Instytut Podstawowych Problemów Techniki PAN

Piotr Kotlarz

"Sieci neuronowe we wspomaganiu rozwiązywania problemów kryptologii"

Rozprawa doktorska wykonana
pod kierunkiem
doc. dr. hab. Zbigniewa Kotulskiego

Warszawa 2008

Spis Treści

Rozdział 1.

1.1. Wstęp

W historii ludzkości wiele metod i narzędzi stosowano jeszcze przed ich formalnym zdefiniowaniem. Nie ominęło to również kryptografii. Większość starożytnych cywilizacji rozwinęło na swoje potrzeby kryptografię. Osobliwym wyjątkiem są tutaj starożytne Chiny, w których nie były szeroko rozwijane narzędzia kryptograficzne. Powodów, dla których nie rozwijano tak szeroko jak w innych cywilizacjach metod utajniania informacji można doszukiwać się w specyfice języka pisanego składającego się z skomplikowanego układu ideogramów. Po zapisaniu w tym języku informacji krąg ludzi, którzy byliby w stanie taką wiadomość odczytać był mocno ograniczony. Egipcjanie stosowali tajne pismo do zapisywania informacji na nagrobkach, natomiast w starożytnych Indiach powstało wiele do dziś nie zidentyfikowanych sposobów zapisu informacji, które służyły ówczesnej kryptologii. Pełny opis historii kryptografii i kryptologii można znaleźć w książce Dawida Kahna „Łamacze kodów" [1]. W historii kryptografii powstało wiele metod zapisu informacji w sposób niejawny, równolegle rozwijały się metody łamania szyfrów. Poza samym bezpieczeństwem szyfru ważna była również łatwość jego stosowania. Z czasem zaczynały powstawać pierwsze urządzenia szyfrujące, jak chociażby dysk szyfrowy Leona Battisty Albertiego, czy najsławniejsze niemiecka: Enigma i japońska Purple.

Rozpowszechnienie wiedzy na temat kryptologii zawsze rodziło wiele kontrowersji. W początkowej fazie rozwoju nauki o szyfrach najważniejszą sprawą było zachowanie tajności stosowanych rozwiązań. Podczas pierwszej wojny światowej nastąpił jednak znaczny postęp w badaniach kryptologicznych. Zaczęły się pojawiać pierwsze publikacje, najważniejszą z nich była monografia z 1918 roku autorstwa Williama F. Friedmana „The Index of Coincidence and its Applications in Cryptography" [2]. W tym samym czasie został zgłoszony pierwszy patent w USA urządzenia szyfrującego, którego autorem był Edward H. Herben. Patent ten dotyczył maszyny rotorowej; koncepcja ta stała się fundamentalną dla kryptografii militarnej i dyplomatycznej na następne półwiecze. W okresie międzywojennym i podczas samej II wojny światowej kryptografia była domeną wojska. Z okresem tym wiąże się najpowszechniej znane wydarzenie ze świata kryptologii, czyli złamanie w 1933 roku szyfru Enigma przez trójkę polskich matematyków: Mariana Rejewskiego, Jerzego Różyckiego i Henryka Zygalskiego. W dalszej fazie prac nad Enigmą brał w nich udział, między innymi, również Alan Turing. We wspomnianym okresie sporadycznie pojawiały się publikacje dotyczące metod utajniania informacji, jednak nie odzwierciedlały one

faktycznego stanu wiedzy i dokonań naukowców. Pierwszą pracą, jaka po wojnie została upubliczniona, był zbiór artykułów Claude'a Shannona „The Communication Theory of Secrecy Systems"[3]. Publikacja ukazała się w 1949 roku. Okres lat pięćdziesiątych oraz sześćdziesiątych nie wniósł zbyt wiele do literatury przedmiotu, co nie oznacza, że zaprzestano badań, a jedynie, że ich wyniki było objęte ścisłą tajemnicą. W latach sześćdziesiątych firma IBM zainicjowała program badawczy w dziedzinie kryptografii komputerowej pod nazwą „Lucyfer", którego kierownikiem został Horst Feistel. W dalszej fazie projektu kierownictwo objął Walt Tuchman. Efektem tego programu stał się algorytm szyfrujący pod taką samą nazwą jak nazwa wspomnianego programu badawczego. Feistel [4] do zaprojektowania Lucyfera wykorzystał koncepcję sieci S-P, o której mowa w dalszych rozdziałach, oraz permutacje i podstawienia. Do początku lat siedemdziesiątych nie istniały żadne ogólnodostępne publikacje na temat nowoczesnych metod szyfrowania. W USA istniało kilka firm, które prowadziły badania na własną rękę. Produkowały i sprzedawały różnego rodzaju urządzenia szyfrujące. Kupujący je jednak nie byli w stanie stwierdzić, czy rzeczywiście są one bezpieczne. Nie istniała instytucja, która byłaby w stanie określić i ogłosić poziom zapewnianego bezpieczeństwa przez określone urządzenie kryptograficzne. 15 maja 1972 roku ogłoszony został pierwszy konkurs na standard szyfrowania danych. Wymagania konkursowe oraz inne szczegóły historyczne znaleźć można na stronach internetowych [5] agencji rządowej USA. Jak możemy dowiedzieć się z lektury raportów zamieszczonych na wcześniej wspomnianych stronach www, zgłaszanych było wiele algorytmów, jednak żaden z nich nie spełniał stawianych wymagań. Dopiero w połowie roku 1974 ośrodek badawczy firmy IBM zgłosił algorytm oparty na wspomnianym wcześniej algorytmie Lucyfer. Po wielu dyskusjach na forum świata naukowego NSA (National Security Agency) ogłosiło 23 listopada 1976 roku algorytm DES jako standard, co zostało opublikowane w [6]. Był to standard wyjątkowy, NSA przypadkowo opublikowało algorytm uważając DES-a za urządzenie, co zaowocowało tym, że każdy mógł napisać program, który będzie realizował szyfr uznany przez agencję rządu USA za bezpieczny. Algorytm ten był powszechnie stosowanym standardem, więc stał się obiektem intensywnych badań. DES doczekał się kilku następców, których konstrukcja oparta była również na permutacjach Feistela oraz S-blokach. Powstały więc między innymi szyfry: FEAL (szybki algorytm szyfrujący) [7] i IDEA [8]. O ile ogłoszenie standardu DES stało się przełomem w zakresie dostępności technik szyfrujących, o tyle pracę z 1976 roku Diffie i Hellman [9] można nazwać rewolucyjną. Opublikowali oni artykuł [9] o jednokierunkowych funkcjach z zapadką. Autorzy zaproponowali protokół zdalnego uzgadniania klucza, tym samym

udowodnili, że osoby chcące wymieniać szyfrowaną korespondencję wcale nie muszą się spotykać w celu bezpiecznego uzgodnienia klucza. W 1978 roku opublikowano pracę, która przedstawiała kryptosystem z kluczem publicznym o nazwie RSA [10]. Bezpieczeństwo tego algorytmu opiera się na trudności rozkładu dużych liczb na czynniki pierwsze. Kryptografia asymetryczna nie spowodowała wyparcia metod symetrycznych. Obecnie obie te metody doskonale się uzupełniają w wielu protokołach kryptograficznych.

Koniec wieku XX oraz początek XXI to czas, kiedy kryptologia stała się dostępną i powszechną dyscypliną naukową. Powstaje wiele publikacji na temat kryptografii, w Polsce oraz na świecie organizowane są konferencję naukowe tematycznie związane z bezpieczeństwem informacji. Ostatnich kilka lat to okres intensywnych badań prowadzonych na całym świecie w zakresie kryptologii. Na lata 1999/2000 przypadło rozstrzygnięcie kolejnego konkursu na nowy standard szyfrowania, następcę DES-a. W finale konkursu znalazły się szyfry: Mars, RC6, Twofish, Rijndael oraz Serpent. Algorytmy te poddane zostały ocenie pod względem odporności na ataki kryptograficzne, łatwości implementacji programowej jak i sprzętowej oraz wydajności. Na trzeciej konferencji AES w kwietniu 2000 roku największą przychylność zyskał algorytm Rijndael. 2 października 2000 roku rozstrzygnięty został przez Narodowy Instytutu Standardów i Technologii USA konkurs AES (Advance Encryption Standard). Zwycięzcą został algorytm, o nazwie Rijndael, stworzony przez belgijski zespół. Ważne wydarzenia w zakresie kryptologii to nie tylko powstanie nowych szyfrów, ale również osiągnięcia w dziedzinie ataków na szyfry. Firma RSA Data Security prowadzi nieustający konkurs na faktoryzację dużych liczb. Skuteczna i wydajna faktoryzacja jest zagrożeniem dla kryptografii asymetrycznej. Obecnie wyzwaniem są liczby rzędu 600 do 2048 bitów. Lata 2002 i 2003 przyniosły publikacje na temat nowego podejścia do problemu ataku na RSA poprzez przyspieszenie faktoryzacji. W roku 2001 profesor Daniel Bernstein opublikował [11] koncepcję Maszyny Bernsteina. W 2003 na "International Workshop on Practice and Theory in Public Key Cryptography" opublikowany [12] został projekt urządzenia, które jest w stanie przespieszyć faktoryzację. W roku 2002 (konferencja Asiacrypt, Eurocrypt 2003) pojawiło się również kilka prac na temat ataku na AES. Rok 2005 przyniósł natomiast doniesienia o fakcie złamania kilku popularnych funkcji skrótu.

Kilka (wybranych) wymienionych powyżej wydarzeń w historii współczesnej kryptografii pokazuje, jak aktywną i dostępną dziedziną nauki stała się kryptografia. Można to także określić mianem przełomu na miarę publikacji standardu DES-a, czy nawet opracowania koncepcji szyfrowania asymetrycznego. W stosunku do pierwszej połowy wieku

XX obecny stan wiedzy z zakresu bezpieczeństwa informacji, a co ważniejsze, jej dostępność, zdecydowanie się podwyższył. Można zaryzykować stwierdzenie, że gdyby wiedza na temat narzędzi kryptograficznych oraz ich dostępność nie była na obecnym poziomie, to dzisiejszy tak zwany wolny świat nie wyglądałby tak jak dzisiaj. Choć z drugiej strony kryptografia niesie ze sobą ciągle jeszcze sporo kontrowersji. Ponadto, poza badaniami „cywilnymi" z całą pewnością prowadzone są prace w zakresie kryptografii „militarnej", których wyniki nie muszą być wcale publikowane. Wobec tego nadal z całą pewnością nie wszystkie wyniki są ogólnie dostępne. Kryptografia to nie tylko dyscyplina naukowe, ale również narzędzie w rękach służb wywiadu, dyplomacji i biznesu.

1.2. Cel i zakres pracy

W powyższym rozdziale przedstawiono, w dużym skrócie, historię kryptografii. Zostało tam wspomniane, że już starożytne cywilizacje stosowały techniki tajnego przekazywania informacji. Ludzi tych cywilizacji cechował, jak na czasy, w których żyli, wysoki stopień rozwoju społecznego i technologicznego. Przez wiele wieków przywilej korzystania z narzędzi kryptograficznych należał jedynie do osób wykształconych. Z czasem oczywiście następowało upowszechnienie technik ochrony informacji, a zdecydowany rozwój kryptografii powszechnej nastąpił w erze komputerów PC.
W czasach obecnych oczywiście trudno wyobrazić sobie zastosowania kryptografii bez wykorzystania techniki cyfrowej. Biorąc pod uwagę fakt, że coraz powszechniej informacja jest przetwarzana, przechowywana i przesyłana w postaci elektronicznej daje się zaobserwować nieustający proces powstawania nowych algorytmów i protokołów kryptograficznych. Implementacja tych rozwiązań odbywa się poprzez implementację konkretnych metod kryptograficznych w różnego rodzaju środowiskach programistycznych lub z wykorzystaniem układów programowalnych dla rozwiązań typowo sprzętowych. Tak więc dzisiaj najpowszechniej stosowaną metodą realizacji algorytmów szyfrujących, funkcji skrótu i podpisów elektronicznych jest programowanie. W tej pracy zaproponowane zostanie podejście alternatywne. Określony algorytm kryptograficzny ma być realizowany poprzez uniwersalny układ, który nie będzie programowany, a uczony nowego algorytmu szyfrującego, który ma realizować.

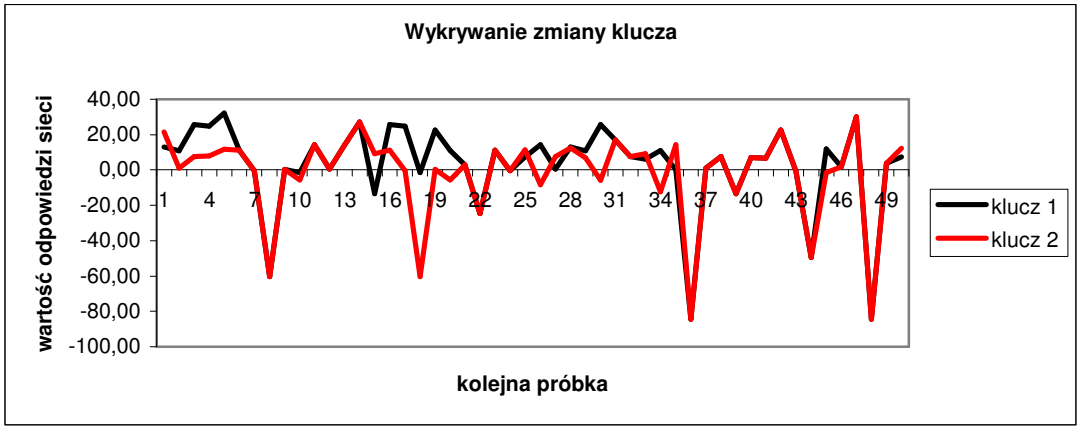
Celem tej pracy jest zaproponowanie wykorzystania narzędzi sztucznej inteligencji, jakim są sieci neuronowe, jako elementu wspomagającego i wzbogacającego metody kryptologii. W ostatnim czasie metody sztucznej inteligencji znajdują zastosowanie w różnych naukach technicznych. Coraz częściej pojawiają się doniesienia o prowadzonych badaniach w zakresie zastosowania wybranych metod sztucznej inteligencji w bezpieczeństwie informatycznym.

Sztuczna inteligencja (Artificial Intelligence, AI) jako nauka zaczęła rozwijać się w latach 50 dwudziestego wieku. Na początku lat pięćdziesiątych powstały pierwsze ośrodki zajmujące się sztuczną inteligencją. Allen Newell i Herbert Simon założyli laboratorium Artificial Intelligence na Uniwersytecie Carnegie Mellon, jako drugie powstało laboratorium w Massachusetts Institute of Technology założone przez Johna McCarthy'ego. Według niego sztuczna inteligencja to:

"konstruowanie maszyn, o których działaniu dałoby się powiedzieć, że są podobne do ludzkich przejawów inteligencji".

Tak więc badaczom sztucznej inteligencji zależy na tym, aby skonstruować maszynę, która radziłaby sobie przynajmniej tak dobrze jak człowiek z rozwiązywaniem zadań polegających na kojarzeniu, przewidywaniu, uogólnianiu i wnioskowaniu. Wszystkie te zadania, które są trudne do rozwiązania metodami maszynowymi są trudno algorytminizowalne.

Kierując się tym, co zostało wyżej powiedziane, widzimy, że punktem wyjścia do rozważań możliwości wykorzystania sztucznej inteligencji w kryptografii, mogą być dwa kierunki poszukiwań. Pierwszym mogłaby być kryptoanaliza, w której narzędzia sztucznej inteligencji mogłyby pomagać w poszukiwaniu różnego rodzaju cech w kryptogramach. Cechy te służyłyby jako informacja pomocnicza ułatwiająca kryptoanalizę. Prostym przykładem tego może być próba wykorzystania sieci neuronowej składającej się z dwóch warstw, na które składa się 9 neuronów, do stwierdzenia faktu możliwej zmiany klucza na podstawie przechwytywanych szyfrogramów. Wykonany został taki eksperyment z bardzo prostym szyfrem Playfair [13]. Polegał on na uczeniu sieci neuronowej parami tekst jawny-szyfrogram, który generowany był poprzez wykorzystanie szyfru Playfair. Po zakończeniu procesu uczenia na wejście sieci podane zostały dwa różne ciągi szyfrogramów wygenerowane za pomocą dwóch różnych kluczy. Po zmianie klucza dało się zaobserwować różnice w działaniu sieci, co pokazuje rysunek 1.1.



Rys 1.1 Wykrywanie zmiany klucza w szyfrogramie.

Powyższy przykład oczywiście bazuje na szyfrowaniu z wykorzystaniem prostego algorytmu. Ten kierunek wykorzystania sieci neuronowych w kryptografii nie jest przedmiotem badań opisanych w tej pracy. Z całą pewnością celowe byłoby rozpoznanie potencjalnych

możliwości wykorzystania tego rodzaju podejścia do wspomagania ataków na szyfry współczesne.

Tego rodzaju podejście, czyli postępowanie według zasady: uczymy sieć neuronową w określonych warunkach, następnie obserwujemy jej działania i ewentualne anomalie pojawiające się podczas jej pracy, wykorzystuje się w systemach wykrywania intruzów (IDS) [14], [15], [16]. Obecnie często w tego rodzaju systemach wykorzystuje się sieci neuronowe, których zadaniem jest wspomaganie wykrywania anomalii w zachowaniu użytkowników sieci.

Praca ta to próba wykorzystania sieci neuronowych do realizacji algorytmów szyfrujących. Obecnie model programu lub implementacji sprzętowej jest następujący. Po wyborze algorytmu, jakiego chcemy używać do szyfrowania, przystępujemy do jego implementacji, po czym następuje kompilacja i powstaje gotowe narzędzie pozwalające zamieniać tekst jawny na tajny. W przypadku rozwiązania sprzętowego przez dane urządzenie realizowany jest jeden wybrany algorytm, lub przy zastosowaniu układów programowalnych uzyskuje się uniwersalny moduł kryptograficzny, który może być programowany. Takie podejście do realizacji algorytmów szyfrujących jest obecnie powszechne. Zamiarem tej pracy jest zaproponowania rozwiązanie alternatywnego.

W pracy tej chcemy dokonać przeglądu narzędzi sztucznej inteligencji, oraz zaproponować potencjalne możliwości zastosowania ich w kryptologii. Algorytmy i protokoły kryptograficzne kojarzą się z dużą dokładnością działania oraz precyzją uzyskiwanych wyników. Przykładem są tutaj chociażby algorytmy szyfrujące, oraz odszyfrowujące. Określić je można odpowiednio jako funkcję szyfrującą F oraz odszyfrowującą D. Argumentem funkcji F jest tekst jawny M, a funkcji D szyfrogram C. Nawiązując więc do tej dokładności i precyzji: $F(M) = D(F(M)) = D(C)$. Z tego wynika, że wartości funkcji F i D dla danych argumentów muszą być dokładne i jednoznaczne. Zastosowania sztucznej inteligencji kojarzą się natomiast raczej z rozwiązaniami przybliżonymi.

Głównym celem pracy jest pokazanie alternatywy dla obecnie powszechnego podejścia do projektowania i implementacji algorytmów szyfrujących. Tą alternatywą mają być układy uczące się. Istotą pracy jest udowodnienie, że pisanie programu komputerowego, lub programowanie układu programowalnego to nie jedyny bezpieczny sposób realizacji szyfrów. Zamiast programowania wykorzystany ma zostać proces uczenia. Poza samą propozycją realizacji wybranego algorytmu szyfrującego przeprowadzona zostanie również analiza bezpieczeństwa takiego rozwiązania. Zaproponowany zostanie protokół, który pozwoli korzystać w sposób wydajny i bezpieczny z proponowanego rozwiązania.

1.3. Teza pracy

Ogólnie sformułowaną tezę pracy doktorskiej można przedstawić następująco:

Można pokazać, że istnieje alternatywa dla obecnie powszechnego podejścia do implementacji algorytmów kryptograficznych.
W szczególności, można wykazać, że wybrane narzędzia sztucznej inteligencji mogą zostać zastosowane do realizacji funkcji kryptograficznych służących do konstrukcji szyfrów blokowych.

W celu wykazania tezy pracy, przeprowadzone zostały studia literaturowe dotyczące aktualnego stanu badań światowych w zakresie możliwości zastosowania sieci neuronowych w kryptologii. Następnie sformułowano koncepcję wykorzystania sieci neuronowych do budowy szyfrów blokowych i opracowane zostało autorskie oprogramowanie, umożliwiające prowadzenie badań eksperymentalnych dotyczących budowy neuronowego układu szyfrującego. Rezultatem prac jest konstrukcja sieci neuronowych zdolnych do realizacji przekształceń liniowych oraz nieliniowych wykorzystywanych przez szyfry symetryczne. W rozprawie wskazano również dodatkowe możliwości, jakie powstają dzięki neuronowej implementacji szyfrów blokowych.

1.4. Struktura pracy

Rozdział 1.

(str. 3-12)

W tym rozdziale przedstawiono: cel i zakres pracy, sformułowana została teza. Przedstawiona została również motywacja do podjęcia tematyki badawczej, będącej przedmiotem rozprawy.

Rozdział 2.

(str. 13-24)

Rozdział 2 to wprowadzenie podstaw teoretycznych z zakresu kryptologii, dotyczących wyników badań przedstawionych w tej pracy.

Rozdział 3.

(str. 26-39)

W tym miejscu wprowadzono zagadnienia z zakresu podstaw matematycznych dla permutacji oraz S-bloków. Poruszona została również tematyka projektowania szyfrów. Opisano także podstawy matematyczne dotyczące sieci neuronowych.

Rozdział 4.

(str. 40-45)

Rozdział nr 4 zawiera przegląd metod implementacji szyfrów, począwszy od historycznych do współczesnych implementacji programowych i sprzętowych.

Rozdział 5.

(str. 46-48)

W tym rozdziale przedstawiony został przegląd obszarów kryptologii, w których wykorzystywane są sieci neuronowe.

Rozdział 6.

(str. 49-88)

W tym miejscu przedstawione zostały oryginalne rozwiązania, w zakresie realizacji elementarnych przekształceń szyfrujących za pomocą metody matematycznej zwanej "sieć neuronowa".

Rozdział 7.

(str. 89-96)

Rozdział nr 7, to propozycja realizacji szyfru blokowego opartego na modelu sieci S-P, za pomocą neuronowych układów realizujących funkcję S-blok oraz permutacje.

Rozdział 8.

(str. 97-101)

W tym rozdziale przedstawione zostały propozycje możliwych rozwiązań w zakresie konstrukcji protokołu kryptograficznego umożliwiającego wykorzystanie neuronowego układu szyfrującego w rozwiązaniach opartych na architekturze klient-serwer.

Rozdział 9.

(str. 102-107)

W tym miejscu przedstawiono możliwości wykorzystania neuronowych realizacji funkcji S-blok oraz permutacji do realizacji szyfrów.

