

Politechnika Częstochowska  
Wydział Elektryczny

Mateusz Iwańczak

Nr albumu 123167

Aplikacja klient-serwer z algorytmami  
szyfrowania symetrycznego.

Praca dyplomowa magisterska

Kierunek: Informatyka, studia stacjonarne

Specjalność: Systemy i sieci komputerowe

Promotor:

Prof. dr hab. inż. Andriy Kityk

Częstochowa, 2018/2019

## Spis treści

1. Wstęp .....	3
2. Cel i zakres pracy .....	4
3. Język C# i platforma .NET .....	5
3.1. Technologia .NET .....	5
3.2. Język C# .....	9
4. Problematyka architektury klient-serwer .....	12
4.1. Sieci komputerowe .....	12
4.2. Protokoły TCP i UDP .....	15
4.3. Protokół IP i adresy MAC .....	17
4.4. Programowanie klient-serwer .....	18
4.5. Aplikacje TCP i UDP .....	20
4.5.1. Połączenie TCP a architektura klient-serwer .....	20
4.5.2. Połączenie UDP a architektura klient-serwer .....	23
5. Algorytmy szyfrowania symetrycznego .....	26
5.1. Pojęcie szyfrowania symetrycznego .....	27
5.2. Szyfr strumieniowy .....	27
5.2.1. RC4 .....	28
5.3. Szyfr blokowy .....	29
5.3.1. Tryby szyfrowania blokowego .....	31
5.3.2. Blowfish .....	35
5.3.3. DES .....	37
5.3.4. 3DES .....	45
6. Projekt aplikacji klient-serwer z algorytmami szyfrowania symetrycznego .....	48
6.1. Założenia .....	48
6.2. Wymagania .....	49
6.3. Opis aplikacji .....	50
6.4. Testy programu .....	62
6.4.1. Testowanie na jednym sprzęcie .....	63
6.4.2. Testowanie na dwóch maszynach .....	71
7. Podsumowanie .....	80
Literatura .....	82
Spis rysunków .....	83
Spis tabel .....	87
Spis załączników .....	88
Dodatek A Spis zawartości dołączonej płyty CD .....	88

## 1. Wstęp

Współczesny świat i ludzie, którzy go tworzą, różnią się sposobem bytu od tego, jaki prezentowali ich przodkowie w czasach prehistorii, starożytności, średniowiecza czy nowożytności. Prezentowali inne poglądy na życie. Kreowali odmienną architekturę, budownictwo, ubiór, sztukę, kulturę. Sposób spędzania wolnego czasu przeżywali odkrywczo i oryginalnie. Praojcowie obecnego człowieka odkrywali, wynaleźli i opracowali najwięcej potrzebnych aspektów, stanowiących fundament dalszego istnienia ludzkiego. Jest pewna prawidłowość, jaka łączyła wszystkie pokolenia od zawsze - stanowiła nią komunikacja. Przedstawiciele gatunku ludzkiego porozumiewały się ze sobą mową i pismem, czyli wysyłano listy. Skutkowało to pragnieniem, aby zachować poufność informacji, którą jako nadawca chce zakomunikować odbiorcy. Miało to szczególne znaczenie w przypadku wojen, gdzie siły każdej ze stron starały się ukryć cenne dla przeciwnika zawartości wiadomości. Zmierzając do realizacji tego założenia, należało zwrócić uwagę na to, w jaki sposób to uczynić?

Zaczął opracowywać schematy, które miały pozwolić zrealizować tenże cel. W taki oto sposób powstał szyfr. Postać tajności informacji było istotne też w przypadku prywatnych listów rodzinnych, przyjacielskich itp. Juliusz Cezar skonstruował szyfr, który miał potencjalnie także przeznaczenie. Z biegiem czasu powstał komputer, a następnie Internet, stanowiący połączenie między nimi. Początkowo skonstruowano maszynę do pisania. Ewoluuował komputer w znacznym tempie, stając się maszyną liczącą, potem do łamania szyfrów, by stać się tym, czym jest dzisiaj. Jest środkiem do komunikacji między ludźmi, wymianą danych, zdobywaniem wiedzy, a co za tym idzie, zatarciem granic pomiędzy ludźmi, ich kulturami, tradycją i różnorodnymi dziedzinami wiedzy. Komputery, łącząc się ze sobą w sieci lokalne, bezprzewodowe i rozległe, potrzebują tego samego, co ludzie od zawsze potrzebowali, czyli ochronić komunikaty nadawane pomiędzy sobą. Zabezpieczenie informacji w sieciach komputerowych funkcjonuje pod nazwą szyfrowania. Nadawcą, szyfrującym wiadomość, jest klient; natomiast odbiorcą, deszyfrującym dane, jest serwer.

## 2. Cel i zakres pracy

Celem tenże pracy jest przybliżenie technologii bezpiecznej komunikacji architektury systemu komputerowego, jaką jest klient-serwer, a więc implementację mechanizmu szyfrowania danych z użyciem algorytmów symetrycznych w aplikacji klienta i serwera.

Praca obejmuje siedem rozdziałów wraz ze spisem bibliografii i dodatków w postaci płyty CD. Pierwszy rozdział obejmuje wstęp do niniejszej tematyki, drugi odsłania cel i zakres zadania do realizacji, trzeci przedstawia język do implementacji koncepcji tematu wraz z platformą, czwarty bazuje na strukturze klient-serwer, z kolei piąty opisuje algorytmy zawarte w projekcie, w szóstym zostały omówione najważniejsze aspekty w kreowaniu aplikacji, siódmy stanowi podsumowanie całego toku rozważań.

Zakres pracy składa się z następujących treści:

- Przegląd literatury w obrębie obiektowego języka programowania C# i platformy .NET, a także przygotowywanie aplikacji w rozdziale trzecim.
- Oględziny piśmiennictwa na temat architektury sieci komputerowej klient – serwer, czyli kluczowych pojęć (klient, serwer, TCP, UDP), oraz esencji tej oto struktury zawarte w czwartym rozdziale.
- Zestawienie materiałów w rozdziale piątym, obejmującym sferę algorytmów szyfrowania symetrycznego, mając na względzie termin szyfrowania symetrycznego, jego podziału na szyfr strumieniowy i blokowy, jak również charakterystykę ich wybranych przykładów (RC4 dla strumieniowego i dla blokowego: Blowfish, DES i 3DES), jak też trybów kodowania.
- Opracowanie programów, prezentujących działanie zabezpieczonej łączności układu klient – serwer, korzystających z systemu szyfrującego dane za pomocą metody szyfrowania symetrycznego.
- Badanie aplikacji, stanowiącej koncepcję metody szyfrowania symetrycznego w strukturze sieci komputerowej klient-serwer.

### **3. Język C# i platforma .NET**

Odnosząc się do sfery technicznej projektu, z pewnością pożądane staje się poruszenie tematyki języka programowania i tego, co za sobą niesie w wyzwaniu implementacji. Niewątpliwie są pozycje na liście potencjalnego wyznacznika. W związku z tym, iż projekt wykonano w języku C#, będzie obligatoryjne omówienie tego oto języka, a także platformy .NET, jako jego nieodłącznego towarzysza.

#### **3.1. Technologia .NET**

Na początku warto zapoznać się z platformą .NET, będącą sugestią firmy Microsoft w sierpniu 2000 roku jako nowatorski standard w kreowaniu aplikacji, przeznaczonych na platformę systemu operacyjnego Windows. Pospolite ustosunkowanie do tegoż wydarzenia cechuje stwierdzenie, iż jest to riposta na technologię Java przedsiębiorcy Sun i połączony z tą oto korporacją zakaz sądowy swobodnej edycji Javy. Elementami przytoczonej technologii jest środowisko uruchomieniowe nazywane .NET Framework, biblioteki klas, a także komponentów. Została przygotowana łącznie z językiem C#, który określono jako zasadniczy język programowania, dedykowanego dla nowego rozwiązania. Kreowane aplikacje nie mają wprost kompilacji do profilu maszynowego, charakterystycznego dla wybranej architektury sprzętu, którą może być procesor, system operacyjny, itp. Oprogramowanie zostaje przetłumaczone na pośredni język Microsoft Intermediate Language (MSIL), mający też nazewnictwo Common Intermediate Language (CIL), potem wspomniany kod pośredni zostaje zainicjowany za pośrednictwem środowiska Common Language Runtime (CLR). Reasumując powstaje warstwa pośrednia, dzieląca system operacyjny i utworzone oprogramowanie, stanowiąc utożsamienie z Javą. Plusem tego jest niewątpliwie zwiększenie bezpieczeństwa i ustabilizowania systemu operacyjnego, a także przenośność kodu w różnorodnych strukturach procesorów, a ponadto systemami operacyjnymi. Ze względu na to, że .NET stanowi standard European Computer Manufacturers Association (ECMA), tworzą się samodzielne implementacje wspomnianego środowiska, którego przykładami są projekt Mono ze sponsorem Novell i projekt dotGNU. Nie są to rozwiązania w

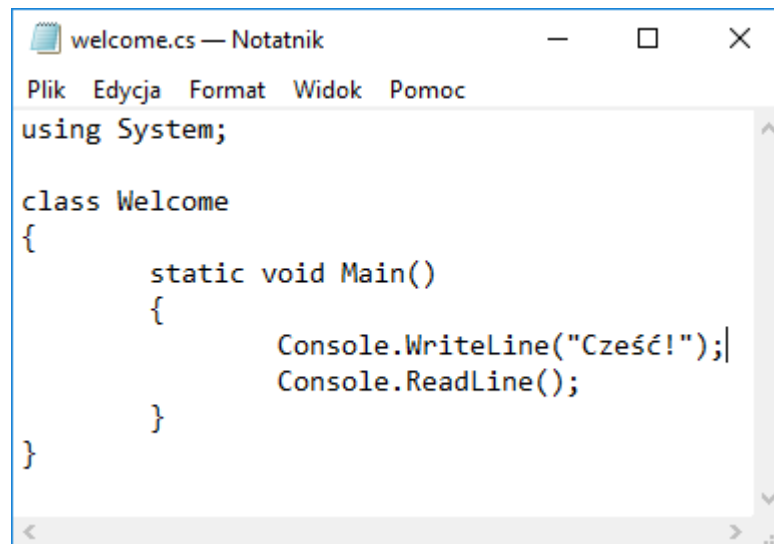
żadnym stopniu wspierane przez koncern Microsoft. Spółka nie ma postulatu, dotyczącego przenośności programu na system operacyjny inny od platformy Windows, co powoduje minimalne zaległości w wersjach rozwojowych od .NET i języka z nim powiązanego. Istnieje implementacja technologii dedykowana przenośnym urządzeniom z platformą Windows Mobile. Uruchamiając pierwszy raz oprogramowanie na określonym sprzęcie, rozpoczyna kompilacja w locie, transformująca kod pośredni w ten maszynowy, uzależniony od rodzaju platformy sprzętu i jego architektury, gdzie jest startowany. Technologia dysponuje dodatkowymi technologiami, jest możliwość tworzenia aktywnych stron internetowych za pośrednictwem ASP.NET, obsługa baz danych ADO.NET i XML, LINQ, a także mechanizm odpowiedzialny za implementację WMI, którym można przeprowadzić diagnostykę systemu operacyjnego i maszyny. Warto podkreślić, że wśród możliwości .NET są usługi sieciowe i Remoting. Zbiór kontrolek i komponentów stanowią biblioteki Windows Forms i Web Forms, gdzie jest możliwość kreowania aplikacji, a także skorzystania z odmiennej struktury kreowania graficznych interfejsów użytkownika, którą jest niewątpliwie XAML. Język C# w połączeniu z .NET zmierza w kierunku dynamicznego rozwoju, zwracając szczególną uwagę na wymagania, pochodzące od autorów oprogramowania, a także ukazujące się nowinki w technologii. Korporacja Microsoft skupia się na tym, by technologia, towarzysząca C#, była fundamentalną platformą dedykowaną dla OS Windows. Osiągnięcia są w połowie zrealizowane ze względu na fakt, iż w dziale programów o typie desktop w .NET w dalszym ciągu jest trudne do zdobycia dominowanie nad konkurencją taką jak WinAPI, czy Delphi VCL. Posługując się przykładem Microsoft Office 2007, sporządzonym w WinAPI, to jest szczególnie widoczne. Nie wynika to platformy samej w sobie, lecz korporacji w słabym tempie migrujących do nowatorskiego środowiska, co nie jest niczym tuzinkowym. Przykładowo o znacznych rozmiarach biblioteka Delphi VCL jest niezawodna z bardzo dobrze wyedukowanymi programistami Delphi. Mając sytuację idei migracji kodu źródłowego do C#.NET, jest popyt na cały rok implementacji przez programistów, by osiągnąć jednakowe rozwiązania w tymże języku. Personel do programowania musi zostać odpowiednio wyszkolony. Podsumowując składowe na cel są liczne, ponieważ rok implementowania, odpowiedni budżet na pracę, przeszkolenie i konsultacje, co kończy się powrotem do pozycji startowej. Ten argument

powoduje, że tylko nieliczne spółki mogą pozwolić sobie na tego typu inwestycje, ponieważ nie mają wystarczających środków. Istota wobec nowych firm wygląda inaczej i w tym przypadku korzystnie, ponieważ rozpoczynając działalność można dokonać wyboru, stawiając na C#.NET. Osiągnięcia technologii .NET zdobyła wysoki cel w dziale programów powiązanych bezpośrednio z siecią.[5]

Pragnąc zdefiniować według firmy Microsoft, czym jest .NET, należy interpretować pojęcie jako rozwiązanie skierowane pod usługi sieciowe, ponadto tworzy nowatorską generację oprogramowania, które scala się ze światem informacji, urządzeniami, a także osobami korzystającymi z produktu znormalizowaną i podmiotową metodą. O stworzeniu platformy .NET przeważały te oto postulaty:

- Łatwiejsze kreowanie programów
- Posiadanie możliwości komunikacji klient-serwer
- Trywialne przysposabianie aplikacji

Częściami składowymi platformy .NET są niewątpliwie środowiska .NET Framework, narzędzia konieczne dla deweloperów, a także serwery .NET Enterprise. Środowisko .NET Framework tworzą biblioteki, technologie wspomagające funkcjonowanie aplikacji, które odpowiadają za kierowanie pamięcią w oprogramowaniu. Narzędzia obligatoryjne dla deweloperów, służące do projektowania programów, tworzą pakiet Visual Studio.NET. W skład narzędzi wchodzi poza językiem programowania nowatorska baza danych, czyli MS SQL Server i technologia bazodanowa ADO.NET, ponadto w przypadku chęci projektowania w oparciu o technologie powiązane z siecią globalną, jest możliwość zarządzać technologią kreowania stron dynamicznych WWW, a mianowicie ASP.NET. Kompilator csc.exe odnosi się bezpośrednio do C#, został udostępniony bezpłatnie razem z .NET Framework. Zasadniczy kod przedstawia Rys. 3.1.

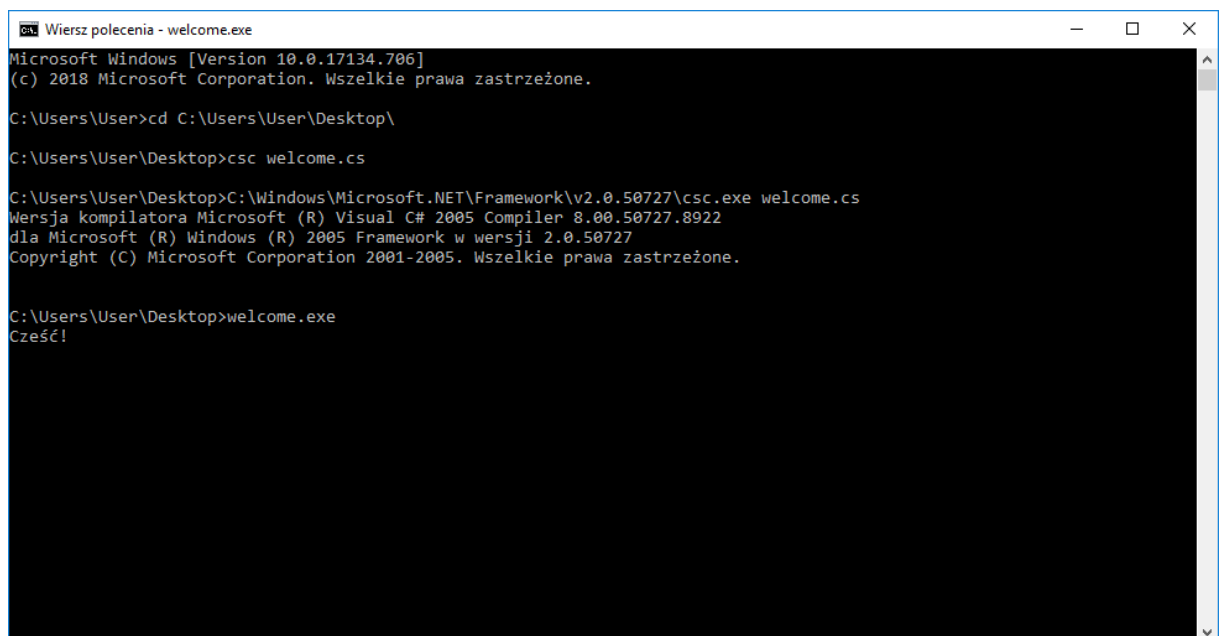


```
welcome.cs — Notatnik
Plik  Edycja  Format  Widok  Pomoc
using System;

class Welcome
{
    static void Main()
    {
        Console.WriteLine("Cześć!");
        Console.ReadLine();
    }
}
```

Rys. 3.1. Treść prostego programu sporządzonego w systemowym Notatniku [2]

Po zapisie kodu źródłowego z rozszerzeniem \*.cs należy odnaleźć kompilator w katalogu C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727. Po znalezieniu należy uruchomić wiersz poleceń (cmd.exe). Ustawić lokalizację w konsoli, a następnie wpisać polecenie csc nazwa\_pliku.cs, co skompiluje projekt. Każdy krok prezentuje Rys. 3.2.



```
Wiersz polecenia - welcome.exe
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\User>cd C:\Users\User\Desktop\

C:\Users\User\Desktop>csc welcome.cs

C:\Users\User\Desktop>C:\Windows\Microsoft.NET\Framework\v2.0.50727\csc.exe welcome.cs
Wersja kompilatora Microsoft (R) Visual C# 2005 Compiler 8.00.50727.8922
dla Microsoft (R) Windows (R) 2005 Framework w wersji 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. Wszelkie prawa zastrzeżone.

C:\Users\User\Desktop>welcome.exe
Cześć!
```

Rys. 3.2. Kompilacja i uruchomienie programu w Wierszu poleceń [2]

Należy odnaleźć plik \*.exe o tej samej nazwie, którą posiada plik z kodem źródłowym.



Serwer .NET Enterprise tworzy program z możliwością skorzystania z niego w połączeniu z aplikacjami .NET.

Niewątpliwie składowymi .NET Framework są Class Libraries (CL), czyli bibliotekami klas, gdzie znajdują się gotowe do użycia komponenty, funkcje, a także biblioteki z potencjalną możliwością użycia w klarowaniu projektu. Częścią CL jest między innymi WinForms – wizualna biblioteka. Kolejną składową jest Common Language Runtime (CLR), będące wspólnym środowiskiem uruchomieniowym (wykonywalnym). Ta oto technologia kontroluje uruchamianie programu .NET, prawidłowość pracy, również zwalnianie pamięci itp. Ostatnim elementem jest Common Type system (CTS), określane jako kolektywny system typów. Za jego pośrednictwem jest możliwość kooperacji między odmiennymi programami na platformie .NET.

Reasumując .NET tworzy progresywną strategię opracowaną przez Microsoft, wyznaczając przy tym nowe reguły w projektowaniu aplikacji. Odnosi się do infrastruktury, odnoszącej się do Internetu, zakłada pracę na wszystkich platformach i systemach. Warto zaznaczyć, iż nie reprezentuje grupy systemów operacyjnych. [2]

### **3.2. Język C#**

Język C# posiada różnorodne edytory kodu źródłowego. Pisać aplikacje można w trywialnej wersji Notatnika w systemie operacyjnym Windows, choć na rynku jest wiele darmowych i komercyjnych rozwiązań, które ułatwiają i umożliwiają pracę nad implementacją, udostępniając proste w obsłudze funkcje niezbędne w edycji kodu źródłowego. Przykładem zaawansowanego oprogramowania, dającego możliwość pisania i kompilacji programów w języku C# jest z pewnością pakiet Borland Developer Studio , którego twórcą jest Borland, będący produktem komercyjnym. Innym rozwiązaniem, które również stanowi reprezentację oprogramowania płatnego jest Visual Studio, które pozwala programować i kompilować, lecz posiada też uproszczone darmowe edycje, o które zatroszczyła się korporacja Microsoft.

Obligatoryjne w implementacji projektów jest z pewnością kompilator, dający możliwości generowania programów realizowanych w oparciu o kod w danym języku.[2] Kompilatorem jest program, który tłumaczy (kompiluje) kod

sporządzony w danym języku programowania na kod wiadomy dla określonego środowiska uruchomieniowego, którym jest sprzęt i system operacyjny, co rozwiązuje problem z tym, że sprzęt bez tej tot operacji nie ma możliwości zrozumienia kodu źródłowego w bezpośredni sposób.[7] Warto przypomnieć, iż kompilator dedykowany językowi C# musi być związany ściśle ze środowiskiem .NET. Niniejszy Framework stanowi aplikację z nazewnictwem csc.exe, które jest możliwy do realizacji z wiersza poleceń.

Plikiem wykonywalnym jest pospolity program z przeznaczeniem na system Windows z rozszerzeniem \*.exe, będący plikiem w formacie Portable Executable (PE), składający się ze skompilowanej wersji kodu źródłowego, czyli kodu maszynowego. Są również dane cechujące daną platformę, uniemożliwiając pracę oprogramowania na innym OS.

Pojęcie kod maszynowy obejmuje postać już przygotowaną do bezpośredniego, bądź pośredniego działania, realizowanego przez procesor.

Chcąc dokonać kompilacji należy sporządzić kod za pomocą właściwego języka programowania, następnie weryfikowana jest składnia przez kompilator. W przypadku błędów, wyświetlony zostaje błąd i następuje koniec pracy.

Proces prekompilacji opiera się na kasowaniu komentarzy z zawartości kodu źródłowego, są też odczytywane makrodefinicje dla języka ANSI C.

Fundamentalny język w tej oto pracy jest nowatorskim językiem programowania, którego stworzyła korporacja Microsoft, będąca nieodłącznie w relacji z platformą .NET. Jest sumą plusów, przemawiających za językiem C++ i Java. Ma podobną postać, co czyni go prostszym do opanowania przez programistów właśnie tych oto języków.[2] W odniesieniu do Javy, elementem składowym wspólnym dla tych języków są z pewnością mechanizmy odpowiedzialne za automatyczne odzyskiwanie pamięci. Jest językiem zorientowanym obiektowo, oprócz mechanizmów posługujących do odzyskiwania pamięci, posiada również możliwość obsługi wyjątków.[7] Przewodnim architektem C# jest Anders Hejlsberg, pochodzącym z Danii. Prowadził życie zawodowe w przedsiębiorstwie Borland, tam miał udział w kreowaniu środowiska Delphi, a także Turbo Pascal. Język C# cechuje specyficzność, programy koniecznie muszą posiadać środowisko uruchomieniowe CLR. Implementując w C#, wykorzystuje się funkcje, oferowane przez .NET Framework, więc programy potrzebują tegoż środowiska na sprzęcie, na którym są uruchamiane.

Rodzaje aplikacji w C# są trzy, w skład grupy wchodzi: aplikacja konsolowa, Windows Forms, a także formularze Web Forms. Aplikacje konsolowe uruchamiane są w oknie konsoli, nie mają interfejsu użytkownika, będąc użytecznymi w trywialnych problemach, nad którymi pracuje zawężona grupa osób z wiedzą adekwatną do korzystania z niego z linii poleceń. Windows Forms jest wizualną biblioteką, dzięki której projektuje się graficzny interfejs użytkownika, są tam formularze, a także komponenty. Natomiast formularze Web Forms są powiązane z kreowaniem stron dynamicznych WWW, wykorzystując przy tym technologię ASP.NET. Deweloper dysponuje formularzem obrazującym stronę WWW, która powstanie, a także grupę kontrolek. Komponenty mogą być wykorzystane, posiłkując się dynamicznym generowaniem kodu HTML, przeznaczonym dla przyszłej strony. Tworzenie przebiega wizualnie, umieszczając pożądaną element na stronie, natomiast środowisko programistyczne wykonuje w tle generację należytego kodu HTML, przetwarzanym przez przeglądarkę internetową.[2]

Nawiązując do standardów tegoż języka zorientowanego obiektowo, warto zwrócić uwagę, iż działanie projektowanych programów będzie możliwe tak na jednej z nowszych wersji np. 6.0 z .NET 4.6 i Visual Studio 2015, jak i w starszych wersjach, które mają swoje korzenie wiele lat temu, a dziś są rzadkością, czyli edycja 1.0 i 1.2, choć mogą zdarzyć się przypadki aplikacji, które mogą nie pracować. Zachowana jest niewątpliwie zgodność z wersją 4.0 z platformą .NET Framework 4 i środowiskiem Visual Studio 2010, a także 5.0 z cechującymi ją .NET Framework 4.5 i Visual Studio edycji 2012 i 2013.[7]

## 4. Problematyka architektury klient-serwer

Architektura klient-serwer jest ściśle związana z programowaniem sieciowym, a projektowanie aplikacji sieciowych obligatoryjnie powinno mieć fundament w postaci wiedzy na temat pojęć, obejmujących sieci komputerowe. Brak opanowania materiału dotyczącego protokołów sieciowych stanowi sporo problemu niż ułatwień. W hipotetycznej sytuacji może pojawić się fakt, iż oprogramowanie kreuje w dużym zakresie ruch w sieci, jest niestabilne, nie posiada odpowiednich zabezpieczeń. Suma trzech przytoczonych czynników sprawia, że poufne dane stają się łatwym celem dla osób, które nie powinny mieć najmniejszego dostępu do nich.[5]

### 4.1. Sieci komputerowe

Koncepcja połączenia komputerów, by istniała możliwość wymiany informacji, ukazała się wtedy, gdy komputery stały się pospolite w globie. Chcąc dokonać charakterystyki pojęcia sieci komputerowe, należy odnieść się na początek do modelu Open System Interconnection (OSI). Prezentuje on zasadniczą strukturę sieci komputerowych, gdzie znaczące funkcje zostają przyporządkowane do określonej warstwy ze zbioru, liczącego łącznie siedem poziomów. Rozdzielenie jest sferą umowną, lecz daje to spore ułatwienie w przyswojeniu pracy każdego ze składowych sieci komputerowej, rozpoczynając od kabla sieciowego, kończąc na programie. Model DoD składa się z czterech warstw i w przystępniejszy sposób obrazuje strukturę sieci globalnej, definiowany jest też jako model odniesienia TCP/IP. Porównanie warstw dla obydwu modeli OSI i DoD obrazuje Tab. 4.1.

Tab. 4.1. Modele warstwowe sieci [5]

Aplikacja	Aplikacja
Prezentacja	
Sesja	
Transport	Transport
Sieć	Sieć
Łączy danych	Interfejs sieciowy
Warstwa fizyczna	
Model OSI	Model DoD

Warstwy aplikacji, prezentacji i sesji w modelu OSI, tworzą jedną wspólną warstwę modelu TCP/IP, a poziom łącza danych i warstwa fizyczna są łącznie interfejsem sieciowym. Warstwa aplikacji jest w największym stopniu znana dla użytkownika, ponieważ w niej funkcjonuje każde oprogramowanie, dotyczące sieci. Doskonałym przykładem jest z pewnością przeglądarka internetowa. Zbiorowy format dostępu do informacji, które są przekazywane między komputerami tworzy warstwa prezentacji, za jej pomocą informacje są jasne dla warstwy aplikacji w innej strukturze, a także ponosi odpowiedzialność w kwestii kompresowania i szyfrowania danych. W warstwie, dotyczącej sesji, zostaje tworzona sesja wymiany danych w układzie dwóch komputerów, zarządzana, a także zamykana. W tejże warstwie są jawne informacje, jakie oprogramowanie komunikuje się z innym, jak również zostaje tam działanie tychże programów synchronizowane. W kwestii warstwy transportowej jest możliwość określić, iż dokonuje się tam segregacja przesyłanych danych na segmenty na komputerze nadawcy, następnie są one scalane na komputerze odbiorcy. We wszystkich transmisjach można wyodrębnić jej własny priorytet i opisane pasmo transmisji. Dane, które zostają wysłane, a także odbierane, podlegają kolejkowaniu. W przypadku pojawienia się błędu, istnieje możliwość wznowienia procesu transmisji. Jest to punkt wychwycenia i zreperowania błędów, łączących się z transmisją danych. Odpowiedzialność za połączenie między hostami i selekcja trasy dla pakietów danych, opierających się na topologii sieci, służy na warstwę sieciową. Nie ma gwarancji, że transmisja dojdzie do skutku. Zostaje przekazany logiczny system adresowania, co pozwala routerom opisać trajektoria podróży pakietów, czyli numerom IP, a przykładem protokołów pracujących na tym poziomie jest IP, ICMP, ARP. Na adresie MAC karty sieciowej funkcjonuje warstwa, powiązana z łączem danych, która pracuje nad fizyczną dostępnością do mediów sieciowych typu karta sieciowa, czy kabel. Mają swój istotny udział też w tym miejscu technologie Ethernet, Wi-Fi, czy Token ring. Fizyczny poziom w modelu stanowią karty sieciowe, kable, światłowody, czy fale radiowe, które koncentrują się na wykonywaniu przesyłu bitu po bicie danych od nadawcy do odbiorcy, a zbiór standardów tej oto warstwy obejmuje 10Base-T, 100Base-T, czy 802.11. Trzy warstwy, najwyżej znajdujące się w modelu OSI, obejmują aplikacje związane z siecią, a cztery dolne pilnują poprawność przepływu danych. Protokoły internetowe posiłkują się różnymi protokołami sieciowymi, przykładem tejże relacji jest DNS wykorzystujący protokół

UDP, pochodzący od warstwy odpowiedzialnej za dzielenie wysyłanych danych na kawałki u nadawcy i składanie ich u odbiorcy.

