacji oraz wyboru sprawdzenia sumy kontrolnej i podpisu. Klikni cie na rozwini ty element drzewa powoduje otworzenie si okna dialogowego z tre ci wiadomo ci, oraz, je li zaznaczony jest checkbox, informacj o poprawno ci lub niepoprawno ci weryfikacji podpisu i sumy kontrolnej.

Koncepcja i realizacja zagadnień bezpieczeństwa - szyfrowanie

Pierwsza część projektu dotyczy zagadnienia szyfrowania. Wykorzystany szyfr blokowy z kluczem publicznym to szyfr AES z kluczem 256-bitowym, zapewniający wystarczający poziom bezpieczeństwa do komunikacji personalnej i korporacyjnej (*[3]*).

Najskuteczniejsze znane ataki pozwalają wykorzystać korelację bitów klucza do zredukowania jego efektywnej długości o maksymalnie dwa bity, co dalej stanowi potrzebę wykorzystania metody bruteforce dla klucza 126-bitowego. Jest to wyzwanie przekraczające możliwości współczesnej krypto analizy i możliwości obliczeniowych maszyn oraz zapotrzebowania na pamięć. *[2]*

Szczegółowy opis działania metody szyfrującej

W projekcie użyto trzech algorytmów:

- AES – odpowiada za zabezpieczenie treści wiadomości, jednak ze swojej symetrycznej natury jest średnio użyteczny w tym zastosowaniu, dlatego został niejako zagnieżdżony w RSA; użyto klasy RijndaelManaged*[4]*

- RSA – ze względu na złożoność obliczeń, jest użyte nie do szyfrowania całej wiadomości, a jedynie klucza AES oraz skrótów MD5: wiadomości (suma kontrolna pozwalająca się upewnić, że wiadomość nie została zmanipulowana) oraz skrótu podpisu. Użyto klasy RSACryptoServiceProvider *[6]*

- MD5 – użyte zostało tak jak napisano wyżej. Wykorzystano klasę MD5. *[7]*

Wymienione klasy znajdują się w przestrzeni nazw System.Security.Cryptography.

Szyfrowanie zasadniczo składa się z kilku głównych kroków:

1. Utworzenie kryptogramu AES z wiadomości.
2. Zaszyfrowanie klucza AES algorytmem RSA i dołączenie do kryptogramu z punktu 1 wraz z kilkoma dodatkowymi informacjami (szczegóły w komentarzach w kodzie)
3. Dołączenie do wiadomości skrótu MD5 podpisu i wiadomości (szczegółowe wyjaśnienie na następnej stronie przy omówieniu odszyfrowania)
4. Konwersja bajtów na CSV (omówione 3 akapity niżej).

Szczegółowy opis działania programu krok po kroku znajduje się w komentarzach w kodzie oraz wygenerowanej dokumentacji. Zaniechano próby opisu kodu w tym opracowaniu, ze względu na objętość kodu i jednoczesną niemożność omówienia zasady jego działania bez przytoczenia odpowiednich, sporych gabarytowo, fragmentów.

Został użyty również mechanizm strumieni pamięci *[5]* w celu łatwego operowania na ciągach bajtów.

Po zaszyfrowaniu wiadomość jest konwertowana bajt po bajcie (z użyciem konwersji byte->string) do postaci plain-tekstu w formacie CSV (Comma-Separated Values), gdzie przecinki oddzielają od siebie kolejne bajty i wysyłana z wykorzystaniem klienta pocztowego omówionego w poprzednim punkcie.

Przyjęcie formatu CSV zostało podyktowane tym, że **bajty wychodzące z enkryptora posiadają wartości 0-255, czyli pełnego zakresu Byte, a ich bezpośrednie przedstawienie spowodowałoby liczne błędy**, m.in. z powodu „zastrzeżonego" użycia zakresu 0-31 w standardzie ASCII. Rozwiązanie z konwersją byte->string->CSV powoduje wykorzystanie jedynie znaków ASCII z zakresu „bezpiecznego" dla usług pocztowych (należy mieć również na względzie to, że wiele serwerów pocztowych dokonuje konwersji z 8-bitowego standardu na 7-bitowy, co również może powodować problemy z kodowaniem)*[15].* Zastosowanie CSV powoduje maksymalnie około 4-krotny wzrost objętości danych (3bajty na cyfry + 1 bajt na przecinek), co nie powinno stanowić problemu przy współczesnej przepustowości łącz.