Rozdział 10.

(str. 108-109)

Rozdział nr 10 to podsumowanie prowadzonych rozważań w całej pracy, oraz przedstawienie obszarów, w których badania będą kontynuowane w przyszłości.

Rozdział 2. Elementy Kryptologii

2.1. Wprowadzenie

Przygotowaniem do rozpoczęcia badań było dokonanie przeglądu obecnie stosowanych metod sztucznej inteligencji w kryptologii oraz w rozwiązaniach z zakresu bezpieczeństwa informatycznego. W ostatnich kilku latach zaczęły pojawiać się publikacje pokazujące możliwości wykorzystania narzędzi sztucznej inteligencji w kryptografii. Szczególnie dotyczy to sieci neuronowych. Jedną z pierwszych prac, w 2001 roku, opublikowali trzej autorzy Ido Kanter, Wolfgang Kiszel i Eran Kanter [17],[18]. Dotyczyła ona kryptografii asymetrycznej. Została opisana tam koncepcja wykorzystania sieci neuronowej, uczonej według metody Hebba, do realizacji protokołu bezpiecznej wymiany klucza. We współczesnej kryptografii, niezwykle ważnym zagadnieniem jest budowanie bezpiecznych kanałów komunikacji. Dotychczas znane protokoły, które to umożliwiały, takie jak chociażby jeden z bardziej popularnych SSL, wykorzystują narzędzia kryptografii asymetrycznej. Wspomniane tu rozwiązanie Secure Socket Layer, wykorzystuje protokół uzgadniania klucza Diffie-Hellman [9]. Tego rodzaju protokoły swoje bezpieczeństwo opierają na trudności rozwiązywania w krótkim czasie różnego rodzaju problemów z zakresu teorii liczb. Autorzy pracy [17] w swoim rozwiązaniu nie wykorzystują teorii liczb, opierają się natomiast na prowadzonym w określonych warunkach uczenie dwóch sieci i synchronizacji wag sieci neuronowych podczas tego procesu. Obie strony (nadawca i odbiorca) początkowo dysponują każdy własną siecią neuronową, jednak o różnych parametrach początkowych. Podczas ustalania klucza, informacje wymieniane pomiędzy nadawcą a odbiorcą nie są tajne. W rezultacie przeprowadzenia procesu uczenia po obu stronach, wagi sieci ulegają synchronizacji, co staje się podstawą ustalenia wspólnego klucza. Zwykle w kryptografii bywa tak, że jeśli ktoś zaproponuje nowy algorytm czy protokół, to po jakimś czasie powoduje to ukazanie się prac przedstawiających dyskusję możliwych potencjalnych ataków. Podobnie stało się z wyżej przytaczaną pracą [17]. Alexander Klimov, Anton Mityaguine i Adi Shamir opublikowali pracę [19], w której prezentują trzy potencjalne metody ataku na neuronowy protokół uzgadniania klucza. Jedna z zaproponowanych metod wykorzystuje narzędzie sztucznej inteligencji, jakim są algorytmy genetyczne. Poza propozycją wykorzystania zjawiska synchronizacji sieci neuronowych do budowy protokołów uzgadniania klucza, sztuczna inteligencja jest również wykorzystywana w systemach wykrywania intruzów. W systemach tych również wykorzystywane są sieci neuronowe do tworzenia tak zwanych profili użytkowników sieci [20].

Celem badań opisanych w tej rozprawie było pokazanie możliwości wykorzystania sieci neuronowych do realizacji algorytmów szyfrujących. Jedną z inspiracji do powstania tej pracy jest fakt pojawienia się w ostatnim czasie prac na temat wykorzystania układów programowalnych do realizacji algorytmów kryptograficznych. Dla implementacji sprzętowych ma to szczególne znacznie, ponieważ zmiana algorytmu realizowanego sprzętowo jest szczególnie kłopotliwa i kosztowna. Jedną z pierwszych prac była [21], gdzie zaproponowano realizację szyfrowania w oparciu o technologię układów programowalnych jako modułu akceleratora kryptograficznego. Wykorzystując rekonfigurowany procesor RipeRench [22] zrealizowano między innymi takie algorytmy jak CRYPTON [23], [24], czy RC6 [25].

W budowie układów programowalnych oraz strukturze sieci neuronowych zachodzą pewne podobieństwa. Zarówno sieci jak i układy składają się z elementarnych, powtarzalnych w strukturze bloków. Założeniem tej pracy jest zaproponowanie takich sieci neuronowych oraz opracowanie metod ich uczenia lub adaptacji, aby możliwe było realizowanie za ich pomocą algorytmów szyfrujących. Cel jaki ma być osiągnięty to zaproponowanie nowego podejścia do problemu realizacji algorytmów szyfrowania. Przy klasycznym podejściu do realizacji algorytmów szyfrujących ewentualna zmiana algorytmu realizowanego przez program lub układ programowalny wymaga przeprowadzenia procesu przeprogramowania. Natomiast dysponując neuronowym układem szyfrującym proces programowania można zmienić na naukę sieci neuronowej nowej metody szyfrowania.

W kolejnych rozdziałach opisana jest wspomniana wyżej koncepcja, jednak najpierw dokonane zostaną niezbędne wprowadzenia z zakresu metod szyfrowania.

2.2. Wstęp do teorii informacji

Matematyczne podstawy dla współczesnej kryptografii określił w 1949 roku Claude Shannon [26],[3]. W pracach tych przedstawił koncepcję zastosowania teorii informacji do analizy szyfrów. Wprowadzenie teorii informacji oraz komputeryzacja spowodowały zmianę podejścia do problemu szyfrowania. Przed erą komputerów permutacje i podstawienia realizowane były na znakach alfabetu. W współczesnej kryptografii podstawową jednostką operacji kryptograficznych jest bit a nie znak alfanumeryczny. We wspomnianych pracach zdefiniowane zostały pojęcia podstawowe dla całej kryptografii. Są to:

Entropia wiadomości M jest to ilość informacji zawartej w wiadomości M. [3]

Np. entropia wiadomości niezbędnej do zapisania informacji o płci wynosi 1 bit, entropia wiadomości oznaczającej dzień tygodnia: 000 niedziela, 001 poniedziałek, …, 110 sobota, 111 – stan niewykorzystany, wynosi 3 bity.

Entropia [26] $H(M)$ wiadomości M, która może przyjmować n równoprawdopodobnych znaczeń jest wyrażona wzorem:

$$H(M) = \log_2 n .$$

Zawartość informacyjna języka

Shannon, w swoich pracach pisze, że entropia [27] tekstu zależy od jego długości N. Entropię tekstów 16-literowych określić można jako 1,3 bit/znak. Zawartość informacyjna języka wyrażona jest wzorem:

$$r = H(M)/N .$$

Wprowadzona została również definicja bezwzględnej zawartości informacyjnej [27] języka, co definiuje się jako założoną liczbę bitów, równo prawdopodobnych, które mogą być przyporządkowane każdemu znakowi. Dla języka, który używa alfabetu L-znakowego jego bezwzględną wartość informacyjną [27] wyraża wzór:

$$R = \log_2 L .$$

Po wprowadzeniu powyższych elementarnych definicji można przejść do omówienia pojęcia nadmiarowości języka. Jest ona bezpośrednio wykorzystywana przez kryptoanalizę, a co za tym idzie ma wpływ na bezpieczeństwo projektowanych algorytmów szyfrujących.

Nadmiarowość języka [27] wyrażona jest wzorem:

$$D = R - r .$$

Dla języka angielskiego nadmiarowość można, więc wyznaczyć w następujący sposób [27]:

$$R = \log_2 26 = 4{,}7,$$
$$r = 1{,}3,$$
$$D = R - r = 3{,}4.$$

Z wyliczeń powyższych wynika więc, że każdy znak języka angielskiego przenosi 3,4 bitu nadmiarowej informacji. Z powyższego wynika prawidłowość, że im bardziej nadmiarowy język w jakim zapisano tekst jawny, tym łatwiejsza kryptoanaliza szyfrogramu. Bierze się to z faktu, że naturalna nadmiarowość języka wykorzystana może zostać w procesie kryptoanalizy do zawężenia liczby możliwych tekstów jawnych. Konsekwencją tego jest częste wykorzystywanie przez programy szyfrujące algorytmów kompresji przed zastosowaniem szyfrowania tekstu jawnego.

W samym procesie szyfrowania w celu zmniejszenia nadmiarowości tekstu jawnego wykorzystuje się dwie podstawowe operacje wprowadzone i zdefiniowane w pracy [3]:

Mieszanie (podstawienie) – którego, zadaniem jest dokonanie zmniejszenia związku pomiędzy tekstem jawnym a szyfrogramem.

Rozproszenie (permutacja) – którego, zadaniem jest rozproszenie nadmiarowości po całym szyfrogramie.

Często przy projektowaniu blokowych algorytmów szyfrujących stosuje się zarówno mieszanie jak i rozproszenie. Tego rodzaju algorytmy szyfrujące określane są jako szyfry złożeniowe lub jako sieci SP (substitution-permutation network). Dla przykładu, w algorytmie DES liniowe operacje permutacji wstępnej, permutacji z rozszerzeniem to rozproszenie. S-bloki realizują natomiast mieszanie. Wspominając o DES-ie zwrócić należy uwagę na kolejną ważną cechę, którą jest iteracyjność samego algorytmu. Wspomniany wyżej szyfr oraz zdecydowana większość współczesnych szyfrów blokowych to Sieci Feistela. Zostały one zdefiniowane i wprowadzone w pracy [28]. Pomysł opiera się na założeniu, że blok o długości n dzielony jest na dwie połowy L i P (długość L i P $n/2$). Dalej Horst Feistel

pisze że można skonstruować iteracyjny szyfr blokowy, czyli taki, w którym wynik danego cyklu zależy od wyniku cyklu go poprzedzającego. Definicję iteracji szyfru blokowego można wyrazić w zapisie :

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i).$$

Wynik i-tego cyklu zależy od wyniku cyklu poprzedniego. Oczywiście poza wynikiem cyklu poprzedniego wpływ na wynik cyklu i-tego ma również klucz cyklu K_i oraz funkcja cyklu f. Dzięki zastosowaniu sumy modulo 2 ciągu lewego (L) z wartością funkcji cyklu, szyfr oparty na tej zasadzie będzie zawsze odwracalny. Na tej zasadzie opartych zostało wiele współczesnych algorytmów szyfrujących, między innymi DES[6], czy FEAL[7]. Rozpatrując bezpieczeństwo szyfrów opartych na zasadzie sieci Feistela widzimy że główne kierunki badań to kryptoanaliza różnicowa i liniowa. Lars R. Knudsen w swojej pracy [29] wprowadza nawet pojęcie praktycznie bezpiecznej sieci Feistela.

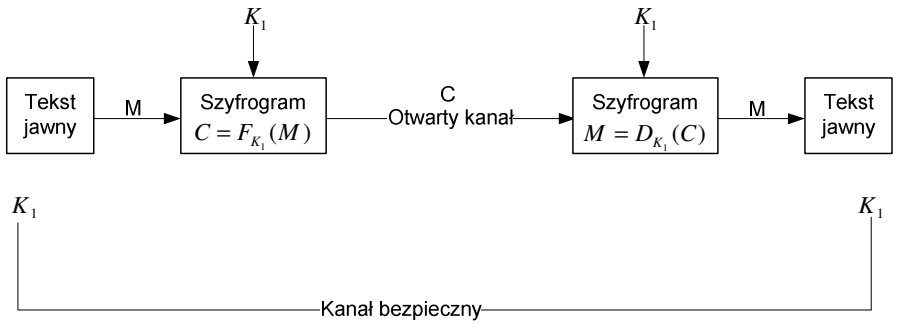
Z punktu widzenia odporności szyfru na wyżej wspomniane rodzaje kryptoanalizy, największe znaczenie ma prawidłowy dobór algorytmów które mają realizować wspomniane już wcześniej mieszanie. W wielu współczesnych szyfrach mieszanie realizowane jest przez specyficzny rodzaj podstawienia, jakim jest S-blok. Tak więc w procesie projektowania algorytmu szyfrującego ważnym elementem jest takie zaprojektowanie S-bloku aby spełniał on wszystkie wymagania bezpieczeństwa, o których będzie mowa w kolejnych rozdziałach.

2.3. Wstęp do algorytmów szyfrujących

Metody szyfrowania w procesie rozwoju kryptografii ewoluowały, jednak zasadniczy przełom nastąpił w momencie opracowania koncepcji kryptografii klucza publicznego. Obecnie algorytmy szyfrujące dzieli się na dwie podstawowy grupy, symetryczne i asymetryczne. Podstawowa różnica pomiędzy nimi wynika z podejścia do problemu dystrybucji klucza. Poniżej przedstawione zostaną koncepcje obydwu metod.

2.3.1. Szyfrowanie symetryczne

Kryptografia symetryczna jest jedyną koncepcją konstrukcji algorytmów szyfrujących, jaka obowiązywała do końca lat siedemdziesiątych. Jej ogólny charakter przedstawia rysunek 2.1.



Rys. 2.1 Kryptosystem symetryczny.

Szyfr symetryczny pozwala na przesyłanie informacji pomiędzy nadawcą i odbiorcą za pośrednictwem otwartego kanału wymiany informacji. Przed rozpoczęciem transmisji nadawca i odbiorca muszą ustalić wspólny klucz $K_n \in K$, według którego będzie się odbywało szyfrowanie i deszyfrowanie, jak pokazane jest to na rysunku 2.1. Jeśli jedna ze stron wygeneruje klucz K_1 z przestrzeni kluczy K, to musi przekazać go drugiej stronie. Ponieważ transmisja ma się odbywać nie chronionym kanałem wymiany informacji, przekazanie klucza (lub jego uzgodnienie) musi zostać dokonane kanałem bezpiecznym, np. za pośrednictwem zaufanego kuriera. Po zakończeniu procesu dystrybucji klucza nadawca może wiadomość M przekształcić w szyfrogram C posługując się funkcją szyfrującą:

$$C = F_{K_1}(M).$$

Tak przygotowaną wiadomość można przesłać do odbiorcy, który posiadając ten sam klucz przekształci szyfrogram do postaci wiadomości M, korzystając z funkcji deszyfrującej:

$$M = D_{K_1}(C).$$

Warunkiem poprawnego działania całego kryptosystemu jest spełnienie warunku:

$$M = D_{K_n}(E_{K_n}(M)).$$

To znaczy szyfrogram, jaki powstaje w wyniku działania funkcji szyfrującej E przy udziale klucza K_n powinien dać się przekształcić za pomocą funkcji deszyfrującej D, przy kluczy K_n, do postaci tekstu jawnego M.

Wymiana informacji jest bezpieczna do momentu aż klucz K_n nie zostanie ujawniony osobie trzeciej. Zmiany klucza można dokonywać w zależności od ustaleń stron biorących udział w wymianie informacji, jednak brać należy zawsze pod uwagę fakt, że przy kryptosystemach symetrycznych wiąże się to z koniecznością użycia bezpiecznego kanału.

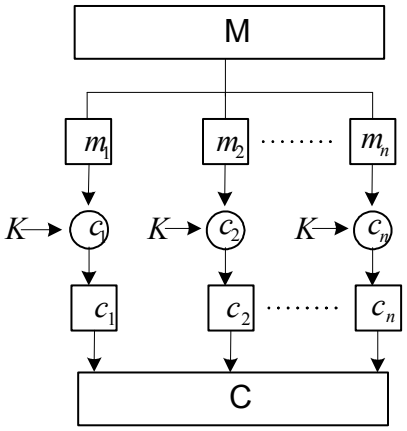
Bezpieczeństwo kryptosystemu nie zależy jedynie od tajności klucza. Ważna jest też jakość stosowanego klucza. Znaczenie ma przestrzeń kluczy K dla danego kryptosystemu. Najprostszą metodą ataku na system szyfrowania jest atak brutalny. Polega ona na uzyskaniu próbki szyfrogramu oraz odpowiadającego mu tekstu jawnego (w przypadku szyfrów blokowych jeden blok to najczęściej 64 lub 128 bity), oraz wypróbowywaniu wszystkich możliwych kluczy. Dla przykładu posługując się kluczem 16-bitowym potrzeba maksymalnie 2^{16} = 65536 prób, co przy możliwościach obecnych komputerów nie jest wielkością astronomiczną. Długość klucza ma więc zasadnicze znaczenie. Bezpieczeństwo współczesnych algorytmów szyfrujących powinno opierać się na tajności klucza. Podstawowe znaczenie ma też jakość samego algorytmu. Każdy nowo projektowany algorytm powinien być odporny na wszystkie znane w danej chwili metody kryptoanalityczne.

Szyfry blokowe

Można dokonać podziału szyfrów symetrycznych na blokowe i strumieniowe. Te pierwsze, które teraz zostaną omówione, działają na bloku tekstu jawnego. Jeden blok tekstu jawnego zamieniany jest na blok tekstu tajnego tej samej długości. Szyfrowanie bitów jednego bloku odbywa się przy udziale tego samego klucza. Sposoby przetwarzania sąsiednich bloków długiego tekstu określane są mianem kryptograficznych trybów działania.

Tryb elektronicznej książki kodowej (ECB)

Tryb ten [13] funkcjonuje w następujący sposób. Tekst jawny M dzielony jest na bloki m_i o długości k. Ostatni blok, jeśli nie ma długości k jest dopełniany. Następnie każdy z tych bloków jest szyfrowany przy użyciu tego samego klucza K.



Rys 2.2 Elektroniczna książka kodowa.

Schemat działania ECB przestawia rysunek 2.2. Szyfrowanie zgodnie z tym trybem można wyrazić wzorem:

$$C = (c_1, c_2, ..., c_n) = (E_K(m_1), E_K(m_2), ..., E_K(m_n)).$$

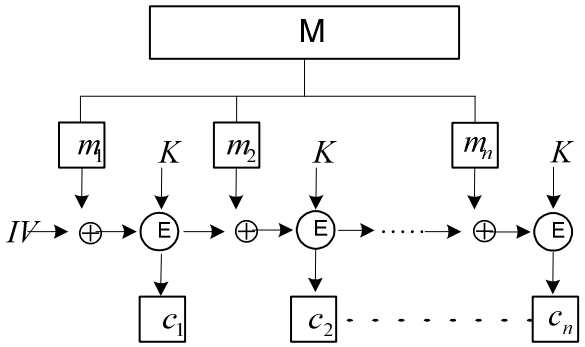
Proces odszyfrowania przebiega analogicznie:

$$M = (m_1, m_2, ..., m_n) = (D_K(c_1), D_K(c_2), ..., D_K(c_n)).$$

Zaletą takiej realizacji szyfrowania jest niezależność poszczególnych bloków szyfrogramu. Uszkodzenie lub utrata jednego bloku nie uniemożliwia odszyfrowania widomości. Wykorzystując niezależność bloków, proces szyfrowania jak i deszyfrowania łatwo można zrównoleglić. Tryb ECB jest także wygodny z punktu widzenia baz danych, można wykonywać operacje na poszczególnych rekordach bez konieczności każdorazowego odszyfrowania i szyfrowania całej bazy.
Elektroniczna książka kodowa ma też jednak poważną wadą, możliwe jest przeprowadzenie ataku bez znajomości klucza.

Tryb wiązanie bloków zaszyfrowanych (Cipher Block Chaining (CBC))

Jest to tryb [13], który wymusza szyfrowanie tych samych bloków tekstu jawnego w różnych miejscach w różny sposób. Ponadto to wprowadza powiązanie pomiędzy poszczególnymi blokami szyfrogramu. Zasadę działa trybu CBC przedstawia rysunek 2.3.



Rys 2.3 Tryb wiązanie bloków zaszyfrowanych.

Wektor inicjujący IV, początkowo musi być znany obu stronom biorącym udział w wymianie informacji, jednak nie musi być on utajniony. Przy trybie wiązanie bloków zaszyfrowanych poszczególne kryptogramy poszczególnych bloków są ze sobą powiązane. Dla otrzymania kolejnego bloku szyfrogramu C_n wykorzystuje się blok poprzedni C_{n-1}. Szyfrowanie przebiega więc według wzoru:

$$C = (c_1, c_2, ..., c_n) = (E_k(m_1 \oplus IV), E_k(m_2 \oplus c_1), ..., E_k(m_n \oplus c_{n-1})),$$

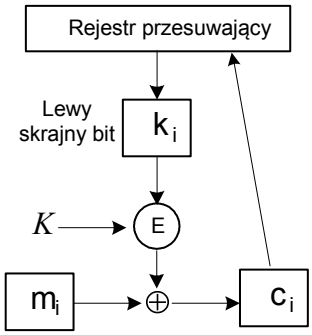
gdzie c_n oznacza kolejne bloki szyfrogramu, a m_n bloki tekstu jawnego. Odszyfrowanie przebiega oczywiście w sposób analogiczny, dla n-tego bloku przedstawia się według wzoru:

$$m_n = D_K(c_n) \oplus c_{n-1}.$$

Przy takim sposobie szyfrowania ważne jest to, że dwa takie same bloki tekstu jawnego będą reprezentowane przez dwa różne bloki szyfrogramu. Żadnego z bloku C_n nie można usunąć, co jest niewygodne z punktu widzenia wykorzystania tego trybu np. w bazach danych.

Tryb sprzężenia zwrotnego zaszyfrowanego tekstu (Cipher Feedback (CFB))

Szyfrowanie tym trybem [13] polega na szyfrowaniu nie całych bloków, ale ich fragmentów, np. 8 bitów. Takie rozwiązanie przydatne jest w sytuacji, gdy nie można czekać na wprowadzenie całego bloku, a przesyłane muszą być już pojedyncze litery, obrazuje to rysunek 2.4.



Rys 2.4 Tryb sprzężenia *zwrotne*go *zaszyfrowanego tekstu.*

Szyfrowanie według CFB przebiega następująco, najpierw bitami wektora początkowego jest wypełniany rejestr przesuwający i wysyłana do drugiej strona biorącej udział w wymianie informacji. W drugim kroku zawartość rejestru szyfrowana jest przy użyciu klucza K. Po zaszyfrowaniu rejestru 8 bitów skrajnych dodaje się modulo 2 z bitami tekstu jawnego. Otrzymując pierwsze 8 bitów szyfrogramu, które mogą być teraz przesłane na drugą stronę. Te same bity trafiają do rejestru przesuwającego. Cykl ten powtarzany jest aż do wyczerpania tekstu jawnego.

Tryby CBC i CFB stosowane mogą być do kontroli integralności przesyłanych danych, ponieważ w obydwu trybach jakakolwiek zmiana w szyfrogramie podczas transmisji spowoduje niemożność przeprowadzenia udanego odszyfrowani.