Rozpatrując przekaz danych między dwoma hostami w sieci globalnej, biorąc pod uwagę model TCP/IP, warto rozpocząć od warstwy aplikacji, w której wysyłane są dane, tam działania posilkują się na strumieniach danych. W technologii .NET klasa `NetworkStream` pełni funkcję, uosabiającą strumień sieciowy, daje możliwość wymiany danych. Warstwa transportowa dzieli strumienie na właściwe segmenty, a sieciowa dostarcza im nagłówki sieciowy wraz z adresem celu i nadawcy, ponadto wszystkie segmenty są transformowane w pakiety. Poziom, który reprezentuje interfejs sieciowy, niniejsze dane są przekazywane bit po drugim do odbiorcy. Odbiorca, mając zamiar odczytać informację, jest zobligowany wykonać proces odwrotny do wcześniej omówionego. Cała ścieżka od warstwy aplikacji do tej fizycznej określa się mianem inkapsulacji, z kolei w odwrotnym kierunku nazywa się deinkapsulacją.

Tab. 4.2. Model DoD i popularne protokoły sieciowe [5]

FTP, HTTP, SMTP, POP3, WINS, SSL	DNS, TFTP	Aplikacja
TCP	UDP	Transport
IP		Sieć
Internet	LAN, WAN	Interfejs sieciowy

Powyższa Tab. 4.2 odzwierciedla z pomocą modelu DoD, jak funkcjonują protokoły sieciowe. Przykładowym protokołem jest FTP, gdzie jest uruchomiony klient FTP na sprzęcie, nawiązuje połączenie z serwerem FTP i zaczyna przysyłać plik. Klient wykonuje swoje zadanie w warstwie aplikacji, gdzie ma miejsce metamorfoza pliku na strumień danych. Protokół FTP korzysta w warstwie transportowej z protokołu TCP w celu dokonania klasyfikacji strumienia na segmenty, aby w późniejszym etapie już w warstwie sieciowej dodać do każdego z nich przypisany nagłówek IP, następnie przekazać ten oto plik do serwera FTP. Tenże serwer, biorąc go, w dalszym ciągu wykonuje operację odwrotną, osiąga pakiety IP z szeregu bitów, z tychże pakietów otrzymuje segmenty, a z nich strumień, potem ze strumienia wynikiem staje się plik.[5]

## 4.2. Protokoły TCP i UDP

Fundament istnienia Internetu z pewnością stanowi para protokołów, którymi są TCP/IP, a także UDP. Termin protokołu obejmuje szereg reguł, którym obligatoryjnie podporządkowują się wszystkie maszyny, prowadzące komunikację ze sobą nawzajem. W sieci rozległej pierwsze miejsce pod względem komunikacji sprzętów osiąga TCP/IP, który stworzony został na początku lat 70. XX wieku przez amerykańską agencję Defense Advanced Research Projects Agency (DARPA). Zwracano szczególną uwagę na utrzymanie transmisji danych mimo to, iż stracono wybiórcze fizyczne połączenia. Niniejszy zbiór zasad winien samodzielnie odnaleźć adekwatną trasę pomiędzy parą sprzętów, które mają miejsce w sieci., którą pierwszą stanowi niewątpliwie ARPANET. W oparciu o protokół TCP/IP w wersji czwartej osiągnięto skutecznie połączenie między uniwersytetem Stanforda, a University College London, co miało miejsce w 1975 roku. W następnych latach w stosunku do sieci komputerowych postęp zmierzał, dając jako medium pośredniczące w wymianie informacji między zakładami badawczymi, a także naukowymi. Jak sieć komputerowa stała się pospolita, w niesamowitym tempie nastąpiła ewolucja ogólnosiwiatowej sieci komputerowej, określonej mianem Internetu, która scala lokalną sieć komputerową (LAN) z bezprzewodową siecią lokalną (WLAN). Protokół TCP/IP w wersji czwartej stanowi w obecnym czasie jest pełnomocną normą w sieci rozległej, a wersja szósta nieprzerwanie nie urzeczywistnia się na masową skalę. Innym nazewnictwem dla zbioru TCP/IP jest stos TCP/IP. Zasadniczymi protokołami, reprezentującymi gmach warstwy transportowej, są niewątpliwie TCP i UDP. Ten pierwszy protokół transportowy cechuje się połączeniowością i niezawodnością, daje rzetelną łączność wykonywaną dzięki sumom kontrolnym i numerom następczym pakietów. W przypadku zniszczenia lub zaginięcia w czasie podróży pakietu, sprzęt odbiorca zawiadamia o konieczności retransmisji. Wnioskiem jest to, iż protokół daje pewność dostarczenia wszelkich pakietów. Nawiązanie połączenia realizuje się dzięki uzgadnianiu trójstopniowemu. W celu przeprowadzenia połączenia, sprzęty koniecznie aktualizują numery sekwencyjne TCP, czyli numery INS, które stosowane są do rozpoznania kolejności pakietów i weryfikacji tego, czy wszelkie przesłane pakiety odebrano przez docelowego hosta. Rozpoczynający komputer komunikuje pakiet z określonym numerem ISN i z uniesioną wyłącznie flagą ACK. Warto zaznaczyć, iż warstwę

transportową cechują segmenty, a każdy pojedynczy ma możliwość potencjalnie mieć jeden pakiet IP lub więcej. Niewątpliwie chcąc skutecznie przekazać informacji, TCP używa okna, by sprecyzować liczbę pakietów, mogących odbierać docelowy host zanim zostanie przesłane potwierdzenie, wspomniane okna posiadają opcję dynamicznej edycji w trakcie transmisji, choć najlepiej zachować wielkość okna równą jeden, gdyż jest to najbardziej optymalne. Protokół bezpołączeniowy UDP nie posiada mechanizmu, weryfikującego tak pakiety wysyłane, jak i te odbierane, a maszyna, przesyłająca pakiet UDP, nie zachowuje go w pamięci. Tenże protokół porównany z TCP tworzy w dużo mniejszym wymiarze ruch sieciowy, co działa na jego korzyść, dając większą prędkość. Wadą jest niewątpliwie brak gwarancji tego, że dane zostaną przekazane. W UDP pakiety mogą potencjalnie posłużyć do różnorakiego typu rozgłoszenia w lokalnych sieciach komputerowych, a tam właśnie prawdopodobieństwo zaginięcia pakietu jest niskie. Dające wymianę informacji protokoły warstwy aplikacji, czyli HTTP lub FTP mają obowiązek działać w oparciu o łączność TCP. Jeśli protokół TCP wykazuje próbę nawiązania połączenia, a host na celu nie jest dostępny lub odmawia łączności, wtedy funkcja kreująca połączenie wykazuje wyjątek, inaczej wygląda obraz sytuacji w przypadku UDP, gdzie nie ma żadnych zgłaszanych wyjątków. Niniejsze rozwiązanie jest skuteczne w sytuacji, gdy jedno oprogramowanie osiąga dostępność do sieci, a kilka programów z warstwy

Tab. 4.3. Lista najpopularniejszych portów i usług do nich przypisanych [5]

20	FTP dane
21	FTP
22	SSH
23	Telnet
25	SMTP
53	DNS
80	HTTP
110	POP3
115	SFTP
119	NNTP
143	IMAP
161	SNMP
222	RSH
443	HTTPS
995	POP3S



aplikacji w modelu TCP, pragnąc wykazać wolę wykorzystania sieci, stosuje porty. Wszystkie programy mają przypisany swój numer portu, gdzie może otrzymywać dane. Segment TCP ma składowe takie jak pola, gdzie są przetrzymywane numery portu źródłowego i docelowego, co jest dowodem na poparcie tezy, iż warstwa transportowa ma informacje, w które miejsce ma zostać przekazany już odebrany segment. Numer portu może być w zakresie od 0 do 65535. Wybrane porty, które są tymi najpopularniejszymi, wraz z przypisanymi do nich usługami zostały umieszczone w Tab. 4.3. [5]

### **4.3. Protokół IP i adresy MAC**

Na poziomie warstwy sieciowej wykorzystuje się do identyfikowania komputerów adresy IP, który prezentuje się w formacie dziesiętnym o długości 32 bitów, gdzie poszczególne z czterech oktetów zostają odseparowane od siebie kropkami. Stanowią adresy logiczne, czyli funkcjonują poprzez nie aktywne urządzenia sieciowe, do których z pewnością można zaliczyć routery. Przed zdobyciem określonego komputera przez pakiet zostaje osiągnięta sieć, której elementem składowym jest dany komputer. Adres MAC karty sieciowej znajduje się w warstwie fizycznej, opisany jest jako unikatowy kod długości 48 bitów, gdzie 24 bity na początku są odpowiedzialne za kod producenta, natomiast pozostałe na końcu 24 bity stanowią kod, bezpośrednio odnoszący się w stosunku do karty. Adresy IP posiadają podział na klasy:

- Klasa A – w niej pierwszy oktet opisuje adres sieci, następne określają unikalny adres dla węzła sieci. Powoduje to, iż nie ma wiele sieci tej klasy ( $2^7 = 127$ ), z kolei pojedyncze sieci mają możliwość zaadresować przeszło 16 milionów węzłów, czyli  $2^{24}$ . Adresy tego typu klasy przypisane zostają znaczącym instytucjom, a także organizacjom rządowym. Zakres obejmuje 10.0.0.0 – 126.255.255.255.
- Klasa B – posiada dwa pierwsze oktety do identyfikowania sieci, co daje do dyspozycji 16284 sieci, gdzie pojedyncza może zaadresować liczbę komputerów o wartości 65535. Możliwość opisanie mieści się w przedziale 128.1.0.0 – 191.255.255.255.

- Klasa C – ma największą ilość, bo  $2^{21}$ , lecz na każdą sieć przypadają maksymalnie 254 komputery.

W odniesieniu do przestrzeni adresowych dla opisanych klas istnieje możliwość wystąpienia adresów prywatnych, wykorzystywanych wyłącznie w lokalnych sieciach komputerowych, którymi są:

- Adresy prywatne z klasy A: są w zakresie 10.0.0.0 – 10.255.255.255
- Adresy prywatne z klasy B: posiadają przedział 172.16.0.0 – 172.31.0.0
- Adresy prywatne z klasy C: mieszczą się w przedziale 192.168.0.0 – 192.168.255.0

Niniejsze adresy nie posiadają widoczności w Internecie. Istotne jest to, iż adres pętli zwrotnej dla interfejsu sieciowego 127.0.0.0/8. Z terminem adresu IP posiada związek również z pojęciem maski podsieci, a także adresu sieci. Maskę podsieci definiuje, jakie bity adresu IP ponoszą odpowiedzialność w kwestii adresu sieci, jak również adres węzła, za jej pośrednictwem urządzenia sieciowe mają możliwość opisać sieć, której składową jest określony komputer. Zwiększenie liczebności komputerów z łącznością do sieci rozległej powoduje wyczerpywanie się zasobów możliwych adresów IP, co skutkuje tym, iż wybiórcze sieci lokalne wykorzystują jeden publiczny adres IP, bądź jest ich kilka. Z kolei, każdy z komputerów w tej oto sieci ma swój adres prywatny, natomiast komputer lub router, stanowiący łącznik tejże sieci z siecią rozległą, posiada adres IP o statusie publicznym, co określane jest mianem techniki Network Address Translation (NAT), bądź maskarada. Istotą tego jest translacja adresów sieciowych, a co warto podkreślić komputery we wnętrzu sieci mają utrudnienie funkcjonowania w roli serwerów internetowych, ponieważ przyjęcie tegoż serwera łączy się z należytą konfiguracją urządzenia, które ma udostępnić drogę do sieci globalnej.[5]

#### **4.4. Programowanie klient-serwer**

Model klient-serwer stanowi fundament wielu z usług w Internecie. Klient jest programem, który nawiązuje połączenie z serwerem i otrzymuje wejście do tej oto świadczenia, a także przesyła żądanie. Serwer stanowi oprogramowanie lub grupę z tymi składowymi, dający dostęp do określonego świadczenia, obsługuje żądanie

klienta. Przykład klienta odzwierciedla przeglądarka internetowa, natomiast serwerem jest WWW. Z usług, oferowanych przez serwer, może korzystać więcej niż jeden klient, a pojedynczy klient ma możliwość nawiązania relacji z wieloma serwerami. Ten fakt daje łatwiejszą edycję danych w dowolnym miejscu, lepszą wydajność, a także skalowalność. Główny serwer upraszcza zarządzanie grupą użytkowników, jednak stanowi słaby punkt dla danej struktury w przypadku awarii, a wtedy klienci nie mogą się w ogóle połączyć. Jeśli są istotne usługi, pojawiają się dodatkowe serwery z możliwością natychmiastowego przejęcia kontroli w sytuacji uszkodzenia zasadniczego serwera. Pospolicie programy serwerowe zostają zainstalowane na komputerach, cechujących się dużą mocą. Klienci mogą zaskładać na dowolnym sprzęcie do użytku domowego. Jeśli serwer ma być widoczny w sieci rozległej, koniecznie powinien mieć zarejestrowany niezmienny i publiczny adres Internet Protocol. Nie ma możliwości takie rozwiązanie w przypadku neostrady, w której niniejszy adres zostaje przyporządkowany dynamicznie na jedną sesję do danego komputera. Gdy nie istnieje dostęp do sieci, wtedy oprogramowania funkcjonują na adresie pętli zwrotnej.

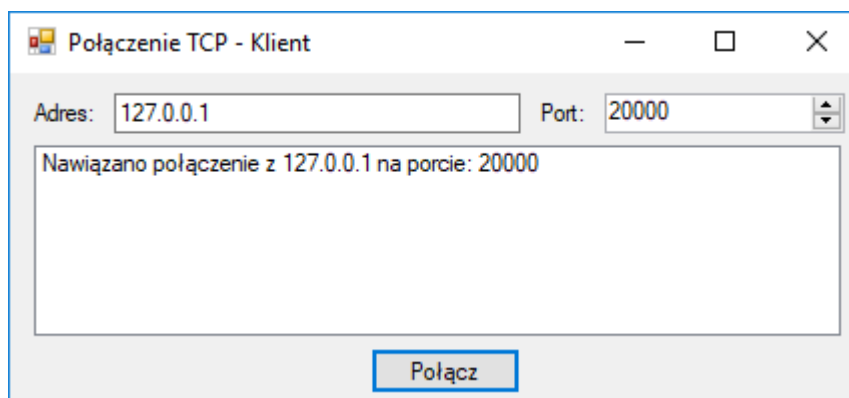
Rozważając architekturę klient-serwer, warto scharakteryzować połączenie peer-to-peer (P2P), który daje pewność każdej stronie równorzędnych praw, czyli nie istnieje centralny serwer, a wszystkie komputery mogą być jednocześnie tak serwerem, jak i klientem. Najpowszechniejszą implementacją tegoż rozwiązania są aplikacje z przeznaczeniem do wymiany plików w sieci internetowej. Działanie wymiany danych przebiega bez uczestnictwa fundamentalnego serwera, a także każdy komputer w sieci w tym samym czasie odbiera i przesyła informacje. Struktura sieci scharakteryzowanej cechuje się płynnością i podlega bieżącym komputerom, które tą oto sieć tworzą. Nie mając głównego serwera nadzorującego sieć, nie ma żadnej opcji tego, by kontrolować treść upublicznianych informacji. Są to sieci, które są najbardziej podatne na łamanie praw autorskich, ponieważ tam dochodzi do transakcji nielegalnych plików typu filmy, gry czy mp3. Warto zaznaczyć, iż mając aplikacje, służącą do wymiany plików w sieci peer-to-peer, nie stanowi przestępstwa, lecz szerzenie plików bez praw autorskich obligatoryjnych do ich rozpowszechniania.[5]

## 4.5. Aplikacje TCP i UDP

Ten oto podrozdział omawia zasadniczą strukturę i wszelkie niezbędne składowe konieczne do utworzenia prostych projektów o układzie klient-serwer w odniesieniu od protokołów TCP i UDP.

### 4.5.1. Połączenie TCP a architektura klient-serwer

Pragnąc stworzyć program klienta, nawiązujący połączenie za pomocą protokołu TCP wraz z wyznaczonym komputerem w sieci, należy pamiętać o tym, by użyć do jego budowania przestrzeni nazw System.Net.Sockets, ponieważ dzięki niej można skorzystać z klasy TcpClient. Przykład trywialny oprogramowania klient składa się z kontrolki textBox, numericUpDown, listBox, button i dwóch label'ów, pokazany na Rys. 4.1. Pole tekstowe służy do wpisania adresu hosta, kontrolka numericUpDown odpowiada za wybór portu sieciowego z zakresu od 0 do 65535.



Rys. 4.1. Wygląd działającego projektu aplikacji TCP Klient [5]

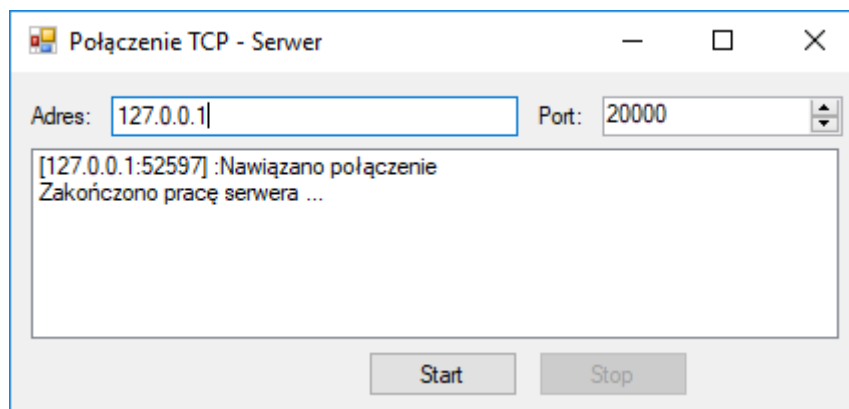
W listBox są zawarte wszelkie komunikaty na temat nawiązania połączenia, a przycisk ma funkcję, związaną z próbą nawiązania połączenia. Opisy pól, wprowadzających adres IP i port należą do obydwu kontrolki label. Kod źródłowy do niniejszej aplikacji przedstawia Rys. 4.2. Jeśli nie ma połączenia z Internetem, można alternatywnie do połączenia ze zdalnym hostem, za pomocą adresu pętli zwrotnej, a mianowicie 127.0.0.1, nawiązać łączność ze samym sobą. W polu tekstowym adres można wpisywać zarówno nazwy DNS, jak i numer IP. W celu skorzystania z

aplikacji może okazać się ustawienie zapory sieciowej w taki sposób, by możliwe stało się nawiązanie łączności aplikacji z Internetem.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
namespace PolaczenieTCPKlient
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            string host = textBox1.Text;
            int port = System.Convert.ToInt16(numericUpDown1.Value);
            try
            {
                TcpClient klient = new TcpClient(host, port);
                listBox1.Items.Add("Nawiązano połączenie z " + host + " na porcie: " + port);
                klient.Close();
            }
            catch (Exception ex)
            {
                listBox1.Items.Add("Błąd: Nie udało się nawiązać połączenia!");
                MessageBox.Show(ex.ToString());
            }
        }
    }
}
```

Rys. 4.2: Kod klienta TCP [5]

Serwer jest tym oprogramowaniem, które oczekuje na połączenie, a gdy nawiąże je, następuje wyświetlenie należytego komunikatu po jego stronie. Poniższa aplikacja zawiera zasadnicze aspekty obsługi połączenia z perspektywy serwera. Są identyczne komponenty, co w przypadku klienta, a ponadto jest jeden przycisk więcej. Przyciski odpowiadają za następujące funkcje. Przycisk Start zamraża interfejs i czeka na połączenie ze stroną klienta, natomiast Stop kończy działanie z perspektywy serwera.



Rys. 4.3. Wygląd działającego projektu aplikacji TCP Serwer [5]

Do projektu należy oprócz przestrzeni nazw System.Net.Sockets dodać również System.Net, ponieważ on dostarcza konieczną klasę IPAddress (Rys. 4.4). Wciskając Start na podstawie adresu IP i numeru portu jest otwierany port i przyłączany jest klient po stronie serwera do przyszłej wymiany informacji. Z kolei klikając stop dokonuje się wyłączenie serwera i zamykany zostaje klient po stronie serwerowej.

Test relacji obu projektów polega na wpisaniu w oprogramowania tak serwera, jak i klienckiego w pola adresów 127.0.0.1, czyli adresu pętli zwrotnej w interfejsie sieciowym, następnie wybranie portu sieciowego np. 20000. Klikając Start na serwerze, następuje oczekiwanie na relacje ze strony klientów (Rys. 4.3). W następnej kolejności należy kliknąć Połącz po stronie klienckiej, co kończy testowanie struktury.[5]

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
using System.Net;
namespace PolaczenieTCPserver
{
    public partial class Form1 : Form
    {
        //pola prywatne
        private TcpListener serwer;
        private TcpClient klient;
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            IPAddress adresIP = null;
            try
            {
                adresIP = IPAddress.Parse(textBox1.Text);
            }
            catch
            {
                MessageBox.Show("Błędny format adresu IP!", "Błąd");
                textBox1.Text = String.Empty;
                return;
            }
            int port = System.Convert.ToInt16(numericUpDown1.Value);
            try
            {
                serwer = new TcpListener(adresIP, port);
                serwer.Start();
                klient = serwer.AcceptTcpClient();
                IPEndPoint IP = (IPEndPoint)klient.Client.RemoteEndPoint;
                listBox1.Items.Add("[ "+IP.ToString()+" ] :Nawiązano połączenie");
                button1.Enabled = false;
                button2.Enabled = true;
                klient.Close();
                serwer.Stop();
            }
            catch (Exception ex)
            {
                listBox1.Items.Add("inicjacja serwera nie powiodła się!");
                MessageBox.Show(ex.ToString(), "Błąd");
            }
        }

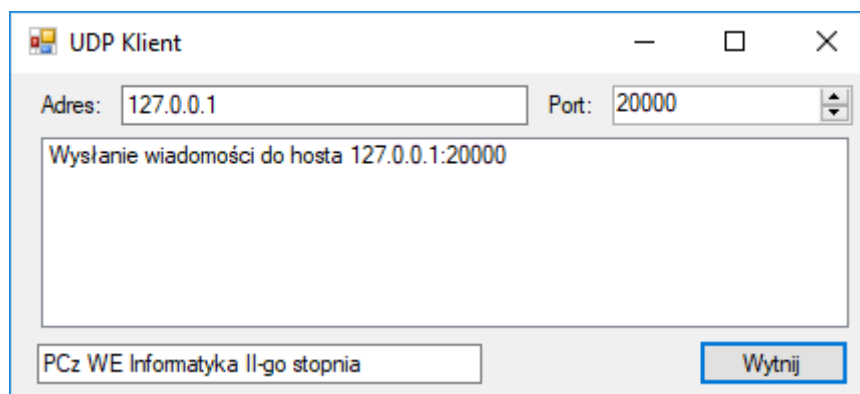
        private void button2_Click(object sender, EventArgs e)
        {
            serwer.Stop();
            klient.Close();
            listBox1.Items.Add("zakończono pracę serwera ...");
            button1.Enabled = true;
            button2.Enabled = false;
        }
    }
}
```

Rys. 4.4. Kod serwera TCP [5]

#### 4.5.2. Połączenie UDP a architektura klient - serwer

Projekty tak klienta, jak i serwera z wykorzystaniem protokołu UDP są zamiennikami poprzednio omówionymi aplikacjami opartymi na protokole TCP, są jednak pewne niuanse, którymi się różnią. Działają podobnie jak poprzednicy z przestrzenią nazw System.Net.Sockets, która udostępnia potrzebne klasy w implementacji (Rys. 4.6).

Do projektu klienckiego jest dodana kontrolka pola tekstowego, gdzie należy wpisać wiadomość do przesłania, przycisk Wytnij pobiera nazwę hosta, port sieciowy i wiadomość, by przesłać ją pod odpowiedniego adresata. Jeśli jest niewłaściwy adres wpisany w pole tekstowe, wtedy nie ma zgłoszenia wyjątku, ponieważ protokół UDP nie odpowiada za dotarcie wysłanego pakietu do odbiorcy i czy nastąpi jego odczyt. Przykładowy wygląd interfejsu graficznego działającego programu zawarto w Rys. 4.5.

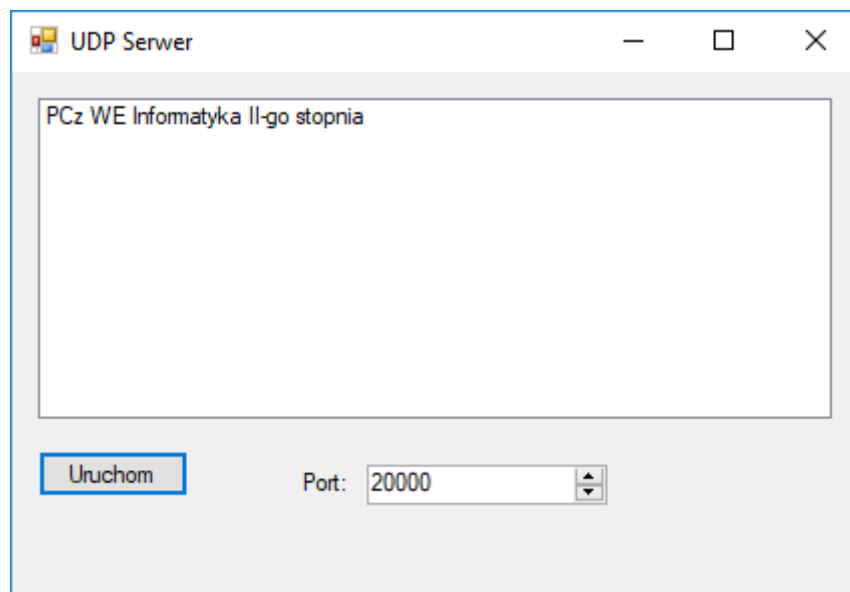


Rys. 4.5. Wygląd działającego projektu aplikacji UDP Klient [5]

```
private void button1_Click(object sender, EventArgs e)
{
    string host = textBox1.Text;
    int port = (int)numericUpDown1.Value;
    try
    {
        udpClient klient = new UdpClient(host, port);
        Byte[] dane = Encoding.ASCII.GetBytes(textBox2.Text);
        klient.Send(dane, dane.Length);
        listBox1.Items.Add("wysłanie wiadomości do hosta " + host + ":" + port);
        klient.Close();
    }
    catch(Exception ex)
    {
        listBox1.Items.Add("Błąd: Nie udało się wysłać wiadomości!");
        MessageBox.Show(ex.ToString(), "Błąd");
    }
}
```

Rys. 4.6. Kod przycisku Wytnij w programie UDP Klient [5]

Program, pełniący rolę serwera, ma pewne istotne różnice w stosunku do swojego odpowiednika TCP. Warto powtórnie zaznaczyć, że protokół bazowy nie ponosi żadnej odpowiedzialności za wszystko, co mogłoby dać minimalną gwarancję skutecznej wymiany danych. Plusem tezy jest z pewnością generowanie mniejszej liczby ruchu w sieci, co znacznie usprawnia operacje, a minusem przyczyna braku pewności na to, że nie utracono danych lub poprawny odczyt, czyli brak kontroli pakietów.



Rys. 4.7. Wygląd działającego projektu aplikacji UDP Server [5]

```
private void button1_Click(object sender, EventArgs e)
{
    int port = (int)numericUpDown1.Value;
    IPEndPoint zdalnyIP = new IPEndPoint(IPAddress.Any, 0);
    try
    {
        UdpClient serwer = new UdpClient(port);
        Byte[] odczyt = serwer.Receive(ref zdalnyIP);
        string dane = Encoding.ASCII.GetString(odczyt);
        listBox1.Items.Add(dane);
        serwer.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Błąd");
    }
}
```

Rys. 4.8. Kod przycisku Uruchom w programie UDP Server [5]

Interfejs graficzny (Rys. 4.7) nie posiada pola tekstowego, w którego można wpisać adres, co daje wniosek, iż dozwolone jest, by połączenia pochodziły z każdego



komputera, który jest ulokowany w sieci. Jeśli wiadomość zostanie odebrana przez serwer, zostaje on zamknięty. Kod źródłowy dla przycisku uruchom jest przedstawiony na Rys. 4.8.