Wyjściowa wiadomość może wyglądać przykładowo:

<message>128,0,0,0,32,0,0,0,2,110,174,207,75,236,72,113,107,0,36,115,21,145,127,197,116,79,214,244,143,131,178,46,79,243,190,144,203,37,11,186,197,228,76,147,118,31,171,238,92,37,27,175,177,67,175,129,5,136,134,73,45,102,144,166,62,149,108,33,245,13,212,2,129,230,14,170,100,243,129,154,218,159,195,229,105,94,182,200,89,149,196,89,108,225,149,128,145,247,106,206,117,141,217,204,111,157,78,218,132,234,115,121,219,84,169,16,25,144,47,41,211,187,38,130,207,0,246,19,228,103,101,233,213,251,18,19,231,236,206,191,39,50,104,119,237,205,81,185,219,114,125,203,161,170,8,247,190,245,233,163,145,186,68,70,151,238,252,94,233,50,136,81,127,132,75,118,197,156,59,185,64,58,125,71,53,13,118,66,6,28,57,170,183,70,129,163,135,40,231,11,40,102,245,17,75,172,217,158,76,10,66,132,212,198,14,154,198,227,18,77,17,60,148,10,11,19,182,202,66,160,99,83,|24,61,158,51,47,178,139,50,97,141,80,132,162,55,37,19,72,125,162,39,129,14,21,108,238,180,165,72,85,25,69,147,176,141,254,149,47,28,54,44,45,154,238,30,100,73,218,174,88,29,232,204,53,12,144,250,143,44,61,214,211,206,39,105,86,112,237,13,181,138,48,232,18,66,226,202,62,108,81,156,46,53,172,241,47,112,249,251,99,50,89,252,41,123,229,151,108,20,245,209,77,154,233,180,245,137,185,204,33,178,224,18,121,134,1,63,236,180,177,3,122,113,40,57,51,100,150,214,|194,141,189,59,191,122,123,17,222,254,150,73,236,231,216,202,20,171,242,209,182,253,121,233,10,217,172,172,246,255,188,249,116,233,253,190,8,75,50,94,180,239,49,119,215,253,80,12,35,156,91,14,168,100,225,114,72,93,88,238,139,120,1,160,113,179,81,187,59,225,87,122,130,51,191,174,71,73,37,108,16,229,226,111,90,17,188,4,172,13,179,235,179,179,168,211,23,18,120,34,237,229,156,33,124,115,32,16,107,200,203,38,24,14,65,205,165,50,239,212,22,157,140,172,229,108,151,119,</message>

Zatem wzór wiadomości jest następujący:

<message> [bajty wiadomości CSV] | [bajty sumy kontrolnej CSV] | [bajty podpisu] </message>

Odszyfrowanie jest procesem odwrotnym do szyfrowania i składa się z części:

1. Rozdzielenie wiadomości metodą Split na części z wiadomością, sumą kontrolną i podpisem

2. Odszyfrowanie zaszyfrowanego RSA klucza AES oraz zaszyfrowanych: podpisu i sumy kontr.

3. Odszyfrowanie wiadomości i wygenerowanie jej skrótu MD5

4. Wygenerowanie skrótu podpisu wprowadzonego przez adresata

5. Porównanie skrótów z punktów 2,3 i 4 i weryfikacja sumy kontrolnej oraz podpisu z wyświetleniem stosownej wiadomości o skutku weryfikacji.

Wszystkie zastosowane algorytmy pochodzą z bibliotek środowiska .NET. Odszyfrowanie odbywa się po podaniu ustalonego podpisu przez odbiorcę. Proces odbywa się dokładnie odwrotnie do procesu szyfrowania. Zaszyfrowany plaintext w formacie CSV zostaje zamieniony na ciąg bitów i

podany na wejście funkcji odszyfrowującej. Po konwersji bitów na znaki ASCII odbiorca posługujący się poprawnym kluczem otrzymuje pierwotną wiadomość. Odbiorca może również otrzymać od programu powiadomienie o udanej lub nie weryfikacji podpisu.

Realizacja zagadnień bezpieczeństwa – podpisywanie

Jak opisano w poprzednim punkcie, szyfrowana wiadomość jest poddana działaniu algorytmu AES, którego klucz jest następnie szyfrowany algorytmem RSA. Do takiego kryptogramu dołączana jest suma kontrolna MD5 zaszyfrowana RSA o tym samym kluczu oraz podpis, definiowany przez nadawcę, będący stringiem dowolnej długości, który jest również skracany MD5 i szyfrowany RSA. Podpisem takim może być na przykład imię nadawcy.

Ponadto, **szyfrowana** suma kontrolna i fakt jej sprawdzenia po stronie odbiorcy, eliminuje możliwość ataków, m.in. z grupy man-in-the-middle.