Szyfry strumieniowe

Wspomniane powyżej szyfry blokowe szyfrują teksty jawne o ściśle określonej długości, co determinuje długość bloku do jakiej przetwarzania dany algorytm szyfrujący został zaprojektowany. Szyfrowanie dłuższych wiadomości wymaga stosowania dodatkowych technik. Koncepcja szyfru strumieniowego prezentuje inne podejście. Wiadomość szyfrowana jest bit po bicie. Szyfry strumieniowe sprawdzają się wszędzie tam gdzie trudno jest określić długość tekstu jawnego i gdzie ograniczone są możliwość buforowania danych. Ogólna konstrukcja szyfru strumieniowego wywodzi się z idei jednorazowego szyfru Vernama z 1917 roku, o nazwie „One-time pad". Samo przekształcenie tekstu jawnego odbywa się z wykorzystaniem operacji $\oplus$ (suma modulo 2). Klucz wybierany jest losowo, najważniejszą zasadą tego rozwiązania jest to, aby przy każdym kolejnym szyfrowaniu używany był inny niezależnie, losowy wybrany klucz. Mając wiadomość, która składa się z ciągu bitów $M = b_1, b_2, ..., b_i$, generowany jest ciąg klucza $K = k_1, k_2, ..., k_i$, o tej samej długości, co sama wiadomość. Szyfrowanie przebiega według poniższego schematu:

$$C = M \oplus K, \quad c_i = b_i \oplus k_i,$$
$$M = C \oplus K, \quad m_i = c_i \oplus k_i.$$

Szyfr Vernama określany jest jako szyfr doskonałej poufności, kluczowe dla tego określenia są dwie najważniejsze jego właściwości:

- generowany kryptogram jest ciągiem losowym, ponieważ prawdopodobieństwo, że w kryptogramie wystąpi 0 $(c_i = 0)$ jest ½, niezależnie od wartość bitu tekstu jawnego

(a_i). Ponieważ i-ty bit klucza jest losowy, tak więc i i-ty bit kryptogramu będzie losowy i niezależny ponieważ kolejne bity klucza generowane są niezależnie,

- na podstawie analizy szyfrogramu bez znajomości klucza, nie można uzyskać żadnych informacji na temat tekstu jawnego. W metodzie „One-time pad" spełniony musi być warunek:

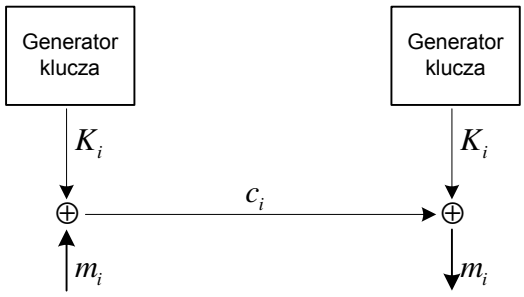
$$C = D \oplus K, \quad K = C \oplus D .$$

Ponieważ ciąg bitów C odpowiada z takim samym prawdopodobieństwem dowolnemu ciągowi bitów M, więc wyciągnięcie jakichkolwiek wniosków na temat tekstu jawnego na podstawie szyfrogramu jest niemożliwe.

Szyfr ten ma jednak mocno ograniczone zastosowania. Z kilku względów:

- problem generowania ciągów losowych,
- długość klucza musi być taka sama jak długość wiadomości,
- klucz musi być uzgodniony przez osoby biorące udział w wymianie informacji.

Współczesne szyfry strumieniowe posługują się generatorem kluczy pseudolosowych, które wyglądają jak losowe, ale w rzeczywistości są na tyle zdeterminowane aby można je było odtworzyć po stronie odbiorcy. Ogólny schemat [13] działania szyfru strumieniowego przedstawiony jest na rysunku 2.5.



Rys 2.5 Szyfrowanie strumieniowe.

Szyfr strumieniowy przekształca tekst jawny w szyfrogram bit po bicie. Generator strumienia klucza generuje strumień bitów $k_1, k_2, ..., k_i$. Jest on poddawany działaniu funkcji logicznej „suma modulo 2" ze strumieniem bitów tekstu jawnego $b_1, b_2, ..., b_i$. Szyfrowanie i deszyfrowanie wyrażone jest wzorami:

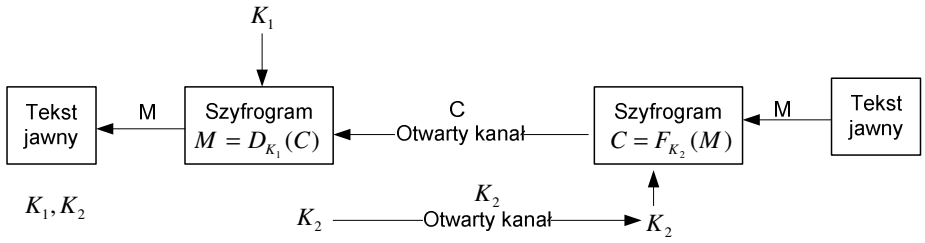
$$\text{szyfrowanie} : c_i = b_i \oplus k_i,$$
$$\text{deszyfrowanie } b_i = c_i \oplus k_i .$$

Bezpieczeństwo przy takim szyfrowaniu zależy całkowicie od jakości generatora kluczy. Jeśli np. jako klucza generowany będzie ciąg samych zer to szyfrowanie właściwie w ogóle nie

będzie miało miejsca. Jeśli generowany będzie strumień składający się z pewnej powtarzalnej sekwencji to bezpieczeństwo takiego układu również będzie na bardzo niskim poziomie.

2.3.2. Szyfrowanie asymetryczne

W swojej pracy [30] Rivest, Shamir i Adleman opublikowali podstawy teoretyczne opierające się na wykorzystaniu pewnych specyficznych funkcji jednokierunkowych do konstrukcji algorytmów szyfrujących z kluczem publicznym. We wspomnianej pracy udowadniają Oni, że jest możliwe stworzenie takiego kryptosystemu, którym będzie można się posługiwać w bezpieczny sposób bez konieczności używania bezpiecznego kanału do wymiany klucza. Wszystkie algorytmy symetryczne wykorzystują jeden klucza do szyfrowania i deszyfrowania. Kryptografia asymetryczna wprowadziła pojęcia klucza publicznego i prywatnego, z których jeden służy do szyfrowania a drugi do odszyfrowania, co pokazuje rysunek 2.6.



Rys. 2.6 Kryptosystem asymetryczny.

Jeśli nadawca zamierza przesłać do odbiorcy zaszyfrowaną wiadomość, musi posiadać klucz publiczny K_2 odbiorcy, który może mu zostać dostarczony otwartym kanałem transmisyjnym, tym samym, którym ma się odbyć przesłanie szyfrogramu. Klucz publiczny nie stanowi tajemnicy, ponieważ przy dobrze zaprojektowanym kryptosystemie asymetrycznym jego znajomość nie pozwoli na uzyskanie klucza prywatnego ani na odszyfrowanie wiadomości w „rozsądnie" długim czasie. Nadawca po otrzymaniu klucza publicznego K_2 szyfruje wiadomość M korzystając z funkcji szyfrującej:

$$C = F_{K_2}(M).$$

Tak otrzymany szyfrogram przesyłany jest do odbiorcy, który jest w posiadaniu klucza prywatnego K_1 stanowiącego parę z kluczem K_2 co pozwala mu na przekształcenie otrzymanego szyfrogramu C na tekst jawny M. Kryptosystem działa poprawnie jeśli spełniony jest warunek:

$$M = D_{K_1}(E_{K_2}(M)).$$

Rozdział 3. Podstawy matematyczne

3.1. Zasady projektowania szyfrów blokowych

Projektowanie algorytmów szyfrujących jest mimo obecnego rozwoju technologicznego nadal zadaniem złożonym i trudnym. Już samo zdefiniowanie pojęcia bezpieczny szyfr nie jest zadaniem łatwym. Jedynie o szyfrze Vernama z 1917 roku, o nazwie „One-time pad" można powiedzieć, że jest bezpieczny, oczywiście o ile stosowany będzie z zachowaniem wszystkich zasad prawidłowego użytkowania tego szyfru. Jednak z punktu widzenia praktycznego jest on mało użyteczny.

3.1.1. Zasady Kerkhoff'a

Ogólne wytyczne dotyczące bezpieczeństwa i funkcjonalności algorytmów szyfrujących znaleźć można w pracy z 1883 rok [31]. Autor sformułował tam „Zasady budowy dobrej maszyny szyfrującej" - zasady Kerkhoff'a:

1. Szyfr nie do złamania,
2. Utajniony jest klucz, a nie sama budowa maszyny,
3. Klucz powinien być łatwy do zapamiętania,
4. Kryptogram powinien być łatwy do przekazania,
5. Maszyna szyfrująca powinna być łatwa do przenoszenia,
6. System powinien być łatwy w użyciu.

Szczególnie ważną zasadą z punktu widzenia dzisiejszych rozwiązań w zakresie bezpieczeństwa informacji jest zasada 2. Wskazuje ona na to, że funkcja szyfrująca E i deszyfrująca D powinny być ogólnie znane i dobrze udokumentowane. Wykonanie przekształcenia szyfrogramu C do postaci tekstu jawnego M powinno być proste pod warunkiem znajomości klucza K. Brak znajomości klucza K, a jedynie posiadanie szyfrogramu oraz funkcji deszyfrującej nie powinno dawać możliwość wyznaczenia tekstu jawnego M. Spełnienie tych postulatów bezpieczeństwa realizowane jest poprzez zastosowanie przekształceń szyfrujących, jakimi są przekształcenia liniowe (permutacje) oraz nieliniowe (podstawienia).

W szyfrach kaskadowych, głównie w tej tematyce osadzona jest ta praca, bezpieczeństwo zależy w przeważającej mierze od jakości zaprojektowanych S-bloków. Projektowanie bezpiecznych S-bloków nie jest zadaniem łatwym. Zestaw kryteriów projektowych, jakie powinny spełniać S-bloki również ulega ciągłym zmianie. Wynika to z faktu rozwoju metod ataków na szyfry. Większość ostatnio zaproponowanych metod ataków na różne algorytmy

wykorzystuje słabości tkwiące właśnie w S-blokach. Nowo wykryta słabość staje się kryterium projektowym. Jeśli S-bloki wybranego szyfru spełniają wszystkie aktualne kryteria, to szyfr uznawany jest za bezpieczny. W procesie projektowania szyfrów, a dokładniej bezpiecznych S-bloków pewnym fundamentem jest teoria informacji; wybrane jej zagadnienia omówione zostały w rozdziale 2.2

3.1.2. Permutacje

Permutacje to funkcje, wykorzystywane są w konstrukcji szyfrów. Permutacja p na S jest bijekcją z S na ten sam zbiór ($S \to S$). Przez S rozumiemy skończony zbiór elementów. Bijekcja jest tu rozumiana jako funkcja typu 1-1 i "na". Funkcję i przekształcenie określa się jako 1-1 (jeden do jednego) jeśli każdy element w przeciwdziedzinie jest obrazem najwyżej jednego elementu z dziedziny. Funkcja jest "na", jeżeli każdy element przeciwdziedziny jest obrazem pewnego elementu dziedziny.
Tak więc dla przykładu: S={1,2,3,4,5}, permutacja $p : S \to S$, może zostać zapisana jako:

p(1)= 3, p(2)= 5, p(3)= 4, p(4)= 2, p(5)= 1 lub

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 2 & 1 \end{pmatrix}.$$

Permutacja jest bijekcją, wobec tego odwrotność permutacji p to:

$$p^{-1} = \begin{pmatrix} 3 & 5 & 4 & 2 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}.$$

Permutacje często poddawane są operacji składania, oznaczanej dalej jako $\circ$. Mając permutacje p_1 i $p_2 \in S(x)$ jej złożeniem jest permutacja $p_1 \circ p_2 \in X$, co zostało pokazane na przykładzie:

$$p_1 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}, p_2 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}, \quad p_1 \circ p_2 = p\,, \quad p = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}.$$

Dla permutacji ważnym pojęciem jest cykl. Jeżeli przyjąć, że $i_1, i_2, \ldots, i_k \in \{1,2,\ldots,n\}$ będą różnymi wartościami, to cyklem o długości *k* nazywamy permutację $p \in S_n$ taką, że:

$$p_{i_1} = i_2, p_{i_2} = i_2, \ldots, p_{i_k} = i, p_j = j \quad dla \quad j \neq i_1, i_2, \ldots, i_k\,.$$

Cykl można zapisać na k różnych sposobów, zaczynając od dowolnego i_l, gdzie $l = 1,2,...,k$.

Przykład:

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 3 & 4 & 5 & 1 & 7 & 6 & 8 \end{pmatrix} \in S_8,$$

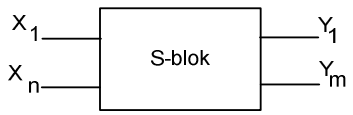
zapisując jako cykl: $p = (12345)(67)(8)$ cykle o długości 1 można pominąć w zapisie $p = (12345)(67)$.

Na podstawie cykli bardzo łatwo można określić rząd danej permutacji. Rzędem permutacji nazywamy najmniejszą wspólną wielokrotność długości cykli, przy czym brane są pod uwagę cykle od długości większej lub równej dwa.

Kolejne zagadnienia to transpozycja, tym mianem nazywamy cykl o długości 2. Z tego wynika, że transpozycja dokonuje przestawienia dwóch elementów ze zbioru.

3.1.3. S-bloki

S-bloki są często występującym elementem w szyfrach blokowych. Ich zadaniem jest realizowanie określonego podstawienia, czyli zamiany ciągu n bitów wejściowych na ciąg m bitów wyjściowych. S-blok można przedstawić tak jak to pokazano na rysunku 3.1.



Rys. 3.1 Schemat działania S-blok-u.

Rozpatrując S-blok jako funkcję boolowską widzimy, że realizuje ona odwzorowanie:

$S : \{0,1\}^n \to \{0,1\}^m$, gdzie m,n – wymiary S-bloku.

Tak więc S-bloki można traktować jako tablice dwuwymiarowe zawierające wartości binarne. Dobór wartości, jakimi „wypełniany" jest S-blok nie jest bez znaczenia. Bardzo istotnym elementem jest to, aby wektor wyjściowy nie był liniowym odwzorowaniem wektora wejściowego. Dla S-bloków określone zostały ścisłe wymagania, jakie powinny one spełniać, aby nie były słabym ogniwem bezpieczeństwa szyfrów, w których zostały wykorzystane. Wartości umieszczane w poszczególnych komórkach tablicy mogą się powtarzać w obrębie całego S-bloku. Istnieje kilka metod, które pomagają w odpowiednim

doborze wartości dla S-bloków. Jednak metody te nie są metodami stuprocentowymi, pozostawiają też wiele do życzenia pod względem złożoności czasowej. Jedną z takich metod jest metoda doświadczalna. Polega ona na wypełnienie S-bloku w sposób losowy i późniejszym przetestowaniu go pod względem spełnianych wymagań dla doskonałego S-bloku.

3.1.4. Kryteria projektowe

S-blok, we współczesnych algorytmach szyfrujących realizuje on operację mieszania. Ze względu na swój nieliniowy charakter często jest elementem, który faktycznie decyduje o jakości projektowanego algorytmu szyfrującego. S-blok zwyczajowo zapisywany jako tablica składająca się określonej liczby kolumn i wierszy, zawierająca w swoich komórkach wartości liczbowe. Funkcja jaką realizuje ten element powinna być nieliniowa. Projektując, więc S-blok należy brać pod uwagę wiele właściwości jakie powinien spełniać. To w jakim stopniu wymagania te zostaną spełnione decyduje o bezpieczeństwie całego szyfru. Siła różnych sieci Feistela, co za tym idzie ich odporność na metody kryptoanalizy wiąże się bezpośrednio z siłą S-bloków, jako elementów nieliniowych.

Rozmiar S-bloku. Generalnie istnieje zasada, że im większy s-blok tym lepiej, ponieważ trudniejsze jest wyznaczenia odpowiednich charakterystyk na potrzeby kryptoanalizy liniowej czy różnicowej. Znaczenie ma też stosunek liczby bitów wejściowych do wyjściowych, jeśli n oznaczać będzie liczbę wejść, a m liczbę wyjść to:

Jeżeli: $n \geq 2^m - m, \Rightarrow$ S-blok realizuje funkcję liniową [32],

Jeżeli: $n \geq 2^m \Rightarrow$ to między bitami n zachodzi zależność liniowa [32].

Zupełność. Kam i David w pracy [33] w odniesieniu do sieci S-P zdefiniowali pojęcie zupełności funkcji szyfrującej. Funkcja $E : \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^l$ jest zupełna, jeśli dla każdej pary pozycji bitów $i, j \varepsilon \{0,1,...,t-1\}$ istnieją przynajmniej dwa bloki wejściowe X_1, X_2 takie, że $X_1 \oplus X_2 = i$, dla których dane wyjściowe Y_1, Y_2 takie że $Y_1 \oplus Y_2 = j$. Inaczej mówiąc, dla kryptograficznej funkcji zupełnej każdy bit szyfrogramu zależy od wszystkich bitów tekstu jawnego.

Większość kryteriów projektowych dotyczy właściwości funkcji Boole'owskie, wprowadzona więc zostanie definicja tej funkcji.

Funkcje Boolowskie

Rozważając właściwości, zastosowania oraz inne aspekty funkcjonowania S-bloków najczęściej sprowadza się to wszystko do badania funkcji boolowskich. Funkcje takie muszą spełniać określone wymagania i kryteria, które zostaną tu zdefiniowane.

Funkcja boolowska n zmiennych to funkcja :

$$f : B^n \to B \text{ , gdzie } B = \{0,1\} \text{ (*).}$$

W definiowaniu właściwości funkcji binarnych często posługujemy się pojęciem tabeli prawdy, jako sposobu zapisu funkcji. Tak więc, jeśli rozpatrzymy funkcję (*), która danym wektorom:

$$\begin{aligned} &\alpha_0 = (00...00), \\ &\alpha_1 = (00...01), \\ &..., \\ &\alpha_{2^n-1} = (11...11), \end{aligned}$$

przypisuje wartości binarne, takie że $f(\alpha_n) \in B$ to powstający w ten sposób ciąg binarny:

$$\left(f(\alpha_0), f(\alpha_1), ..., f\left(\alpha_{2^n-1}\right)\right),$$

nazywamy tabelą prawdy.

Z kryptograficznego punktu widzenia bardzo ważną cechą funkcji Boole'owskie jest ich zrównoważanie oraz wartość wagi Hemminga. Funkcja boolowska jest zrównoważona jeśli jej tabela prawdy zawiera 2^{n-1} jedynek lub zer, inaczej mówiąc tyle samo zer co jedynek.

Wagą Hamminga wektora v nazywamy liczbę jedynek w nim występujących, $hwt(v)$. W odniesieniu do funkcji boolowskiej waga Haminga to liczba jedynek w jej tabeli prawdy.

Odległość Haminga między dwiema funkcjami jest parametrem, który określa liczbę wejść dwóch funkcji, na których te funkcje się różnią. Zdefiniować ją można jako:

$$d(f,g) = hwt(f \oplus g) = \sum_x f(x) \oplus g(x) \quad .$$

Funkcja binarna jest afiniczna jeśli możliwe jest zapisanie jej jako:

$$f(x_1,...,x_n) = a_0 \oplus a_1 x_1 \oplus ... \oplus a_n x_n \text{ , gdzie } a_i \in B, B = \{0,1\}, i = 0,...,n \text{ .}$$

Funkcja afiniczna jest określana jako liniowa jeśli a_0=0.

Zrównoważenie : kryterium to zostało scharakteryzowane wcześniej w tej pracy dla funkcji boolowskiej. Traktując S-blok jako funkcję możemy pojęcie zrównoważenia funkcji boolowskiej odnieść do S-bloku. Nie zrównoważenie powoduje podatność na wszelkiego rodzaju metody kryptoanalizy, które bazują na analizie prawdopodobieństw ciągu wyjściowego. Niebezpieczeństwo to wzrasta wraz ze wzrostem liczby rund, w których wykorzystywany jest niezrównoważony S-blok, ponieważ powstaje pewne odchylenie, przy którym niektóre ciągi wyjściowe są bardziej prawdopodobne niż inne. Stanowi to tym samym punkt zaczepienia dla kryptoanalizy.

Efekt lawinowy – Z czysto praktycznego punktu widzenia S-blok spełnia kryterium lawinowości, jeśli zmiana jednego bitu w ciągu wejściowym powoduje zmianę przynajmniej dwóch bitów na wyjściu.

Definicja właściwości jaką jest efekt lawinowy wiąże się z wyżej wspomnianą właściwością zupełności. W pracach [28], [34] Feistel zdefiniował kryterium lawinowości:

Funkcja $f : B^n \to B$ spełnia kryterium lawinowości, jeśli zmiana jednego bitu w ciągu wejściowym powoduje zmianę przynajmniej połowy bitów na wyjściu.

Ścisłe kryterium lawinowości - Rozwinięciem powyższej definicji jest ścisłe kryterium lawinowości. Właściwość tę określa się często skrótem SAC (Strict Avalanche Criterion) zdefiniowali ją A.F. Webster i S.E. Tavaresw 1986 [35].

Funkcja $f : B^n \to B$, spełnia warunek SAC jeśli $f(x) \oplus f(x \oplus \alpha)$ jest zrównoważona dla wszystkich wektorów α takich, że ich waga Haminga równa jest 1. Funkcjonuje również pojęcie warunku SAC k-tego rzędu. Warunek ten to niejako uogólnienie zwykłego warunku SAC, do przypadku gdy zmianie w wektorze wejściowym ulega więcej niż jeden bit. W takim przypadku funkcja f spełnia warunek SAC rzędu K, jeśli $f(x) \oplus f(x \oplus \alpha)$ jest zrównoważone dla każdego $\alpha \in B$, oraz $1 \le W(\alpha) \le k$ (W-waga Hamminga).

Nieliniowości funkcji boolowskiej f, którą można zdefiniować jako odległość tej funkcji od zbioru wszystkich jej funkcji afinicznych oznaczanego jako A_n, zdefiniowanych nad B^n.

$$N_f = \min_{g \in A_n} d(f, g),$$

gdzie N – stopień nieliniowości, g - funkcja afiniczna.

W pracy [36] wprowadzone zostało pojęcie bent funkcji, określanej jako funkcja doskonale nieliniowe. Funkcja binarna jest doskonale nieliniowa jeśli dla $f(x) \oplus f(x \oplus \alpha)$ jest

zrównoważona dla każdego wektora α takiego, że jego waga Hamminga zawiera się w przedziale od 1 do n.

Bent funkcje cechują się dużymi stopniami nieliniowości oraz spełniają warunek SAC, lecz nie są funkcjami zrównoważonymi. Mimo to często są wykorzystywane w procesie konstrukcji S-bloków poprzez wykorzystanie odpowiednich metod numerycznych.