Z powodu niedogodności zobrazowanych na podstawie powyższych przykładów tenże protokół zazwyczaj nie znajduje swojego użycia w strukturze klient-serwer, innymi słowy w relacji żądanie – odpowiedź.[5]

## 5. Algorytmy szyfrowania symetrycznego

Na początku warto wspomnieć, jaka dyscyplina nauki zajmuje się rozpatrywaniem jednego z fundamentalnych członów niniejszej pracy. Działem tym jest kryptografia, pracująca nad przeobrażeniem informacji, określanej mianem tekstu jawnego czy wiadomości w szyfrogram, tożsamym z terminem informacja zaszyfrowana. Jest w tym wypadku stosowany klucz, sformułowany przez użytkownika. Reorganizację danych należy zrobić tak, by ten, kto jest w posiadaniu szyfrogramu, nie był w stanie odczytać tekstu jawnego bez wiedzy na temat klucza w przeciągu stosunkowo niewielkiego odstępu czasu. Przerabianie wiadomości, określane mianem szyfru lub algorytmem kryptograficznym, jest wykonywane przez algorytmy. W przeważającej liczbie algorytmów kryptograficznych są realizowane algorytmy blokowe, cechujące się jednorazowym przerobieniem bloku, czyli fragmentu jawnej wiadomości. Z kolei algorytmy strumieniowe są przypuszczalnie określane mianem generatorów pseudolosowych z ziarnem, będącym kluczem szyfrującym. Liczby pseudolosowe kreowane przez generator podlegają działaniu XOR, czyli operacji sumy modulo 2 z wiadomością, czego wynikiem staje się szyfrogram. Cechą przewodnią algorytmów kryptograficznych jest zwykle znaczna złożoność obliczeniowa, powodując status realizacji sprzętowe jako w największym stopniu wydajne wytwory tychże algorytmów. Dla porównania algorytmy blokowe w zestawieniu ze strumieniowymi są w wyższym stopniu złożone niż te drugie. Dotyczy to zarówno aspektu realizacji, jak również idei funkcjonowania.[8]

Realizacja właściwych konfrontacji pomiędzy algorytmami symetrycznymi i niesymetrycznymi jest wyzwaniem ze względu na różnorodność charakterystyki i użycia w praktyce, czego dowodem jest kompatybilność standardów symetrycznych z szyfrowaniem danych, włączając w to sygnał mowy, w odróżnieniu do asymetrycznych czy tych z kluczem publicznym, cechujących się większą wydajnością czy lepszą odpornością na atak. Symetryczne szyfrowanie jest rodzajem zsumowanego sposobu ochrony danych samym w sobie. Choć nie jest w stanie dać gwarancji na nierozzerwalność wiadomości, to zapewnia pewny sposób na niebezpieczeństwa w kierunku poufności wiadomości i implikacji uwierzytelnienia do pewnego momentu.[10]

## **5.1. Pojęcie szyfrowania symetrycznego**

Termin szyfrowania symetrycznego odnosi się do tego, gdzie stacje nadające i te odbierające stosują jeden klucz tajny.[10] Inaczej szyfrowanie symetryczne można nazwać jako algorytm z kluczem prywatnym, czy z kluczem symetrycznym.[3] Obligatoryjne jest w tym przypadku to, by w nadajniku i odbiorniku znalazły się jednakowe algorytmy i identyczne strumienie kluczy. Fundamentem tegoż typu szyfrowania jest z pewnością zabezpieczona dostawa ogólnego klucza do obydwóch miejsc, innymi słowy winien dotrzeć do wszystkich kompetentnych użytkowników w sieci. Można przyjąć, iż w pełni nadzorowana komunikacja za pośrednictwem dystrybucji kluczy musi odbywać się na całym obszarze sieci z zastosowaniem zabezpieczonej trasy, co jest niezwykle trudne, a co za tym idzie, stanowi wyzwanie dla administratora, odpowiedzialnego za aspekt bezpieczeństwa. Największą bolączką jest z pewnością to, że w trakcie dystrybucji wszyscy z dostępem do kluczy posiadają wgląd do zawartości danych, mających podlegać tejże ochronie.[10] Szyfry z kluczem prywatnym mają istotny udział w ochronie danych na dyskach komputerowych, bądź szyfrowania wiadomości, które mają zostać przesłane pomiędzy parą sprzętu, a także szyfrowania sygnałów mowy i innych informacji przekazywanych w łańcuchach telekomunikacyjnych. Podział szyfrów symetrycznych opiera się na strumieniowych i blokowych.[3]

## **5.2. Szyfr strumieniowy**

Szyfr strumieniowy charakteryzuje się utajnianiem informacji pod formą sekwencji pojedynczych bitów a także znaków, nie zwiększa liczby błędów, konieczna jest synchronizacja instrukcji warunkowych dla początkowej fazy pracy i w momencie rozpoczęcia, jest konieczność ponownej synchronizacji w chwili zgubienia bądź dołożenia bitów czy znaków w sekwencji kryptogramów, są proste i wydajne w implementacji dla sprzętu, a skomplikowane i powolne w implementacji programowej, są podatne na ataki w formie powieleń, usunięć czy wtrąceń bitów lub znaków z kryptogramów, a także mają słabość na ataki aktywne o formie zmian w kryptogramach.[3] Tenże szyfr realizuje szyfrowanie, bazując na pojedynczych bitach, gdzie wszystkie bity z osobna ulega modyfikacji w zgodzie z konkretnym

schematem, dając kryptogram. Fundamenty tychże szyfrów tworzą generatory strumienia klucza. Wspomniane generatory kreują kolejno bity klucza stosowane w przyszłości do szyfrowania bitów tekstu jawnego. Zasadnicze zabezpieczenie określonej metody zależy od jakości generatora w myśl, że większe poziom wynika ze znacznie losowego charakteru wykreowanego strumienia bitów klucza. Można ująć, że należy posiadać odpowiedni generator liczb losowych, do nie jest proste do realizacji. Strumień klucza nie może być zwyczajny i periodyczny, czego uzasadnieniem staje się ułatwienie złamania szyfru, gdy znajduje się tam jakakolwiek prawidłowość. Kolejną wytyczną powinna być z pewnością kreowanie innego strumienia klucza po każdym uruchomieniu. Nie może mieć miejsca przypadek, gdy dwa teksty jawne podlegają zaszyfrowaniu jednakowym kluczem nawet wtedy, gdyby odznaczały się zgodnie z wymogami losowości. Ma możliwość przekształcenia do blokowej postaci, cechuje się też prostszą analizą matematyczną, co w konsekwencji ułatwia ich projektowanie, lecz są mniej skomplikowane w kryptoanalizie. Kwalifikuje się pozytywnie do implementacji sprzętowej, jest czasochłonny, gdyż realizowane są działania obliczeń na pojedynczych bitach lub bajtach.[6] Ten oto rodzaj szyfru symetrycznego mogą mieć zastosowanie w szczególności tam, gdzie są rygorystyczne wymagania na temat prędkości transmisji, są prostsze w realizacji bardziej pod kątem sprzętowym niż ma to miejsce programowo.[3]

#### **5.2.1. RC4**

Jest to algorytm, którego twórcą jest Ron Rivest, funkcjonującym w trybie OFB, działającym w oparciu o 256 S-bloków.[6] Stanowi człon grupy symetrycznych szyfrów strumieniowych, pospolity i znamienity dzięki trywialności w implementacji, a także tempo działania. Rozmiar klucza, na którym operuje ma długość do 2048 bitów. Ma swój udział w szeregu protokołów, czego doskonałymi przykładami są ochrona komunikacji w sieci rozległej Internet, gdzie występuje protokół Transport Layer Security lub zabezpieczenia używane w sieciach bezprzewodowych Wired Equivalent Privacy.[12] Innym zastosowaniem może być szyfrowanie dokumentów Excel i Word. Przebieg szyfrowania tejże metody opiera się na dwóch etapach, gdzie opisana wiadomość, która posiada określony ciąg bitów zostaje poddana niniejszej operacji, w wyniku czego tworzy się szyfrogram. Rozważając głębiej tą sentencję działanie rozpoczyna generowanie tablicy pomocniczej, mającej rozmiar o wartości

256 bajtów. Polega to na utworzeniu tablicy jednostkowej, następnie kreowane są permutacje. Dla przykładu, mając tablicę  $S = \{0,1,2,\dots,255\}$ , tworzy się permutacje według następującego schematu (Rys. 5.1).

```
j=0
od i=0 do i=255:
    j = (j + Si + klucz[i modulo długości klucza]) modulo 256
    zamień Si z Sj
```

Rys. 5.1. Tworzenie permutacji S [13]

W kolejnym etapie tworzony zostaje ciąg liczb pseudolosowych, w wyniku czego jest możliwość do tego, by dokonać szyfrowania tekstu jawnego. Pseudokod szyfrowania według niniejszej metody zawiera Rys. 5.2.[13]

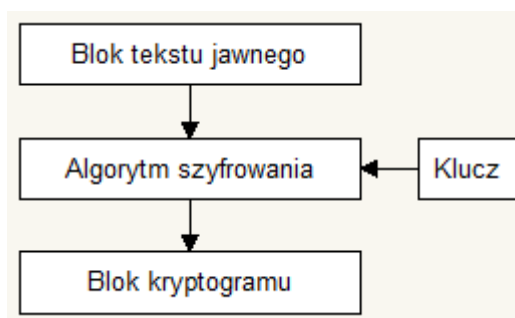
```
i = 0
j = 0
od k = 1 do długości wiadomości :
    i = (i + 1) modulo 256
    j = (j + Si) modulo 256
    zamień Si z Sj
    xk = S[(Si + Sj) modulo 256] XOR wk
```

Rys. 5.2. Szyfrowanie wiadomości w RC4 [13]

### 5.3. Szyfr blokowy

Algorytm blokowy jest rodzajem szyfru, funkcjonującego na blokach danych, posiadających różną długość, mogą zostać sprowadzone do postaci strumieniowej. Im starszy algorytm, tym pracują na mniejszej liczbie bitów.[6] Jest to metoda jedna z fundamentalnych składowych, używanych w strukturach kryptograficznych.[4] Argumentem na poparcie przytoczonego stwierdzenia jest fakt, iż starsze schematy działają na 64 bitach, a dla porównania relatywnie nowy AES posługuje się blokami maksymalnie o rozmiarze 256 bitów. Procedura szyfrowania opiera się na tym, że odrębne bloki danych przebudowywane są w procesie działania

schematu na postać bloku szyfrogramu.[6] Jednakże te oto bloki mają stały rozmiar.[4] Zasadniczą postać szyfru blokowego, gdzie realizacja algorytmu skutkuje zaszyfrowaniem fragmentu wiadomości, mającej określoną długość prezentuje Rys. 5.3. W obszarze stosowanych w sferze praktycznej zasadniczych cech dla blokowych szyfrów znajdują się: długość bloku tekstu jawnego o wartości 64 lub 128 bitów, a także długość przeznaczona dla klucza początkowego w zakresie od 40 do 256 bitów.[3]



Rys. 5.3. Ogólna struktura szyfratora blokowego [3]

Porównywalnie przebiega tok postępowania deszyfrowania, wyróżniający się od poprzednika tym, że rolę danych wejściowych realizują bloki szyfrogramu.[6] Klucz powinien być tajny, tworzy go zazwyczaj ciąg 128 i 256 bitów.[4] Niewątpliwa jest kwestia niezmiennego klucza, co jest charakterystyczne dla wzorców symetrycznych. Dobrą stroną algorytmów tego typu jest ich kwalifikacja do implementacji programowej. Proces szyfrowania, przebiegając na blokach danych, zostaje znacznie skrócony.[6] Szyfr blokowy jest w pełni odwracalny, co odzwierciedla istnienie funkcji, pobierające pożądaną liczbę bitów szyfrogramu i zwraca tą samą liczbę bitów jawnej wiadomości. Ujmując to stwierdzenie inaczej, oznacza to, że hasła umieszczone w bitach są parami różne, w innym razie nie byłaby opcja odszyfrowania wybranych z tegoż zbioru.

Rozpatrując aspekt bezpieczeństwa tegoż szyfru, warto zaznaczyć pewne reguły, które dają temu gwarancję. Algorytm staje się bezpieczny, gdy nie ma w stosunku do niego żadnych ataków, zarazem określony może zostać taki schemat jako idealny, co z pewnością staje się trudnym w realizacji wyzwaniem.[4]

### 5.3.1. Tryby szyfrowania blokowego

Szyfru blokowe mają swoje przeznaczenie głównie w szyfrowaniu danych, co w przypadku krótkich tekstów jawnych mogą używane być w sposób bezpośredni, tak przy rozległych wiadomości w porównaniu z długością bloku, daje podstawę do tego, by wykorzystać jeden z trybów. Szyfr blokowy umożliwia szyfrowanie co najwyżej o opisanym rozmiarze bloki. Pragnąc zaszyfrować dane z odmienną długością od rozmiaru jednego bloku, nieuniknione staje się skorzystanie z trybu.[4]

Chcąc dokonać selekcji krypto systemu z przeznaczeniem ochrony danych może stać się nieuchronne spotkanie z wielorakimi trybami schematów blokowych. Choć warto kierować się tym, co jest pewne i pospolite, to tryb nieodłączny z algorytmem jest niezwykle wart uwagi jako parametr, ponieważ twórca wdrożył sposób ze słabościami zasadniczo oddziałującymi niekorzystnie na stopień bezpieczeństwa.[6] Tryb przeszkadza zupełnie w podsłuchiowaniu, jednak nie uwierzytelnia, co skutkować może dowolną edycją wysyłanych danych, wykonywaną przez niepożądaną osobę.

W szyfrze blokowym dokonuje się transformacja tekstu otwartego na zaszyfrowany, a oboje mają narzuconą długość, a większość trybów narzuca rozmiar jawnego tekstu jako wielokrotność rozmiaru bloku, co zobowiązuje do skorzystania z dopełnienia, gdzie chodzi o poręczenie odwracalności tegoż działania, innymi słowy sposób na nieomylny podział oryginalnego tekstu z jego dopełnieniem jest nieuchronne ze względu na to, iż musi onno zostać wyrzucone od jawnego tekstu po odszyfrowaniu. Najlepiej jest, by dopełnienie nie wydłużało jawnej wiadomości, co nie zawsze da się uczynić.[4]

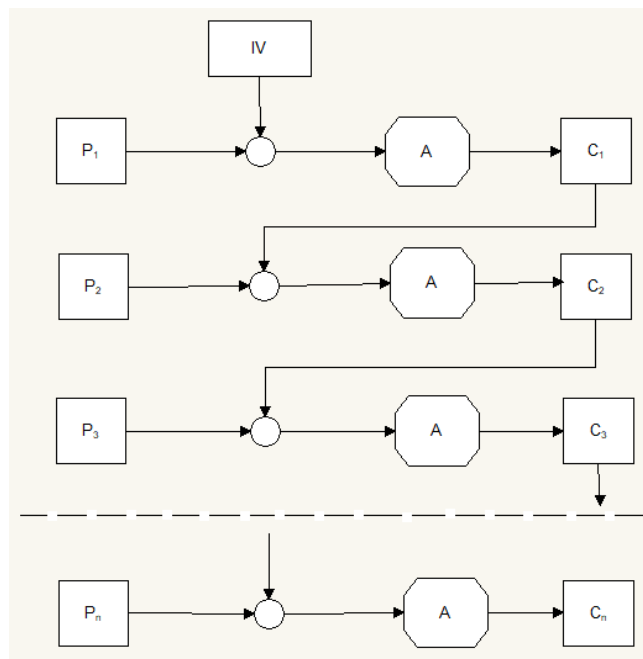
Pierwszym trybem, który podlegnie rozważaniu, jest CBC. Rozwinięcie skrótu pochodzi z języka angielskiego Cipher Block Chaining, co oznacza w tłumaczeniu na język polski wiązanie bloków zaszyfrowanych. Charakteryzuje się konwencjonalnością i niesamowitą skutecznością. W nim pojedynczy blok wiadomości jawnej podlega operacji XOR z wcześniej wykreowanym kryptogramem według wzoru na Rys. 5.4.

$$C = A(K, P_k \oplus C_{k-1}) \text{ dla } k = 1, 2, \dots, n$$

Rys. 5.4. Wzór operacji XOR [6]

Może pojawić się zastanowienie nad pochodzeniem  $C_0$  w trakcie pierwszej rundy procesu, jednak za to odpowiada wektor inicjujący (IV).[6] Wektor inicjujący lepiej, by nie był wartością niezmienną, bo może w konsekwencji prowadzić do trudności, związanych z pierwszym blokiem we wszystkich wiadomościach zbliżonych do używanych w trybie ECB. Rozpatrując dwa odmienne tekstu jawne z tym samym tekstem otwartym mogłyby mieć te same bloki na początku po procesie szyfrowania. Lepszym rozwiązaniem jest skorzystanie z innej opcji, a mianowicie użyciu licznika do IV. Choć nie jest najlepszy, to IV jako pospolity licznik ma możliwość po skorzystaniu z XOR zamaskować niezgodności i poskutkować tymi samymi blokami po zaszyfrowaniu. Pochodzenie utrapienia ze skorzystaniem z trybu ECB i CBC, ze stałym IV lub w formie licznika wynika z braku podobieństwa zwyczajnych wiadomości z tworzonymi losowo ciągami bajtów. Są dwie słabości pracy z losowym IV, czyli algorytm szyfrujący koniecznie ma wgląd do któregoś źródła losowości, a budowa skutecznego generatora liczb losowych jest dużym wyzwaniem, co argumentuje to obligatoryjność skorzystania z tegoż rozwiązania. Następnym minusem jest to, iż szyfrogram ma jeden blok dłuższy od tekstu otwartego, co nie jest skuteczne w krótkich wiadomościach, gdzie jeden blok stanowi potencjalnie znaczne rozszerzenie, a nie jest to właściwe. Najlepiej jest postąpić, mianując IV jako jednorazową, odpowiednio zrobioną wartość, co można osiągnąć poprzez to, by każda wiadomość podlegająca szyfrowaniu uzyskuje niepowtarzalny numer.[4] Zupełny przebieg algorytmu A w obecnie analizowanym trybie obrazuje poniższy Rys. 5.5. Wszystkie wytworzone bloki kryptogramu jest podległy od wszelkich poprzedników, w konsekwencji pomimo przerabiania tego samego bloku wiadomości wynikowy blok kryptogramu zostanie innym od tego, który ukazał się pierwszy raz. Nieprawdopodobny będzie też przypadek wsadzenia wyselekcjonowanego bloku do kryptogramu, co przyczyniłoby się na kształt następnych bloków w jawnym tekście. Przypuszczalny test przekształcenia kryptogramów istnieje możliwość wykluczyć za pomocą podpięcia do wszystkich kodu MAC, czyli kodu uwierzytelnienia wiadomości. W tym wypadku zarówno pierwszy blok szyfrowanych danych można powiedzieć, iż po zaszyfrowaniu stanie w innym profilu, co jest podporządkowane realizowanemu wektorowi inicjującemu. Mają miejsce nieliczne złamania algorytmów dzięki temuż schematowi, dzięki tekstowi jawnemu, jeśli użytkowany jest ten oto tryb w sekwencji szyfrowania.





Rys. 5.5. Szyfrowanie w trybie CBC [6]

Plusem tejże metody jest z pewnością to, że przyćmienie jednego bloku kryptogramu niesie ze sobą nieczytelność co najwyżej dwóch bloków w tekście jawnym mimo relacji pomiędzy blokiem tekstu tajnego a zupełną wcześniejszą częścią kryptogramu, a daje to sposobność odczytu danych w sytuacji pojawienia się błędów w momencie przesłania, bądź zapisu. Ten aspekt ma swoją wartość, choć nie jest taka ważna, ponieważ poprawność danych weryfikują sumy kontrolne, a protokoły połączeń internetowych zawiadamiają o nieobecności obligatoryjnych pakietów. Hipotetyczna luka w bezpieczeństwie ma prawo pojawić się w trybie omawianym z zastosowaniem ataku urodzinowego jedynie wtedy, gdy pokładają się dwa bloki kryptogramów. Jeśli atakujący wie, jakie wartości są w odrębnych blokach kryptogramu, ma możliwość obliczeń wartości dla ustalonej sumy, nie zapewnia to pewnego złamania szyfru, lecz nadaje statut pierwszego etapu. Przyczynowość nie występuje po stronie samego CBC, ale długość użytego w tym oto szyfrze bloku. Jeśli krótszy pojawia się ten blok, daje to sposobność do ukazania się jednorakich kryptogramów. Szyfry od 64-bitowej długości bloku mają nadzieję zostać złamane wtedy, gdy zarządza się ponad 30 gigabajtami kryptogramów, co daje obecnie konieczną dla bezpieczeństwa przed owym atakiem rozmiar bloku na 128 bitów. W kryterium bezpieczeństwa lokuje się wyżej niż ECB z prawdopodobieństwem jednolitych bloków kryptogramu w przypadku, gdy do dwóch tożsamyh wiadomości użyty zostanie bliźniaczy wektor inicjujący, a także klucz. Jednak w przypadku

wytworzenia wektora IV będzie zastosowany generator liczb losowych, nie ma prawa mieć miejsca taka sytuacja. Ten oto tryb, mając zależność każdego bloku kryptogramu od poprzedników skuteczniej maskuje właściwości wiadomości.

Drugim trybem ulegającym analizie jest ECB, którego nazwa pochodzi z języka angielskiego Electronic Code Book, co w tłumaczeniu oznacza elektroniczną książkę kodową.[4] nazewnictwo elektronicznej książki kodów ma swoje korzenie w tym, że mając identyczny tekst jawny dokonuje się szyfrowanie w taki sposób, że powstaje tożsamy szyfrogram, powodując możliwość wykonania książki kodów wszystkich możliwych wiadomości i przypisywanych do nich szyfrogramów dla pojedynczego klucza. Szyfrowany zostaje każdy z bloków wiadomości metodą niezależną, dając sposobność do zastosowania w szyfrowaniu plików, gdzie egzekwuje się nieograniczonego dostępu z opcją edycji, czego doskonałym przykładem jest baza danych.[10] Tenże tryb tworzy najprostszą postać szyfru blokowego, którego istota obejmuje szyfrowanie analogiczne każdego z bloków danych, co można zapisać następująco w sposób matematyczny (Rys. 5.6).

$$C_k = A(K, P_k) \text{ dla } k = 1, 2, \dots, n$$

Rys. 5.6. Sposób matematyczny obrazu szyfrowania pojedynczego bloku danych [6]

P oznacza tekst jawny, tekst tajny stanowi C, K jest kluczem szyfrowania, A będzie dowolnym algorytmem blokowym, natomiast n cechuje liczbę bloków wiadomości. Omawiany tryb gwarantuje najmniejszy poziom bezpieczeństwa z grona wszystkich trybów szyfrowania, ponieważ są tożsame bloki tekstu jawnego w tekście tajnym, co obniża poziom trudności krypto analizy kryptogramów, a poza tym daje możliwość niedostrzegalnego zastępowania bloków danych. Powyższe argumenty przemawiają za tym, iż ten tryb pełni drugorzędną rolę i nie zalecany jest wtedy, gdy w kryterium zawarte jest zapewnienie profesjonalnej ochrony.[6] Proces szyfrowania obejmuje osobne działanie na każdy blok. Tryb ECB jest niepożądany ze względu na dwa te same bloki tekstu otwartego i analogiczne dla nich bloki w szyfrogramie, z czego atakujący może sporo uzyskać danych. Repetycja bloków długich otwartego tekstu pojawia się często, mogą to być powtórki fraz, wiele łańcuchów znaków w Unicode ma w co drugim bajcie zero, dużo formatów plików ma znaczne fragmenty pod względem długości, gdzie są w ciągach prawie tylko zera.[4] Słabościami są skłonność na atak statystyczny w stosunku do szyfrogramu, zagrożenie uszkodzenia

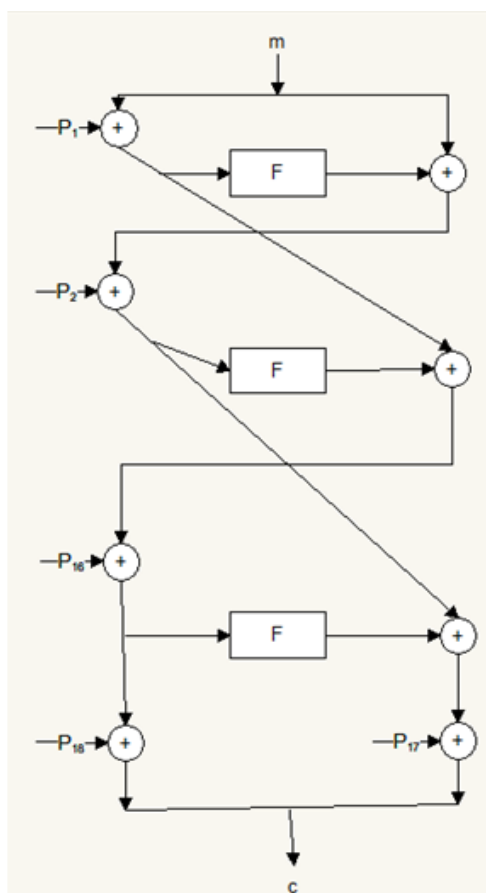
całego bloku odszyfrowanej wiadomości w przypadku odszyfrowania błędu bitu w szyfrogramie i bez możliwości wnikania w klucz atakujący ma możliwość edycji bloku danych. Plusami są z pewnością szyfrowanie więcej niż pojedynczej wiadomości z korzystaniem z jednego klucza, a także duży poziom szybkości pracy.[10]

Tryb CTR jest ostatnim z wybranych do omówienia, jest ich więcej. Ma zastosowanie w generowaniu strumienia klucza szyfrującego, co prowadzone zostaje dzięki numerowi jednorazowemu wiadomości, a także licznika szyfrowanych bloków. Numer jednorazowy zazwyczaj wywodzi się w oparciu o numer wiadomości i danych gwarantujących unikalność. Zważywszy na odpowiednie zabezpieczenie ważne staje się, by relacja klucza z numerem jednorazowym było niepowtarzalne dla określonej wiadomości, w przeciwnym przypadku skutkuje to zaszyfrowaniem więcej niż jednej wiadomości jednolitym strumieniem klucza. Tryb jest trywialny do implementacji, co można argumentować równoznacznym przebiegiem procedury szyfrowania i deszyfrowania, a ponadto zezwala na jednoczesne przetwarzanie strumienia klucza, dając tym samym wyższą szybkość w dostępie do strzeżonych danych.[6] Ten oto tryb ma inne nazewnictwo jako tryb licznika.[4]

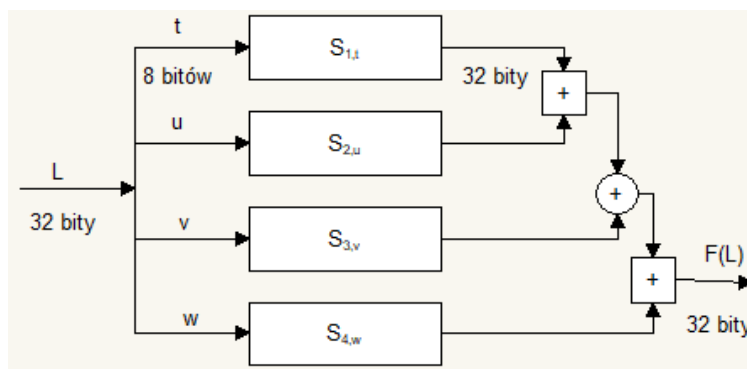
### 5.3.2. Blowfish

Algorytm blokowy o nazwie Blowfish przekształca bloki długości 64 bitów wraz z kluczem nie dłuższym niż 448 bitów, przemiany są realizowane w szesnastu rundach. Dla pojedynczej rundy używana jest permutacja selekcyonowana dzięki kluczowi  $k$ , a także podstawień, uzależnionych od podlegających danych, jak również klucza. W dodatku pracą, wykonywaną we wszystkich rundach stanowi przeglądanie cztery razy tablicy indeksowanej. Klucze pomocnicze  $P_1, P_2, \dots, P_{18}$ , a także bloki podstawień  $S_{1,t}, S_{2,u}, S_{3,v}, S_4$  długości 32 bitów o liczące 256 wejść są definiowane przed wszystkimi działaniami szyfrowania i deszyfrowania. Pracę szyfrowania rozpoczyna dzielenie bloku wejściowego 64-bitowego na dwa bloki jednakowego rozmiaru. Następnie realizowane jest szesnaście razy przerobienie za pomocą funkcji  $F$  oraz kluczami pomocniczymi  $P_1, P_2, \dots, P_{18}$ , a ostatnia runda nie zamienia bloku lewego z prawym, dodając w zamian modulo 2 do bloku lewego klucz  $P_{18}$ , a do prawego  $P_{17}$ , a także połączenie dwóch wspomnianych bloków w blok wyjściowy  $c$  o długości 64 bity (Rys. 5.7). Działanie deszyfrowania pracuje na tej samej zasadzie jak szyfrowanie, różniąc się kwestią kolejności kluczy  $P_1, P_2, \dots, P_{18}$ , które

służą w przeciwnym szyku. Zobrazowanie pracy funkcji  $F$  przedstawia rysunek 5.8. W celu definicji wartości używa się bloków podstawień  $S_{1,t}$ ,  $S_{2,u}$ ,  $S_{3,v}$ ,  $S_{4,w}$ , działanie sumowania modulo 2 dla stosownych bitów, a także dodawanie modulo  $2^{32}$ . Lewy blok  $L$  o rozmiarze 32 bitów zostaje podzielony na 4 bajty  $t$ ,  $u$ ,  $v$ ,  $w$ . Tak klucze pomocnicze, jak i bloki podstawień są określane dzięki algorytmowi Blowfish, klucza użytkownika  $k$  o rozmiarze 448 bitów, a także cyfr po przecinku dla liczby  $\pi$ , ukazanej w systemie szesnastkowym.[1]



Rys. 5.7. Schemat blokowy szyfru Blowfish [1]



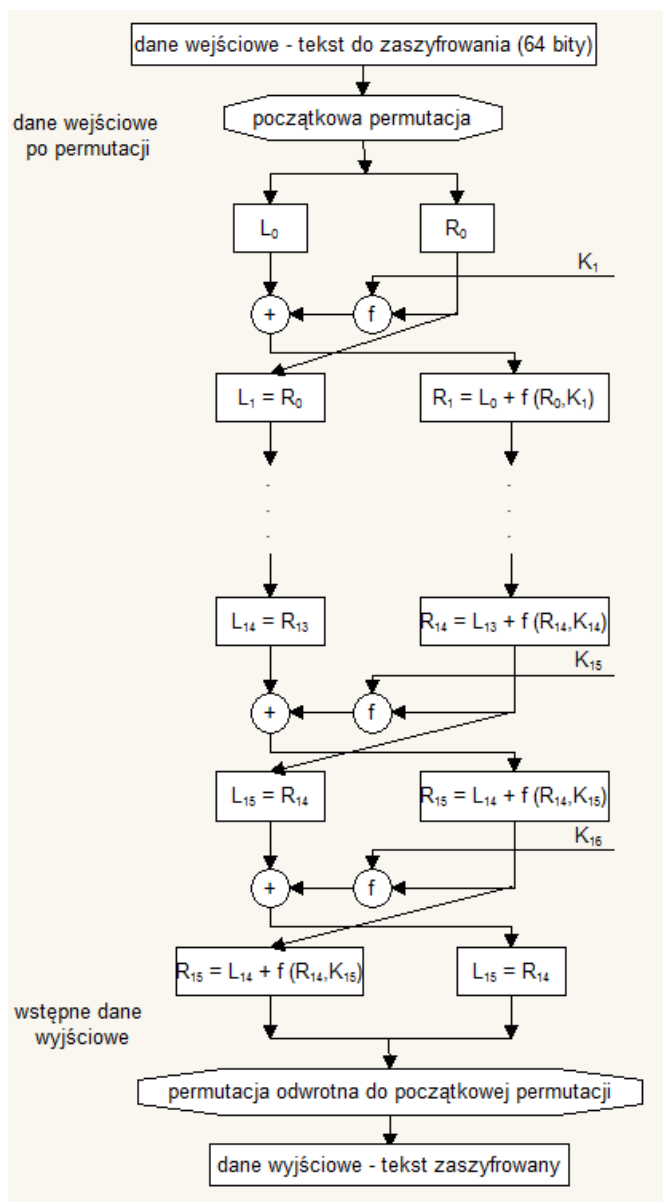
Rys. 5.8. Funkcja  $F$  [1]

### 5.3.3. DES

Skrót DES pochodzi z języka angielskiego Data Encryption Standard, stanowi nazewnictwo jednego z zasadniczych szyfrów blokowych. [6] Ma status najpowszechniejszego systemu z kryptografii.[9] Długość 64 bity charakteryzuje pojedynczy szyfrowany blok wiadomości, z kolei 56 bitów określana jest niniejsza wartość dla klucza szyfrowania, lecz jego zapis dokonuje się w formie liczby 64-bitowej.[6] Staje się to powodem do tego, by określić, iż niniejsze długości dają mu status nieprzydatności w czasach obecnych, gdzie swoje miejsce odnalazły sprawne komputery, a także przetwarzanie masowe.[4] Pod względem szyfrowania i deszyfrowania należy podkreślić, iż jest używana identyczna metoda, wsparta na kompozycji kryptograficznych sposobów rozpraszania i mieszania. Algorytm liczy szesnaście cykli, a także dwie permutacje, to znaczy początkową i końcową.[7] Tury te są indeksowane od 1 do 16, gdzie każda przemienia lewą i prawą połówkę na nową w kompatybilności z kluczem w turze. Funkcja, która rozszerza transformuje prawą stronę, ulega zmianie liczba bitów z 32 na wartość 48, zaś on zostaje przemianowany za pośrednictwem XOR z kluczem tury o długości 48 bitów. Efekt działania tejże operacji stanowi wejście do S-boksu. Przekształcenia ogółem w pojedynczym procesie szyfrowania DES ulega szesnaście razy. DES potrzebuje szesnaście kluczy tur, posiadających długość po 48 bitów, mających korzenie w selekcji 48 bitów z klucza o długości 56 bitów. Warto zaznaczyć, że dobór jest odmienny dla wszystkich kluczy.[4] Na pojedynczy cykl realizowane są nieodmienne obliczenia, bazujące na rozwiązaniach z poprzednich cykli. Tekst na wejściu zostaje podzielony na 64-bitowe bloki, potem pojedynczy blok podlega operacji szyfrowania według kolejno następujących po sobie kroków:

- 1) Permutacja początkowa
- 2) Podział tekstu na lewy i prawy blok
- 3) Permutacja rozszerzona
- 4) Szyfrowanie dzięki S-blokom
- 5) Szyfrowanie za pomocą P-bloków
- 6) Permutacja końcowa

W postaci schematycznej został zobrazowany algorytm na Rys. 5.9.