W ten sposób adresat (podając ustalony podpis, np. imię nadawcy) jest w stanie potwierdzić tożsamość nadawcy. Potencjalny atakujący niewiele jest w stanie w tym momencie zrobić, gdyż zarówno suma kontrolna, jak i podpis są zaszyfrowane algorytmem RSA i bez znajomości klucza nie jest możliwe manipulowanie nimi w efektywny dla atakującego sposób.

Metody klas

Zostały wylistowane i opisane w oddzielnym pliku wygenerowanym przez program do automatycznego generowania dokumentacji.

Testy aplikacji

Aplikacja została przetestowana w kilku scenariuszach.

1. Otworzenie aplikacji z opcją wygenerowania klucza, wysłanie wiadomości oraz odszyfrowanie w ramach jednej sesji.
2. Otworzenie aplikacji z opcją wczytania istniejącego klucza, otworzenie istniejącej na serwerze wiadomości i jej poprawne odszyfrowanie.
3. Próba otworzenia (odszyfrowania) wiadomości z użyciem innego klucza prywatnego niż pasujący.
4. Próba odszyfrowania wiadomości z podaniem innego podpisu niż podpis nadawcy.

Zgodnie z oczekiwaniami, punkty 1 i 2 zakończyły się sukcesem, punkt 3 – błędem i nieotrzymaniem treści wiadomości, a punkt 4 – pomyślnym wyświetleniem wiadomości, z jednoczesnym wyświetleniem komunikatu o błędzie sprawdzenia podpisu cyfrowego.

Sprawdzono również, czy klucze generowane przez program są faktycznie losowe (przykładowe rozwiązanie ze strony Microsoft Developers Network zawierało istotne uchybienie polegające na generowaniu zawsze jednakowego klucza, co zostało naprawione).

Zastosowania praktyczne implementowanego zagadnienia i podsumowanie

Temat projektu w istocie sam odpowiada praktycznemu zastosowaniu. Projekt, rozwiązany za pomocą utworzonej aplikacji, może znaleźć zastosowania wszędzie, gdzie konieczna jest szybka i efektywna wymiana informacji tajnych lub poufnych. Poziom bezpieczeństwa jest gwarantowany na poziomie wystarczającym nawet dla zastosowań rządowych. Zagadnienie jest o tyle istotne, że z zebranych informacji [16] wynika, że niestety wiele serwerów pocztowych nie używa bezpiecznych połączeń do komunikacji między sobą, a jedynie na drodze klient-serwer, co jest istotną luką w systemie poczty elektronicznej, tworząc jednocześnie obszar dla zastosowania tego projektu

Przeprowadzone testy wykazały, że algorytm pozwala na szyfrowanie z wydajnością do 4megabitów na sekundę, co jest wystarczające nawet do szyfrowania niewielkich plików. W istocie, utworzony moduł szyfrujący ma możliwość podania na wejście nie plaintekstu, a ścieżki do szyfrowanego pliku, co przy niedużym nakładzie pracy może być użyte jako gotowe rozwiązanie również do celów komunikacji z szyfrowanymi załącznikami.

Bibliografia

[1] Joan Daemen, Vincent Rijmen, "The Design of Rijndael: AES – The Advanced Encryption Standard." Springer, 2002.

[2] "Kryptografia: teoria i praktyka zabezpieczania systemów komputerowych" . Mirosław Kutyłowski, Willy-B. Strothmann

[3] Henri Gilbert; Thomas Peyrin (2009-11-09). "Super-Sbox Cryptanalysis: Improved Attacks for AES-like permutations"

[4] Ou, George (April 30, 2006). "Is encryption really crackable?". Ziff-Davis. Archived fromthe original on August 7, 2010. Retrieved August 7, 2010.

[5] https://msdn.microsoft.com/pl-pl/library/system.security.cryptography.rijndaelmanaged(v=vs.110).aspx

[6] https://msdn.microsoft.com/pl-pl/library/system.io.memorystream(v=vs.110).aspx

[7] https://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider(v=vs.90).aspx

[8] https://msdn.microsoft.com/pl-pl/library/system.security.cryptography.md5(v=vs.110).aspx

[9] http://www.formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng.swf

[10] http://smrithisomanna.blogspot.com/2012/06/send-and-receive-mail-from-c-smtp-and.html

[11] http://www.crypto-it.net

[12] http://students.mimuw.edu.pl/~mr214564/smieci/AESmr214564/

[13] http://www.drzewo-wiedzy.pl/?page=artykul&id=105

[14] http://www.drzewo-wiedzy.pl/?page=artykul&id=88

[15] https://tools.ietf.org/html/rfc5721 Appendix A. akapit 3

[16] http://security.stackexchange.com/questions/51552/how-insecure-is-pop-imap-smtp