Profile XOR. Są podstawowym narzędziem kryptoanlizy różnicowej. Kryptoanaliza różnicowa została opracowana dla algorytmu DES i w takiej wersji opublikowana została przez E. Bihama i A. Shamira [37]. Z czasem adaptowano ją do innych szyfrów i obecnie jest jedną z najpopularniejszych form ataków na kryptosystemy. Metoda to należy do grupy ataków ze znanym tekstem jawnym. Polega ona na analizowaniu rezultatu kolejnych iteracji podczas szyfrowania. Analiza prowadzona jest przy wykorzystaniu pary znanych tekstów jawnych różniących się w określony sposób. Różnicę tę określa się za pomocą funkcji $\oplus$. Główne spostrzeżenie, jakiego dokonali twórcy tej metody to:

$$(X \oplus K) \oplus (X' \oplus K) = X \oplus X', \text{ gdzie } \oplus \text{ oznacza funkcje XOR}$$

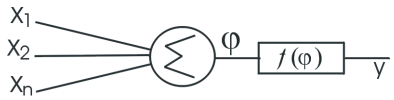
Całość sprowadza się więc to tego, że ciągi bitów wejściowych są znane, ale nieznany jest ciąg użytego klucza. Przy realizacji kryptoanalizy pomijane są operacje liniowe. Tak więc rozpatrywanym fragmentem szyfru są S-bloki. Wartość $X \oplus K$ i $X' \oplus K$ ulega zmianie po przejściu przez S-blok. Nowo postała różnica jest zależna zarówno od $X \otimes X'$, jak również od $X \oplus K$ i $X' \oplus K$. To co daje równanie kryptoanalizy różnicowej to fakt, że pomimo tego, że nie znamy danych wejściowych S-bloku (ponieważ nie znamy klucza) znamy ich różnicę, ponieważ jest ona taka sama jak różnica ciągów X i X'. Z punktu widzenia projektowania S-bloku należy dążyć do tego, aby tablica XOR-profilu dla danego S-bloku nie zawierała dużych liczb, co znacznie ułatwiłoby wyznaczenie klucza rundy. Problem dużych liczb występujących w XOR-profilu można rozwiązać częściowo poprzez zwiększanie liczby rund, jednak dzieje się to kosztem wzrostu złożoności czasowej.

Kompletność: dotyczy szyfru jako całości. Przy odpowiednio przygotowanych S-blokach oraz p-blokach należy zadbać o właściwe ich „otoczenie". Między innymi, należy dobrze określić liczbę rund, sposoby generowania kluczy dla poszczególnych cykli tak, aby osiągnięty został określony poziom odporności na znane metody kryptoanalityczne.

3.2. Sieci neuronowe

3.2.1. Wprowadzenie

W sieciach neuronowych przetwarzanie informacji odbywa się w sposób lokalny, poprzez realizowanie elementarnych procesów, czyli poprzez neurony. Pierwowzorem matematycznego modelu neuronu jest jego biologiczny odpowiednik. Dokładne działanie biologicznego neuronu nie jest do końca zbadane, więc wszystkie modele matematyczne są w pewnym sensie próbą przybliżonego odwzorowania działania biologicznego neuronu. Jednym z pierwszych modeli neuronów jest zaproponowany w 1943 roku model McCulloch-Pittsa [38]. Przyjmuje on neuron jako jednostkę binarną. Model neurony przedstawiony został na rysunku 3.2.



Rys. 3.2 Model neuronu.

Istota działania tego modelu polega na sumowania sygnałów wejściowych x przemnożonych przez wartości wag w. Otrzymana suma iloczynów φ nazywana jest potencjałem membranowym, to poprzez analogie z biologicznym neuronem. Działanie matematycznego motelu neuronu wyrazić można jako:

$$\varphi = \sum_{i=1}^{N} x_i * w_i = x * w^T \quad , \quad \begin{matrix} x = [x_1, x_2, ..., x_i], \\ w = [w_1, w_2, ..., w_i]. \end{matrix}$$

Ostatni etap to działanie funkcji aktywacji f, dla której argumentem jest właśnie potencjał membranowy. Otrzymana wartość jest sygnałem wyjściowym neuronu y. W neuronie McCulloch-Pittsa jako funkcję aktywacji f zaproponowano funkcję progową, wyrażoną wzorem:

$$f = \begin{cases} 1, & \varphi > p \\ 0, & \varphi < p \end{cases}.$$

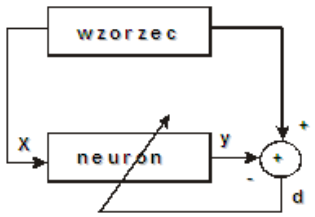
Stosowane są również modelu neuronów z innymi funkcjami aktywacji. Najpopularniejszą z nich jest funkcja sigmoidalna.

W ogólnym przypadku można przyjąć, że neuron realizuje pewne nieliniowe odwzorowanie $n : X \to Y$. X reprezentuje tu przestrzeń wejść, a Y przestrzeń wyjść. Rodzaj odwzorowania i przestrzeni zależy od problemu i funkcji, jaką realizuje dany neuron. Najczęściej przyjmuje

się w literaturze, że neuron realizuje operacje na liczbach rzeczywistych, wejście określa się zwykle jako wektor $x = [x_1, x_2, ..., x_i]$, natomiast wyjście jest skalarem y. Przyjmuje się również, że odwzorowanie n realizowane przez neuron powinno być możliwie elementarne. Wymaganą złożoność na potrzeby realizowanego zadania uzyskać należy przez odpowiednią skonstruowaną sieć neuronową.

3.2.2. Reguła perceptronu

Perceptron prosty to model neuronu McCulloch-Pittsa z opracowaną tak zwaną regułą perceptronu [39]. Proces uczenia zgodnie z tą regułą polega na realizacji procesu uczenia według zasady uczenia z nauczycielem. Ważnym elementem w procesie uczenia jest tzw. zbiór uczący, na który składają się pary wzorcowe, czyli komplet wartości wejściowych x oraz sygnał wyjściowy y. Zbiór uczący zawiera oczywiście poprawne wartości sygnałów wyjściowych. Zbiór uczący zawiera wiele takich par. Zbiór ten konstruowany jest często na podstawie wyników pochodzących z eksperymentów lub w inny sposób pozwalający zgromadzić reprezentatywne przypadki dla odwzorowanie, jakie ma realizować perceptron. Proces uczenia sieci neuronowej przedstawiony został na rysunku 3.3.



Rys. 3.3 Proces uczenia sieci neuronowej.

Na rysunku 3.3 przez wzorzec rozumiemy zbiór uczący składający się z dwóch wektorów:

$$x(n) = [x_1(n), x_2(n), ..., x_i(n)]^T,$$
$$y(n) = [y_1(n), y_2(n), ..., y_j(n)]^T.$$

W powyższych wzorach *n* to numer, kolejnego zestawu wektorów uczących (cykl procesu uczenia), *i* natomiast to kolejny nr wejścia i wyjścia neuronu.

Kolejne wektory x_i ze zbioru uczącego trafiają na wejście neuronu, na podstawie wartości odpowiedzi neuronu oraz wartości wzorcowej y_i pochodzącej z wektora y przeprowadzana jest korekta wartości wag dokonywana jest według następującego schematu [39]:

- jeśli błąd $d = 0$ to korekta wag nie następuje,
- jeśli odpowiedź neuronu równa jest 0, a $y_n = 1$, to korekta wag realizowana jest według wzoru :

$$w_i(n+1) = w_i(n) + x_i(n),$$

gdzie $n+1$ symbolizuje cykl bieżący a n to cykl poprzedni,

- jeśli odpowiedź neuronu równa jest 1, a $y_n = 0$, to korekta wag realizowana jest według schematu :

$$w_i(n+1) = w_i(n) - x_i(n).$$

Proces ten powtarzany jest dla wszystkich próbek uczących, do mementu uzyskania zakładanej minimalizacji różnic pomiędzy uzyskiwanymi odpowiedziami neuronu a wartościami żądanymi pochodzącymi ze zbioru uczącego.

3.3.3. Reguła Hebba

Po koncepcji perceptronu Donald O. Hebb zaproponował własną teorię uczenia. Dokonał on pewnej obserwacji związanej z połączeniami między neuronowymi. Hebb zaobserwował, że wagi połączeń pomiędzy neuronami zwiększane są przy uaktywnieniach poszczególnych neuronów, czyli stopień korekty wag zależy, między innymi, od poziomu sygnału wyjściowego. Korekta wag w modelu Hebba przebiega według zależności:

$$w_i(n+1) = w_i(n) + \eta x_j(n) y_j(n).$$

W ogólnym przypadku, reguła uczenia Hebba może być zapisana jako:

$$\Delta W_i(n) = F(x_i, y_j),$$

gdzie F to funkcja sygnału wejściowego zwanego presynaptycznym oraz wyjściowego y zwanego postsynaptycznym. Często regułę Hebba przedstawia się jako funkcję iloczynową sygnałów x i y:

$$\Delta W_i(n) = \eta x_i(n) y_j(n).$$

Współczynnik uczenia η decyduje o tym jaki wpływ w cyklu n ma sygnał uczący na wartość korekty wag.

Korzystając z modelu Hebba można realizować uczenie na wiele sposobów. Sam model Hebba doczekał się wielu wersji dostosowanych do konkretnych przypadków. Uczenie odbywać się może według dwóch schematów, z nauczycielem i bez nauczyciela. W tej pracy wykorzystywana przez nas będzie metoda uczenia z nauczycielem, więc skupimy się na opisie tej koncepcji. Ogólny schemat reprezentujący przebieg procesu uczenia z nauczycielem został pokazany na rysunek 3.3, Reguła Hebba dla tego przypadku jest zapisana w postaci:

$$\Delta W_i(n) = \eta x_i(n) d_i(n) .$$

W dalszej części pracy używana będzie forma zapisu:

$$w_i^{'} = w_i + \eta d(n) x_i ,$$

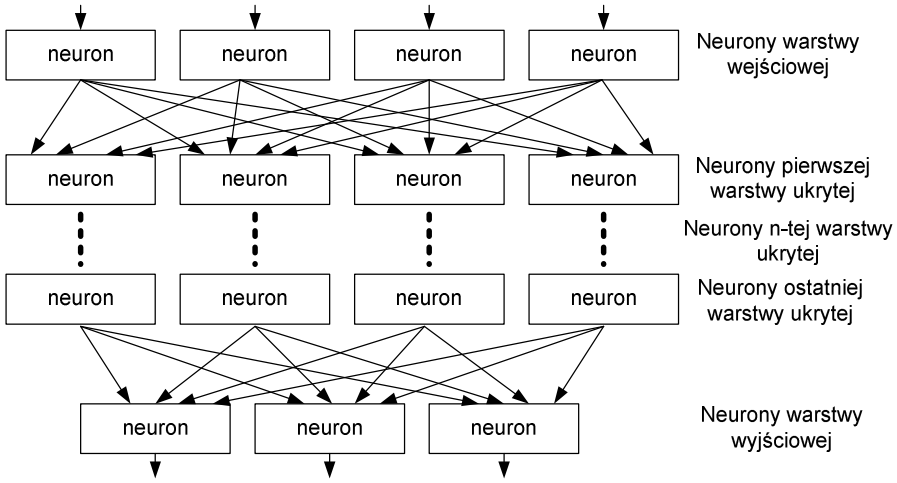
gdzie $d(n)$ to wartość różnicy pomiędzy odpowiedzią neurony na wymuszenie wektora $X(n)$ a wartością wzorca $y(n)$.

W procesie uczenia często wykorzystuje się również zbiór testujący, który służy do oceny stopnia nauczenia. Proces uczenia prowadzony jest do momentu aż błąd popełniany przez sieć dla sygnałów pochodzących ze zbioru testującego osiągnie założony (zadawalający) poziom. Możliwości pojedynczego neuronu są oczywiście bardzo ograniczone. Łączenia pojedynczych neuronów w sieci można dokonać na wiele sposobów.

3.3.4. Sieć neuronowa

W powyższych podrozdziałach opisany został matematyczny model neuronu, czyli elementarnej jednostki przetwarzającej informacje w sieci neuronowej. Na przestrzeni kilkudziesięciu lat powstało wiele modeli sieci neuronowych, dlatego w tym podrozdziale zostanie opisany elementarny podział struktur sieci neuronowych.

Sieć neuronowa jako układ przetwarzający sygnały posiada wejścia, poprzez które dostarczany są dane wejściowe oraz wyjścia, poprzez które sieć przekazuje wyniki swojej pracy. Odpowiada to budowie ludzkiego mózgu. Wejścia sieci reprezentują nerwy sensoryczne, natomiast wyjścia sieci neuronowej odpowiadają tak zwanym nerwom motorycznym. Sieć neuronowa może posiadać jeszcze pewną liczbę neuronów, które biorą udział w przetwarzaniu sygnałów wejściowych na wyjściowe. Do neuronów pośredniczących nie ma dostępu „z zewnątrz" sieci, stąd ten obszar sieci nazywany jest warstwą ukrytą.

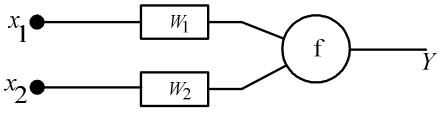


Rys. 3.4 Struktura ogólna sieci neuronowej.

Rysunek 3.4 prezentuje ogólny schemat wielowarstwowej, jednokierunkowej sieci neuronowej. Stosowane są również w rozwiązaniach praktycznych sieci, które realizują przetwarzanie informacji w dwóch kierunkach. Są to sieci ze sprzężeniem zwrotnym. Ponieważ w pracy tej nie są one wykorzystywane, więc nie będą szczegółowo opisywane. Tym bardziej, że w literaturze przedmiotu można znaleźć wiele informacji na ich temat. [39],[40],[41].

3.3.5. Adaptacyjne sieci logiczne

W powyższych rozdziałach opisana została koncepcja sieci neuronowych opartych na najpowszechniej stosowanym modelu neuronu zdolnym do przetwarzania danych o charakterze zmienno przecinkowym. Wartości wag również mogą przyjmować wartości o takim charakterze. W latach siedemdziesiątych XX wieku pojawiła się koncepcja adaptacyjnych sieci logicznych oraz sieci wielowarstwowych realizujących funkcje boolowskie [42]. Sieci takie składają się z binarnych neuronów operujących na danych postaci binarnej. Wejścia, wyjście oraz wagi takiego neuronu mogą przyjmować wartości 0 lub 1. Rysunek 3.5 przedstawia neuron binarny.



Rys. 3.5 Schemat neuronu boolowski.

Neuron binarny posiada zawsze tylko dwa wejścia, a blok f realizuje funkcję [43]:

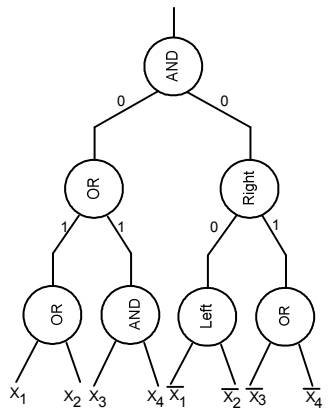
$$y = \begin{cases} 1, & dla \quad (w_1 + 1)x_1 + (w_2 + 1)x_2 \geq 2 \\ 0, & dla \quad (w_1 + 1)x_1 + (w_2 + 1)x_2 < 2 \end{cases}.$$

W sytuacji, gdy wartości wag i wejść neuronu mogą przyjmować jedynie wartości 0 lub 1, neuron będzie realizował przy danej kombinacji stanów wag oraz wejść jedną z czterech funkcji boolowskich. Zapisano to w tabeli 3.1.

Tabela 3.1 Neuron boolowski – realizowane funkcje.

W_1, W_2	X_1, X_2:Y	Realizowana funkcja
0,0	0,0:0 0,1:0 1,0:0 1,1:1	AND
0,1	0,0:0 0,1:1 1,0:0 1,1:1	LEFT
1,0	0,0:0 0,1:0 1,0:1 1,1:1	RIGHT
1,1	0,0:0 0,1:1 1,0:1 1,1:1	OR

Sieci konstruowane z takich neuronów to zwykle sieci wielowarstwowe posiadające jedno wyjście. Wejścia sieci są natomiast podwajane, poprzez negowanie. Przedstawia to rysunek 3.6.



Rys. 3.6 Struktura ogólna sieci neuronowej – boolowskiej.

Liczba wejść sieci jest zależna od rozmiaru wektorów wejściowych, na których sieć ma operować. W sieciach boolowskich każde wejście zwykłe (x_i) jest podwajane przez wejście negowane ($\overline{x_i}$). Wykorzystanie sieci zbudowanej z takich neuronów rozpoczyna się od inicjalizacji podczas, której następuje losowanie wartości wag poszczególnych neuronów. W rezultacie następuje przyporządkowanie realizowanych funkcji logicznych (AND, OR, Left, Right) do poszczególnych neuronów w sieci. Do procesu inicjalizacji sieci, jeszcze przed

ustalaniem wartości wag, następuje losowe łączenie każdego neuronu warstwy wstępnej z jednym wejściem prostym i jednym negowanym.

Proces uczenia (adaptacji) takiej sieci prowadzony może być według trzech różnych metod:

- modyfikacji poddawane są wagi neuronów,
- rozmiar sieci jest redukowany, przy pewnej nadmiarowości inicjalizowanej sieci,
- rozbudowa struktury sieci poprzez dodawanie pojedynczych neuronów lub małych podsieci.

Większość ze stosowanych sposobów modyfikacji wag lub architektury sieci wykorzystuje różne metody heurystyczne. W przypadku neuronowych sieci logicznych stosuje się również metody ewolucyjne, które wykorzystują do projektowania sieci boolowskich takie operacje jak:

- mutacja wag (zmiana wartości losowo dobieranej wagi),
- mutacja struktury (dodanie małej podsieci w losowym miejscu),
- krzyżowanie (tworzenie sieci potomnej na podstawie sieci rodzicielskich).

Wytrenowana sieć boolowska ma tę zaletę, że łatwo implementuje się tego rodzaju struktury, również za pomocą rozwiązań sprzętowych. Wykorzystanie jako elementarnych jednostek neuronów logicznych powoduje dużą wydajność tego rodzaju sieci. Tego rodzaju struktury nie wykonują żadnych z tak czasochłonnych operacji jak mnożenie, co ma miejsce w przypadku innego rodzaju neuronów. Prostota ta wprowadza jednocześnie ograniczenia co do potencjalnych zastosowań takich sieci.

Rozdział 4. Metody implementacji szyfrów

W pracy tej proponowane są rozwiązania pozwalające implementować algorytmy szyfrujące za pomocą sieci neuronowych. Współcześnie taka metoda nie jest wykorzystywana. Szyfrująca sieć neuronowa ma stanowić alternatywę dla znanych metod realizacji szyfrów. W rozdziale tym przedstawione zostały stosowane obecnie i historycznie metody implementacji algorytmów kryptograficznych.

Zaprojektowanie algorytmu szyfrującego, opisanie go w postaci funkcji matematycznych, przeprowadzenie badań mających na celu ocenę jego poziomu bezpieczeństwa to dopiero połowa sukcesu. Jeśli szyfr ma być używany do utajania informacji konieczna jest jego właściwa implementacja. Rozumieć należy przez to wskazanie odpowiednich narzędzi oraz opisanie sposobu ich użycia w celu realizacji algorytmu szyfrującego. Na przestrzeni długiej historii kryptografii pojawiało się wiele rozwiązań pozwalających na realizowanie konkretnych szyfrów. O wyborze sposobu korzystania z algorytmu kryptograficznego decyduje kilka czynników. Często sam algorytm determinuje sposób jego implementacji. Częste są również przypadki, że szyfr był projektowany z myślą o konkretnym sposobie realizacji. Ważnym czynnikiem jest również to, w jakich warunkach ma przebiegać szyfrowanie, kto i gdzie będzie szyfru używał. Inaczej problem przedstawia się w przypadku warunków np. pola walki czy działań tajnych agentów, a zupełnie inaczej gdy myślimy o szyfrowaniu elektronicznych przelewów bankowych. Z powyższego wynika również ograniczenie związane z technologią dostępna w określonych warunkach. Istotne jest również to, jakiego rodzaju, jakiej długości oraz jak długo informacja ma być chroniona. W kolejnych podrozdziałach przedstawiamy przegląd wybranych metod realizacji algorytmów szyfrujących.

4.1. Historyczne implementacje szyfrów

Kryptografia posiada wielowiekową historię. Krótka historia kryptografii przedstawiona została w rozdziale 1.1 tej pracy. Patrząc na to jak rozwijała się kryptografia, a w szczególności na sposoby realizacji szyfrów, ich jakość i funkcjonalność wyróżnić można dwa zasadnicze okresy: czasy sprzed ery komputerów i okres współczesny. Do realizacji pierwszych szyfrów przez cywilizacje egipskie, chińskie czy indyjskie wykorzystywano głównie takie narzędzia, jak: tabliczki gliniane, klinowe, kamień, papirus. Pierwszy przełom w rozwoju metod szyfrowania nastąpił w okresie gdy zaprzestano wykorzystywać hieroglify, pismo obrazkowe. W czasach gdy podstawowym nośnikiem informacji był pergamin i papier powstawały takie algorytmy jak szyfr Cezara (IX wiek), Vigenère'a (1553) czy Playfair (1854). Do stosowania ich wystarczyła znajomość algorytmu oraz coś na czym szyfrogram można było zapisać. W niektórych przypadkach, jak to ma miejsce np. w przypadku szyfru Vigenère'a, koniecznym było posiadanie tak zwanej tablicy alfabetów. Były to zapisane (np. na papierze) z różnym przesunięciem alfabety. Z biegiem czasu szyfry stawały się coraz bardziej złożone, rosła ilość czynności, jakie należało wykonać aby przekształcić tekst jawny w szyfrogram. Z czasem więc zaczęły się pojawiać rozwiązania sprzętowe, które miały wspomagać żmudny proces szyfrowania. Jednym z pierwszych takich urządzeń była, wykonana z dwóch okrągłych talerzy blachy miedzianej połączonych centralnie, tarcza Leone Battisty Albertiego [61]. Powstawały również inne rozwiązania takie, jak „Cryptograph Wheatstone'a" urządzenie w formie zegara zaprezentowane na Światowej wystawie w Paryżu w 1867 roku. Urządzenie szyfrujące M-94 Armii USA, mające postać cylindra, wprowadzone w 1922 roku i stosowane do 1942 roku, a w marynarce wojennej USA niemal do czasów współczesnych [1]. Wraz z rozwojem metod kryptografii rozwijała się również kryptoanaliza. Postęp w badaniach krytpoanalitycznych powodował, że szyfry musiały być coraz bardziej skomplikowane, pojawił się wymóg stosowania szyfrów polialfabetycznych. Jednocześnie pojawienie się radia wymusiło konieczność stosowania wydajnych i bezpiecznych metod szyfrowania. Przełom nastąpił wraz z początkiem budowy elektro-mechanicznych maszyn rotorowych wraz z najbardziej rozpoznawalną Enigmą.
Kolejnym kamieniem milowym implementacji szyfrów było wykorzystanie komputera.