Rys. 5.9. Schemat standardu (algorytmu) szyfrowania danych DES [11]

Znak + w kole oznacza operator logiczny XOR, innymi słowy alternatywę wyłączającą. Permutacja przekształcająca dla szyfrowania nie jest znacząco istotna.[11]

Permutacja początkowa polega na tym, że bity podlegającym szyfrowaniu blokowi zostają permutowane, kierując się macierzą przestawień, gdzie znajdują się numery bitów, mające ukazać się na określonych pozycjach już po doprowadzeniu do permutacji. Potem odczyt wykonuje się według zasad od strony lewej do prawej i z góry na dół, a bit o numerze 58 zostaje przesunięty w konsekwencji na początkową pozycję, bit o numerze 50 przechodzi na drugą pozycję, a 42 na miejsce trzecie, jak przedstawia poniższa Tab. 5.1, inne bity zachowują się podobnie według tabelki.

Tab. 5.1. Macierz permutacji początkowej IP [6]

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Podział tekstu na bloki rozpoczyna podział na dwie części bloku po 32 bity, czyli na lewą i prawą. Dalej szesnaście razy powstałe podbloki ulegają działaniu funkcji  $f$ , która scala z kluczem szyfrowania. Ilość przesunięć na każdy pojedynczy cykl przedstawia Tab. 5.3. Przy wszystkich cyklach transformowany zostaje klucz dzięki operacjom redukcji, podziału, przesunięcia, a także permutacji z działaniem kompresji.[6] Zbliżona modyfikacja w postaci zamiany bitów dzieje się w końcowej fazie szyfrowania, a dzieje się to, gdy kreowane są z połówki lewej i prawej szyfrogram o długości 64 bitów.[4] Najpierw klucz o długości 64 bitów zostaje zredukowany do wartości 56 bitów za pomocą usunięcia bitów parzystości, dalej zamienione miejscami są pozostałe według macierzy permutacji klucza (Tab. 5.2).

Tab. 5.2. Macierz permutacji klucza [6]

57	49	41	33	25	17	9
1	58	50	42	43	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Później klucz zostaje podzielony na dwa bloki o długości 28 bitów, przestawianych w lewo o skok jeden lub dwa bity.

Tab. 5.3. Liczba przesunięć przypadająca na każdy z cykli [6]

Cykl	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Liczba przesunięć	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

W następnej kolejności dokonana zostaje selekcja 48 bitów klucza, mających zastosowanie w zaszyfrowaniu tekstu, a typowane są według macierzy kompresji (Rys.5.4).

Tab. 5.4. Macierz kompresji [6]

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Mechanizm jest określany mianem permutacji z kompresją ze względu na liczbę bitów klucza, która ulega zmniejszeniu podczas jego trwania. Wspomniana macierz ma nazewnictwo też jako permutowany wybór. Edytuje następny raz szereg bitów, ignorując wybrane ze zbioru, co daje skutek w postaci efektu kompresji, z kolei poprzez permutację pojedynczy cykl metody ma inną wykorzystywaną kombinację bitów klucza zasadniczego, oferując tym samym lepsze zabezpieczenie systemu. Wszystkie bity są z osobna wykorzystywane w liczbie od 14 do 16 podkluczy.

Etap permutacji rozszerzonej realizowany jest na prawym bloku danych. Skutkiem tego postępowania ulega zmianie rozmiar z 32 do 48 bitów. Należy rozumieć to, jako pokrycie ilości bitów bloku do klucza, posiadających przeznaczenie w szyfrowaniu danych. W następnej kolejności blok zostaje przesunięty, a także zwiększony do tego stopnia, by mieć możliwość ulegnięcia kompresji w momencie dokonywania w przyszłości podstawiania. W pierwotnej postaci dane są

Tab. 5.5. Macierz alokacji permutacji z rozszerzeniem [9]

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1



Tab. 5.6. Struktura S-bloków [6]

Wiersz	Kolumna															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	2	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

członkowane na bloki, mające długość 4 bitów każdy. Na ich fundamencie dokonywane są obliczenia danych wyjściowych. Generowane są po dwa bity wyjściowe przez pierwszy i końcowy bit z każdej grupy, z kolei drugi i trzeci zobowiązane do za jeden bit na wyjściu. W kolejnym kroku blok wynikowy podlega permutacji, biorąc pod uwagę macierz alokacji (Tab. 5.5). Ciąg bitów wykreowany w ten oto schemat zostaje zsumowany modulo 2 z bitami, należącymi do klucza, potem zaczyna się proces podstawiania, która zostaje przeprowadzona dzięki S-blokom, których struktura jest odzwierciedlona w Tab. 5.6.

Zanim przeobrażony przez S-bloki podlegający szyfrowaniu ciąg danych zostaje grupowany na osiem bloków o długości 6 bitów, a pojedynczy z nich zostaje przerobiony za pośrednictwem jednego z ośmiu możliwych S-bloków, te S-bloki stanowią macierze alokacji, podsuwające ciąg 4-bitowy na wyjściu. Ciąg bitów na wejściu zostaje przerobiony tak, że startowy i końcowy bit stanowią liczbę binarną o znaczeniu numeru wiersza danego S-bloku, z kolei centralne 4 bity są liczbą binarną wskazującą numer kolumny, a w miejscu rozcięcia opisanej kolumny i wiersza powstaje wartość wyjściowa. Liczby, zawierające się w S-blokach, są w przedziale  $<0, 15>$ , co daje wynik w postaci liczby o długości 4-bitowej. Fundament bezpieczeństwa niniejszego algorytmu stanowią S-bloki, a reszta działań dokonują przekształcenia liniowe, wszakże w S-blokach mają miejsce transformacje nieliniowe, co stanowi argumentację w stwierdzeniu, iż jest to stopień trudności dla kryptoanalizy. Ważne jest też to, że edycja jednego z bitów w ciągu wejściowym, oddziałuje na modyfikację wszystkich bitów na wejściu, a także dwie odmienne wartości na wejściu potencjalnie mogą przynieść identyczną wartość na pozycji wyjściowej.

P-bloki (Tab. 5.7) stanowią macierz alokacji, w której 4-bitowe ciągi na wejściu S-bloków scalane są w 32-bitowy blok danych i permutowane. Proces ten ma

Tab. 5.7. P-blok [9]

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

nazewnictwo permutacji zwykłej, gdyż obrazuje wszystkie pojedyncze bity na wejściu na jeden wyjściowy bit i nie bagatelizuje żadnego bitu.

Na koniec ma miejsce sumowanie modulo, utworzonego po zdefiniowanych przeobrażeniach prawego bloku danych z częścią lewą początkowego o długości 64 bitów bloku wiadomości, a potem obydwie połowy zostają zastąpione wzajemnie i daje to początek następnego cyklu.

Jeśli ostatni cykl będzie zrealizowany w algorytmie DES, ma miejsce łączenie bitów z lewej i prawej połówki zmienionego bloku danych, a blok wynikowy podlega permutacji uformowanej macierzą permutacji końcowej (Tab. 5.8).

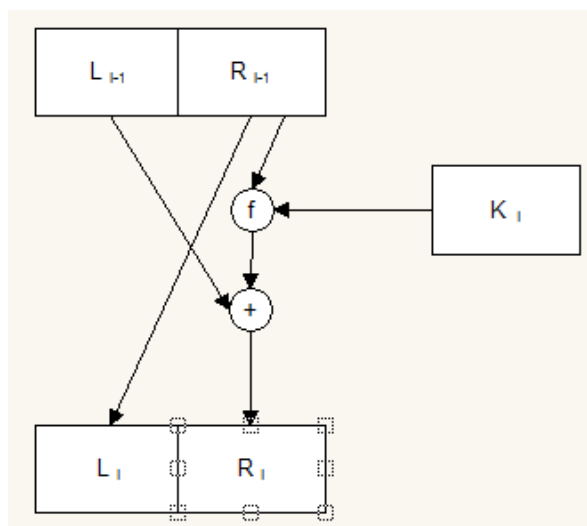
Tab. 5.8. Macierz permutacji końcowej  $IP^{-1}$  [9]

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

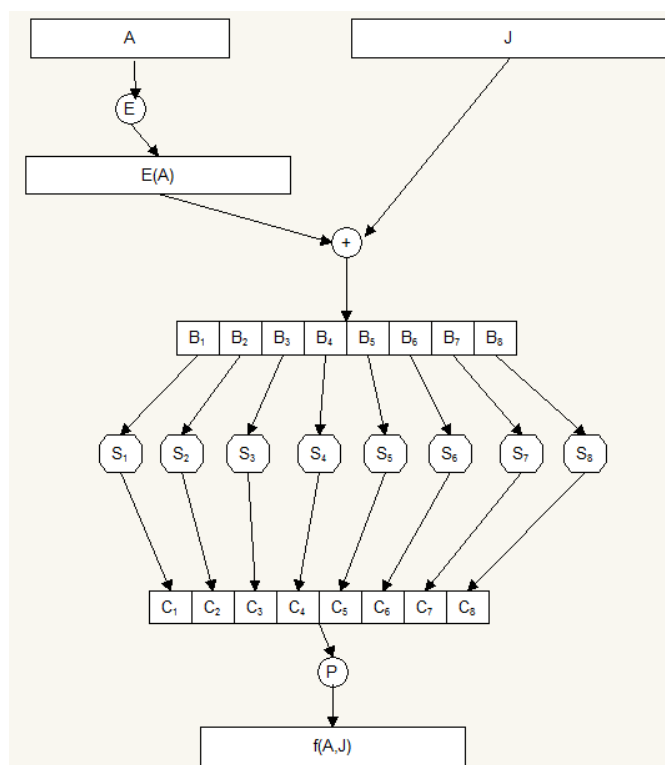
Do wykonania deszyfracji szyfrogramu za pomocą DES używana zostaje identyczna metoda, kolidująca jedynie w kluczach, realizowanych w każdym etapie, użycanych w przeciwstawnej kolejce i przekładany w prawą stronę, co w przypadku szyfrowania przebiega w lewą stronę.[6]

DES pomimo faktu, iż jest wolnym w tempie, stanowi składową szeregów struktur, a wartość klucza ma zbyt mało bitów.[4] Reasumując algorytm DES dzieli się na trzy etapy, rozpoczynając od tworzenia ciągu bitów dla ustalonego tekstu jawnego za pomocą permutacji bitów ciągu w zgodzie z permutacją początkową  $IP$ , przez obliczanie 16 iteracji pewnej funkcji, kończąc na realizacji permutacji  $IP^{-1}$ , która jest przeciwstawna do tej początkowej.

Wielkość opisująca funkcję  $f$  dla ciągu o długości 32 bity jest podległa do dwóch motywów, a mianowicie ciągu  $A$  32-bitowego, a także ciągu  $J$ , mającego 48 bitów. Zobrazowanie wspomnianej funkcji przedstawia Rys. 5.10. W podstawowym ujęciu funkcja  $f$  jest zbudowana z podstawienia z zastosowaniem bloku podstawienia, a po nim ma miejsce określona permutacja  $P$  (Rys. 5.11). Charakterystyka algorytmu



Rys. 5.10. Pojedyncza runda szyfrowania w systemie DES [9]



Rys. 5.11. Funkcja f systemu DES [9]

DES jest bardzo długa, jednak nie przeszkadza to, by zachował łatwy poziom trudności w implementacji na poziomie programistycznym i sprzętowym. Różnica symetryczna, znana też jako alternatywa wykluczająca, ciągu bitów stanowi jedyną operację arytmetyczną. Algorytm DES ma zastosowanie w systemach bankowych, szyfrowaniu osobistych numerów identyfikacyjnych, jak również na działaniach

wykonywanych za pomocą bankomatów, układach rozliczeń międzybankowych w celu zatwierdzenia transakcji i ma swój pospolity udział w instytucjach rządowych.[9]

#### 5.3.4. 3DES

Potrójny DES (Triple DES) lub 3DES jest kolejną po 2DES edycją metody DES.[6] Ta oto modyfikacja w znacznym stopniu wzmacnia algorytm DES.[3] Jego działanie polega na czymś innym niż zaszyfrowanie identycznego tekstu za pomocą trzech różnych kluczy.[6] Dało to poprawę w długości klucza, która była za mała, ale nie rozwiązało to kwestii długości bloku. Składa się z analogicznych, po sobie przebiegających metod DES.[4] Pracuje według wzoru z Rys. 5.12.

$$C = \text{DES}_K(\text{DES}_{K^{-1}}(\text{DES}_K(P)))$$

Rys. 5.12. Szyfrowanie metodą 3DES [6]

Początkowo ma miejsce zaszyfrowanie danych kluczem K, potem deszyfrowanie kluczem K', a koniec tworzy szyfrowanie kluczem K. Warto podkreślić, iż dla  $K=K'$ , powstaje klasyczny DES, co jest następstwem woli zaręczenia kompatybilności z urządzeniami szyfrującymi, korzystając ze starej wersji metody. Triple DES nie jest uchroniony od ataku typu Meet in the Middle, daje to wskazania, sformułowane tak, by w czasie pojedynczego kroku pracy został wykorzystany odmienny klucz, jednak są to jedynie wywody, algorytm spełnia swoje przeznaczenie. Minusem metody pozostaje flegmatyczność, co można porównać z tempem nowatorskich algorytmów, których doskonałym przykładem jest AES.[6] Kwestia tempa pracy jest spuścizną z metody DES, od której jest trzykrotnie wolniejszy. Są nowe projekty, gdzie nie należy go stosować, jak również DES. 3DES posiada klucz dłuższy od przodka, pomimo to dziedziczy niewystarczające klucze, jak również właściwość dopełnienia. Te oto uzasadnienia dyskwalifikują go z użycia w praktyce, ogranicza go rozmiar bloku o wartości 64 bity, co wstrzymuje znacząco liczbę danych szyfrowanych poprzez jeden klucz. Obligatoryjne jest korzystanie z Triple DES w pewnych wypadkach, gdyż nakłada go aktualna struktura systemu. Powinno się zachować pewny dystans w stosunku do 3DES, ponieważ nie jest on idealnym szyfrem, posiada możliwości ataku na niego.[4] Przyjmując za  $E_K(M)$  szyfrowanie DES oraz  $D_K(C)$  deszyfrowanie DES, natomiast M obierając jako wiadomość, z kolei C szyfrogram z użyciem klucza.

Należy podkreślić, iż obie operacje typu TDES składają się z działań analogicznych DES. Szyfrowanie przebiega, jako przekształcenie 64 bitów bloku wiadomości w szyfrogram o tym samym rozmiarze (Rys. 5.13).

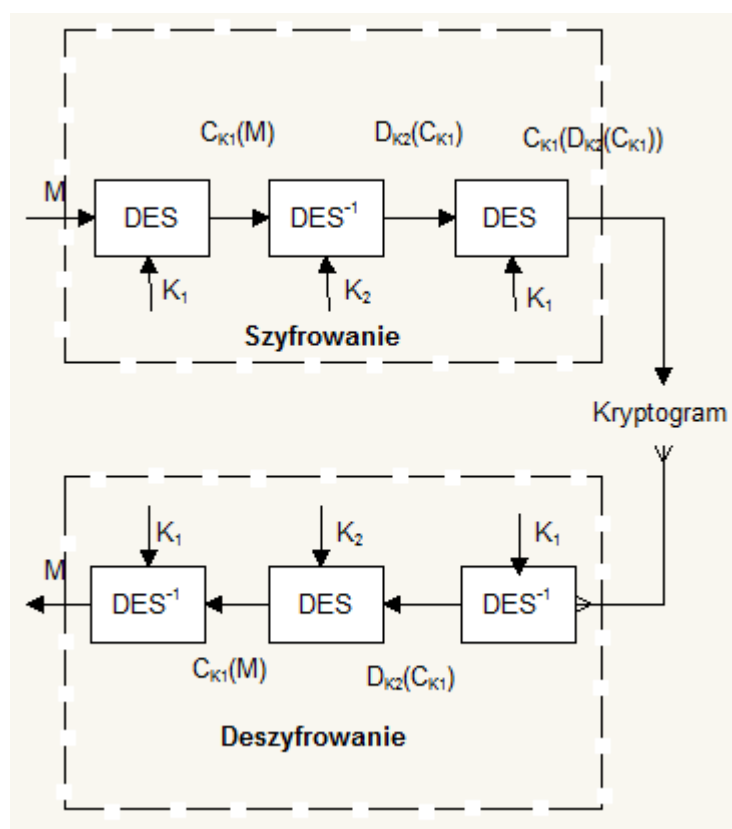
$$C = E_{K3}(D_{K2}(E_{K1}(M)))$$

Rys. 5.13: Przebieg szyfrowania TDES [11]

Natomiast deszyfrowanie przekształca 64 bity blok tekstu zaszyfrowanego w ten jawny (Rys. 5.14).

$$M = D_{K1}(E_{K2}(D_{K3}(C)))$$

Rys. 5.14. Deszyfrowanie metodą TDES [11]



Rys. 5.15. Trzykrotny algorytm DES [3]

Są to wzory, opisujące zasadnicze działania w tymże algorytmie.[11]

Algorytm DES nie kreuje żadnych grup, co znaczy, iż wiadomość wynosi  $2^{112}$  w zamian za  $2^{56}$ ; jawny tekst początkowo podlega szyfrowaniu z kluczem  $K_1$ , z kolei

deszyfracja podporządkowana jest kluczowi  $K_2$ , a następnie powtórnie szyfrowana za pomocą klucza  $K_1$ . Działanie deszyfrowania charakteryzuje się podmianieniem działania szyfrowania z deszyfrowaniem dzięki temu samemu kluczowi. Ta operacja została zaprezentowana na poniższym Rys. 5.15.

Istnieje prawdopodobieństwo spotkania edycji DES z użyciem niezależnych podkluczy, gdzie w zamian za generowanie podkluczy o rozmiarze 48 bitów z kluczem pierwotnym 56 bitów zostaje wykorzystana grupa kluczy o 768-bitowych, co można stwierdzić na podstawie obliczeń 16 cykli z niezależnymi podkluczami 48-bitowymi. W operacji mnożenia  $16 \cdot 48$  iloczyn wynosi 768, stanowiąc tym samym dowód niniejszej tezy. W konsekwencji przestrzeń przeznaczona na klucz osiąga  $2^{768}$ . [3]

## **6. Projekt aplikacji klient – serwer z algorytmami szyfrowania symetrycznego**

Niniejszy rozdział stanowi kwintesencję pracy. Elementami składowymi są założenia, wymagania, charakterystyka koncepcji sprawdzonej architektury klient – serwer, korzystającej z systemu szyfrowania symetrycznego w przesyłaniu wiadomości, zaimplementowanemu w języku C#.

### **6.1. Założenia**

Aplikacje klienta i serwera przygotowane w zakresie pracy pozwala użytkownikowi na poniższe możliwości:

- Zrozumienia sensu funkcjonowania zabezpieczonej struktury klienta – serwera
- Zasyfrowaniu i odszyfrowaniu określonej wiadomości, odzwierciedlającej tekst jawny za pomocą sprecyzowanego klucza.

W pierwszym postulatcie myślą przewodnią jest przede wszystkim poznanie w jasno zobrazowany sposób, jak komunikują się komputery ze sobą w sieciach lokalnych, bezprzewodowych i rozległych. Jest to szczególnie ważne, gdyż zawsze jeden z komputerów pełni rolę klienta, natomiast drugi służy jako serwer. Klient wysyła zapytanie, z kolei serwer udostępnia dane, które posiada. Ten fakt przemawia za tym, by informacje były przekazane w obie strony w pewny sposób, uchroniony przed kradzieżą cennych danych przez stronę trzecią. W codzienności klientem jest internauta, korzystający z przeglądarki, natomiast serwerem daje możliwość przeglądania informacji ze stron, które użytkownik pragnie zobaczyć.

Następna teza podkreśla esencję całego projektu. Dzieje się tak, ponieważ do poprawnego skonstruowania architektury klienta i serwera obligatoryjne jest należyta ochrona wartościowych danych. Środkiem zaradczym w problematyce przytoczonego zagadnienia jest szyfrowanie, mogące być symetrycznym lub z kluczem publicznym. Praca obejmuje szyfrowanie symetryczne z wybranymi przykładami jego rodzajów, a więc szyfru strumieniowego i blokowego.



## 6.2. Wymagania

Mając zamiar uruchomić stworzone aplikacje konieczne jest, by dysponować na własnym urządzeniu, czyli komputerze osobistym PC lub laptopie, z systemem operacyjnym firmy Microsoft o nazewnictwie Windows. Minimalnym wymogiem do skompilowania kodu źródłowego, wykonanego w języku programowania C#, jest posiadanie na swoim urządzeniu platformy .NET wraz z zestawem bibliotek niezbędnych do uruchomienia programu. Musi być zainstalowany pakiet .NET Framework. Ponadto niezbędny do kompilacji jest kompilator csc.exe, jest uruchamiany z wiersza poleceń (cmd), osiągalnym tuż po instalacji pakietu .NET Framework i Visual Studio. Są to obligatoryjne zalecenia, jednak poręczne jest zastosowanie zintegrowanego środowiska programistycznego (IDE), którego przykładem jest Visual Studio (darmowa wersja Community), czy SharpDevelop. Ten pierwszy do projektowania aplikacji potrzebuje instalacji składników, które są potrzebne dla programisty języka C#, a także aktywować licencję na produkt poprzez konto personalizacji Microsoft. Z kolei drugi już ma wszystko, co jest niezbędne do pracy, ponadto jest w wersji portable, czyli nie wymagające instalacji na twardym dysku, a plik konfiguracyjny lokują w folderze domyślnym. Projekt w formie dwóch aplikacji: jedna pełniąc funkcję klienta, a druga będąca serwerem sporządzono w zintegrowanym środowisku programistycznym Visual Studio Community 2017 w Wersji 15.9.5, co przemawia za tym, iż niniejsza wersja jest w największym stopniu kompatybilna. Oprócz tego korzystano z pakietu .NET Framework w Wersji 4.7.03056. W przypadku połączenia z Internetem połączenie odgrywa kluczową rolę, lecz ma to szczególne znaczenie jedynie w sytuacji połączenia z drugim komputerem. Rozwijając tą sentencję, należy rozumieć, że aplikacja klienta jest uruchamiana na jednym urządzeniu, jednakże aplikację serwera na drugiej maszynie. W kwestii skorzystania z obu aplikacji na jednym sprzęcie, nie jest konieczny dostęp do sieci.

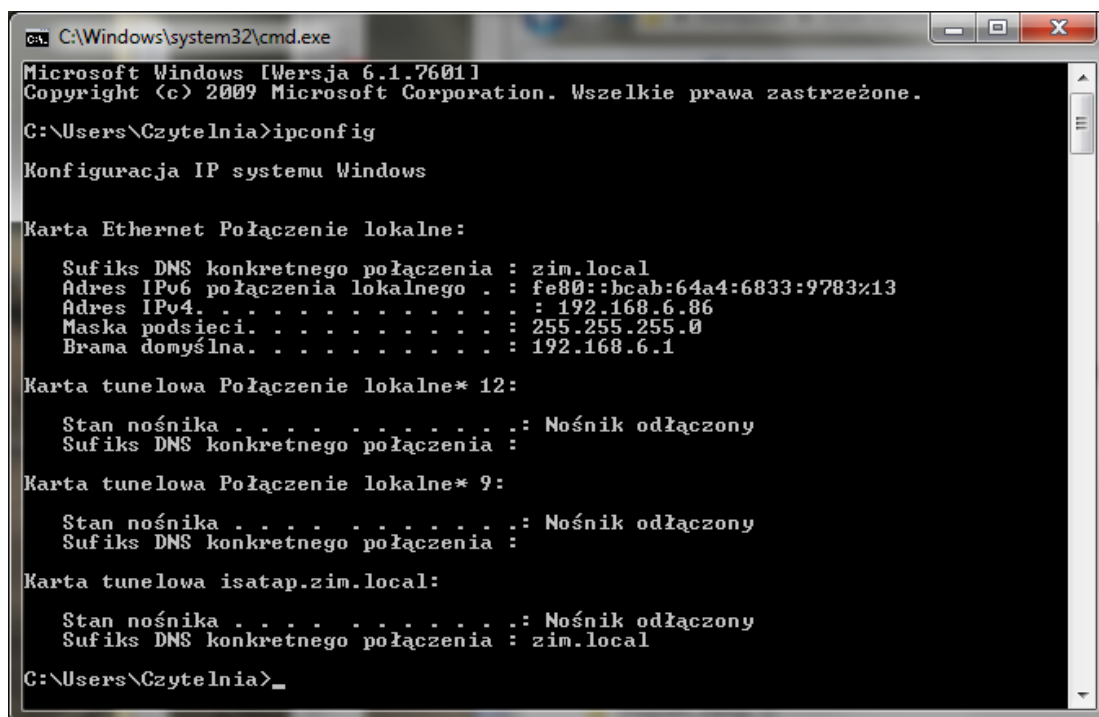
### 6.3. Opis aplikacji

Projekt aplikacji klient – serwer z algorytmami szyfrowania symetrycznego, jak zostało już wcześniej wspomniane, składa się z dwóch programów: pierwszy ma znaczenie klienta, następnie drugi służy jako serwer.

W pierwszej kolejności należy ustalić dwa aspekty niezbędne do działania programu. Pierwszym jest adres IP, stanowiący numer identyfikacyjny komputera lub serwera w sieci, mający fundamentalną rolę w komunikacji wśród urządzeniami. Drugą perspektywą jest port protokołu służącego do identyfikowania procesów, funkcjonujących na oddalonych systemach. Stąd to właśnie należy określić nieużywany port w zakresie od 0 do 65535, będący tym samym dla obydwóch aplikacji, domyślnie ładowany port spełnia kryteria. Odnosząc się do adresu IP, sprawa wygląda dwuznacznie, gdyż jest to zależne w dużej mierze od tego, na ilu sprzętach będzie przeprowadzany test. W przypadku uruchomienia obydwóch aplikacji na jednym sprzęcie domyślny adres IP wystarczy ze względu na to, iż jest to adres pętli zwrotnej, umożliwiającej połączenie z sobą samym. Postać, odnosząca się do dwóch maszyn, wygląda inaczej dlatego, że należy określić numer identyfikacyjny dla urządzenia pełniącego rolę serwera, a w kolejnym kroku wpisać go w pole tekstowe, odnoszące się do adresu IP w aplikacji klient i serwer. Chcąc dokonać także analizy Internet Protocol, trzeba uruchomić wiersz poleceń (cmd), a następnie skorzystać z polecenia ipconfig, wpisując ipconfig, dalej odczytując z danych adres IPv4 z okienka konsoli. Okno wiersza poleceń wraz z potrzebnym adresem ukazuje Rys. 6.1. Po sprawdzeniu adresu Internet Protocol dla serwera jest komplet niezbędnych danych do nawiązania połączenia klienta z serwerem i odwrotnie.