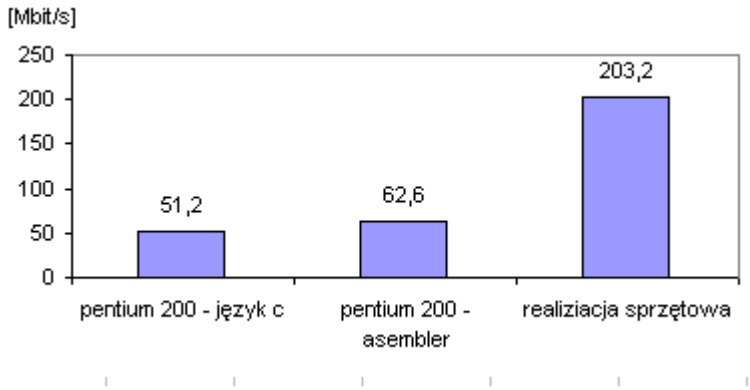
4.2. Współczesne implementacje programowe

Pojawienie się komputerów, a co ważniejszej, ich rozpowszechnienie spowodowało duże zapotrzebowanie na oprogramowanie umożliwiające szyfrowanie przechowywanych i przesyłanych informacji. Wraz z komputerem kryptografowie otrzymali narzędzie, dzięki któremu mogli implementować zupełnie nowe algorytmy szyfrujące. Jednak dostęp do takiej samej mocy obliczeniowej uzyskali również kryptolodzy. Metody kryptoanalizy również przeżyły rozkwit w erze komputerów. Zaczęto więc tworzyć programowe implementacje szyfrów. Z punktu widzenia bezpieczeństwa danego szyfru duże znaczenie ma nie tylko jakość samego algorytmu, ale również środowisko w jakim ten algorytm pracuje. Zatem implementacja algorytmu kryptograficznego ma kluczowe znaczenie dla bezpieczeństwa. W praktyce większość ataków na systemy szyfrowania wykorzystuje różnego rodzaju luki i niedociągnięcia w implementacjach. Przykładem tego może być atak na szyfr AES w implementacji OPENSSL [62]. Atak na sam algorytm, czyli zastosowanie kryptoanalizy jest trudnym przedsięwzięciem, o wiele prościej jest intruzowi poszukać luk w środowisku, w którym konkretny algorytm pracuje. Z drugiej jednak strony atak na algorytm jest dużo bardziej niebezpieczny, ponieważ fakt przeprowadzenia ataku np. z wykorzystaniem kryptoanalizy różnicowej [37] może pozostać niezauważony. Implementacje rozwiązań kryptograficznych mają szczególne znaczenie. Szyfry najczęściej wykorzystywane są w protokołach komunikacyjnych. O ile wprowadzanie zmian w oprogramowaniu pojedynczych stacji roboczych jest czymś prostym i powszechnym, o tyle modyfikacje protokołów sieciowych to już przedsięwzięcie o nieporównywalnej skali. Niezwykle ważnym jest więc pisanie bezpiecznych programów. Implementacje szyfrów wiążą się zwykle z konieczność zachowywania różnych danych w tajemnicy. Programista pisząc program, który realizuje algorytm szyfrujący musi brać pod uwagę na przykład konieczność przechowywania tajnych kluczy w pamięci operacyjnej, plikach czy bazach danych. W zakresie tworzenia oprogramowania powszechne jest dzisiaj korzystanie z bibliotek realizujących określone algorytmy kryptograficzne. Największa ilość bibliotek kryptograficznych dostępna jest w języku C, choć obecnie daje się zauważyć spory wzrost implementacji w języku Java[63]. Projektanci współczesnych algorytmów szyfrujących muszą brać pod uwagę nie tylko ich bezpieczeństwo ale również ich implementowalność zarówno programową, jak i z wykorzystaniem układów programowalnych. Często jest tak, że w toku badań i dyskusji środowiska kryptologów powstają dwie wersje szyfru. Jedna na potrzeby implementacji sprzętowej, a druga programowej. Przykładem może tu być rodzina szyfrów strumieniowych opartych na generatorze FCSR[64],[65]. W pracy [66]

zaproponowano konstrukcję FCSR dla konstrukcji sprzętowych, natomiast w pracy [67] proponowane są dwie wersje szyfru X-FCSR-128 i X-FCSR-256 dostosowane do implementacji programowych. Taka sytuacja jest coraz częstsza. Obecnie w światowym nurcie badań kryptologicznych prym wiodą publikacje dotyczące implementacji sprzętowych algorytmów szyfrujących.

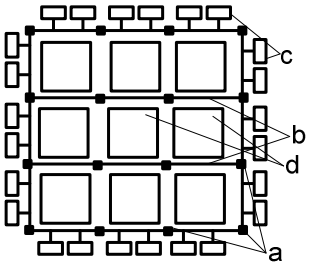
4.3. Współczesne implementacje sprzętowe

Implementacje sprzętowe algorytmów szyfrujących są obecnie bardzo często omawiane w publikacjach naukowych. Przeszukując zawartość baz publikacji z zakresu kryptologii, w okresie ostatnich kilku lat, znajdziemy więcej artykułów na temat realizacji sprzętowej szyfrów, niż prac dotyczących implementacji programowych. Implementacje sprzętowe posiadają wiele zalet w stosunku do rozwiązań programowych. Zasadniczą kwestią jest, szybkość realizacji przekształceń szyfrujących, pokazuje to wykres na rysunku 4.1 pochodzący z pracy [68].



Rys. 4.1 Porównanie wydajności implementacji szyfrów.

W ogólności implementacje sprzętowe są realizowane na dwa sposoby. Pierwszym są dedykowane układy scalone ASIC (Application Specific Integrated Circuit) projektowane do realizacji z góry określonego algorytmu oraz układy programowalne. Przykładem mogą być tutaj implementacje algorytmu AES, w pracy [69] omówiona została odmiana implementacji tego algorytmu w technologii ASIC. Implementacje w tej technologii cechują się dużą szybkością działania oraz najczęściej mieszczą się w jednym układzie scalonym. Natomiast wadą takie podejścia jest koszt wprowadzenia ewentualnych zmian, trzeba projektować układ od nowa. Alternatywą są struktury programowalne FPGA (Field Programmable Gate Array). W ogólnym przypadku taki układ składa się z konfigurowalnych bloków logicznych oraz połączeń pomiędzy nimi. Realizacje sprzętowe szyfrów projektowane są za pomocą języków opisu rozwiązań sprzętowych takich, jak np. VHDL. Następnie można przystąpić do realizacji w technologii ASCI lub FPGA. Różnice pomiędzy jedną a drugą technologią, dotyczące czasu działania, wynikają z faktu pojawiających się opóźnień z powodu programowalnej realizacji połączeń wewnątrz układu. Realizację dwurundowego algorytmu AES w strukturach programowalnych znaleźć można w [70]. Układy rekonfigurowane posiadają strukturę komórkową, pokazane jest to na rysunku 4.2.



Rys. 4.2 Struktura układu w architekturze FPGA.

Analogie do tej budowy można zaobserwować również w neuronowych układach szyfrujących, których konstrukcja jest przedmiotem tej pracy. W rozwiązaniach proponowanych w rozdziale 6 wykorzystano również koncepcję rekonfiguracji układu składającego się z pojedynczych komórek (neuronów) w celu realizacji konkretnego algorytmu szyfrującego.

Rozdział 5. Wykorzystanie sieci neuronowych w kryptologii

Sieć neuronowa (SN) jest to metoda matematyczna, która zaliczana jest przez współczesną naukę do grona narzędzi tak zwanej sztucznej inteligencji. Sposób wykorzystania sieci neuronowych, który opisane zostały w tej pracy nie traktuje ich jako sztucznej inteligencji. SN w rozważaniach tu prowadzonych to przede wszystkim układ adaptacyjny. Choć w niektórych przypadkach wykorzystywane są odwołania do właściwości SN, jaką jest zdolność do uogólniania i klasyfikacji obiektów. Pojawienie się w światowym nurcie badań prac, z zakresu wykorzystania sieci neuronowych w kryptologii, to w zasadzie ostatnie dwadzieścia lat. Wcześniej pojawiały się prace korzystające z metod obliczeniowych dziś określanych mianem metod sztucznej inteligencji. Pierwsze prace dotyczyły wykorzystania algorytmów ewolucyjnych do ataków na proste szyfry podstawieniowe [44],[45]. W pracy [46] zostały porównane metody wykorzystujące algorytmy genetyczne do ataków na szyfry wykorzystujące permutacje, podstawienia i transpozycje. Jedną z ciekawszych cytowanych tam koncepcji jest wykorzystanie algorytmów genetycznych do ekstrakcji klucza z szyfru opartego na koncepcji Feistela [47]. W 1996 roku pojawiła się propozycja użycia algorytmów genetycznych do ataku na szyfr Merkle-Hellman [48]. W 1997 roku pojawiła się praca przedstawiająca pomysł [49] na wykorzystanie metod genetycznych do przeszukiwania dużych przestrzeni kluczy dla czterowirnikowej maszyny rotorowej. Wymienione tu zostały jedynie wybrane zastosowania algorytmów genetycznych w kryptologii.

Praca ta dotyczy wykorzystania sieci neuronowych, więc to ich wykorzystaniu w kryptografii zostanie poświęcone więcej miejsca. Sporo publikacji dotyczy zastosowania sieci neuronowych w kryptografii asymetrycznej, w szczególności problemu logarytmu dyskretnego[50],[51]. Wiele algorytmów asymetrycznych (np. uzgadniane klucza Diffie-Hellman, szyfr El Gamal) wykorzystuje potęgowanie modularne w postaci:

$$a^x \bmod n.$$

Wyznaczanie wartości powyższego wyrażenia jest proste. Problemem odwrotnym do potęgowania modularnego jest wyznaczenie logarytmu dyskretnego:

$$wyznaczenie\ x\ gdzie\ \ a^x = b(\bmod n),$$

jest to zadanie bardzo trudne obliczeniowo. Formalna definicja logarytmu dyskretnego:

Jeśli G jest grupą skończoną, b elementem tej grupy oraz y jest takim elementem G, który jest potęgą b, to logarytmem dyskretnym elementu y o podstawie b jest taka liczba całkowita x, dla której $b^x = y$.

Problem liczenia logarytmu dyskretnego przy module będącym liczbą pierwszą jest porównywalny z faktoryzacją liczb składających się z dwóch czynników pierwszych, przybliżonej długości jak moduł logarytmu. W ostatnim czasie pojawiły się wyniki poszukiwania logarytmu dyskretnego z wykorzystanie SN uczonych według zasady wstecznej propagacji błędów. W pracy [52] zaprezentowane zostały wyniki dla modułów stanowiących małe liczby pierwsze (np. 19), gdzie skuteczność sieci neuronowych była 100 procentowa. Kolejne wyniki dla liczb większych pokazano w pracy [53], tu rezultaty były już na poziomie około 70 %. Innym zagadnieniem, poruszanym również w pracach [52],[53], jest wspomaganie za pomocą sieci neuronowych ataku na algorytm RSA. Zastosowano neuronową aproksymację funkcji Eulera. Rozpatrując szyfr RSA, opiera on swoje działanie na przekształceniu:

$$C = M^d \bmod n, \qquad n = p * q .$$

Funkcja Eulera $\varphi(N) = (p-1)(q-1)$ dla $N = p * q$, jest definiowana jako ilość liczb względnie pierwszych z liczbą N, nie większych od N, przy traktowaniu liczby 1 jako względnie pierwszą ze wszystkimi liczbami. Funkcja Eulera określana jest dla liczb całkowitych dodatnich.

O znaczeniu funkcji Eulera dla bezpieczeństwa szyfrowania i podpisu elektronicznego napisano w pracach [54] i [55]. Autorzy pracy [52] użyli sieci neuronowej do rozwiązania problemu faktoryzacji, wykorzystując do tego aproksymację funkcji Eulera. Celem wykorzystania sieci neuronowej w tym przypadku było uzyskanie klucza prywatnego na podstawie klucza publicznego. Wyniki wydają się być obiecujące jednak 100 % skuteczność udało się autorom uzyskać jedynie dla małych liczb.

Kolejnym obszarem zastosowania sieci neuronowych w kryptografii, jaki można znaleźć w pracach z ostatnich kilkunastu lat, jest wspomaganie kryptologii krzywych eliptycznych poprzez wykorzystanie sieci neuronowych. Kryptosystemy oparte na krzywych eliptycznych są obecnie przedmiotem badań kryptologów. Na potrzeby tworzenia kryptosystemów opartych na krzywych eliptycznych, wykorzystuje się krzywe eliptyczne silne kryptograficznie. Zdefiniowano je w pracy [56].

Krzywą eliptyczną E określoną nad ciałem $GF(p^n)$ nazywamy silnie kryptograficzną jeśli:

1) $n = 1$ lub n jest pierwsze,
2) #E($GF(p^n)$)=k*r, gdzie $r > 2^{150}$ i jest pierwsze, gcd(r,k)=1,
   gdzie: #E rząd krzywej eliptycznej, $1 \le k \le 2^8$,
3) $\left(p^n\right)^s - 1 \neq 1 \bmod r$ dla $1 \le s \le 10$,
4) jeśli $n > 1$ to współczynniki a i b w równaniu krzywej nie należą do ciała $GF(2)$.

W oparciu o teorię krzywych eliptycznych budowane są krytposystemy asymetryczne. Bezpieczeństwo takich systemów determinowane jest poprzez trudność obliczenia logarytmu dyskretnego na tych krzywych. Przyczyn popularności krzywych eliptycznych we współczesnym nurcie badań kryptograficznych można dopatrywać się głównie w tym, że ich zastosowanie pozwala zmniejszyć około 6 krotnie rząd wielkości liczb stosowanych w bezpiecznych rozwiązaniach kryptografii asymetrycznej. Sieci neuronowe natomiast, w przypadku krzywych eliptycznych, znalazły zastosowanie jako narzędzia wspierające kryptoalizę [57]. Szczególną inspiracją do podjęcia tematyki badawczej, która jest przedmiotem tej pracy, związanej z wykorzystaniem sieci neuronowych do realizacji algorytmów szyfrujących była właśnie praca [17]. Wykorzystanie sieci neuronowej w protokole uzgadniania klucza pokazało, że sieci neuronowe mogą być wykorzystana do realizacji zadań precyzyjnych. Podstawową cechą SN jest ich zdolność do uogólniania prezentowanych wyników. W pracy [17] wykorzystano sieć neuronową opartą na modelu dwuwarstwowego perceptronu i regule uczenia Hebba. Nadawca i odbiorca posługują się dwiema sieciami o nieznanych warunkach początkowych i prowadząc proces uczenia przy pomocy publicznego zbioru uczącego, w rezultacie mogą ustalić wspólnych klucz prywatny. Pokazuje to, że zastosowanie sieci neuronowych w kryptografii jest jak najbardziej uzasadnione i możliwe. Kolejnym obszarem zastosowania sieci neuronowych, odnotowanym w ostatniej pracy E.C.Laskari „Cryptography and Cryptanalysis Through Computational Intelligence", jest wykorzystanie SN jako narzędzie wspomagającego projektowanie algorytmów szyfrujących. Wstępną koncepcję tego podejścia przedstawiliśmy w pracy [58]. Koncepcje własne, realizacji algorytmów szyfrujących za pomocą sieci neuronowych przedstawiliśmy między innymi w pracach [59][60]. Wyniki tam zamieszczone stały się punktem wyjścia dla badań, których rezultaty przedstawione są w tej pracy.

Śledząc dzisiejszy stan badań w zakresie wykorzystywania narzędzi sztucznej inteligencji w kryptografii widać, że jest to dopiero obiecujący początek dla dalszych poszukiwań.

Rozdział 6. Realizacja elementarnych funkcji kryptograficznych za pomocą sieci neuronowych

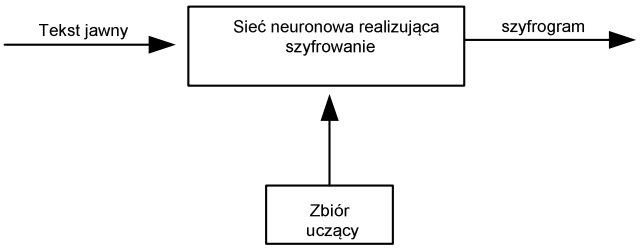
6.1. Wprowadzenie

Wiele popularnych protokołów kryptograficznych, takich jak SSL i SSH oraz oprogramowanie do realizacji usług kryptograficznych posiadają kilka zaimplementowanych algorytmów kryptograficznych. Przyczyn tego rodzaju rozwiązania jest kilka, jednak istotną, z punktu widzenia ciągle rozwijających się metod ataków jest możliwość wyboru z pośród kilku algorytmów szyfrujących. Dobrym przykładem jest tutaj algorytm DES, którego słabości zostały udowodnione. Szyfr ten jest wykorzystywany w realizacji wielu protokołów kryptograficznych, obecnie stosowanych. Jednak większość z nich ma zaimplementowanych kilka innych szyfrów, dzięki czemu użytkownik może zrezygnować z wykorzystywania DES-a.

Problem reagowania na pojawiające się nowe metody ataków na znane szyfry musi być obecnie brany pod uwagę przez twórców wszelkiego rodzaju protokołów i aplikacji kryptograficznych.

Na przestrzeni kilku ostatnich lat pojawiły się publikacje na temat wykorzystania układów programowalnych w kryptografii do realizacji algorytmów kryptograficznych. Dla implementacji sprzętowych ma to szczególne znacznie, ponieważ zmiana algorytmu realizowanego sprzętowo jest szczególnie kłopotliwa i kosztowna w porównaniu z implementacją programową. Dlatego w pewnym sensie naturalnym wydaje się wykorzystanie układów programowalnych. Prace z tego zakresu zostały przytoczone w rozdziale 2.1. Szerzej na temat tego podejścia do realizacji szyfrów oraz ogólnie na temat wykorzystania układów programowalnych w przetwarzaniu sygnałów napisano w [71], [72].

W pracy tej jako układ szyfrujący próbujemy wykorzystać sieć neuronową. Pierwsze skojarzenie możliwości wykorzystania sieci neuronowej do szyfrowania prezentujemy w następnym akapicie. Jest to przykład nieco przewrotny, ale jest najprostszą odpowiedzią stwierdzającą czy sieć neuronową można wykorzystać do realizacji algorytmów szyfrujących. Celem tej pracy jest konstrukcja sieci neuronowej, która byłaby w stanie realizować różne algorytmy szyfrujące. Zmiana realizowanego algorytmu ma następować po przeprowadzonym w określonych warunkach procesie uczenia. Rysunek 6.1 prezentuje w dużym uproszczeniu to, do czego zmierzają badania prowadzone na potrzeby tej pracy.

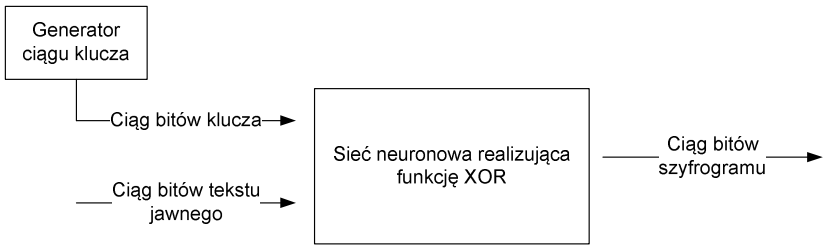


Rys. 6.1 Schemat ogólny neuronowego układu szyfrującego.

Koncentrując się na algorytmach szyfrujących z rodziny algorytmów symetrycznych wybrany został problem realizacji poprzez sieć neuronową dwóch podstawowych przekształceń kryptograficznych: permutacji i podstawienia, rozdział 3.1. Koncepcję realizacji permutacji z wykorzystaniem sieci neuronowej pokazano w rozdziale 6.2. Proponowane są dwa podejścia do konstrukcji permutującej sieci neuronowej. Pierwsze opiera się na koncepcji neuronu, którego wagi przyjmować mogą wartości rzeczywiste, drugie natomiast wykorzystuje neuron boolowski. Zasadę ich działania opisano w rozdziale 6.2. W rozdziale 6.3 przedstawiono propozycję rozwiązania problemu realizacji z wykorzystaniem sieci neuronowej operacji podstawienia (S-bloku).

Czy wykorzystanie sieci neuronowych do szyfrowania jest w ogóle możliwe ?

Rozumiejąc problematykę i charakter kryptografii można by dojść do wniosku, że zastosowanie sieci neuronowych jako narzędzia kryptograficznego jest z góry skazane na niepowodzenie, ponieważ kryptografia wymaga dużej jednoznaczności i dokładności. Sieci neuronowe w ogólnym swoim charakterze są metodami, które dają odpowiedzi bardziej ogólne lub przybliżone do wartości dokładnych. Jednak z drugiej strony znane są konstrukcje sieci neuronowych, które realizują funkcje boolowskie (mowa o tym była wcześniej) takie, jak chociażby suma modulo 2 ($\oplus$). Zatem na pytanie, czy możliwym jest skonstruowanie szyfrującej sieci neuronowej można zaryzykować odpowiedź twierdzącą, ograniczając się na razie w rozważaniach tylko do operacji $\oplus$ i jej realizacji za pomocą sieci neuronowej, to wystarczy do realizacji szyfru. Mowa tu o opisanym wcześniej, w drugim rozdziale, szyfrze Vernama z 1917 r. Jego podstawą jest właśnie funkcja logiczna $\oplus$. Posiadając nauczoną sieć neuronową, rozwiązującą problem alternatywy wykluczającej można ją wykorzystać w realizacji algorytmu szyfrującego „One-time pad". Ogólny schemat prezentujący tą koncepcję przedstawiony jest, na rysunku 6.2.



Rys. 6.2 Schemat ogólny sieci neuronowej realizującej „One-time pad”.

Rysunek 6.2 oraz opisana koncepcja pozwala stwierdzić, że sieć neuronowa może zostać wykorzystana do realizacji algorytmów szyfrujących. Użyteczność tego jest jednak niewielka ze względu na fakt, że algorytm „One-time pad” jest traktowany jako teoretyczny szyfr doskonały. W praktyce wykorzystanie go jest dość kłopotliwe. Pytanie więc jest następujące: Czy możliwe jest wykorzystanie sieci neuronowych do czegoś więcej niż tylko do realizacji funkcji opisanej powyżej?

6.2. Realizacja permutacji

Pomysł wykorzystania sieci neuronowej jako uniwersalnego układu szyfrującego, opiera się na konstrukcji takich sieci neuronowych, które będą w stanie realizować elementarne przekształcenia szyfrujące. W pierwszej kolejności zostanie zaprezentowana koncepcja realizacji za pomocą sieci neuronowej permutacji.