Kolejny krok stanowi wprowadzenie do programu należnych danych. W celu zrealizowania niniejszej wytycznej, należy uruchomić oprogramowanie tak klienta, jak i serwera. Domyślnie podczas załadowania programów adresem IP jest adres pętli zwrotnej 127.0.0.1, a port opisuje się jako 2000, znajdujący się w analogicznych polach tekstowych. Warto zwrócić uwagę na to, że podczas uruchomienia Klienta tekst jawny (wiadomość) i klucz są wpisane wstępnie w należyte pola tekstowe, które użytkownik może sam edytować. Przypisanie tych oto wartości następuje w konstruktorach pokazanych w Rys. 6.3. Jest również możliwość wygenerowania klucza składającego się z losowo wybranych znaków z kodu ASCII, mającego też

losowo wybraną długość. W szacie graficznej znajduje się obszar terminala, gdzie znajduje się przycisk Start (Rys. 6.4), który nawiązuje połączenie z serwerem, szyfruje wiadomość i przesyła ją do serwera.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Wersja 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\Czytelnia>ipconfig

Konfiguracja IP systemu Windows

Karta Ethernet Połączenie lokalne:

    Sufiks DNS konkretnego połączenia : zim.local
    Adres IPv6 połączenia lokalnego . : fe80::bcab:64a4:6833:9783%13
    Adres IPv4. . . . . : 192.168.6.86
    Maska podsieci. . . . . : 255.255.255.0
    Brama domyślna. . . . . : 192.168.6.1

Karta tunelowa Połączenie lokalne* 12:

    Stan nośnika . . . . . : Nośnik odłączony
    Sufiks DNS konkretnego połączenia :

Karta tunelowa Połączenie lokalne* 9:

    Stan nośnika . . . . . : Nośnik odłączony
    Sufiks DNS konkretnego połączenia :

Karta tunelowa isatap.zim.local:

    Stan nośnika . . . . . : Nośnik odłączony
    Sufiks DNS konkretnego połączenia : zim.local

C:\Users\Czytelnia>
```

Rys. 6.1. Okienko Wiersza poleceń z wyświetlonymi parametrami dla komendy ipconfig  
[opracowanie własne]

Przycisk Zamknij (Rys. 6.4) daje możliwość zakończenia działania oprogramowania, jeśli użytkownik nie chce dalej korzystać z tego, co oferuje aplikacja kliencka. Ostatnim przyciskiem Generuj klucz zostaje wytworzony klucz o przypadkowej długości i składający się z przypadkowych znaków drukowalnych, zawartych w kodzie ASCII. Funkcję kreującą klucz opisuje Rys. 6.2.

```
private void generuj_klucz()
{
    int k; char c;
    klucz = "";
    generator = new Random();
    for(int j = 0; j < generator.Next(1, 127-32); j++)
    {
        k = generator.Next(32, 127);
        c = (char)k;
        klucz += c.ToString();
    }
    txtklucz.Text = klucz;
}
```

Rys. 6.2. Funkcja generująca klucz jako alternatywa dla wpisywania klucza przez użytkownika (obiekt generator typu Random zadeklarowany globalnie) [opracowanie własne]

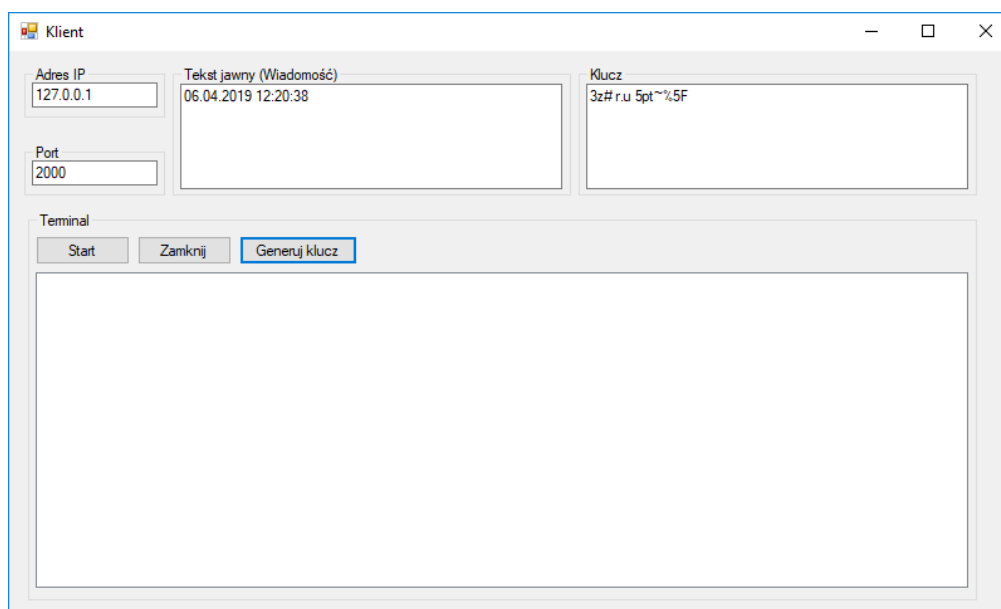
```

public Form1()
{
    InitializeComponent();
    txtIP.Text = "127.0.0.1";
    txtPort.Text = "2000";
    txtwiadomosc.Text = Convert.ToString(DateTime.Now);
    generuj_klucz();
}

public Form1()
{
    InitializeComponent();
    opcja = rB_RC4.Text;
    textBox1.Text = "127.0.0.1";
    textBox2.Text = "2000";
}

```

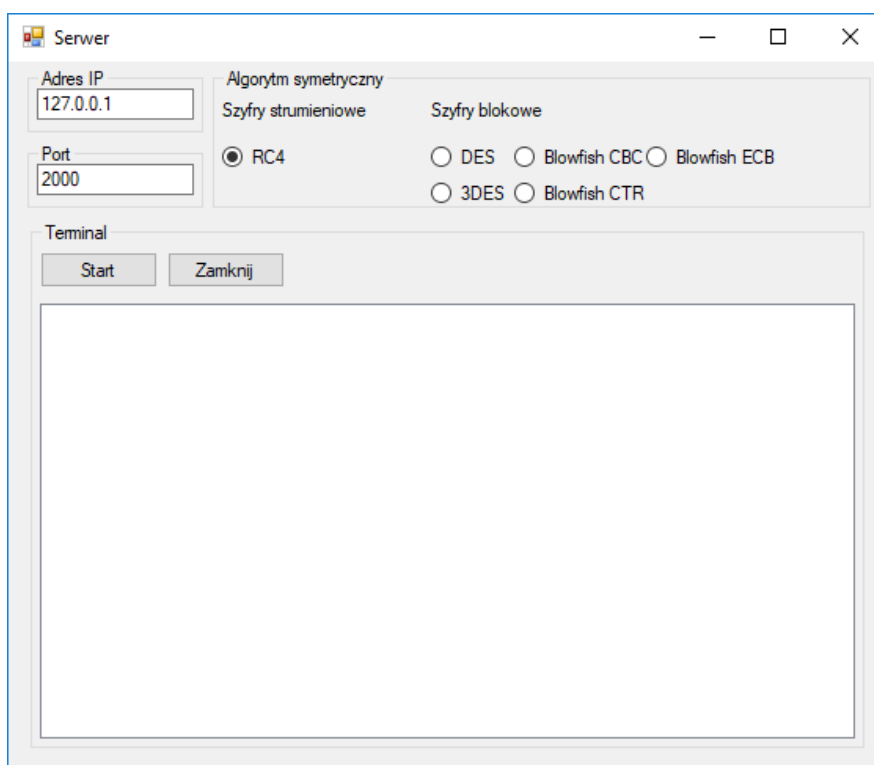
Rys. 6.3. Konstruktry klient (u góry) i serwera (na dole) przypisujące domyślne wartości do odpowiednich komponentów i wywołanie funkcji generowania klucza [opracowanie własne]



Rys. 6.4. Aplikacja Klient po uruchomieniu [opracowanie własne]

Warto zaznaczyć, że podczas załadowania programu klienckiego w polu tekstowym, gdzie można wpisać wiadomość do przesłania, pojawia się data i czas uruchomienia (Rys. 6.4). Stanowi to przypadkową koncepcję, której powód zawarcia w tej części projektu spełnia rolę zapobiegnięcia wyjątkowi, który zostanie opisany w dalszej części tego oto podrozdziału. Program serwera zawiera w swojej szacie graficznej podobnie jak aplikacja kliencka pola tekstowe, w które można wpisać adres IP i port sieciowy. Ponadto znajdują się przyciski opcji, odgrywające fundamentalną rolę w realizacji działania całego projektu ze względu na fakt, iż użytkownik wybiera nimi algorytm symetryczny, mogący być przykładem szyfru strumieniowego lub blokowego, dzięki któremu klient zaszyfruje wiadomość, z kolei serwer będzie ją

deszyfrował. Domyślnie jest wybrana opcja użycia szyfru strumieniowego RC4 w momencie załadowania oprogramowania, co jest jednym z środków ostrożności, mającymi zapobiec nieoczekiwanemu wyjątkowi, polegającemu na braku wybranego algorytmu do szyfrowania i deszyfrowania. Jeden z szyfrów blokowych został zaimplementowany w trzech trybach kodowania, co jest widoczne na szacie graficznej, gdyż znajdują się na niej trzy przyciski opcji dotyczące szyfru Blowfish w różnych trybach kodowania. Za przypisanie konkretnej metody szyfrowania wiadomości odpowiada fragment kodu źródłowego na Rys. 6.6. Na dole znajduje się terminal, gdzie mają swoje miejsce dwa przyciski, posiadające przypisane funkcjonalności. Szatę graficzną po uruchomieniu Serwera przedstawia Rys. 6.5. Przycisk Start uruchamia nasłuchiwanie serwera i wysyła po nawiązaniu połączenia z klientem pierwszą istotną informację, zawierającą daną, jakiego algorytmu ma użyć klient do zaszyfrowania wiadomości.



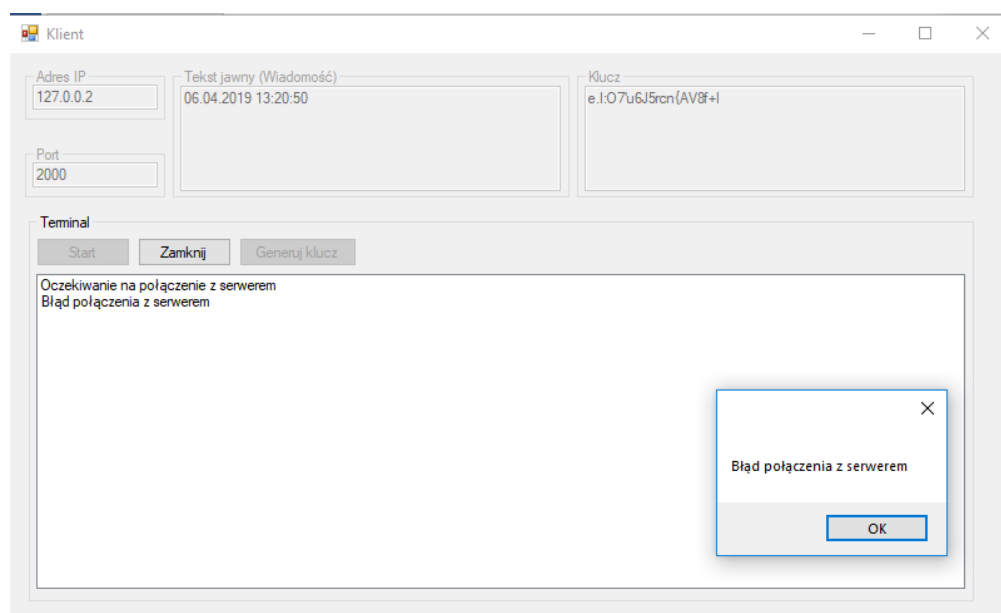
Rys. 6.5. Aplikacja Serwer po uruchomieniu [opracowanie własne]

Drugi przycisk Zamknij (Rys. 6.5) posługuje, jako ten, który kończy działanie oprogramowania serwera. Innymi słowy spełnia identyczną funkcję, jak ma to miejsce w oprogramowaniu klienckim.

Wracając do wątku adresu IP, domyślną wartość parametru, czyli adres pętli zwrotnej, można użyć jedynie w sytuacji, gdy używane są obie aplikacje na jednym komputerze, a co za tym idzie nie jest potrzebne połączenie internetowe. W tej okoliczności musi być ten sam numer identyfikacyjny, w przeciwnym razie pojawi się pierwszy konflikt, który zakończy działanie klienta, a serwer ulegnie zawieszeniu, jak jest to zaprezentowane na Rys. 6.7.

```
private void opcjaalgorytmu(object sender, EventArgs e)
{
    if (rB_RC4.Checked == true) opcja = rB_RC4.Text;
    else if (rB_DES.Checked == true) opcja = rB_DES.Text;
    else if (rB_3DES.Checked == true) opcja = rB_3DES.Text;
    else if (rB_BlowfishECB.Checked == true) opcja = rB_BlowfishECB.Text;
    else if (rB_BlowfishCBC.Checked == true) opcja = rB_BlowfishCBC.Text;
    else if (rB_BlowfishCTR.Checked == true) opcja = rB_BlowfishCTR.Text;
}
```

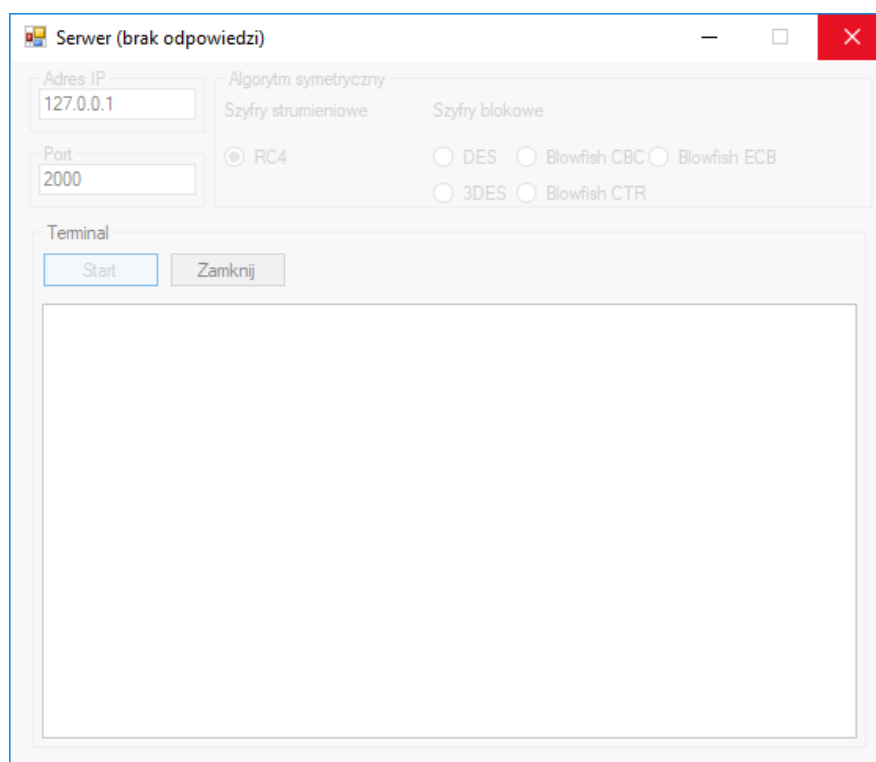
Rys. 6.6. Wybór opcji algorytmu szyfrowania po stronie serwera [opracowanie własne]



Rys. 6.7. Aplikacja Klient w sytuacji innego adresu IP od tego, który ma aplikacja Serwera [opracowanie własne]

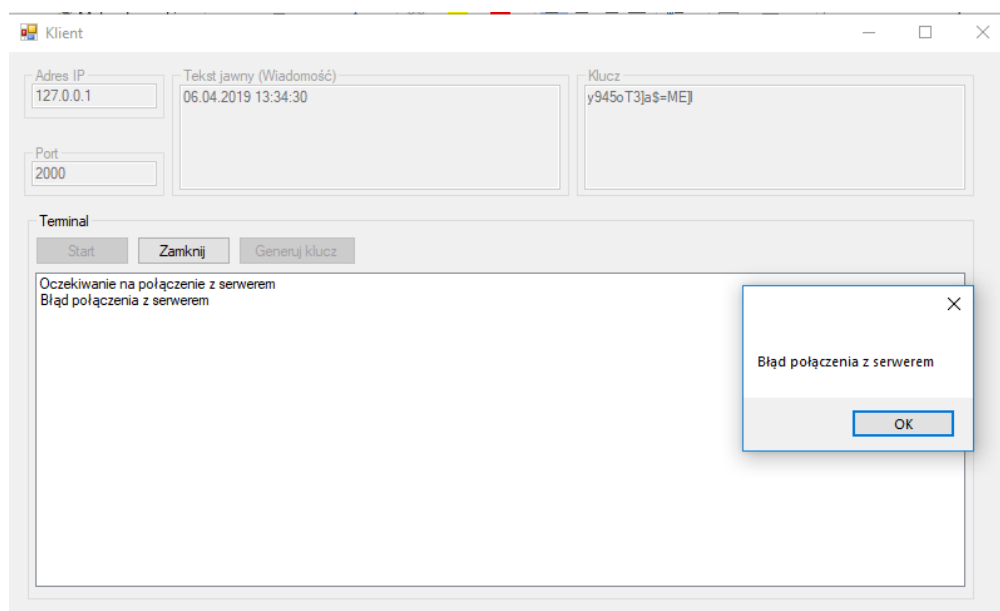
Pojawia się komunikat o błędzie połączenia z serwerem, a także jego treść ukazuje się w zawartości terminala. Po potwierdzeniu przeczytania komunikatu następuje zamknięcie oprogramowania klienckiego. Działanie użytkownika w stosunku do programu Serwer, by skutecznie zakończyć jego pracę, może opierać się na uruchomieniu Menedżera zadań i w obszarze uruchomionych aplikacji zaznaczenie procesu działania Serwera, aby następnie zakończyć jego proces, klikając przycisk

Zakończ zadanie. Opcja zamknięcia programu za pomocą przycisku jest niemożliwa, ponieważ nie ma możliwości jego naciśnięcia ze względu na zawieszenie jego pracy (Rys. 6.8). Drugim rozwiązaniem jest zatrzymanie pracy z poziomu IDE, czyli zintegrowanego środowiska programistycznego, w tym przypadku Visual Studio Community 2017, klikając czerwony kwadrat, zatrzymujący debugowanie lub wciskając kombinację klawiszy Shift + F5. Działa to rozwiązanie jedynie, gdy jest uruchamiane oprogramowanie wraz z kompilacją z poziomu VS, co jest minusem tegoż rozwiązania.

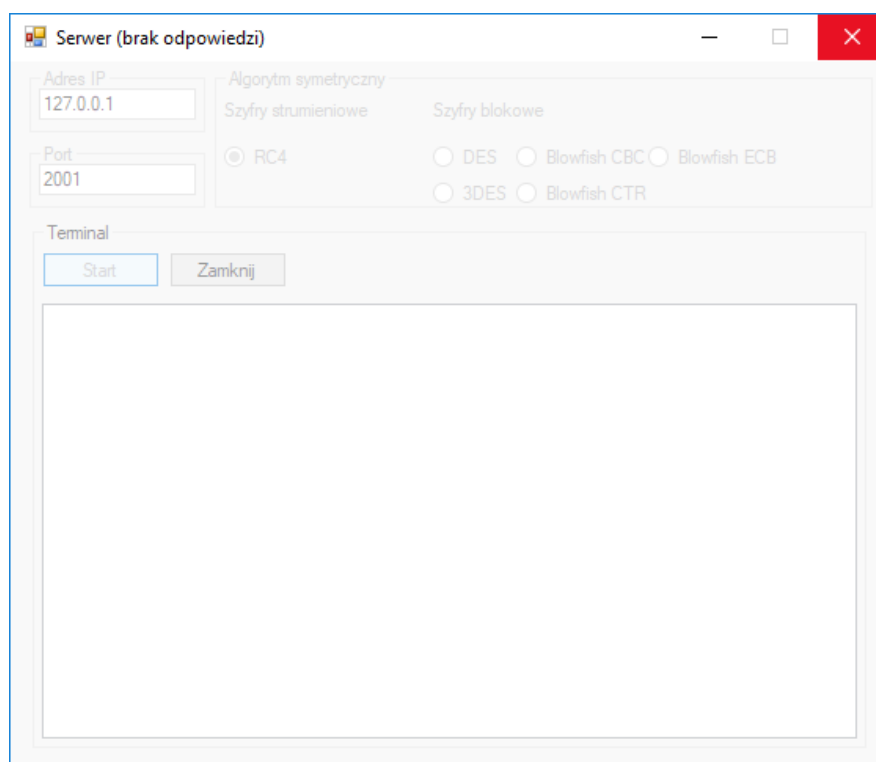


Rys. 6.8. Aplikacja Serwer w sytuacji innego adresu IP od tego, który ma aplikacja Klienta  
[opracowanie własne]

Następnym wyjątkiem, który może pojawić się podczas użytkowania projektu jest różnica we wpisaniu numeru portu sieciowego w obydwu programach, co ukazane jest niżej. W tym przypadku również kończy się to tymi samymi efektami, co w poprzedniej sytuacji. Klient komunikuje, że nastąpił błąd połączenia z serwerem i po kliknięciu przycisku potwierdzającego OK następuje koniec jego działania (Rys. 6.9). Podobnie ma się istota funkcjonowania oprogramowania serwera, praca aplikacji ulega zawieszeniu i nie ma możliwości wciśnięcia przycisku Zamknij (Rys. 6.10). Środki zaradcze na tenże problem są dwa identyczne, jak w poprzednim konflikcie.



Rys. 6.9. Aplikacja Klient w sytuacji innego portu sieciowego w stosunku do tego wprowadzonego w aplikacji Serwera [opracowanie własne]

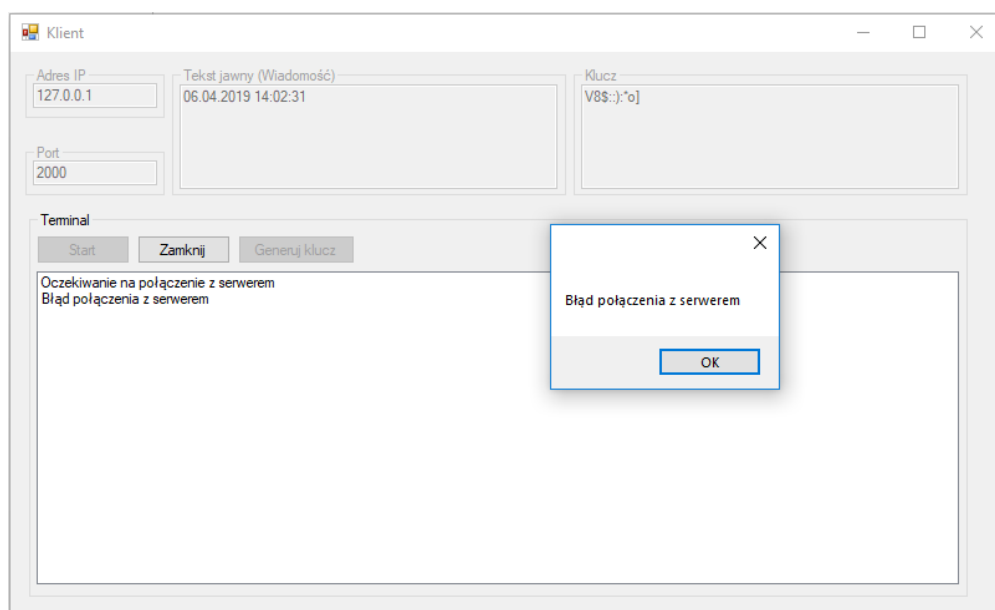


Rys. 6.10. Aplikacja Serwer w sytuacji innego portu sieciowego w stosunku do tego wprowadzonego w aplikacji Klienta [opracowanie własne]

Trzecim przypadkiem konfliktu jest okoliczność, gdy jednocześnie w obydwóch aplikacjach zostaje wpisany różny adres IP i port sieciowy. Pojawia się identyczny

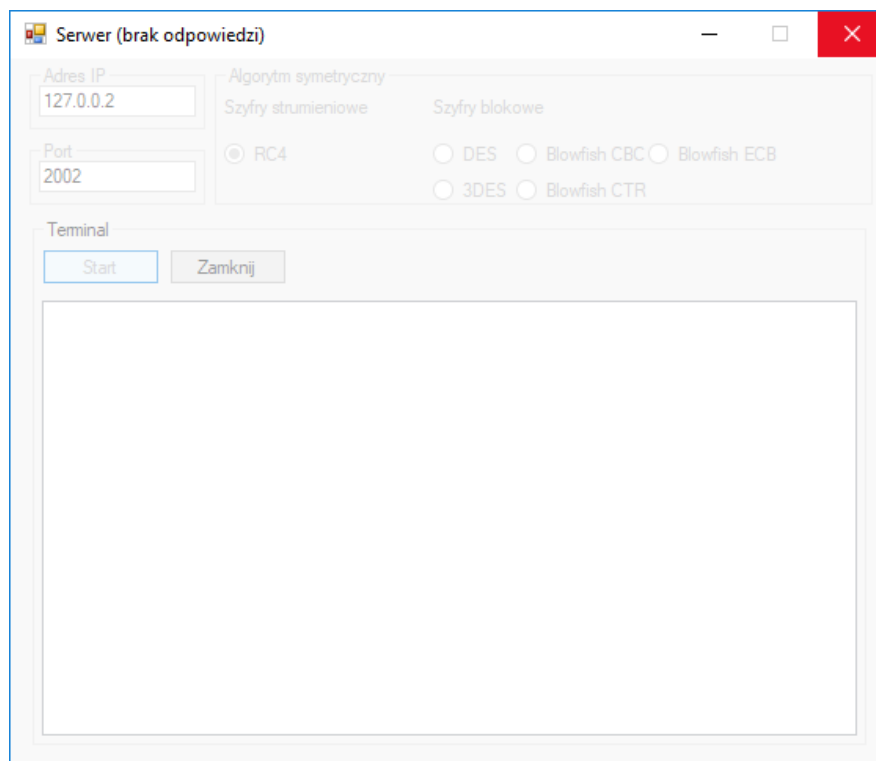


komunikat po stronie klienta i na jego terminalu, a po zatwierdzeniu przyciskiem OK, oprogramowanie kończy pracę (Rys. 6.11).



Rys. 6.11. Aplikacja Klient w sytuacji innego adresu IP i portu sieciowego w stosunku do tych wprowadzonych w aplikacji Serwera [opracowanie własne]

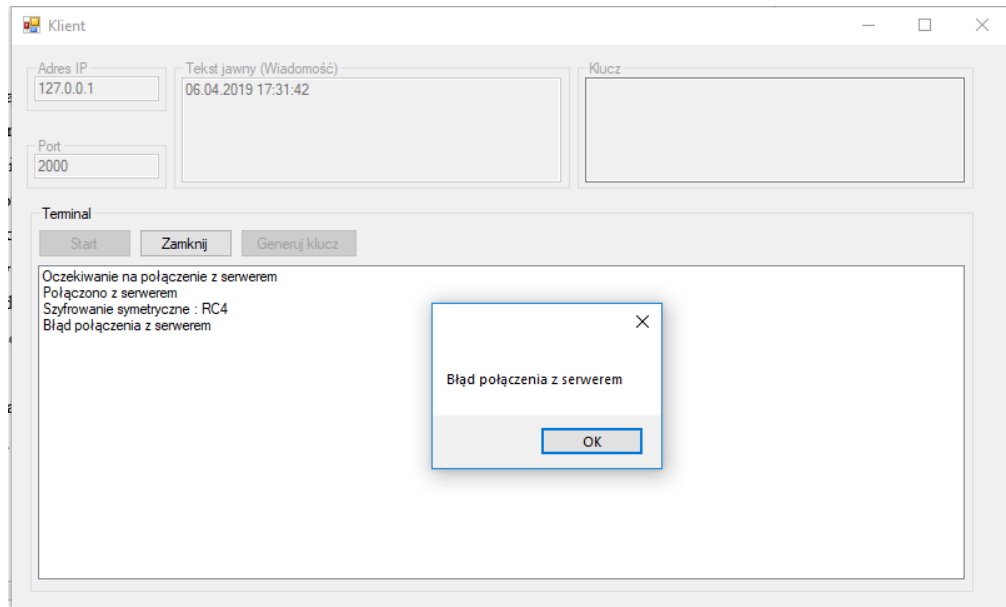
Program, pracujący jako serwer, analogicznie odnosi się do rozbieżności tak numeru identyfikacyjnego, jak i portu jednocześnie. Zachowuje się podobnie, jak ma to miejsce jeśli jest niekompatybilność jednego z dwóch parametrów. Warte podkreślenia jest istota reakcji oprogramowania klienta i serwera na rozbieżności IP i portu, kombinacji liczby poprawnych wartości niniejszych wytycznych. Błędne wpisanie choćby jednej wielkości prowadzi do tych samych konsekwencji (Rys.6.12). Rozpatrując pozostałe trzy przypadki, gdy pojawia się niespójność w działaniu analizowanej architektury, warto powrócić do dwóch koncepcji, na pozór wydających się trywialnymi. W pierwszej idei chodzi o to, dlaczego przy załadowaniu aplikacji w polu tekstowym klucza są przypadkowe znaki i jest ich losowa liczba, a co za tym idzie długość klucza? Druga zaś dotyczy pola tekstowego z tekstem jawnym, a dokładnie dotyczy przyczynowości daty i godziny w tym oto komponencie. Odpowiedź polega na tym, że bez wiadomości szyfrowanie nie ma żadnego sensu ze względu na fakt, iż coś musi podlegać tej procedurze. Brak wiadomości prowadzi do wyświetlenia komunikatu o błędzie w rozpoczęciu nasłuchiwanie po stronie serwera (Rys. 6.16), a klient nie otrzymuje odpowiedzi (Rys. 6.15).



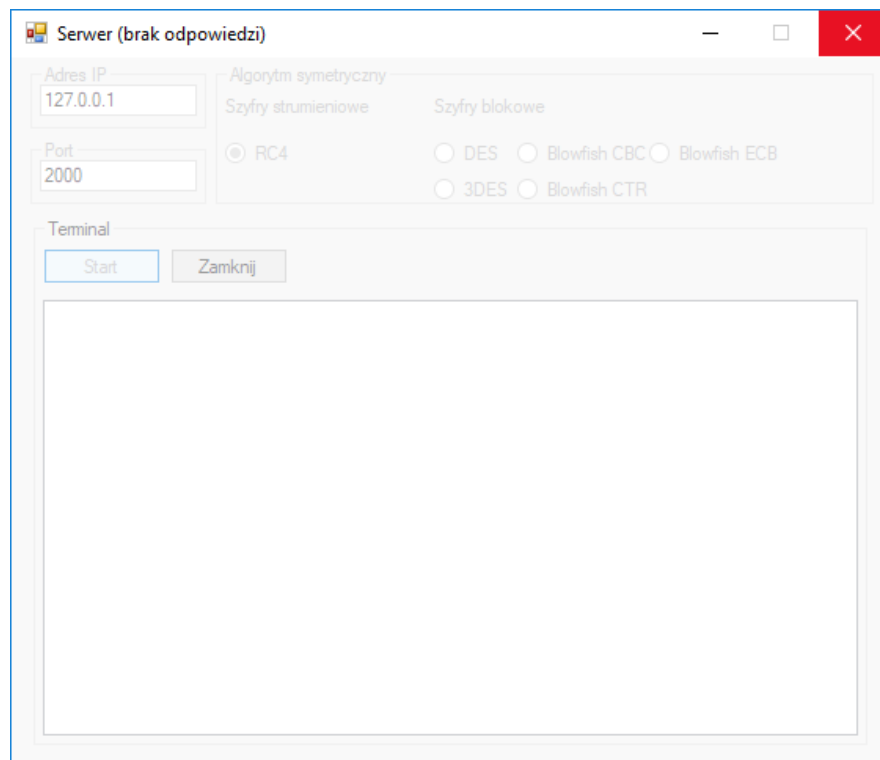
Rys. 6.12. Aplikacja Serwer w sytuacji innego adresu IP i portu sieciowego w stosunku do tych wprowadzonych w aplikacji Klienta [opracowanie własne]

Z kolei brak klucza skutkuje błędem w połączeniu z serwerem po stronie klienta (Rys. 6.13), a oprogramowanie serwera podlega zawieszeniu (Rys. 6.14). Z tego względu podczas uruchamiania te oto dwa pola tekstowe są wypełnione, gdyż mają dać użytkownikowi do myślenia. Mogą w pewien sposób ukazać, że brak poprawnego jednego z tych dwóch parametrów pozbawia sensu działanie algorytmu szyfrującego, nie będącego obligatoryjnie rodzajem algorytmu symetrycznego, bo może dotyczyć również tego, który posiada klucz publiczny i prywatny, muszą mieć długość większą od zera. Przykładem świata rzeczywistego jest szyfr Cezara, którego używał twórca do porozumiewania się ze swoimi przyjaciółmi, a więc nie ustalając i wysyłając do nich „pusty” klucz działałby na własną niekorzyść, gdyż jego wiadomość nie zostałaby zaszyfrowana i stałaby się tym samym tekstem publicznym, dostępnym dla każdego. Nie nastąpiłoby w owym czasie żadne szyfrowanie. Idea wysyłania klucza do przyjaciela, a następnie zaszyfrowanie wiadomości o długości równej zero, podważyłoby to, czy jego intencje są w porządku względem sprzymierzeńca, a

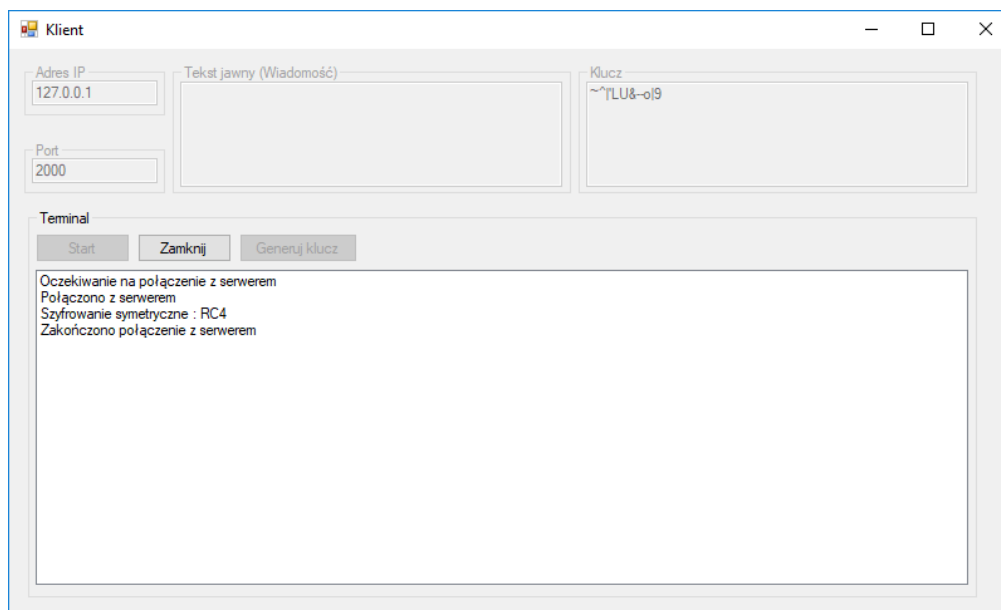
przynajmniej wzbudziłoby obawy o odbiorcy. Kontrowersja z brakiem tekstu jawnego, który ma być szyfrowany i deszyfrowany w przyszłości, po stronie klienta, daje pozorne wrażenie, że wszystko jest w porządku.



Rys. 6.13. Aplikacja Klienta w sytuacji braku wprowadzonego klucza [opracowanie własne]



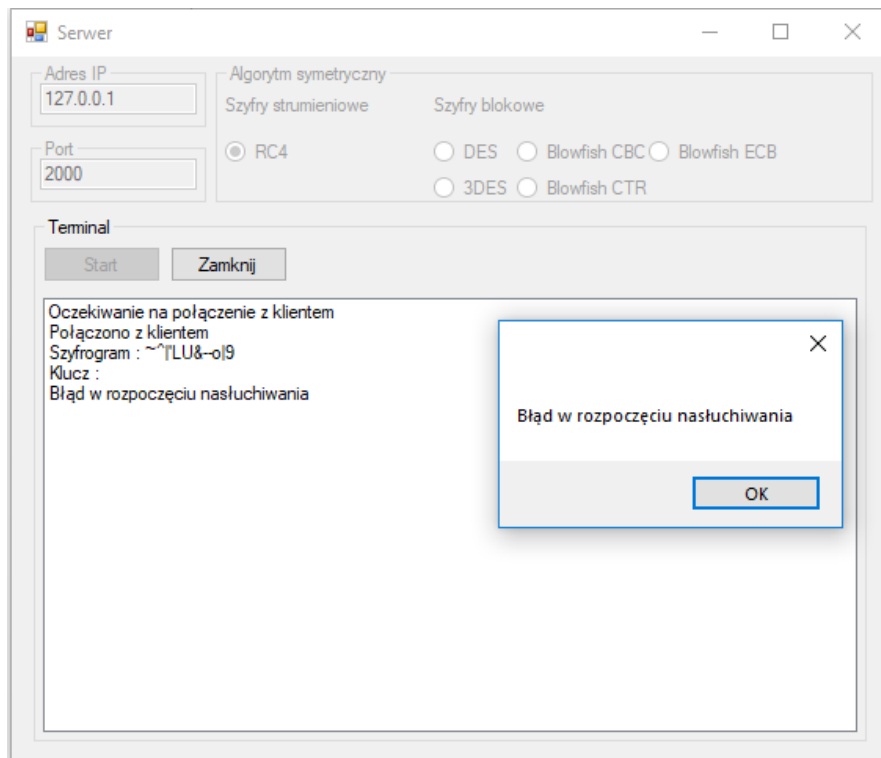
Rys. 6.14. Aplikacja Serwera w sytuacji braku wprowadzonego klucza [opracowanie własne]



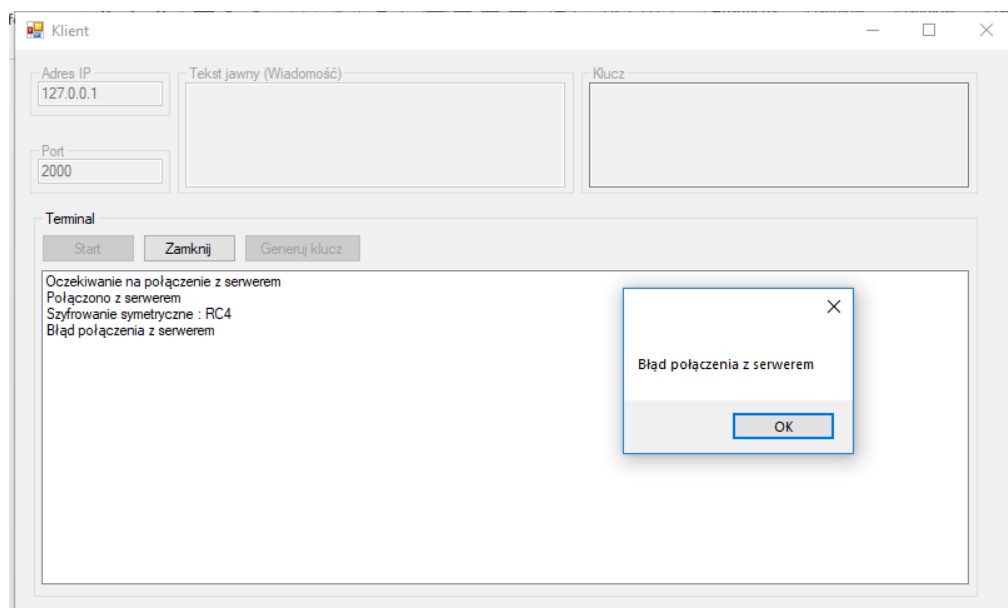
Rys. 6.15. Aplikacja Klienta w sytuacji braku wiadomości [opracowanie własne]

Realny obraz problemu pojawia się w momencie przekazania informacji, będącej pustą wiadomością dopiero po stronie serwera. Zdarzenie to prezentuje z jednej strony przekazanie klucza przez klienta do serwera, a z drugiej wysłanie wątpliwej wiadomości przez klienta do serwera. Należy ją rozumieć jako tą, na której nie można nic dokonać, gdyż jej długość musi być większa od zera. Otrzymanie klucza i wiadomości, w której tekst jest błędny z perspektywy serwera, uniemożliwia proces deszyfrowania, gdyż warunkiem tego procesu jest posiadanie czegoś, co ma podlegać analizie. Ostatni wyjątek jest połączeniem dwóch poprzednich, ponieważ brak wiadomości i klienta wyklucza definitywnie działanie szyfrowania. Należy zwrócić szczególną uwagę na to, że pokazuje się komunikat tak po stronie klienta (Rys. 6.17), jak i po stronie serwera (Rys. 6.18), który po potwierdzeniu przyciskiem OK kończy działanie oprogramowania. Rozwiązuje to problem zamknięcia aplikacji serwer, która w pozostałych przypadkach powodowała zawieszenie pracy programu. Ten przypadek bezapelacyjnie skreśla możliwość przeprowadzenia tekstu jawnego przez proces zaszyfrowania, przesłania, a następnie odbioru i na koniec deszyfrowania.

Reasumując, jest pięć przypadków, gdy oprogramowanie klienta wykazuje błąd połączenia z serwerem, a po stronie serwera dochodzi do zawieszenia jego pracy.

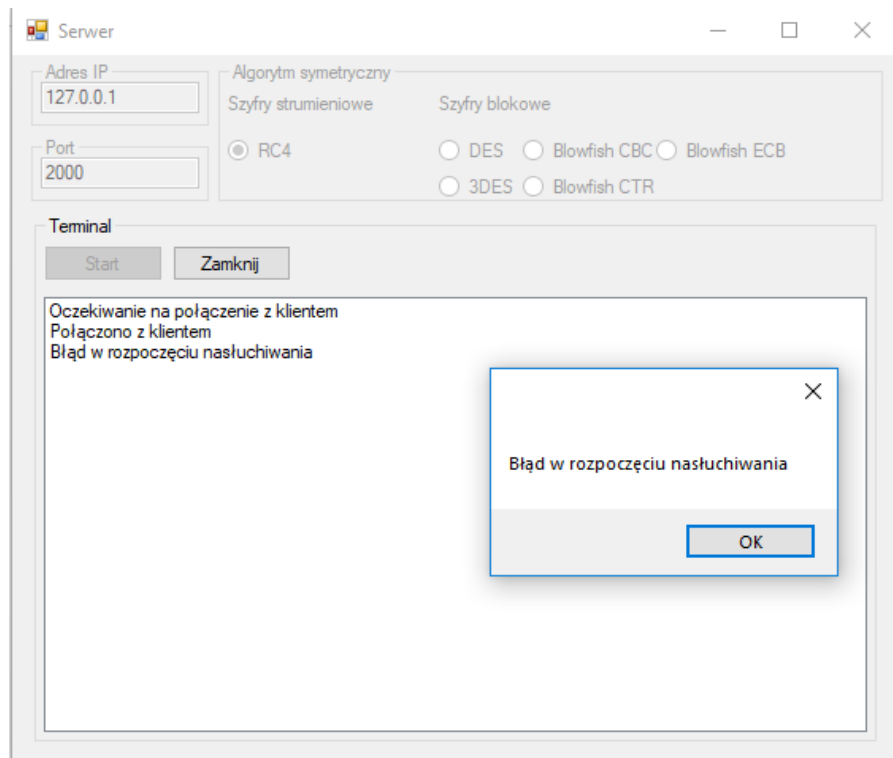


Rys. 6.16. Aplikacja Serwera w sytuacji braku wiadomości [opracowanie własne]



Rys. 6.17. Aplikacja Klienta w sytuacji braku wiadomości i klucza [opracowanie własne]

Wyjątkiem jest szósty przypadek, kiedy jest pusty klucz i wiadomość, wtedy następuje komunikat po stronie serwera o błędzie w rozpoczęciu nasłuchiwania i oprogramowanie nie ulega zawieszeniu.



Rys. 6.18. Aplikacja Serwera w sytuacji braku wiadomości i klucza [opracowanie własne]

Są to:

- a) sprzeczność adresów IP (numeru identyfikacyjnego sprzętu)
- b) rozbieżność portów sieciowych
- c) jednoczesna przeciwstawność adresów IP i portów sieciowych
- d) brak klucza
- e) pusta wiadomość (długość łańcucha znaków równa zero)
- f) połączenie braku klucza i pustej wiadomości

Klient nie reaguje na to, że jest błąd w danych wejściowych jedynie w przypadku pustej wiadomości. Serwer podlega zawieszeniu pracy w prawie każdym przypadku za wyjątkiem, gdy ma pustą wiadomość lub poza nią jest jeszcze pusty klucz. Pojęcie pusty znaczy to samo, co długość zero, więc nie jest spełnione kryterium.

## 6.4. Testy programu

Ten oto podrozdział stanowi prezentację działania oprogramowania dla każdego przykładu szyfrowania symetrycznego, zaimplementowanego w niniejszym projekcie. Testy aplikacji zostały podzielone na dwie części: pierwsza obejmuje testy układu

klient – serwer lokalnie na jednym sprzęcie, natomiast druga zawiera testy przeprowadzone na dwóch komputerach, połączonych ze sobą za pośrednictwem bezprzewodowej sieci Wi-Fi.

#### 6.4.1. Testowanie na jednym sprzęcie

Do przeprowadzenia wspomnianego aktualnie testu właściwie tak aplikacja klienta, jak i serwera jest gotowa ze względu na to, iż domyślny adres IP i port sieciowy jest już ustawiony. Dla przypomnienia adres IP jest ustawiony na adres pętli zwrotnej, umożliwiającą łączenie się ze sobą samym. Z kolei kwestia połączenia z siecią globalną nie jest obowiązkowa, nie wpływa na działanie oprogramowania klienckiego i serwerowego w żaden sposób. Uwarunkowane jest to tym, iż procesy dzieją się na lokalnym hoście, nie ma komunikacji z innym sprzętem, innymi słowy nie uczestniczy w pracy drugi system operacyjny. W poprzednim rozdziale omówiono istotę poszczególnych przykładów metod szyfrowania symetrycznego. Rozpatrując jako pierwszy algorytm RC4, warto wspomnieć o tym, iż przebieg procesu szyfrowania opiera się na permutacji S i fundamentalnej części tworzenia szyfrogramu zaimplementowanej jak na Rys. 6.19.

```
private static byte[] EncryptInitialize(byte[] key)
{
    byte[] s = Enumerable.Range(0, 256)
        .Select(i => (byte)i)
        .ToArray();

    for (int i = 0, j = 0; i < 256; i++)
    {
        j = (j + key[i % key.Length] + s[i]) & 255;
        Swap(s, i, j);
    }

    return s;
}

private static IEnumerable<byte> EncryptOutput(byte[] key, IEnumerable<byte> data)
{
    byte[] s = EncryptInitialize(key);
    int i = 0;
    int j = 0;

    return data.Select((b) =>
    {
        i = (i + 1) & 255;
        j = (j + s[i]) & 255;
        Swap(s, i, j);
        return (byte)(b ^ s[(s[i] + s[j]) & 255]);
    });
}

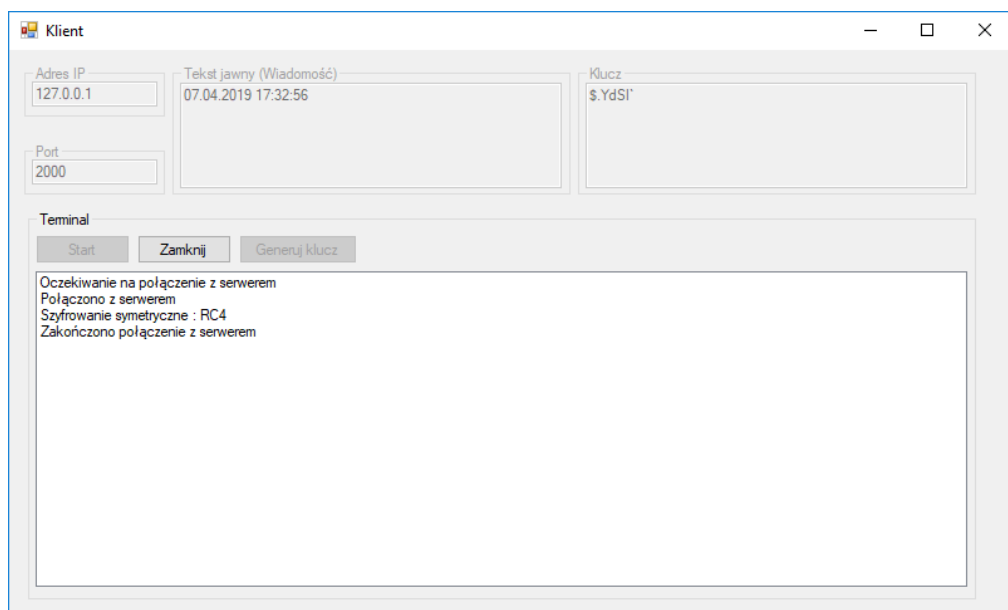
private static void Swap(byte[] s, int i, int j)
{
    byte c = s[i];
    s[i] = s[j];
    s[j] = c;
}
```

Rys. 6.19. Zasadniczy kod odpowiadający za permutację S i szyfrowanie metodą RC4

[opracowanie własne]

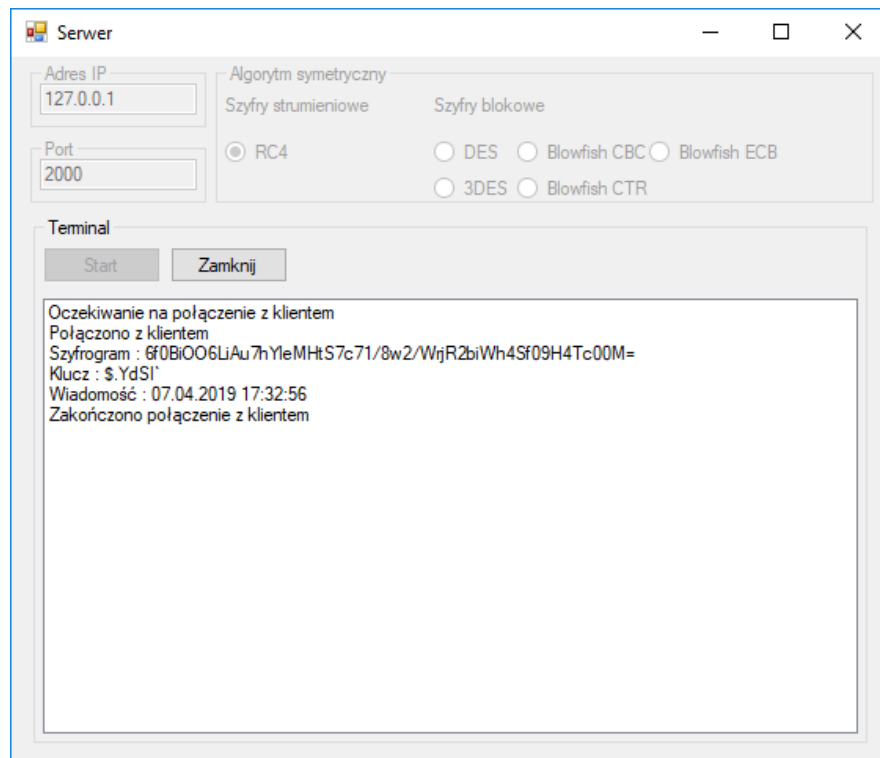
Poniższe zrzuty ekranowe ukazują efekty szyfrowania i deszyfrowania wybranymi przez użytkownika algorytmami po stronie serwera, które mogą być szyfrem strumieniowym lub blokowym.

Na początek są przedstawione konsekwencje wyboru opcji RC4, czyli jedynego szyfru strumieniowego zaimplementowanego w projekcie dla aplikacji klienckiej (Rys. 6.20) i serwerowej (Rys. 6.21). Następnie zaprezentowane zostały skutki wyboru szyfru blokowego DES. Wygląd interfejsu klienta przedstawia Rys. 6.23, natomiast serwera ma postać jak na Rys. 6.25.



Rys. 6.20. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru strumieniowego RC4  
[opracowanie własne]



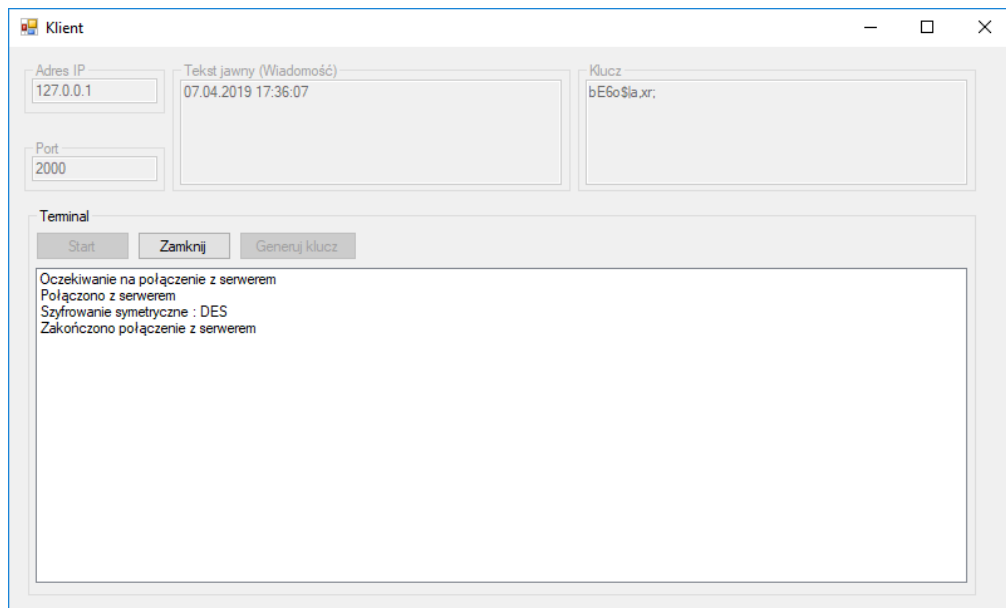


Rys. 6.21. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru strumieniowego RC4  
[opracowanie własne]

Szyfrowanie DES zostało zaimplementowane za pomocą wbudowanych funkcji w języku C# (Rys. 6.22).