6.2.1. Permutacja dwóch bitów

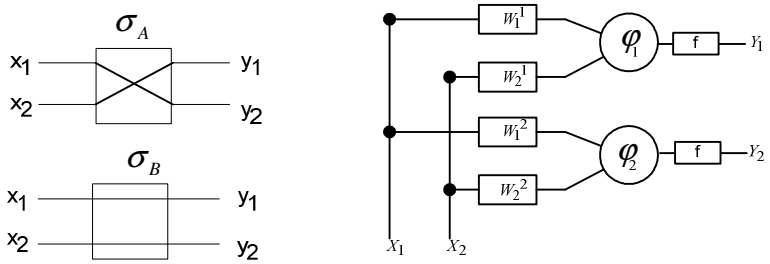
Celem jest przygotowanie sieci neuronowej (czarnej skrzynki), która będzie realizowała permutację dwóch bitów. Sieć będzie posiadać wobec tego dwa wejścia i dwa wyjścia. Permutacje realizowane przez sieć zapisane zostały poniżej:

- układ realizuje permutację, $\sigma_A = \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$,

- układ nie realizuje permutacji, $\sigma_B = \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}$,

gdzie : σ_A, σ_B - permutacja, x, y – odpowiednio: wejście, wyjście układu permutującego.

Posługując się odpowiednio skonstruowanym zbiorem uczącym chcemy mieć możliwość decydowania o tym czy sieć będzie realizowała permutacje σ_A czy σ_B. Na rysunku 6.3 przedstawiona jest sieć zbudowana z dwóch neuronów. Po lewej stronie przedstawiona jest reprezentacja symboliczna, która pokazuje rodzaj realizowanych permutacji. Po prawej stronie rysunku 6.3 przedstawiona została struktura sieci, która realizuje dane przekształcenie.



Rys. 6.3 Schemat sieci neuronowej realizującej permutację dwóch bitów.

Sieć pokazana na rysunku 6.3, składa się z neuronów, których działanie można opisać następująco:

$$\varphi_n = \sum w_i^n x_k \,, \qquad f = \begin{cases} 1, & \varphi > p \\ 0, & \varphi < p \end{cases}, \quad y_l = f(\varphi_n)\,,$$

gdzie : N – nr neuronu, i – nr wejścia neuronu, k – nr wejścia sieci, l – nr wyjścia sieci, φ_n - potencjał membranowy.

Zmiana realizowanej permutacji, pomiędzy σ_A i σ_B dokonywana jest z wykorzystaniem przedstawionych w tabeli 6.1, zbiorów uczących.

Tabela 6.1 Zbiór uczący dla permutacji A i B.

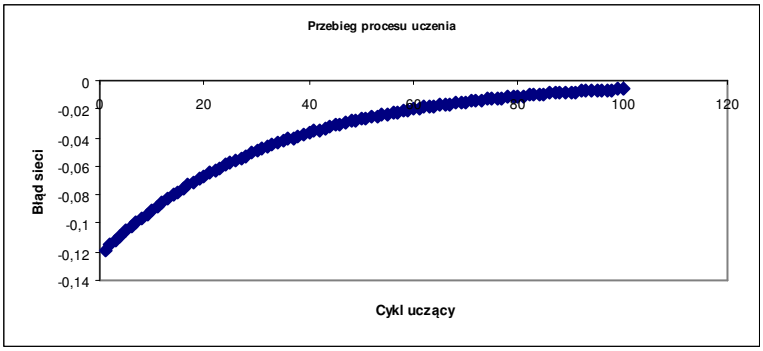
σ_A				σ_B			
x_1	x_2	y_1	y_2	x_1	x_2	y_1	y_2
0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	1
1	0	0	1	1	0	1	0
1	1	1	1	1	1	1	1

Proces uczenia prowadzony jest według modelu uczenia z nauczycielem, co opisano w rozdziale 3.2. Zbiory uczące przedstawione są powyżej. Korekta wag odbywa się według wzoru:

$$w_i^n(c+1) = w_i^n(c) + \eta d(c) x_i^n(c) \quad \text{, gdzie}$$

d – różnica pomiędzy sygnałem wyjściowym a wzorcem,
η – współczynnik uczenia,
C – aktualna wartość,
C+1 – wartość po korekcie.

Przełączanie układu, pomiędzy realizacją jednej i drugiej permutacji, odbywa się poprzez przeprowadzenie uczenia sieci z wykorzystaniem odpowiedniego zbioru uczącego. Przebieg uczenia pokazany jest na rysunku 6.4, przedstawiającym zmieniającą się wartość błędu popełnianego przez sieci.



Rys. 6.4 Proces uczenia sieci permutującej.

6.2.2. Permutacja trzech bitów

Sam moduł dokonujący permutacji na dwóch bitach to zbyt mało, aby można było z kilku takich modułów skonstruować sieć neuronową dokonującą dowolnej permutacji dla bloku składającego się z kilkudziesięciu bitów. W tym podrozdziale opisano propozycję realizacji permutacji trzech bitów za pomocą sieci neuronowej. Ogólne założenie jest takie samo jak w przypadku poprzednim: przygotować sieć neuronową (czarną skrzynkę), która będzie realizowała permutację, tym razem, trzech bitów.

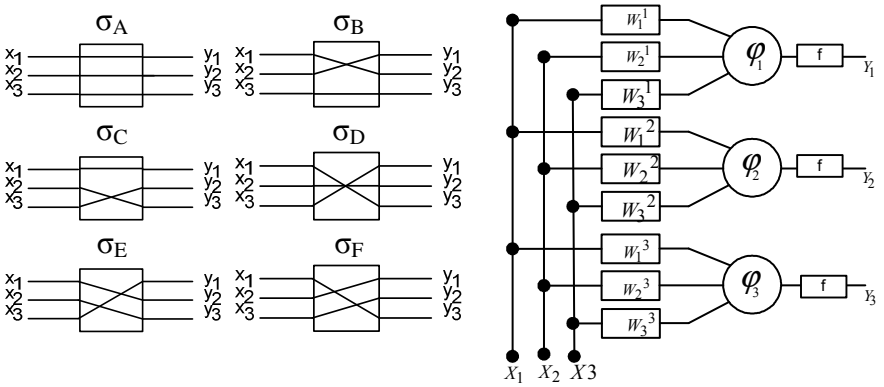
W porównaniu do przypadku z poprzedniego podrozdziału, dla trzech bitów możliwości permutacji jest zdecydowanie więcej:

$$\sigma_A = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}, \quad \sigma_B = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix},$$

$$\sigma_C = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}, \quad \sigma_D = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix},$$

$$\sigma_E = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}, \quad \sigma_F = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}.$$

Sieć neuronowa zdolna do realizacji wyżej wypisanych permutacji przedstawiona jest na rysunku 6.5.



Rys. 6.5 Schemat sieci neuronowej realizującej permutację trzech bitów.

Sieć dla trzech bitów składa się z takich samych neuronów jak sieć dla dwóch bitów. Również proces uczenia przebiega w analogiczny sposób jak w przypadku dwóch bitów. Zbiory uczące dla sieci, za pomocą których realizowane jest przełączanie układu neuronowego pomiędzy permutacjami σ, realizowane jest za pomocą następujących zbiorów uczących budowanych według zasady:

Tabela 6.2 Zbiór uczący dla permutacji trzech bitów.

A					
x1	**x2**	**x3**	**y1**	**y2**	**y3**
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	1

B					
x1	**x2**	**x3**	**y1**	**y2**	**y3**
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	1	0	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	1	1

C					
x1	**x2**	**x3**	**y1**	**y2**	**y3**
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	1

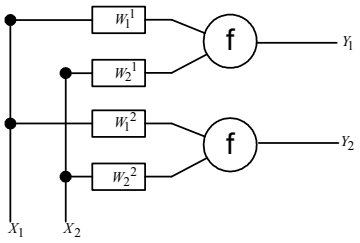
Korzystając z powyższych zbiorów uczących możliwe jest decydowanie o tym, jaką permutację ma realizować w danej chwili układ. Zarówno element dwu i trzybitowy to układy elementarne, za pomocą których możliwe jest budowanie większych sieci, które mogą realizować permutacje na większych blokach bitów tekstu jawnego.

6.2.3. Permutacja realizowana na sieci boolowskiej

Założenie jest takie: skonstruować blok, który będzie realizował dowolną permutację na dwóch bitach tekstu jawnego. W celu konstrukcji takiego bloku należy zbudować sieć neuronową, która będzie posiadała dwa wejścia i dwa wyjścia. Sieć ma umożliwić realizację następujących permutacji:

$$\sigma_A = \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \quad , \sigma_B = \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix},$$

gdzie σ_A, σ_B - to permutacja, x, y – odpowiednio: wejście, wyjście układu permutującego. Posługując się odpowiednio skonstruowanym zbiorem uczącym chcemy mieć możliwość decydowania o tym czy sieć będzie realizowała permutacje σ_A czy σ_B. Na rysunku 6.6 przedstawiona jest sieć zbudowana z dwóch neuronów boolowskich, realizująca permutację dwóch bitów.



Rys. 6.6 Schemat logicznej sieci neuronowej realizującej permutację dwóch bitów.

Sieć składa się dwóch takich samych neuronów. Zmiana realizowanej permutacji, pomiędzy σ_A i σ_B, dokonywana jest z wykorzystanie zbiorów uczących zapisanych w tabeli 6.3:

Tabela 6.3 Zbiór uczący dla permutacji dwóch bitów.

permutacja σ_A:

x_1	x_2	y_1	y_2
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

, permutacja σ_B:

x_1	x_2	y_1	y_2
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

.

W wyniku przeprowadzonego procesu uczenia zmianie ulegają wartości wag dla:

σ_A wagi sieci: $w_1^1 = 0;\quad w_2^1 = 1;\quad w_1^2 = 1;\quad w_2^2 = 0$,

σ_B wagi sieci: $w_1^1 = 1;\quad w_2^1 = 0;\quad w_1^2 = 0;\quad w_2^2 = 1$.

Proces uczenia przebiega według zależności:

$$d = y_w - y,$$

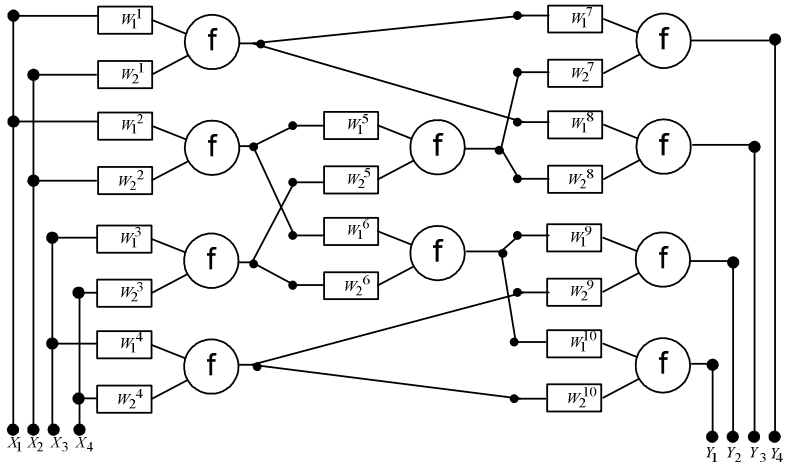
$$w_i^j(n+1) = w_i^j(n) + d * X_i^j, \text{ gdzie:}$$

d– różnica pomiędzy odpowiedzią wzorcową sieci, a rzeczywistą,
y_w - odpowiedź wzorcowa sieci,
y – odpowiedź rzeczywista sieci,
i – nr wejścia neuronu , j – nr neuronu,
x – wejście neuronu,
w – waga neuronu, (n+1) – cykl uczący bieżący a (n) poprzedni.

Zmiana realizowanej permutacji odbywa się poprzez przeprowadzenie uczenia sieci z wykorzystaniem odpowiedniego zbioru uczącego.

6.2.4. Permutacja czterech bitów na sieci boolowskiej

W poprzednich rozdziałach, gdzie wykorzystano sieci neuronowe oparte na neuronach McCulloch-Pittsa oraz regule Hebba, przedstawiono koncepcję realizacji bloku dokonującego permutacji trzech bitów. W przypadku sieci wykorzystującej neurony logiczne, które mogą posiadać tylko dwa wejścia, jako drugi proponowany jest neuronowy blok permutacji czterech bitów. Na rysunku 6.7 przedstawiono neuronową sieć boolowską, realizującą permutację czterech bitów:



Rys. 6.7 Permutacja czterech bitów – neuronowa sieć boolowska.

W tabeli 6.4 przedstawiony jest przykład zbioru uczącego dla jednej permutacji na czterech bitach.

Tabela 6.4 Zbiór uczący dla permutacji czterech bitów.

x1	x2	x3	x4	y1	y2	y3	y4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	1	0	0
0	0	1	1	0	1	0	1
0	1	0	0	0	0	1	0
0	1	0	1	0	0	1	1
0	1	1	0	0	1	1	0
0	1	1	1	0	1	1	1
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	1

$$\sigma_A = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 2 & 4 \end{pmatrix}$$

Po przeprowadzonym procesie uczenia, zbiorem zapisanym w tabeli 6.4, sieć przyjmie wartości wag dla permutacji σ_A:

$$w_1^1 = 0; w_2^1 = 1; w_1^2 = 1; w_2^2 = 0; w_1^3 = 0; w_2^3 = 1; w_1^4 = 1; w_2^4 = 0; w_1^5 = 0; w_2^5 = 1;$$
$$w_1^6 = 1; w_2^6 = 0; w_1^7 = 0; w_2^7 = 1; w_1^8 = 1; w_2^8 = 0; w_1^9 = 0; w_2^9 = 1; w_1^{10} = 1; w_2^{10} = 0.$$

6.3. Realizacja S-bloku

Drugim przekształceniem szyfrującym powszechnie występującym we współczesnych algorytmach szyfrujących jest nieliniowe podstawienie. Podstawienie realizowane jest właśnie za pomocą S-bloków. Zmierzając więc do tego aby realizować algorytm szyfrujący za pomocą układu neuronowego, kolejnym etapem tej pracy jest zaproponowanie sposobu realizacji S-bloku za pomocą sieci neuronowej. S-blok zdefiniowany został w rozdziale 3 tej pracy, wobec tego od razu dokonane zostanie omówienie problemu jego realizacji.

Działanie S-bloku może być interpretowane jako działanie wybierania z tablicy określonej wartości przechowywanej w komórce o określonym adresie. S-bloki stosowane w praktycznych implementacjach składają się z tablicy o kilku kolumnach i wierszach. W przypadku algorytmu DES jest to tablica w wymiarach 4 wiersze i 15 kolumn. Adres pojedynczej komórki składa się z 6 bitów, dwa skrajne określają nr wiersza a 4 środkowe nr kolumny. Jeśli natomiast przyjrzymy się aktualnemu standardowi szyfrowania jakim jest szyfr AES, to stosowane tam S-bloki stanowią tablicę o szesnastu wierszach i kolumnach. Adres pojedynczej komórki w takim S-bloku składa się z ośmiu bitów. W następnych rozdziałach przedstawiona zostanie propozycja realizacji za pomocą sieci neuronowej S-bloku wykorzystywanego w algorytmie DES.

Sama realizacja S-bloku za pomocą sieci neuronowej to nie jedyny cel. Konstrukcja neuronowa ma umożliwić zmianę zawartości S-bloku poprzez dostarczenie określonych informacji do użytkownika urządzenia lub programu szyfrującego. Informacje te mają być przysyłane w postaci jawnej, a ich przechwycenie przez intruza nie może ułatwiać krytpoanalizy. Z punktu widzenia potencjalnych zastosowań, będzie o nich mowa w rozdziale 8. Cel ten jest bardzo ważny, nawet jeśli miałby on być osiągnięty poprzez zwiększenie rozmiarów sieci.

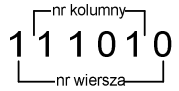
6.3.1. Realizacji jednego wiersza S-bloku

Chcąc realizować S-blok z algorytmu DES, problem można podzielić na dwa etapy. Pierwszy to realizacja pojedynczego wiersza, drugi to rozbudowa zaproponowanej metody do pełnego S-bloku (4 wiersze). Jeśli rozpatrzymy najpierw problem realizacji jednego wiersze to sieć neuronowa powinna dać możliwość realizacji wybierania wskazanej wartości z tablicy. W tabeli 6.5 pokazany jest S-blok nr 1 algorytmu DES, pierwszy wiersz tej tabeli jest przedmiotem rozważań w tym podrozdziale.

Tabela 6.5 Zawartość S-blok nr I algorytmu DES.

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

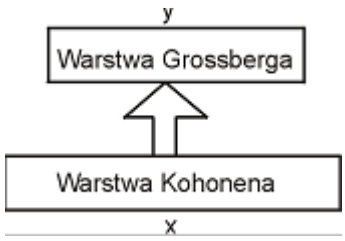
Ciąg wejściowy S-bloku to 6 bitów:



.

Tak więc wobec powyższego, adresowanie komórki w obrębie jednego wiersza odbywa się z wykorzystaniem 4 bitów. Pierwszym zadaniem jest rozwiązanie problemu realizacji za pomocą sieci neuronowej wybierania konkretnej wartości z tablicy 1x16.

W literaturze dotyczącej sieci neuronowych znana jest koncepcja sieci CP [39], tak zwanej sieci przesyłającej żeton. Ogólna budowa tej sieci przedstawiona jest na rysunku 6.8.



Rys. 6.8 Koncepcja budowy sieci CP.

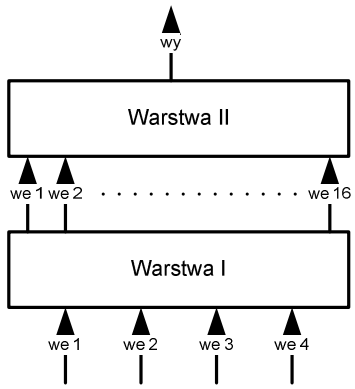
Sieć ta składa się z dwóch warstw posiadających odrębne zadania. Warstwa Kohonena działa na zasadzie konkurencyjności. Wartość jeden pojawia się na wyjściu jednego neuronu.

$$y_i = \begin{cases} 1 & dla \max(\varphi_i) \\ 0 & w \quad inny \quad wypadku \end{cases}, \; gdzie \; \varphi_i = x * w^T.$$

Druga warstwa sieci jest adresowana przez warstwę pierwszą. O wartości na wyjściu sieci CP ostatecznie decyduje warstwa Grossberga.

Pomysł na realizację pojedynczego wiersza S-bloku za pomocą sieci neuronowej bazuje na ogólnej koncepcji sieci CP. Opisując proponowaną konstrukcję sieci neuronowej dla przejrzystości opisu założono, że sieć ma udzielać odpowiedzi w postaci dziesiętnej. Ma to również znaczenie w osiągnięciu celu numer 2, opisanego we wstępie do tego rozdziału. W dalszej części pracy dotyczącej realizacji konkretnego przykładowego szyfru pokazana zostanie sieć, które na wyjściu podaje wynik w postaci binarnej. W toku badań i eksperymentów okazało się, że to czy sieć ma udzielać odpowiedzi w postaci binarnej czy dziesiętnej, jest kwestią konstrukcji ostatniej warstwy sieci.

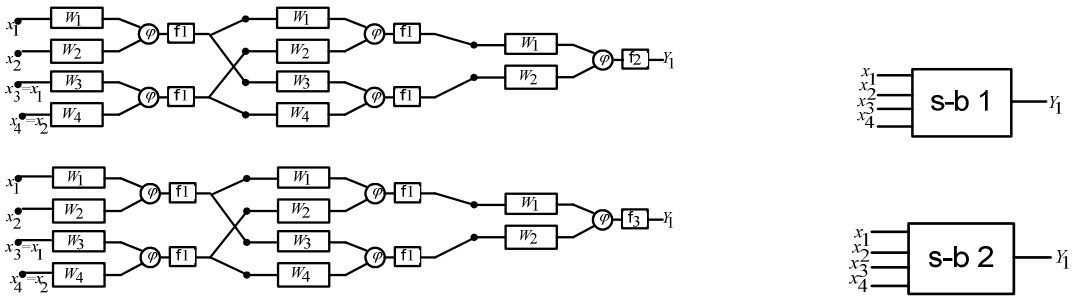
Wychodząc więc z ogólnej koncepcji sieci CP, strukturę układu neuronowego realizującego jeden wiersz S-bloku można przestawić za pomocą schematu blokowego, pokazanego na rysunku 6.9:



Rys. 6.9 Schemat budowy sieci realizującej pierwszy wiersz S-bloku.

Warstwa I miałaby realizować klasyfikację obiektów określanych poprzez odpowiednią kombinację 4 bitów na 16 różnych klas (1 wiersz S-bloku posiada 16 komórek). Skupiając się na pierwszej warstwie, chodzi tu o konstrukcję sieci neuronowej, która dla różnych kombinacji czterech bitów wejściowych dla poszczególnych sygnałów będzie odpowiadała jedynką tylko na jednym wyjściu. Wykorzystane do tego zostaną 4 sieci neuronowe, z których każda będzie realizowała klasyfikację obiektu na 4 klasy przy 4 cechach podawanych na wejście.

Podobnie jak było to rozwiązane w przypadku realizacji permutacji również i przy neuronowej realizacji S-bloku stosowane są do budowy bloki neuronowe, które można nazwać podsieciami. Rozwiązanie takie zostało przyjęte z dwóch powodów. Pozwala w czytelnej i zwartej formie przedstawiać strukturę sieci, a modułowość ułatwia implementację. Ponadto modułowa budowa ułatwia ewentualną rozbudowę rozmiaru S-bloku. Tak więc sieć pozwalająca na klasyfikację na 4 klasy wykorzystuje dwa bloki elementarne, przedstawione na rysunku 6.10.



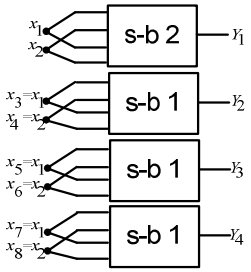
Rys. 6.10 Neuronowe klasyfikatory elementarne.

Bloki „s-b 1” oraz „s-b 2” składają się z neuronów z dwiema różnymi funkcjami aktywacji:

$$f_1 = \varphi,\ f_2 = \begin{cases} 0, & \varphi \le p_1 \\ 1, & \varphi > p_1 \end{cases}, \text{ oraz } f_3 = \begin{cases} 0, & \varphi \ge p_2 \\ 1, & \varphi < p_2 \end{cases},$$

gdzie: $\varphi = \sum w_i x_i$, $y_j = f_k(\varphi)$, $p_1 = 0{,}5$, $p_2 = 0{,}2$, f – funkcja aktywacji, i – nr wejścia/wagi neuronu, φ_n - potencjał membranowy, p - próg funkcji aktywacji, y_j - wyjście kolejnego neuronu, k - nr funkcji aktywacji.

Korzystając z układu powyższych bloków, rysunek 6.10, tworzymy sieć, która działa według tablicy prawdy pokazanej poniżej. Trzymając się zasady konstrukcji bloków elementarnych sieć realizująca poniższą tablicę prawdy oznaczona zostanie jako „4-kl” i pokazana na rysunku 6.11.



x1	x2	y1	y2	y3	Y4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Rys. 6.11Realizacja rozpoznawania czterech obiektów.

Można oczywiście dyskutować z faktem złożoności tej sieci realizującej dość proste zadanie. Jednak, taki układ wydaje się optymalny pod względem dalszych jego zastosowań i wymogów stawianych przed omawianym w dalszej części pracy neuronowym układem szyfrującym.

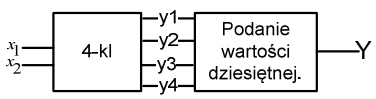
Pojedynczy blok 4-kl realizuje klasyfikację na cztery klasy. Jednak celem rozważań w tym rozdziale jest rozwiązanie problemu klasyfikacji na 16 klas (16 elementowy wiersz S-bloku). Wobec tego skonstruowano sieć składającą się z 4 bloków 4-kl. Sieć ta posiadać będzie 16 wyjść. Biorąc pod uwagę tablicę prawdy dla pojedynczego bloku 4-kl, rysunek 6.11, na wyjściu całej sieci możliwe jest pojawienie się zestawów odpowiedzi przedstawionych w tabeli 6.6.

Tabela 6.6 Wyjście I warstwy sieci realizującej jeden wiesz S-bloku.

y1	y2	Y3	Y4	y5	y6	y7	y8	y9	y10	y11	y12	y13	y14	y15	y16
1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

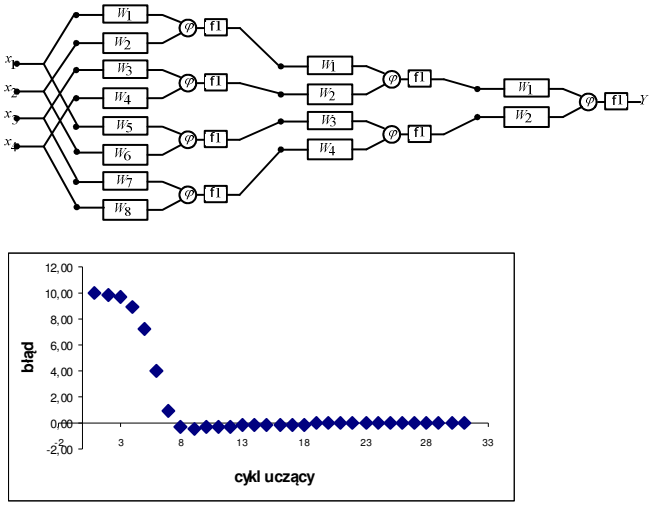
Z powyższego wynika, więc że w tabeli 6.6 zgromadzone zostały ciągi bitów wejściowych dla drugiej warstwy sieci. Na tym etapie rozważań druga warstwa sieci ma realizować podawanie konkretnej wartości (0-15). Dla zaprezentowania ogólnej idei budowy takiej sieci zaczniemy od przykładu elementarnego jakim jest sieć podająca 4 różne wartości. Przykład ten został akurat wybrany ponieważ wskazanie czterech różnych klas realizowane jest przez pojedynczy blok elementarny 4-kl, rysunek 6.11. Zatem problem sprowadza się do tego, aby układ neuronowy realizował tablicę prawdy:

Tabela 6.7 Tablica prawdy, rozpoznawanie czterech klas.



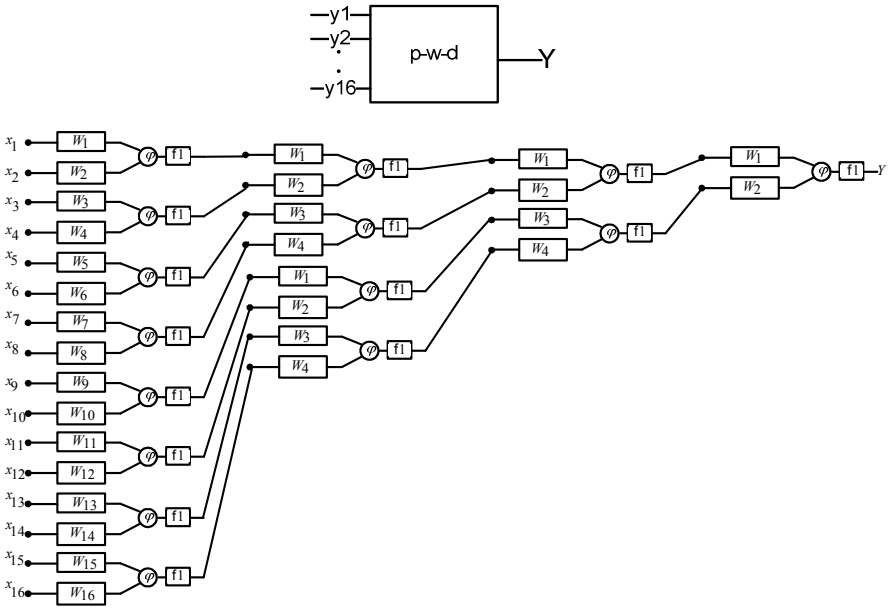
x1	x2	y1	y2	y3	y4	Y
0	0	1	0	0	0	0
0	1	0	1	0	0	1
1	0	0	0	1	0	2
1	1	0	0	0	1	3

Ważny założeniem jest również to, aby gotowa sieć neuronowa mogła realizować S-blok o dowolnej zawartości. Zmiana realizowanego S-bloku powinna być dokonana w możliwie szybki i wygodny sposób bez konieczności zmiany struktury sieci. Po wielu eksperymentach z różnymi konstrukcjami, sieć realizująca podanie czterech różnych wartości przybrała postać jak na rysunku 6.12. Poniżej przedstawiony został również wykres obrazujący przebieg procesu uczenia sieci.



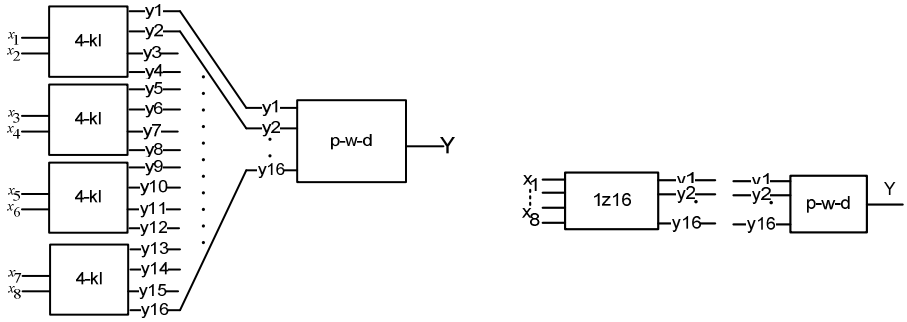
Rys. 6.12 Sieć realizująca podawanie 4 lub 8 wartości dziesiętnych na wyjściu.

Tego rodzaju sieć jest w stanie realizować podanie ośmiu różnych wartości na wyjściu biorąc pod uwagę możliwości sygnałów pojawiających się na wyjściu warstwy pierwszej oraz zapewnienia odpowiedniej separacji (odległości na osi liczbowej) pomiędzy udzielanymi odpowiedziami. Sieć realizująca podawanie na wyjściu wartości od 0 do 15 musi być już rozbudowana. Tu ponownie po przeprowadzeniu wielu prób i eksperymentów dobrany został kształt sieci realizujący podawanie 16 różnych wartości dziesiętnych na wyjściu. Sieć ta pokazana jest na rysunku 6.13. Ponieważ na wyjściu sieci nie znajduje się funkcja progowa, to ważne jest takie dobranie struktury sieci, aby odpowiedzi sieci były podawane z odpowiednią dokładnością.



Rys. 6.13 Sieć realizująca podawanie na wyjściu wartości od 0 do 15.

Został rozwiązany problem realizacji za pomocą sieci neuronowej problemu adresowania komórek wiersza S-bloku oraz rozwiązany problem podawania na wyjściu sieci wartości stanowiącej zawartość komórki o danym adresie. Można teraz przystąpić do złożenia sieci realizującej działanie pojedynczego wiersza S-bloku. Schemat ideowy przedstawiono na rysunku 6.14.

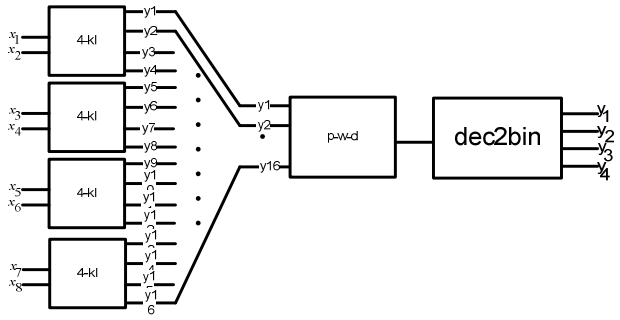


Rys. 6.14 Sieć realizująca jeden wiersz w S-bloku.

Sieć pokazana na rysunku 6.14 realizuje funkcję pojedynczego wiersza S-bloku algorytmu DES. Można ją jednak przyjąć jako pierwszą wersję, bardziej jako konstrukcję teoretyczną, ponieważ ma ona 8 wejść. W praktycznym rozwiązaniu komórki w wierszach S-bloku mają adresy 4 bitowe. Rozwiązanie zmiany adresowania 8-bitowego na 4 bitowe omówione jest w dalszej części, rozdział 6.

6.3.2. Neuronowy dekoder wartości dziesiętnych na dwójkowe

W poprzednim rozdziale pokazana została koncepcja realizacji pojedynczego wiersza S-bloku. Zawartość poszczególnych komórek S-bloku stanowią wartości zmiennoprzecinkowe, jest to rozwiązanie wygodniejsze z punkt widzenia realizacji neuronowej. Charakter działania układu neuronowego jest taki, że zawsze będzie na wyjściu udzielał odpowiedzi w postaci wartości przybliżonych. Z eksperymentów i doświadczeń przeprowadzonych w toku badań na potrzeby tej pracy doktorskiej okazało się, że działanie sieci neuronowej, operującej na wartościach zmiennoprzecinkowych, użytej do realizacji S-bloku jest zasadne pod względem wydajności całego układu. Oczywiście odpowiedź S-bloku jest zawsze w postaci binarnej. Wobec tego koniecznym stało się opracowanie układu neuronowego, który będzie realizował zamianę wartości zmiennoprzecinkowej na binarną. Zakładając, że cały projektowany układ ma być realizowany za pomocą sieci neuronowej, zaistniała więc konieczność konstrukcji neuronowego dekodera. Układ taki na potrzeby przejrzystości dalszego wywodu nazwany został: „dec2bin”. W tym rozdziale celem jest budowa układu przedstawionego na rysunku 6.15.



Rys. 6.15 Sieć realizująca jeden wiersz w S-bloku - odpowiedź binarna.

Realizacja neuronowa pozwoli również na łatwość wprowadzania zmian w działaniu układu w zależności od wprowadzonych modyfikacji w innych częściach sieci realizującej S-blok. Projektowany układ „dec2bin" posiada jedno wejście, na które podawana będzie odpowiedź pochodząca z układu „p-w-d", realizującego podanie wybranej z wiersza S-bloku wartości w postaci binarnej. „Dec2bin" na wyjściu ma udzielić odpowiedzi w postaci 4 bitowej, binarnej. Tablica prawdy dla projektowanego układu, przedstawiona jest w tabeli 6.8,

Tabela 6.8 Tablica prawdy, kodowanie 16 wartości dziesiętnych do postaci binarnej.

x1	y1	y2	y3	y4
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Pamiętając, że głównym celem, do realizacji, którego mają doprowadzić badania opisane w ramach tej pracy, jest budowa „neuronowego układu szyfrującego". Ważnym jest fakt, aby konstrukcja układu kodującego była uniwersalna i mogła być w przyszłości rozbudowywana. W tym celu przyjęta została zasada budowy modułowej, dzięki czemu możliwa jest łatwa modyfikacja układu np. ze względu na wielkość zbioru możliwych sygnałów wejściowych, a co za tym idzie długość binarnego sygnału na wyjściu.

Podobnie jak miało to miejsce w poprzednich rozdziałach, pierwszym krokiem w rozważaniach jest sprowadzenie problemu kodowania do realizacji zadania klasyfikacji,

typowego dla sieci neuronowych. Jeśli zamiast jednej tablicy prawdy skonstruować cztery osobne tabelice zestawiając odpowiednio $x_1 : y_1, x_1 : y_2, x_1 : y_3, x_1 : y_4$, to możemy zaobserwować, że funkcje, które opisują te tabele realizują podział sygnału wejściowego na dwie klasy. Tak więc zadanie konstrukcji neuronowego układu kodującego polegać będzie na konstrukcji czterech odrębnych sieci, które będą realizowały poszczególne funkcje. Ich charakterystyki zapisane zostały w tabeli 6.9 oraz na rysunku 6.16.

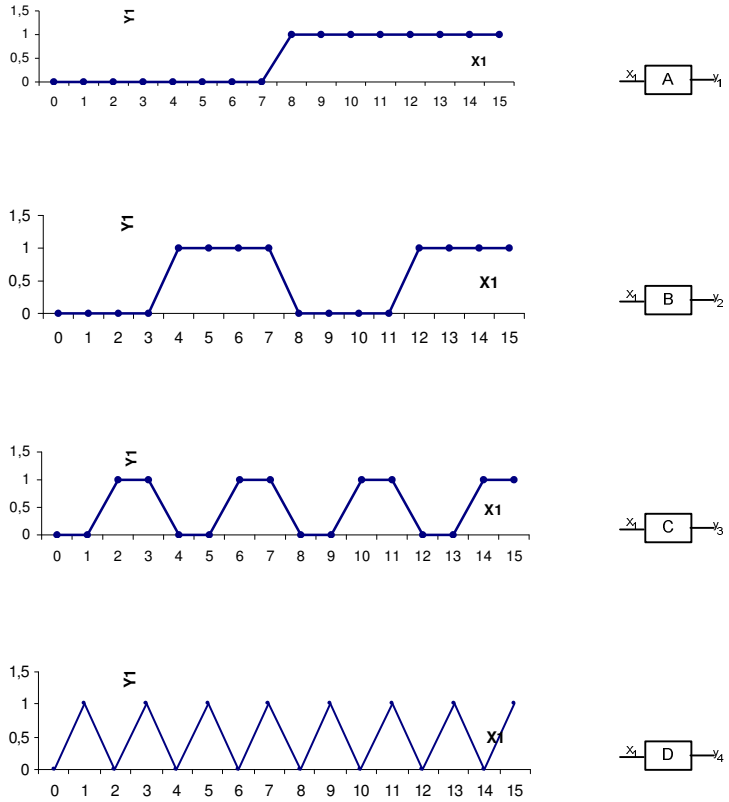
Tabela 6.9. Tablice prawdy dla układu „dec2bin".

A	X1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	y1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

B	X1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	y2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1

C	X1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	y3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

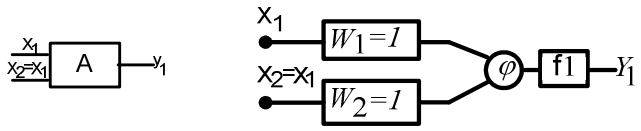
D	X1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	y4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1



Rys. 6.16 Bloki elementarne sieci „dec2bin".

Realizacja bloku A

Ten problem jest stosunkowo prostym do realizacji neuronowej, nie wymaga nawet przeprowadzania procesu uczenia i wystarczy pojedynczy neuron, opisany na rysunku 6.17.

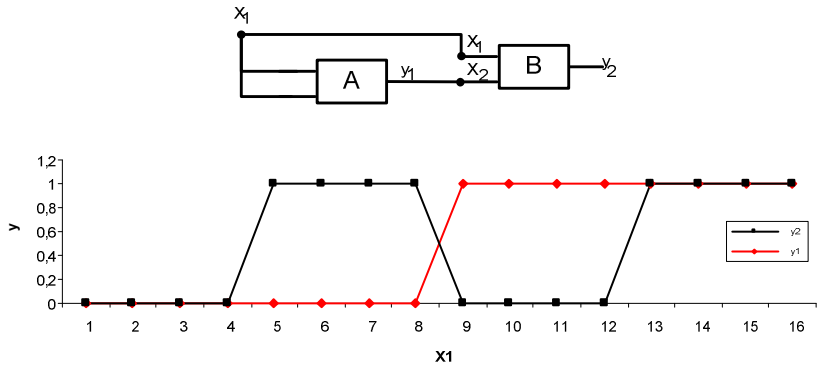


$$f = \begin{cases} 0, & \varphi \le p_1 \\ 1, & \varphi > p_1 \end{cases}, \quad \varphi = \sum w_i x_i, \quad p_1 = 7$$

Rys. 6.17 Sieć realizująca blok A.

Realizacja bloku B

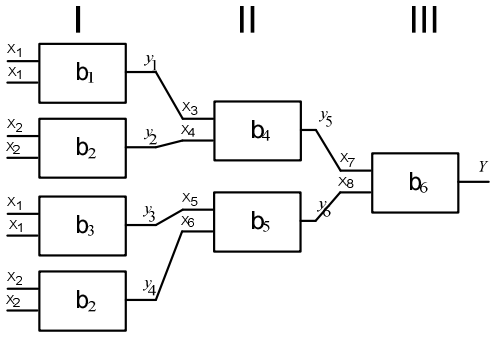
W tym przypadku zadanie jest nieco bardziej złożone, jeden neuron już nie wystarczy. Punktem wyjścia do rozważań jest zweryfikowanie, jakie informacje będą podawane na wejście projektowanego układu. Na pewno na wejście sieci „*B*” podawana będzie wartość zmiennoprzecinkowa (ta sama, która jednocześnie znajdzie się na wejściu układu „*A*”). Ponadto do dyspozycji mamy też odpowiedź układu „*A*”. Układ dokonuje podziału wartości zmiennoprzecinkowej na dwie grupy. Sieć, która jest obecnie przedmiotem rozważań ma za zadanie również dzielenie sygnału wejściowego na grupy. Jak widać na rysunku 6.18 odpowiedź układu „*B*” jest zależna od wartości podawanej na wyjściu bloku „*A*”.



Rys. 6.18 Zależność pomiędzy układami B i A.

Ta zależność właśnie zostanie wykorzystana do budowy układu B. Problem można więc rozpatrywać w taki sposób, że mamy podział sygnału wejściowego na dwie grupy (moduł „*A*”) oraz w każdej grupie podział na dwie podgrupy (moduł „*B*”). W celu realizacji modułu „*B*” skonstruowana została trzywarstwowa siec neuronowa. Poniżej (tabela 6.10) przedstawiona jest ogólna koncepcja tej sieci. Tabela 6.10 zawiera tablice prawdy oddające

zasadę działania funkcji realizowanej przez układ „B”. Schemat budowy modułu B pokazany jest na rysunku 6.19.



Rys. 6.19 Budowa modułu B.

Tabela 6.10 Tablice prawdy dla modułu B.

Warstwa I

x1	x2	y1	y2	y3	y4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	1	0	0	0
5	0	1	0	0	0
6	0	1	0	0	0
7	0	1	0	0	0
8	1	1	1	0	1
9	1	1	1	0	1
10	1	1	1	0	1
11	1	1	1	0	1
12	1	1	1	1	1
13	1	1	1	1	1
14	1	1	1	1	1
15	1	1	1	1	1

Warstwa II

x3	x4	x5	x6	Y5	Y6
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
1	0	0	0	1	0
1	0	0	0	1	0
1	0	0	0	1	0
1	0	0	0	1	0
1	1	0	1	0	0
1	1	0	1	0	0
1	1	0	1	0	0
1	1	0	1	0	0
1	1	1	1	0	1
1	1	1	1	0	1
1	1	1	1	0	1
1	1	1	1	0	1

Warstwa III

x7	x8	Y
0	0	0
0	0	0
0	0	0
0	0	0
1	0	1
1	0	1
1	0	1
1	0	1
0	0	0
0	0	0
0	0	0
0	0	0
0	1	1
0	1	1
0	1	1
0	1	1

B

x1	x2	Y
0	0	0
1	0	0
2	0	0
3	0	0
4	0	1
5	0	1
6	0	1
7	0	1
8	1	0
9	1	0
10	1	0
11	1	0
12	1	1
13	1	1
14	1	1
15	1	1

Tabela 6.10 zawiera tablice prawdy dla poszczególnych warstw składających się na sieć neuronową modułu B. Na poszczególne warstwy składają się moduły elementarne, rysunek 6.19, których zasada działania zostanie teraz przestawiona. Moduł b_2 to neuron, którego zadaniem jest przekazanie na wyjście tego samego sygnału jaki podawany jest na jego wejście. Moduł $b_1 i b_3$ to pojedyncze neurony realizujące funkcję progową, odpowiednio z progiem $p = 3$ oraz $p = 1$. Elementy b_1, b_2, b_3 nie wymagają procesu uczenia, ich wagi przyjmują wartości 0 i 1. Elementy b_4, b_5, b_6 wymagają już przeprowadzenia procesu uczenia. Do realizacji poszczególnych elementów wystarczy również jeden neuron posiadający dwa wejścia i jedno wyjście. W tabeli 6.11 przedstawione zostały tablice prawdy, jakie mają realizować poszczególne bloki b_4, b_5, b_6. Jednocześnie są one reprezentacją zbiorów

uczących, z tym, że dla polepszenia jakości procesu uczenia w zbiorach uczących wartości 0 zostały zastąpione wartościami -1.

Tabela 6.11 Tablice prawdy dla modułów b4, b5, b5.

b4

x1	x2	Y
0	0	0
0	1	0
1	0	1
1	1	0

b5

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

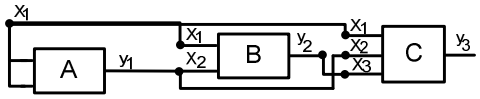
b6

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

Tak więc układ składający się z elementów b1 do b6 realizuje funkcję zapisaną jako tabela prawdy układu B przedstawionego w tabeli 6.10.

Realizacja bloku C.

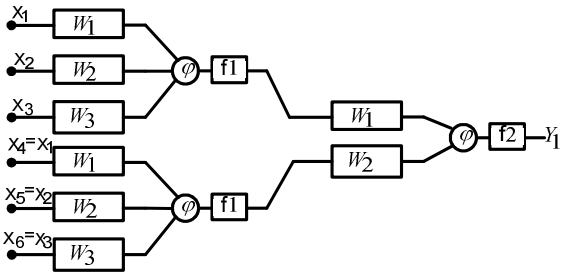
Realizacja kolejnego bloku wymaga użycia sieci neuronowej składającej się z dwóch warstw. Proces uczenia sieci przeprowadzany jest dla całej sieci od razu bez potrzeby „rozbijania" na podbloki. Na tym etapie realizacji kodera „dec2bin" na wejściu bloku „*C*" dysponujemy trzema różnymi wartościami. Dostępna jest wartość zmiennoprzecinkowa, wynik działania bloku *A* i bloku *B*, co obrazuje rysunek 6.20.