```
public static string Encrypt(string originalString)
{
    if (String.IsNullOrEmpty(originalString))
    {
        throw new ArgumentNullException(
            "The string which needs to be encrypted can not be null.");
    }
    DESCryptoServiceProvider cryptoProvider = new DESCryptoServiceProvider();
    MemoryStream memoryStream = new MemoryStream();
    CryptoStream cryptostream = new CryptoStream(memoryStream,
        cryptoProvider.CreateEncryptor(bytes, bytes), CryptoStreamMode.Write);
    StreamWriter writer = new StreamWriter(cryptostream);
    writer.Write(originalString);
    writer.Flush();
    cryptostream.FlushFinalBlock();
    writer.Flush();
    return Convert.ToBase64String(memoryStream.GetBuffer(), 0, (int)memoryStream.Length);
}
```

Rys. 6.22. Fragment kodu źródłowego odpowiedzialnego za szyfrowanie metodą DES [opracowanie własne]

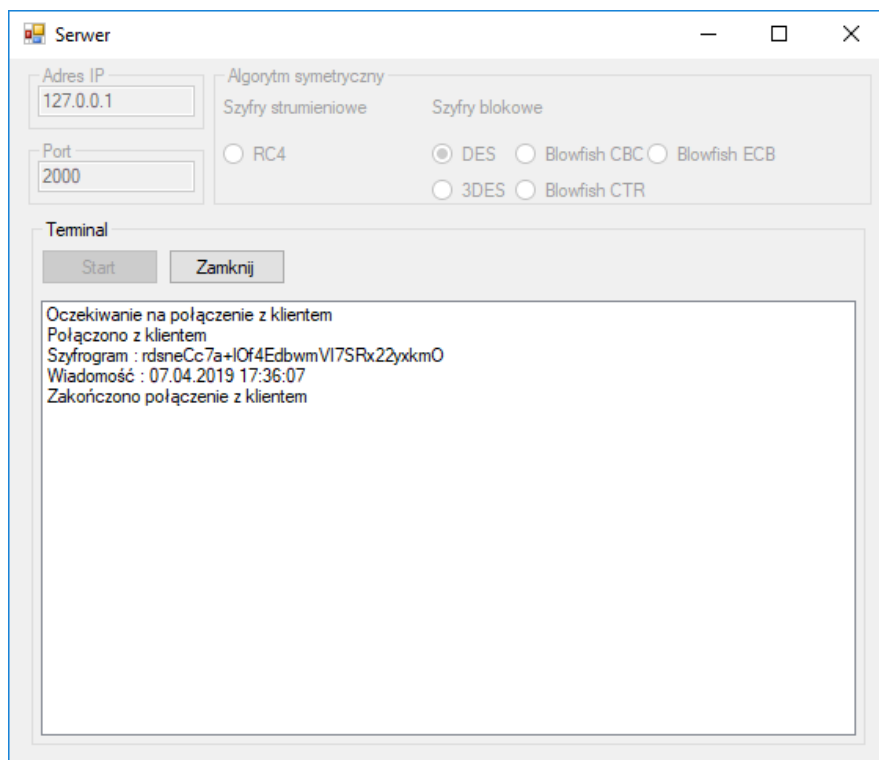


Rys. 6.23. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego DES  
[opracowanie własne]

Kod, który odpowiada za proces deszyfrowania, również korzysta z osadzonych funkcji (Rys. 6.24).

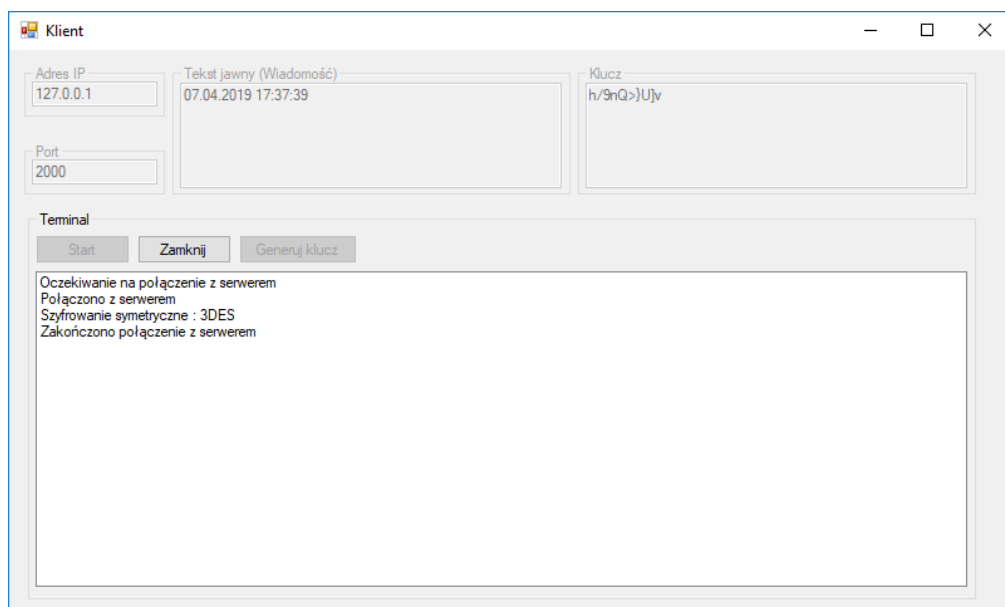
```
public static string Decrypt(string cryptedException)
{
    if (string.IsNullOrEmpty(cryptedException))
    {
        throw new ArgumentNullException(
            "The string which needs to be decrypted can not be null.");
    }
    DESCryptoserviceProvider cryptoProvider = new DESCryptoserviceProvider();
    MemoryStream memoryStream = new MemoryStream(
        Convert.FromBase64String(cryptedException));
    CryptoStream cryptostream = new CryptoStream(memoryStream,
        cryptoProvider.CreateDecryptor(bytes, bytes), CryptoStreamMode.Read);
    StreamReader reader = new StreamReader(cryptostream);
    return reader.ReadToEnd();
}
```

Rys. 6.24. Fragment kodu źródłowego odpowiedzialnego za szyfrowanie metodą DES [opracowanie  
własne]

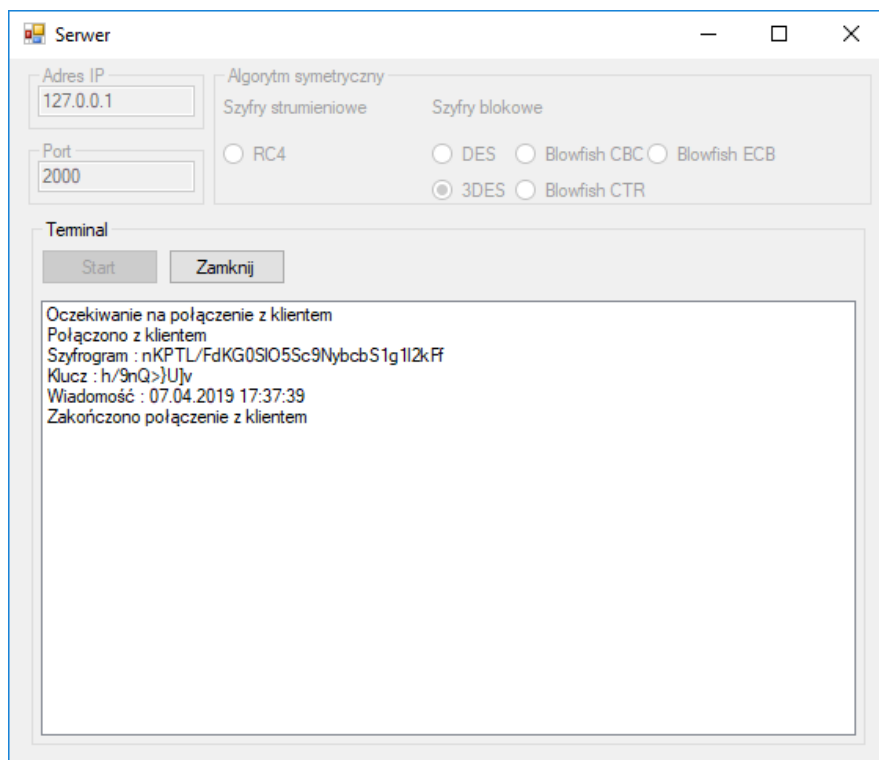


Rys. 6.25. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego DES  
[opracowanie własne]

Podobnież mają się konsekwencje wyboru metody 3DES, gdzie okno klienckie ukazuje Rys. 6.26, a programu serwera Rys. 6.27.

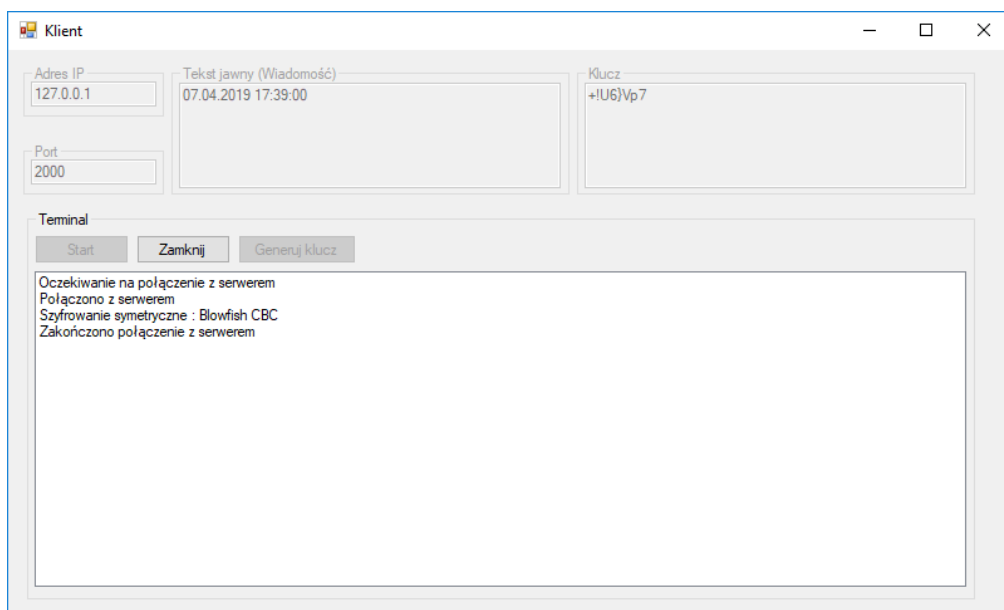


Rys. 6.26. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego 3DES  
[opracowanie własne]

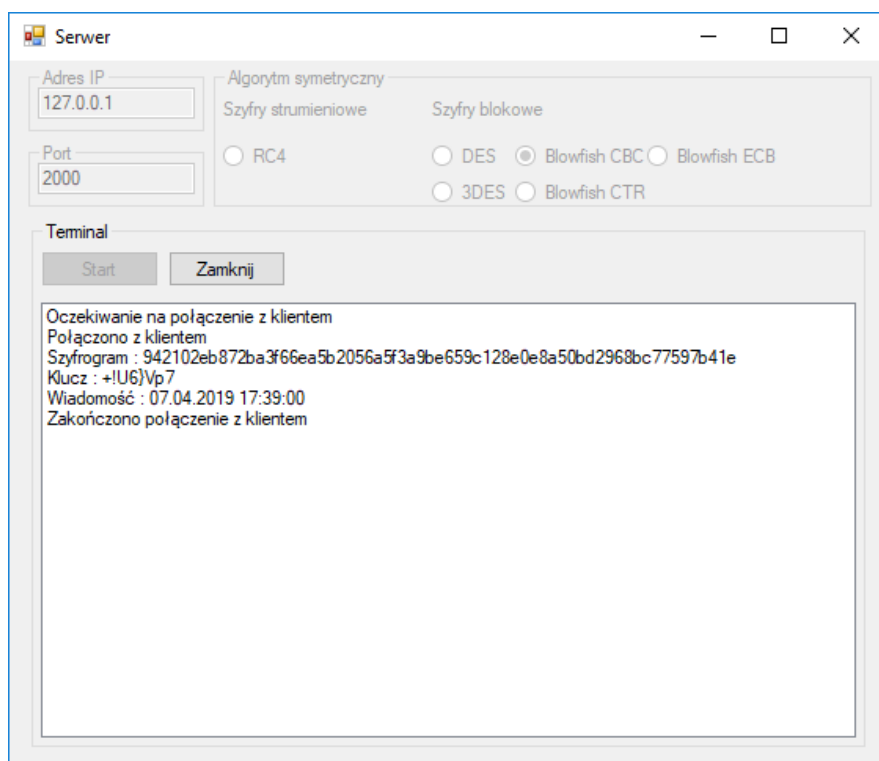


Rys. 6.27. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego 3DES [opracowanie własne]

Algorytmem, który ma największy wkład, jest niewątpliwie Blowfish, ponieważ jest zaimplementowany z możliwością wyboru jednego z trzech trybów kodowania. Interfejs graficzny po wyborze trybu CBC ukazuje Rys. 6.28, a serwera Rys. 6.29.

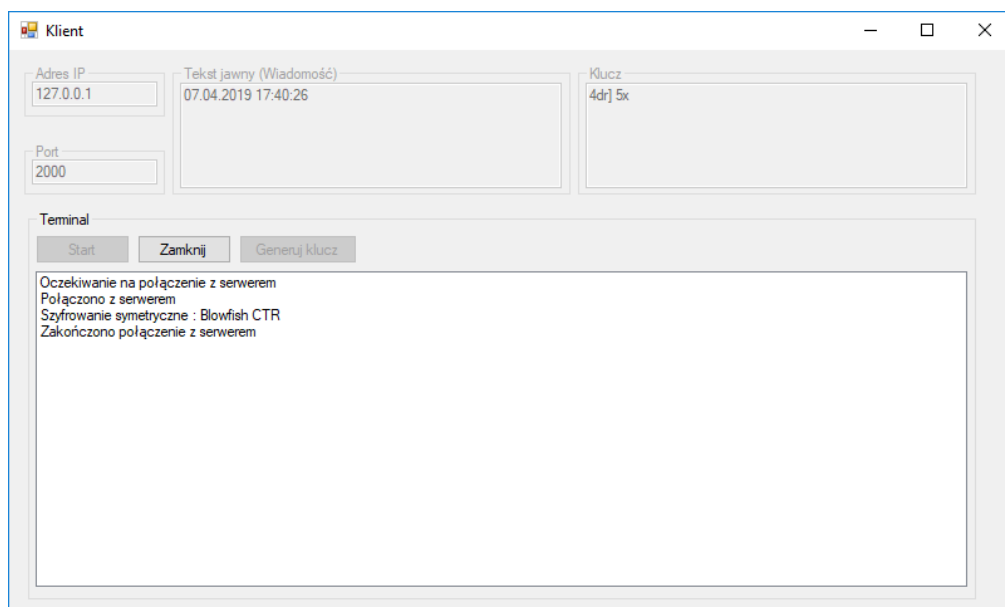


Rys. 6.28. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CBC [opracowanie własne]

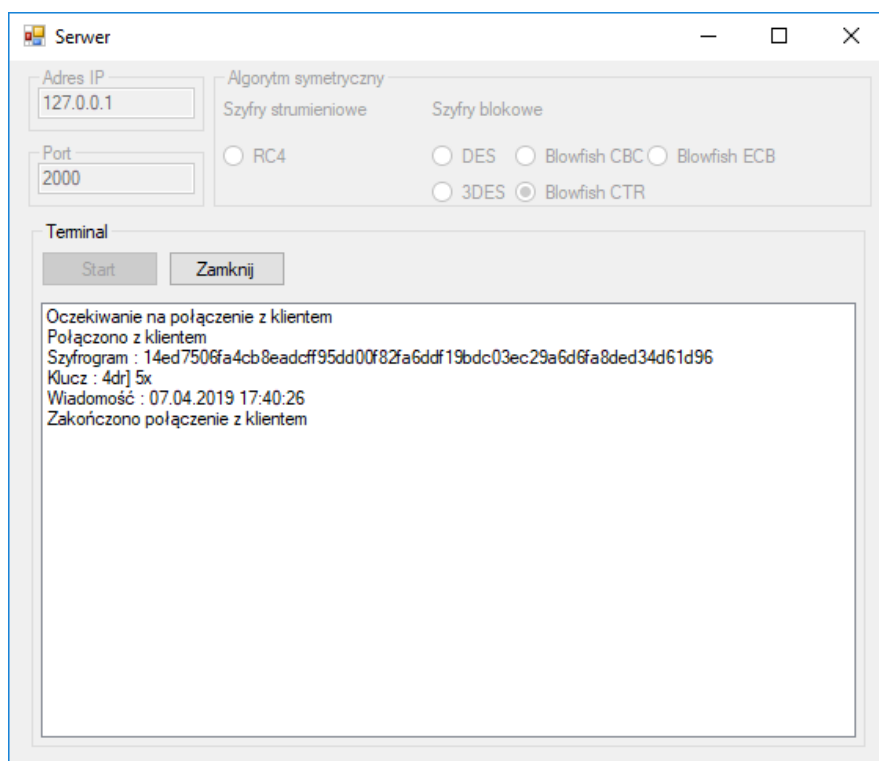


Rys. 6.29. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CBC [opracowanie własne]

Następnym trybem kodowania jest CTR, którego GUI dla klienta wygląda jak na Rys. 6.30, z kolei dla serwera opisuje się w sposób przedstawiony na Rys. 6.31.

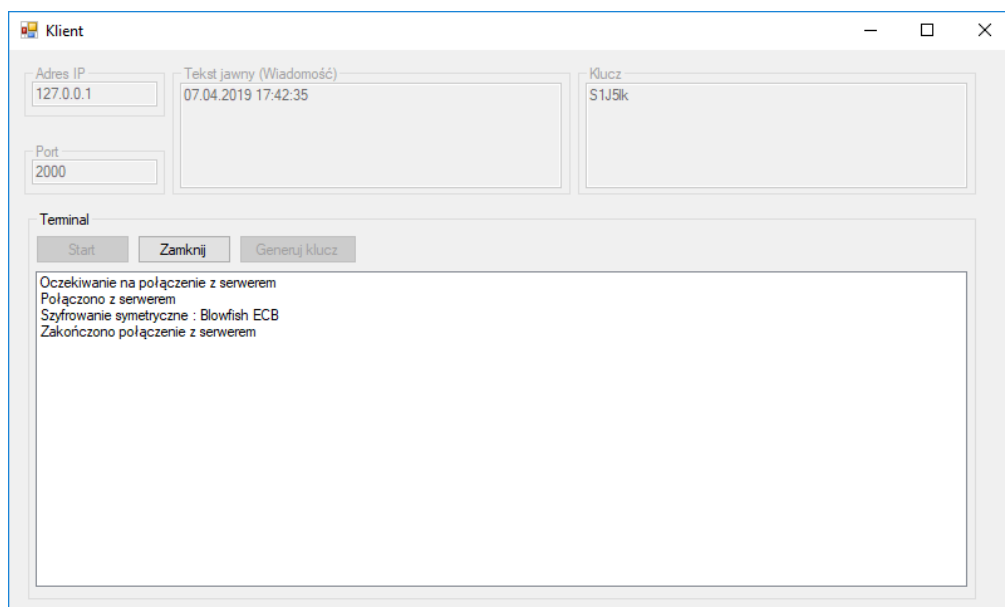


Rys. 6.30. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CTR [opracowanie własne]

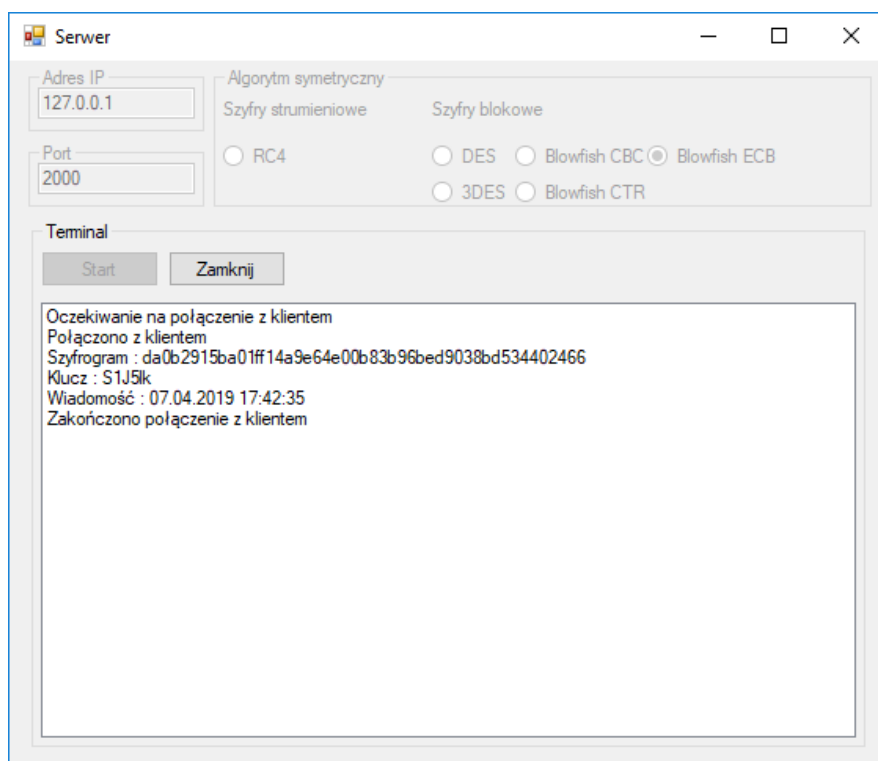


Rys. 6.31. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CTR [opracowanie własne]

Ostatnim trybem kodowania Blowfish jest ECB. Klient po wyborze tej opcji przedstawia się w sposób ujęty na Rys. 6.32, a serwer według Rys. 6.33.



Rys. 6.32. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania ECB [opracowanie własne]



Rys. 6.33. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania ECB [opracowanie własne]

#### 6.4.2. Testowanie na dwóch maszynach

Mając zamiar dokonać test oprogramowania na dwóch komputerach, należy sprawdzić, jaki adres IP posiada sprzęt, który będzie serwerem. W tym celu należy uruchomić wiersz poleceń (cmd). Niezbędne jest również połączenie z Internetem, w przeciwnym razie nie będzie możliwości przeprowadzenia doświadczenia, a co za tym idzie wyświetlenia danych za pomocą komendy ipconfig.

Po uzyskaniu informacji na temat numeru identyfikacyjnego sprzętu, odpowiadającego za rolę serwera (Rys. 6.35), można przystąpić do testu. Poniżej zamieszczone są zawartości wiersza poleceń w przypadku braku połączenia z siecią globalną (Rys. 6.34) i prawidłowym wyświetleniem parametru dzięki komendzie ipconfig. Nieistotne dane o urządzeniu, będącym serwerem, zostały ukryte.

```

C:\ Wiersz polecenia
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\User>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Połączenie lokalne* 11:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Połączenie lokalne* 12:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

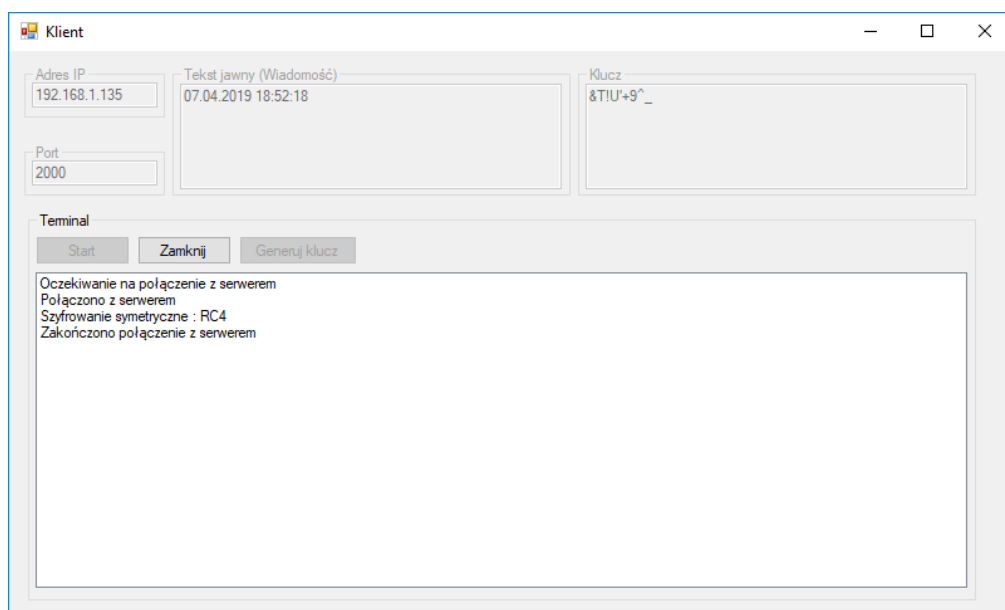
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

C:\Users\User>

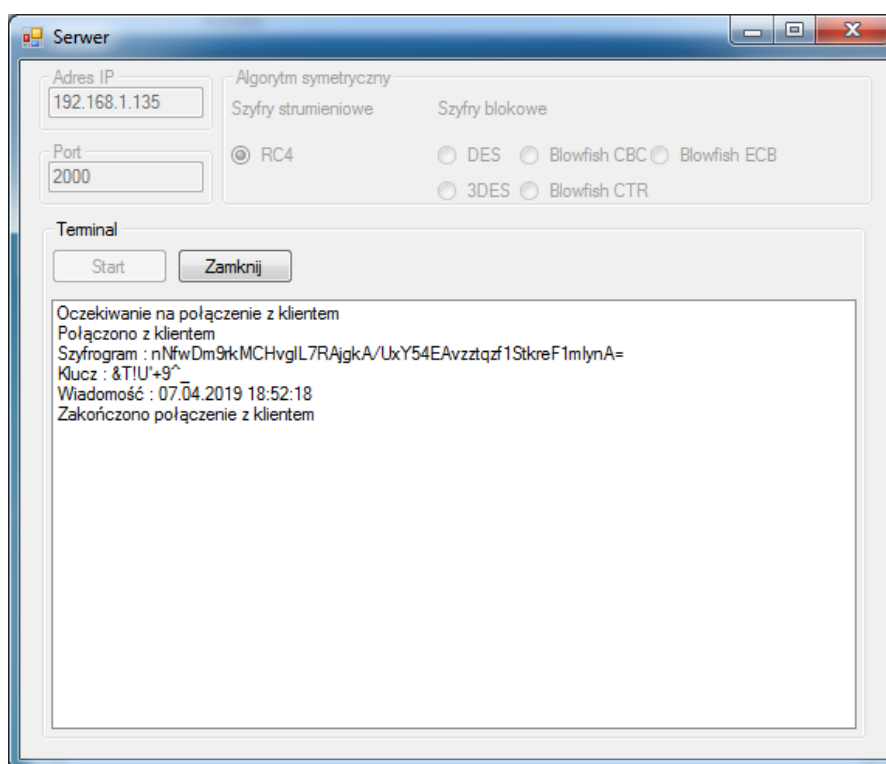
```



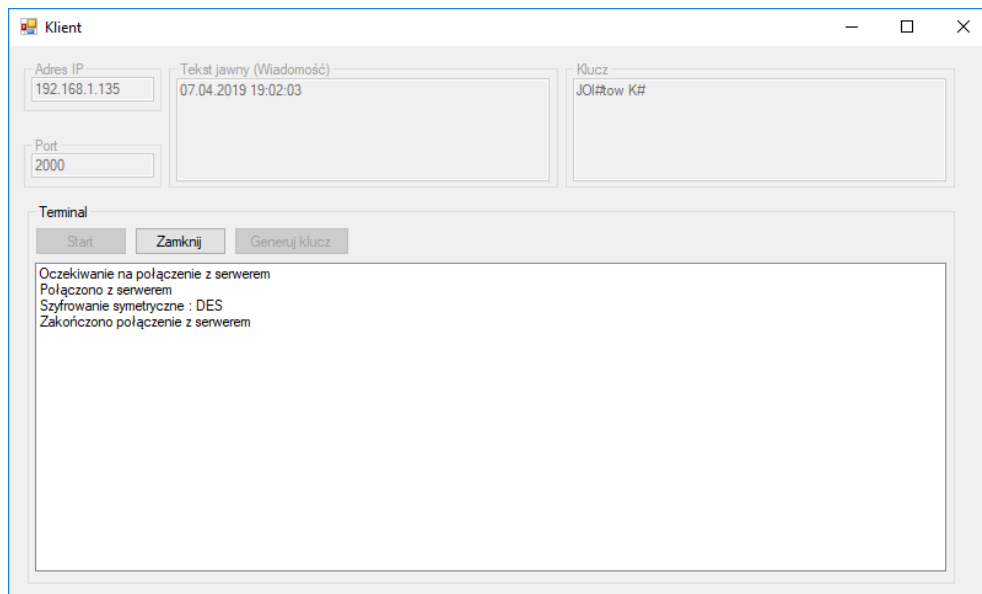
pętli zwrotnej metoda RC4, której efekty po stronie klient (Rys. 6.36) i serwer (Rys. 6.37) zostały pokazane niżej.



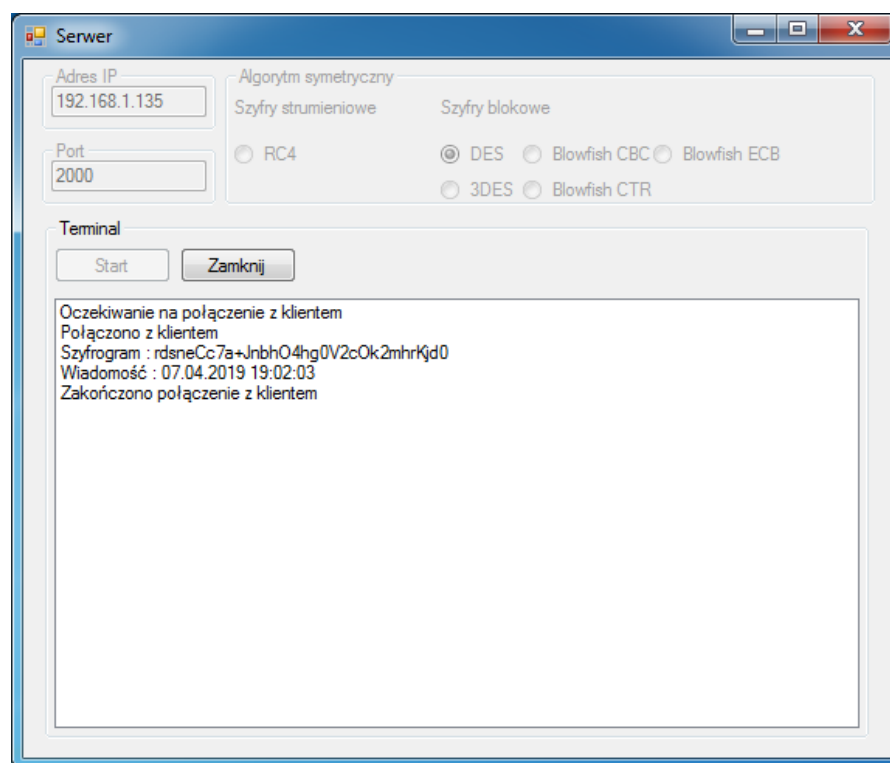
Rys. 6.36. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru strumieniowego RC4  
[opracowanie własne]



Rys. 6.37. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru strumieniowego RC4  
[opracowanie własne]



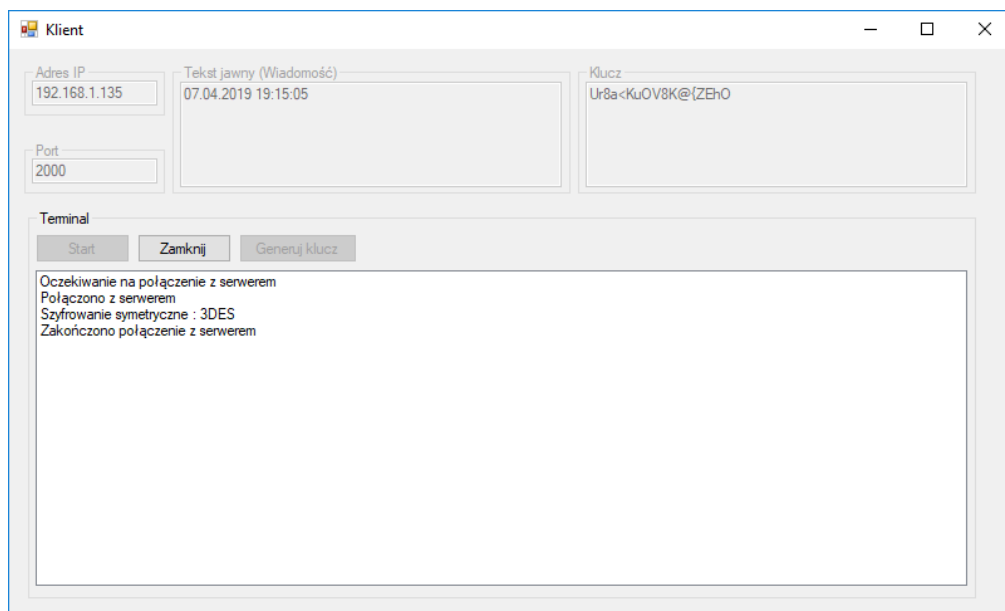
Rys. 6.38. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego DES  
[opracowanie własne]



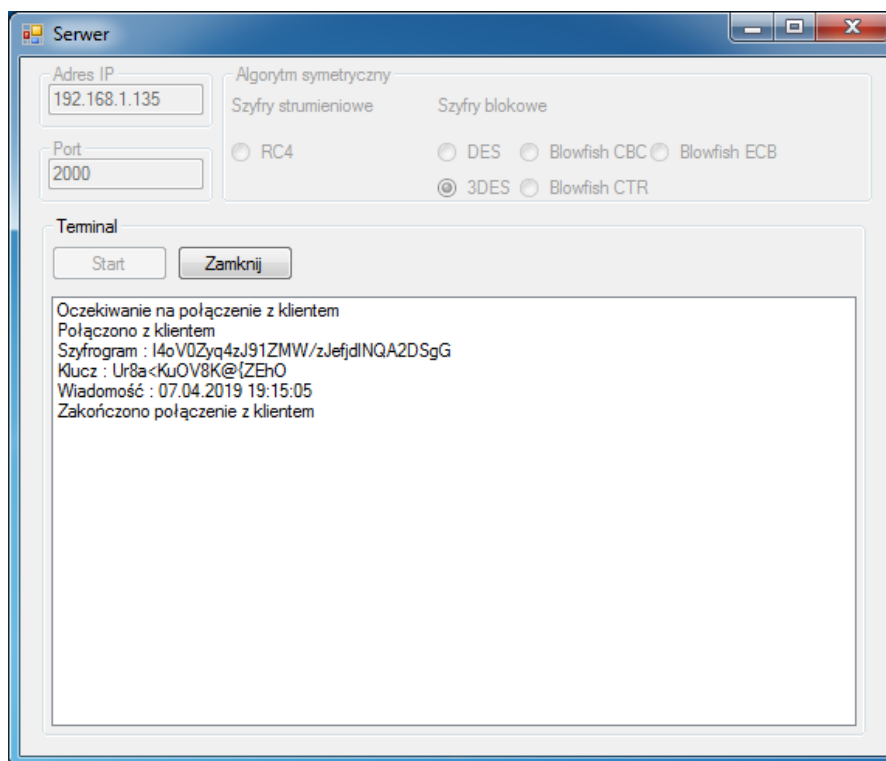
Rys. 6.39. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego DES  
[opracowanie własne]

U góry zostały przedstawione efekty końcowe pracy szyfru blokowego DES dla klienta (Rys. 6.38) i serwera (Rys. 6.39). Z kolei pod spodem na Rys. 6.40

zaprezentowano konsekwencje działania szyfru 3DES dla klienta, a następnie serwera (Rys. 6.41).

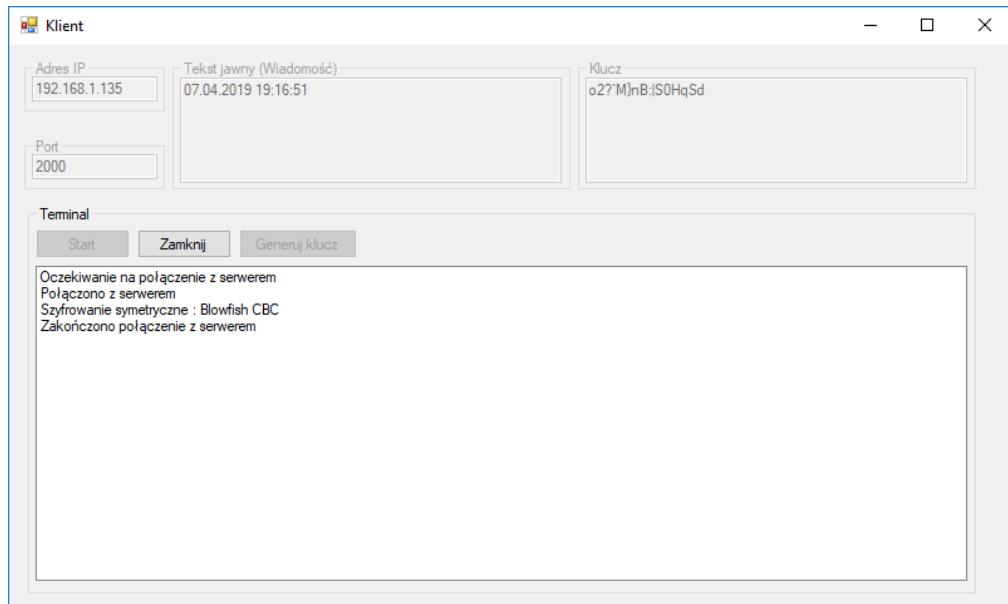


Rys. 6.40. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego 3DES  
[opracowanie własne]

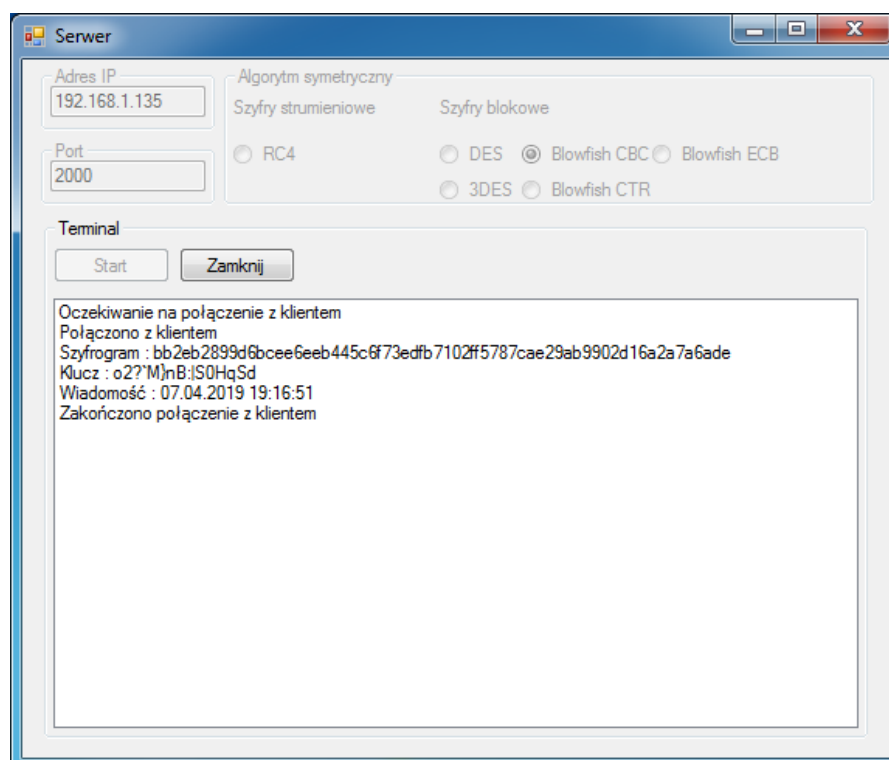


Rys. 6.41. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego 3DES  
[opracowanie własne]

Serię testów działania programów architektury klient-serwer na dwóch sprzętach zamyka jak w przypadku jednej maszyny szyfr Blowfish z zaimplementowanymi trzema trybami kodowania, a mianowicie są nimi CBC, CTR i ECB.

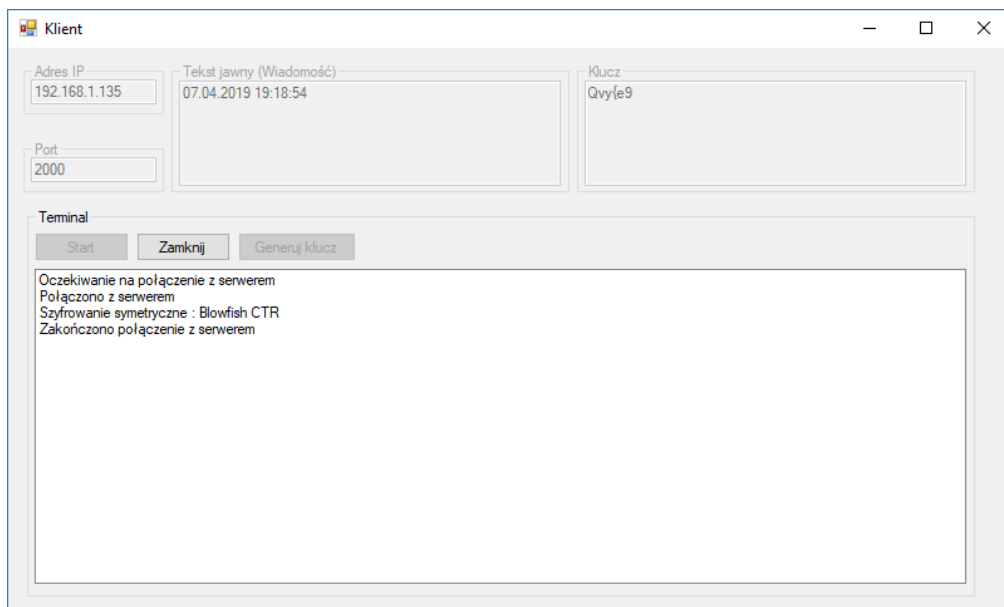


Rys. 6.42. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CBC [opracowanie własne]

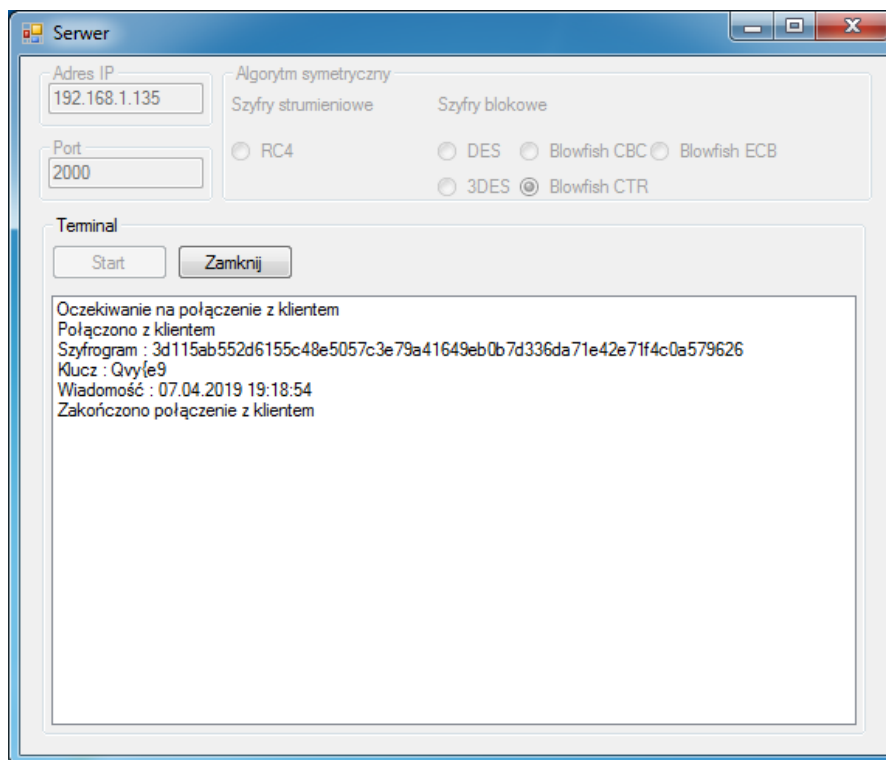


Rys. 6.43. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CBC [opracowanie własne]

Pierwszy z nich otwiera część testu dla metody Blowfish. Oprogramowanie klienta po jego selekcji wygląda w konsekwencji jak na Rys. 6.42. Natomiast aplikacja serwera przedstawia się w sposób ujęty na Rys. 6.43.

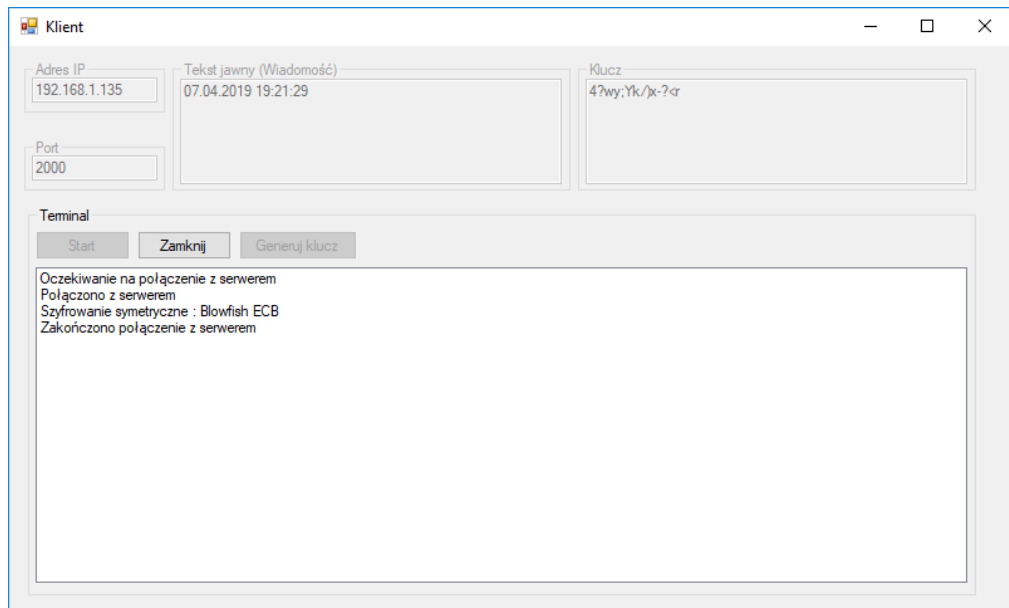


Rys. 6.44. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CTR [opracowanie własne]

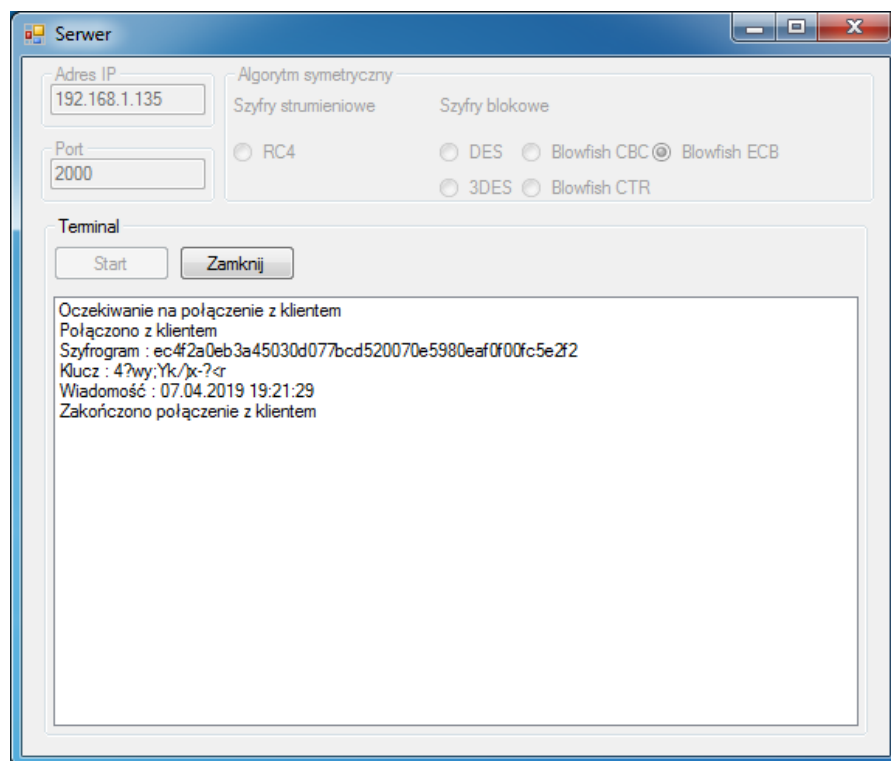


Rys. 6.45. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CTR [opracowanie własne]

Podobnie wygląda sytuacja dla trybu CTR, gdzie klient ukazany jest na Rys. 6.44, a serwer w postaci jaką pokazuje Rys. 6.45.



Rys. 6.46. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania ECB [opracowanie własne]



Rys. 6.47. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania ECB [opracowanie własne]

Kończący prezentację wyników badań tryb kodowania ECB daje efekty pracy dla klienta (Rys. 6.46) i serwera (Rys. 6.47) według powyższej prezentacji.

```
try
{
//próba wykonania instrukcji
}catch(Exception blad)
{
//kod wykonywany w przypadku wystąpienia wyjątku
}
```

Rys. 6.48. Instrukcja try ... catch, która zapobiega błędom w przypadku wciśnięcia przycisku Start u Klienta i Uruchom u Serwera [opracowanie własne]

Tak projekt klienta, jak i serwera nie są programami doskonałymi. Mimo to, starano się, by zapobiec niepożądanym reakcjom aplikacji na błędne użytkowanie, w maksymalnym zakresie. W celu zapobiegnięcia błędom, w przypadku braku poprawnych danych w odniesieniu do adresu IP, portu sieciowego, długości wiadomości i klucza zastosowano jedno z najbardziej adekwatnych rozwiązań, czyli blok try ... catch (Rys. 6.48).

## 7. Podsumowanie

Zrealizowana praca porusza kilka aspektów, którymi są komunikacja w architekturze klient-serwer, a także kwestia zabezpieczenia tegoż układu. Do przeprowadzenia implementacji tych oto problemów wykorzystano język programowania zorientowanego obiektowo Visual C#.

Z wykorzystaniem opracowania i zagłębienia się w literaturę, wszelkich dostępnych pomocy dydaktycznych stworzono strukturę klient-serwer w postaci dwóch projektów, w których jeden pełni rolę klienta, a drugi funkcję serwera. Stanowi to przystępne zobrazowanie działania relacji pomiędzy nadawcą i odbiorcą danych. Aplikacje są typu okienkowego, jest możliwość wyboru algorytmu szyfrującego dowolną wiadomość z odpowiednim kluczem, wybranym przez użytkownika lub wylosowanym przez program. Selekcja metody szyfrowania symetrycznego, który może być strumieniowy, bądź blokowy można dokonać po stronie serwera. W kwestii opisanego klucza i zdefiniowania wybranego tekstu jawnego leży po stronie klienta. Na serwerze jest opcja wyboru algorytmu szyfrującego informację dzięki przyciskom opcji, co jest niewątpliwie plusem ze względu na fakt, iż istnieje alternatywa dla tegoż rozwiązania w postaci edycji kodu źródłowego. Jednak ten sposób mógłby stanowić utrudnienie w użytkowaniu zwłaszcza dla osoby nie mającej znajomości nie tyle języka Visual C#, co samego programowania. Drugą mocną stroną jest wybór dowolnego adresu IP w przypadku doświadczenia przeprowadzanego na jednej maszynie, gdzie nie ma z góry ustalonej jednego adresu pętli zwrotnej, ponieważ są alternatywy z inną cyfrą na końcu ciągu znaków, opisującego tenże adres. Test na dwóch maszynach nie daje takiej możliwości, gdyż musi być uwzględniony adres IP serwera, sprawdzonego wcześniej w wierszu poleceń. Trzecią zaletę tworzy wybór dowolnego portu, który musi w trakcie przeprowadzania testu należeć do zbioru tych, które nie są używane w danym momencie. Czwartym atutem jest to, że użytkownik może wylosować lub wpisać wybrany przez siebie klucz. Piątym plusem jest to, że użytkownik może korzystać na swoim urządzeniu z aplikacji bez instalacji Visual Studio i dodatkowych bibliotek.

Stworzenie mechanizmu szyfrującego za pomocą wybranego algorytmu, a także skuteczna komunikacja między klientem a serwerem stanowiły największe wyzwanie w całej pracy.



Aplikacje klienta i serwera nie należą do grupy programów doskonałych ze względu na swoje niedoskonałości, które przedstawiono w rozdziale poświęconym opisowi aplikacji. Mogą pojawić się również słabe strony, które nie zostały wykryte podczas przeprowadzanych badań, jednak te najważniejsze luki zostały zabezpieczone przed niepożądanymi konsekwencjami. Projekt może zostać rozszerzony o dwa aspekty. Pierwszym jest szyfrowanie więcej niż jednej wiadomości podczas jednego uruchomienia architektury klient-serwer. Z kolei drugą jest używanie większej liczby metod niż jedna podczas pojedynczej sesji działania struktury.

Zaproponowane rozwiązania można zastosować w praktyce, nawiązując do przesyłania poufnych informacji od nadawcy do odbiorcy, zwiększeniu bezpieczeństwa komunikacji sieci komputerowej i wymiany danych w strukturze klient-serwer.

## Literatura

1. Bilski T., Pankowski T., Stokłosa J.: *Bezpieczeństwo danych w systemach informatycznych*. Wydawnictwo Naukowe PWN, Warszawa - Poznań 2001, ISBN 83-01-13452-6.
2. Boduch A.: *Wstęp do programowania w języku C#*. Wydawnictwo Helion, Gliwice 2006, ISBN 83-246-0523-1.
3. Chrzan M., Jackowski S.: *Współczesne systemy telekomunikacyjne*. Tom 2, Wydanie III, Wydawnictwo Politechniki Radomskiej, Radom 2008, ISBN 978-83-7351-274-0.
4. Ferguson N., Schneier B.: *Kryptografia w praktyce*. Wydawnictwo Helion, Gliwice 2004, ISBN 83-7361-211-4.
5. Grabek M., Orłowski S.: *C# Tworzenie aplikacji sieciowych. Gotowe projekty*. Wydawnictwo Helion, Gliwice 2012, ISBN 978-83-246-2910-7.
6. Karbowski M.: *Podstawy kryptografii*. Wydanie III, Wydawnictwo Helion, Gliwice 2006, ISBN 978-83-246-6975-2/ISBN 83-7361-933-X.
7. Lis M.: *C#. Praktyczny kurs*. Wydanie III, Wydawnictwo Helion, Gliwice 2016, ISBN 978-83-283-1456-6.
8. *Programowalne układy przetwarzania sygnałów i informacji*, pod red. Prof. Tadeusza Łuby, Wydanie I, Wydawnictwo Komunikacji i Łączności, Warszawa 2008, ISBN 978-83-206-1711-5.
9. Stinson D. R.: *Kryptografia w teorii i praktyce*. Wydawnictwo Naukowo – Techniczne, Warszawa 2005, ISBN 83-204-2982-X.
10. Sutton R.J.: *Bezpieczeństwo telekomunikacji. Praktyka i zarządzanie*. Wydawnictwo Komunikacji i Łączności, Warszawa 2006, ISBN 83-206-1517-8.
11. Yan S. Y.: *Teoria liczb w informatyce*. Wydawnictwo Naukowe PWN, Warszawa 2006, ISBN 978-83-01-14905-5.
12. <http://www.crypto-it.net/pl/symetryczne/rc4.html?tab=0> [dostęp 2019-04-11.]
13. <http://www.as.up.krakow.pl/~greg/zajecia/sieci/Sieci%20cw%20RC4.pdf> [dostęp 2019-04-11.]

## Spis rysunków

- Rysunek 3.1. Treść prostego programu sporządzonego w systemowym Notatniku [2]
- Rysunek 3.2. Kompilacja i uruchomienie programu w Wierszu poleceń [2]
- Rysunek 4.1. Wygląd działającego projektu aplikacji TCP Klient [5]
- Rysunek 4.2. Kod klienta TCP [5]
- Rysunek 4.3. Wygląd działającego projektu aplikacji TCP Serwer [5]
- Rysunek 4.4. Kod serwera TCP [5]
- Rysunek 4.5. Wygląd działającego projektu aplikacji UDP Klient [5]
- Rysunek 4.6. Kod przycisku Wytnij w programie UDP Klient [5]
- Rysunek 4.7. Wygląd działającego projektu aplikacji UDP Serwer [5]
- Rysunek 4.8. Kod przycisku Uruchom w programie UDP Serwer [5]
- Rysunek 5.1. Tworzenie permutacji S [13]
- Rysunek 5.2. Szyfrowanie wiadomości w RC4 [13]
- Rysunek 5.3. Ogólna struktura szyfratora blokowego [3]
- Rysunek 5.4. Wzór operacji XOR [6]
- Rysunek 5.5. Szyfrowanie w trybie CBC [6]
- Rysunek 5.6. Sposób matematyczny obrazu szyfrowania pojedynczego bloku danych [6]
- Rysunek 5.7. Schemat blokowy szyfru Blowfish [1]
- Rysunek 5.8. Funkcja F [1]
- Rysunek 5.9. Schemat standardu (algorytmu) szyfrowania danych DES [11]
- Rysunek 5.10. Pojedyncza runda szyfrowania w systemie DES [9]
- Rysunek 5.11. Funkcja f systemu DES [9]
- Rysunek 5.12. Szyfrowanie metodą 3DES [6]
- Rysunek 5.13. Przebieg szyfrowania TDES [11]
- Rysunek 5.14. Deszyfrowanie metodą TDES [11]
- Rysunek 5.15. Trzykrotny algorytm DES [3]
- Rysunek 6.1. Okienko Wiersza poleceń z wyświetlonymi parametrami dla komendy ipconfig [opracowanie własne]
- Rysunek 6.2. Funkcja generująca klucz jako alternatywa dla wpisywania klucza przez użytkownika (obiekt generator typu Random zadeklarowany globalnie) [opracowanie własne]

Rysunek 6.3. Konstruktry klient (u góry) i serwera (na dole) przypisujące domyślne wartości do odpowiednich komponentów i wywołanie funkcji generowania klucza [opracowanie własne]

Rysunek 6.4. Aplikacja Klient po uruchomieniu [opracowanie własne]

Rysunek 6.5. Aplikacja Serwer po uruchomieniu [opracowanie własne]

Rysunek 6.6. Wybór opcji algorytmu szyfrowania po stronie serwera [opracowanie własne]

Rysunek 6.7. Aplikacja Klient w sytuacji innego adresu IP od tego, który ma aplikacja Serwera [opracowanie własne]

Rysunek 6.8. Aplikacja Serwer w sytuacji innego adresu IP od tego, który ma aplikacja Klienta [opracowanie własne]

Rysunek 6.9. Aplikacja Klient w sytuacji innego portu sieciowego w stosunku do tego wprowadzonego w aplikacji Serwera [opracowanie własne]

Rysunek 6.10. Aplikacja Serwer w sytuacji innego portu sieciowego w stosunku do tego wprowadzonego w aplikacji Klienta [opracowanie własne]

Rysunek 6.11. Aplikacja Klient w sytuacji innego adresu IP i portu sieciowego w stosunku do tych wprowadzonych w aplikacji Serwera [opracowanie własne]

Rysunek 6.12. Aplikacja Serwer w sytuacji innego adresu IP i portu sieciowego w stosunku do tych wprowadzonych w aplikacji Klienta [opracowanie własne]

Rysunek 6.13. Aplikacja Klienta w sytuacji braku wprowadzonego klucza [opracowanie własne]

Rysunek 6.14. Aplikacja Serwera w sytuacji braku wprowadzonego klucza [opracowanie własne]

Rysunek 6.15. Aplikacja Klienta w sytuacji braku wiadomości [opracowanie własne]

Rysunek 6.16. Aplikacja Serwera w sytuacji braku wiadomości [opracowanie własne]

Rysunek 6.17. Aplikacja Klienta w sytuacji braku wiadomości i klucza [opracowanie własne]

Rysunek 6.18. Aplikacja Serwera w sytuacji braku wiadomości i klucza [opracowanie własne]

Rysunek 6.19. Zasadniczy kod odpowiadający za permutację S i szyfrowanie metodą RC4 [opracowanie własne]

Rysunek 6.20. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru strumieniowego RC4 [opracowanie własne]

Rysunek 6.21. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru strumieniowego RC4 [opracowanie własne]

Rysunek 6.22. Fragment kodu źródłowego odpowiedzialnego za szyfrowanie metodą DES [opracowanie własne]

Rysunek 6.23. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego DES [opracowanie własne]

Rysunek 6.24. Fragment kodu źródłowego odpowiedzialnego za szyfrowanie metodą DES [opracowanie własne]

Rysunek 6.25. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego DES [opracowanie własne]

Rysunek 6.26. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego 3DES [opracowanie własne]

Rysunek 6.27. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego 3DES [opracowanie własne]

Rysunek 6.28. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CBC [opracowanie własne]

Rysunek 6.29. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CBC [opracowanie własne]

Rysunek 6.30. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CTR [opracowanie własne]

Rysunek 6.31. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CTR [opracowanie własne]

Rysunek 6.32. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania ECB [opracowanie własne]

Rysunek 6.33. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania ECB [opracowanie własne]

Rysunek 6.34. Wiersz polecenia w momencie braku połączenia z siecią globalną [opracowanie własne]

Rysunek 6.35. Wiersz poleceń po stronie Serwera, z którego należy odczytać adres IPv4 [opracowanie własne]

Rysunek 6.36. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru strumieniowego RC4 [opracowanie własne]

Rysunek 6.37. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru strumieniowego RC4 [opracowanie własne]

Rysunek 6.38. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego DES [opracowanie własne]

Rysunek 6.39. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego DES [opracowanie własne]

Rysunek 6.40. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego 3DES [opracowanie własne]

Rysunek 6.41. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego 3DES [opracowanie własne]

Rysunek 6.42. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CBC [opracowanie własne]

Rysunek 6.43. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CBC [opracowanie własne]

Rysunek 6.44. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CTR [opracowanie własne]

Rysunek 6.45. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania CTR [opracowanie własne]

Rysunek 6.46. Aplikacja Klienta wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania ECB [opracowanie własne]

Rysunek 6.47. Aplikacja Serwera wykorzystującego szyfrowanie symetryczne szyfru blokowego Blowfish w trybie kodowania ECB [opracowanie własne]

Rysunek 6.48. Instrukcja try ... catch, która zapobiega błędom w przypadku wciśnięcia przycisku Start u Klienta i Uruchom u Serwera [opracowanie własne]

## Spis tabel

Tabela 4.1. Modele warstwowe sieci [5]

Tabela 4.2. Model DoD i popularne protokoły sieciowe [5]

Tabela 4.3. Lista najpopularniejszych portów i usług do nich przypisanych [5]

Tabela 5.1. Macierz permutacji początkowej IP [6]

Tabela 5.2. Macierz permutacji klucza [6]

Tabela 5.3. Liczba przesunięć przypadająca na każdy z cykli [6]

Tabela 5.4. Macierz kompresji [6]

Tabela 5.5. Macierz alokacji permutacji z rozszerzeniem [9]

Tabela 5.6. Struktura S-bloków [6]

Tabela 5.7. P-blok [9]

Tabela 5.8. Macierz permutacji końcowej  $IP^{-1}$  [9]

## **Spis załączników**

### **Dodatek A Spis zawartości dołączonej płyty CD**

1. Praca magisterska – niniejszy dokument
2. Kod źródłowy aplikacji klienta i serwera