C	X1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	X2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	X3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	y3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

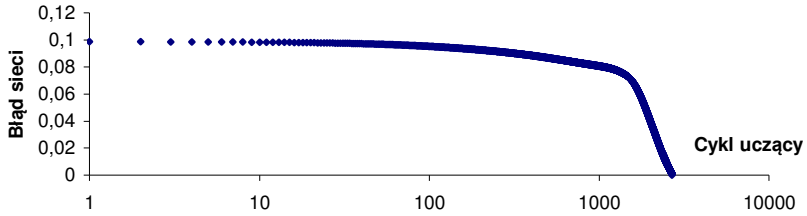
Rys. 6.20 Schemat działania modułu C.

Przy wykorzystaniu trzech sygnałów wejściowych (x1, x2, x3) realizacja tabeli prawdy pokazanej na rysunku 6.20 możliwa jest za pomocą sieci neuronowej, składającej się z trzech neuronów i używając powyższe tablice prawdy jako zbioru uczącego. Szczegółowa konstrukcja sieci pokazana jest na rysunku 6.21. Rysunek 6.22 przedstawia wykres odzwierciedlający proces uczenia.



$$f_1 = \varphi,\ f_2 = \begin{cases} 0, & \varphi \geq p_1 \\ 1, & \varphi < p_1 \end{cases}, \text{ gdzie: } \varphi = \sum w_i x_i,\ y_j = f_k(\varphi),\ p_1 = 0{,}5,$$

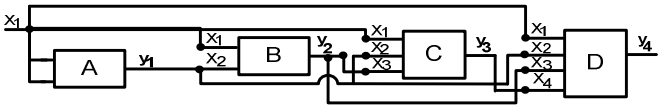
Rys. 6.21 Sieć neuronowa realizująca moduł C.



Rys. 6.22 Przebieg procesu uczenia sieci neuronowej realizującej moduł C.

Realizacja bloku D

Ostatni blok neuronowego kodera wartości dziesiętnych do postaci binarnej zrealizowany został zgodnie z taką samą ideą jak blok C. Różnicą jest tu fakt, że w pierwszej warstwie sieci wykorzystane zostały dwa neurony posiadające cztery wejścia. W celu wykorzystania czterech sygnałów pochodzących z wyjść układów A, B i C oraz wartości dziesiętnej, która poddawana jest kodowaniu.



Rys. 6.23 Schemat układu „dec2bin”

Tabela 6.12 Tablica prawdy dla modułu D

D	X1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	X2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	X3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	X4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	Y4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Tworząc sieć z przygotowanych bloków elementarnych A, B, C i D uzyskujemy sieć neuronową, rysunek 6.23, która realizuje kodowanie wartości zmiennoprzecinkowych do

postaci binarnej czterobitowej. Układ „dec2bin” realizuje więc funkcję, której tablica prawdy przedstawiona została w tabeli 6.8.

Aby zrealizować cel postawiony w tym podrozdziale czyli skonstruować sieć neuronową realizującą pojedyncze wiersze S-bloku koniecznym jest rozwiązanie problemu, o którym wcześniej wspomniano. Do tej pory sieć realizująca jeden wiersz S-bloku posiadała 8 wejść, rysunek 6.15. Natomiast, jeśli rozważamy S-blok pochodzący z szyfru DES tych wejść powinno być cztery (cztery wskazują komórkę w pojedynczym bloku).

6.3.3. Czterobitowe wejście sieci realizującej jeden wiersz

Na rysunku 6.15 przedstawiona została koncepcja sieci realizującej wybieranie jednej wartości zmiennoprzecinkowej z przedziału [0,15] poprzez 8 bitowe wejście sieci. Więc pierwszym zadaniem będzie rozwiązanie problemu redukcji 8 wejść do czterech przy zachowaniu sposobu działania całego układu. Chodzi tu głównie o podawanie na wyjściu Y wartości zmiennoprzecinkowych z dokładnością do dwóch miejsc po przecinku tak, aby przez kolejne warstwy sieci wartość na wyjściu Y mogła być jednoznacznie identyfikowana. Zadanie jest więc następujące, znaleźć taki sposób kodowania sygnału 4 bitowego do 8 bitowego, który zapewni spełnienie wymogu, o którym wspomniałem powyżej. W wyniku przeprowadzonego szeregu eksperymentów i poszukiwań możliwego rozwiązania podjęta została decyzja o rozbudowaniu liczby wejść sieci z 8 do 16. Zrealizowane zostało to poprzez rezygnację w module „4-kl”, pokazanego na rysunku 6.11, z połączeń pomiędzy wejściami modułów „$s-b$ 1”i „$s-b$ 2”, rysunek 6.11. W rezultacie wybór padł na połączenie opisane poniżej, 4 wejścia całego układu realizującego jeden wiersz S-bloku z 16 wejściami pierwszej warstwy sieci.

$$X=\begin{pmatrix} & & & & & & 1 & 2 & 3 & 4 & & & & & & \\ 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 & 3 & 4 & 3 & 4 & 3 & 4 & 3 & 4 \end{pmatrix}$$

Powyższy zapis należy rozumieć w sposób następujący: wejście nr 1 warstwy pierwszej połączone jest z wejściami nr 1,3,5,7 warstwy drugiej. Wejście nr 2 połączone jest z wejściami 2,4,5,8 warstwy drugiej, itd. W rezultacie takiego rozwiązania pierwsza warstwa sieci pokazanej na rys. 6.15, składająca się z 4 modułów 4-kl, realizuje funkcję zapisaną za pomocą tablicy prawdy pokazanej w tabeli 6.13:

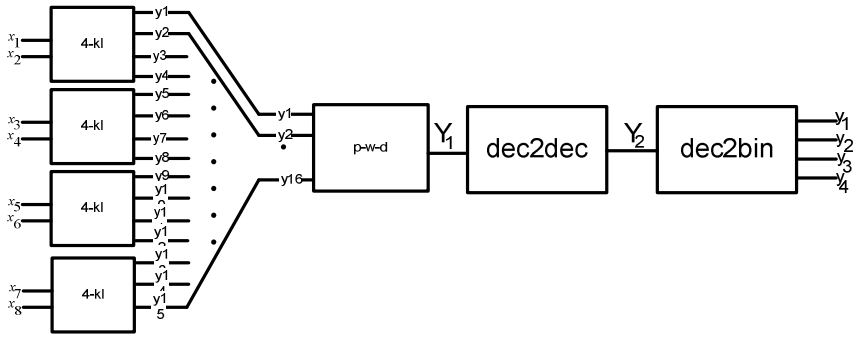
Tabela 6.13. Tablica prawdy pierwszej warstwy sieci.

x1	x2	x3	x4	y1	y2	Y3	y4	y5	y6	y7	y8	y9	y10	y11	y12	y13	y14	y15	y16	Y
0	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	1	1	1	0	0	0	0	1	1	1	1	16
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22
0	0	1	1	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	24
0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	25
0	1	0	1	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	27
0	1	1	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	28
0	1	1	1	0	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	31
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34
1	0	0	1	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	40
1	0	1	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	43
1	0	1	1	0	0	0	1	1	0	1	1	0	0	1	1	0	0	1	1	49
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	51
1	1	0	1	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	52
1	1	1	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	55
1	1	1	1	0	0	1	0	1	1	0	1	0	1	0	1	0	1	0	1	79

W wyniku zmiany pojawiających się sygnałów na wyjściach $y_1 - y_{16}$, rysunek 6.15, inne wartości będą pojawiać się na wyjściu Y układu „p-w-d", rysunek 6.15. Nauczony moduł „p-w-d" podawał wartości na wyjściu Y z przedziału [0,15], przy pojawiających się ciągach bitów wejściowych pokazanych w tabeli 6.13. Przy niezmienionych parametrach nauczonej sieci „p-w-d" i możliwych sygnałach pojawiających się na wejściu tej sieci przedstawionych w tabeli 6.13, cały układ neuronowy realizujący pojedynczy wiersz S-bloku na swoim wyjściu Y, będzie podawał wartości przedstawione w tabeli 6.13. Pojawia się więc kolejny problem: cały układ powinien dla S-bloku algorytmu DES, jako sygnał wyjściowy, podawać wartości z przedziału [0,15]. Wartości z tego przedziału powinny trafiać na wejście opracowanego w rozdziale 6.3.2 tej pracy neuronowego dekodera „dec2bin".

W tabeli 6.13 ostatnia kolumna zawiera zestaw możliwych odpowiedzi, należy więc przygotować sieć neuronową, która będzie realizowała zamianę wartości ze wspomnianej tabeli na odpowiadające jej wartości z przedziału [0,15]. Układ taki na potrzeby tej pracy nazwiemy „bin2bin". Najpierw rozpatrzmy sytuację ze względu na to, jakimi informacjami dysponujemy, czyli rozpoczynamy od konstrukcji tablicy prawdy dla układu „dec2dec".

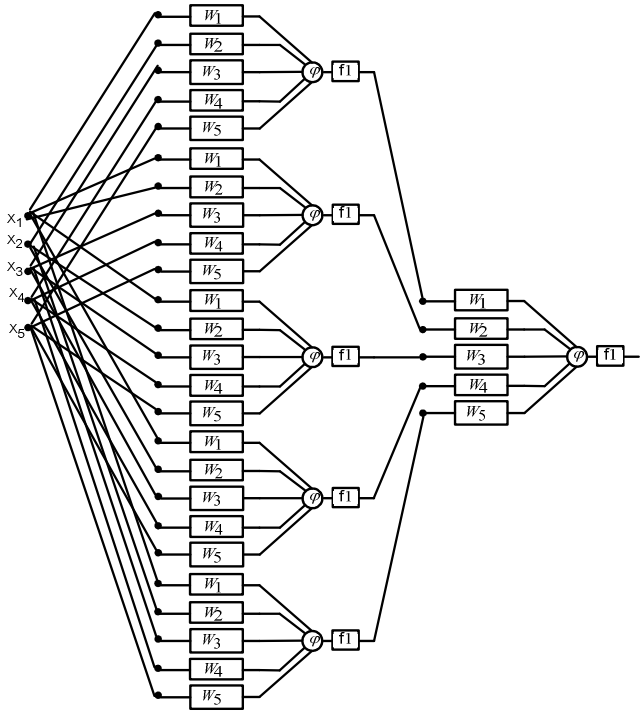
Tablica prawdy i umiejscowienie projektowanego układu w strukturze sieci realizującej pojedynczy S-blok przedstawione zostały na rysunku 6.24.



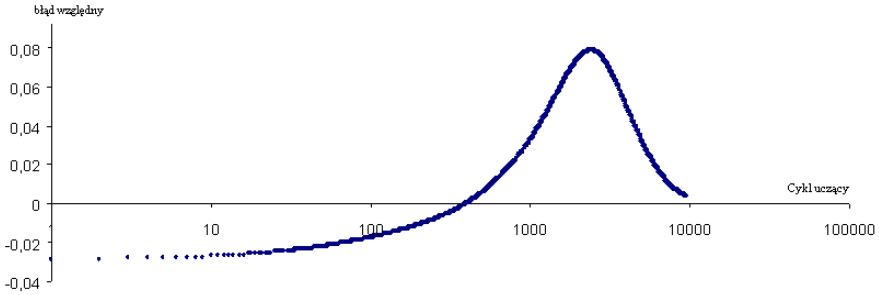
x1	x2	x3	x4	Y1	Y2
0	0	0	0	0	0
0	0	0	1	16	1
0	0	1	0	22	2
0	0	1	1	24	3
0	1	0	0	25	4
0	1	0	1	27	5
0	1	1	0	28	6
0	1	1	1	31	7
1	0	0	0	34	8
1	0	0	1	40	9
1	0	1	0	43	10
1	0	1	1	49	11
1	1	0	0	51	12
1	1	0	1	52	13
1	1	1	0	55	14
1	1	1	1	79	15

Rys. 6.24 Umiejscowienie układu „dec2dec" oraz tabela prawdy tego układu.

Z powyższej tabeli prawdy wynika, że na wejście układu trafia odpowiedź z układu „p-w-d" oraz odpowiadający jej ciąg czterech bitów pochodzący z wejścia całej sieci. Sieć, która będzie realizowała funkcję zapisaną za pomocą tablicy pokazanej na rysunek 6.24 jest prostą w budowie siecią i składa się dwóch warstw, zawierających neurony tego samego typu. Na rysunku 6.25 pokazana została struktura takiej sieci oraz wykres obrazujący przebieg procesu uczenia, przedstawiony na rysunku 6.26.



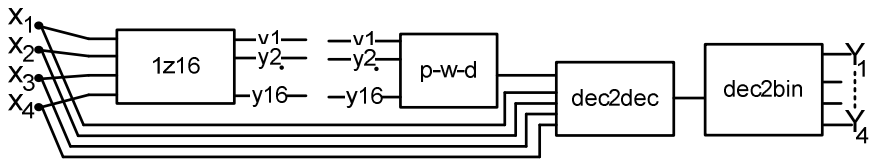
Rys. 6.25 Sieć neuronowa realizująca układ „dec2dec”.



Rys. 6.26 Przebieg procesu uczenia sieci neuronowej realizującej układ „dec2dec”.

Warto zwrócić uwagę na wykres, rysunek 6.26, pokazujący, jak w procesie uczenia sieci „dec2dec” zmienia się błąd względny. Na wykresie można zaobserwować, że w pewnym momencie błąd względny osiąga wartość 0, jednak proces uczenia jest kontynuowany. Wynika to stąd, że w momencie osiągnięcia przez błąd wartości 0 test działania sieci nie dał zadawalających wyników, więc proces uczenia był kontynuowany, co pokazuje wspomniany wcześniej wykres. Został on zakończony w momencie, kiedy co prawda błąd nie osiągnął wartości 0, ale kontrola działania sieci zbiorem testującym dała zadawalające wyniki. Wniosek, jaki wynika z powyższego jest taki, że błąd popełniany przez sieć nie może być zawsze jedynym kryterium oceny jakości procesu uczenia sieci neuronowej. W tym przypadku gdyby kierować się jedynie wartością błędu, to proces uczenia został by przerwany przed osiągnięciem sukcesu.

Na tym etapie możliwe jest już przedstawienie budowy układu neuronowego realizującego pojedynczy wiersz S-bloku. Na rysunku 6.27, przedstawiony został blokowy schemat takiej sieci.



Rys. 6.27 Sieć neuronowa realizująca jeden wiersz S-bloku.

Jeśli założymy, że układ z rysunku 6.27 miałby realizować pierwszy wiersz S-bloku nr 1, uczeniu poddać należałoby moduł „p-w-d" oraz moduł „dec2dec". Pozostałe elementy „1z16" oraz „dec2bin" są niezmienne dla realizacji wierszy z poszczególnych S-bloków algorytmu DES. Dzięki takiej konstrukcji, przy realizacji S-bloku składającego się z czterech wierszy element „1z16" i „dec2bin" wystąpią tylko raz, co w znacznym stopniu zmniejsza stopień złożoności całego układu. Ponadto podejście do konstrukcji całego układu w architekturze modułowej pozwala na ograniczenie liczb, aktualizowanych wartości wag w przypadku zmiany realizowanego algorytmu, o czym szerzej będzie mowa w kolejnych rozdziałach.

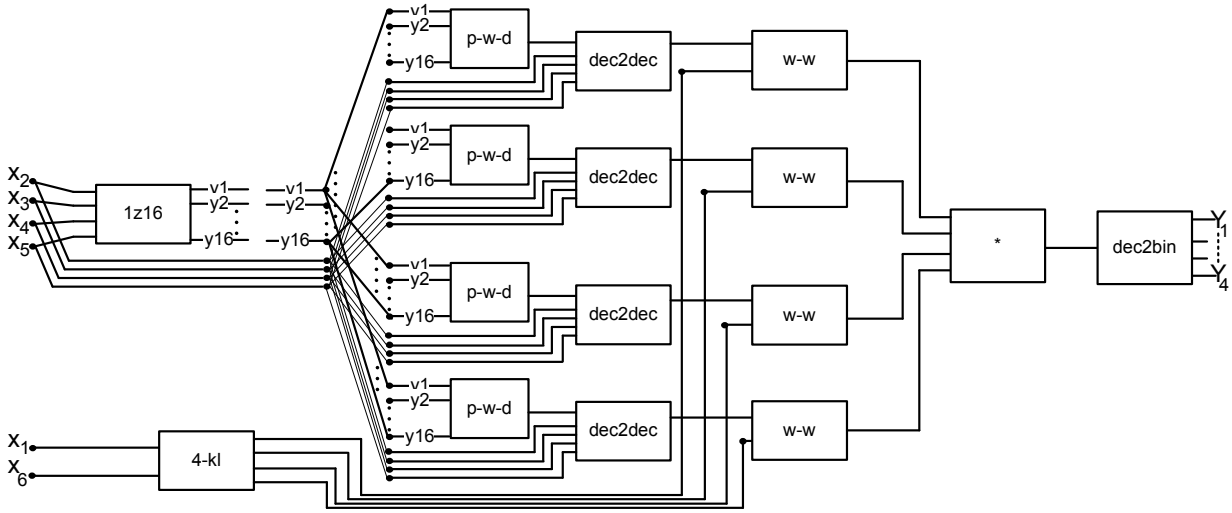
6.3.4. Realizacja kompletnego S-bloku

W chwili obecnej możliwe jest podjęcie próby konstrukcji całego S-bloku składającego się z 4 wierszy. Jako punkt wyjścia przyjmujemy S-blok nr 1 z algorytmu DES. S-blok ten przedstawiony został w tabeli 6.5.

Aby zrealizować kompletny S-blok niezbędnym jest rozwiązanie problemu adresowania wierszy. W 6 bitowym sygnale wejściowych S-bloku dwa skrajne bity wskazują nr wierszy, z którego następnie wybierana jest konkretna wartość 1 z 16. Ponieważ w algorytmie DES S-bloki posiadają cztery wiersze więc mamy tu problem wyboru 1 z 4. Problem budowy sieci realizującej podział na cztery klasy został opisany już w rozdziale 6.3.1. Rysunek 6.11 przedstawia budowę takiej sieci, moduł ten nazwany ostał na potrzeby tej pracy „4-kl". To oczywiście jednak jeszcze nie wystarcza.

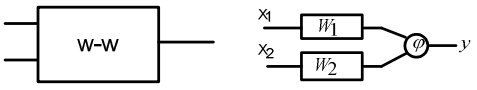
Wracając do koncepcji sieci realizującej pojedynczy wiersz S-bloku, rysunek 6.27, należy rozważyć ogólną koncepcję realizacji nie jednego a czterech wierszy, czyli pełny S-blok algorytmu DES. Przy pierwszych rozważaniach nasuwa się rozwiązanie polegające na wykorzystaniu czterech kompletnych układów neuronowych (rysunek 6.27). Jednak przy dłuższym zastanowieniu się nie jest to konieczne. Można zrealizować pełny S-blok prościej.

Dzięki temu, że od początku tej pracy przyjęto modułową koncepcję budowy sieci, można dokonać wyodrębnienia bloków które mogą być wspólne dla czterech wierszy, a które powinny być w liczbie odpowiadającej liczbie wierszy. Okazuje się więc, że blok „1z16" i „dec2bin" mogą być wspólne, natomiast bloki „p-w-d" oraz „dec2dec" muszą realizować róże przekształcenia na potrzeby poszczególnych wierszy. Wynika to z faktu, że każdy z wierszy posiada różne zestawy wartości w poszczególnych komórkach. Rysunek 6.28 przedstawia ideę konstrukcji pełnego uniwersalnego układu neuronowego, za pomocą którego można zrealizować dowolny S-blok algorytmu DES. Oczywiście, po przeprowadzonym w odpowiednich warunkach procesie uczenia sieci neuronowych „p-w-d" oraz „dec2dec".



Rys. 6.28 Neuronowy układ realizujący S-blok algorytmu DES.

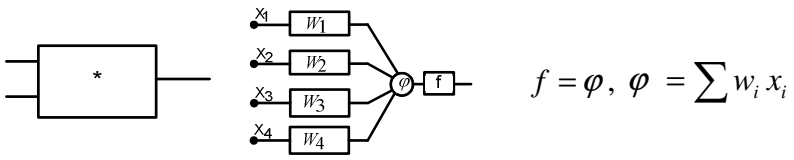
Wyjaśnienia wymaga jeszcze kilka szczegółów z powyższego rysunku. Moduł „w-w" składa się z pojedynczego neuronu, który realizuje funkcję:



$$y = \begin{cases} 0, & x_2 = 0 \\ x_1, & x_2 = 1 \end{cases}$$

Rys. 6.29 Schemat układu „w-w".

Z sygnałów (2 bitowych), jakie mogą się pojawić na wejściu układu „4-kl" wynika, że wartość „<>0" pojawi się tylko na wyjściu jednego z czterech układów „w-w", na pozostałych pojawi się 0. Tą jedną wartością będzie element wskazany przez sygnał wejściowy S-bloku. Kolejnym elementem całego układu jest moduł „*". Jego zadaniem jest przekazanie na wejście modułu „dec2bin" wartości, jaka pojawi się na wyjściu jednego z układów „w-w". Do realizacji tego zadania również wystarczy jeden neuron, tylko że tym razem posiadający 4 wejścia, rysunek 6.30.



Rys. 6.30 Schemat układu „”.*

Działanie ostatniego bloku, czyli neuronowego dekodera wartości dzisiętnych na postać binarną, opisane zostało w rozdziale 6.3.2.

W rozdziale 6.3 szczegółowo opisana została konstrukcja sieci neuronowej, która realizuje funkcję S-blok. Modułowy charakter budowy całego układu pozwala na łatwą realizację S-bloków o innych rozmiarach. Ostatni etap tych rozważań to, krok po kroku, przedstawienie na przykładzie S-bloku nr 1 z algorytmu DES procesu przystosowania powyższego układu do realizacji konkretnej funkcji S-blok. Kluczowym założeniem, jakie poczynione zostało na początku badań, jest to, aby uniwersalność rozwiązania polegała na tym, że jedyną zmianą jest zmiana wartości wag a nie struktury układu. Wobec tego doprowadzenie do sytuacji, że proponowany układ realizuje konkretny S-blok wymaga przeprowadzenia procesu uczenia w określonych warunkach, a nie wymaga przebudowy struktury sieci.

6.3.5. Praktyczna realizacja S-bloku nr 1 algorytmu DES

W procesie uczenia można wyróżnić dwa zasadnicze etapy. Pierwszy to uczenie układów „p-w-d”, drugi to konstrukcja zbiorów uczących dla układów „dec2dec” na podstawie odpowiedzi pochodzących z modułu „p-w-d”.

Proces uczenia modułów „p-w-d”

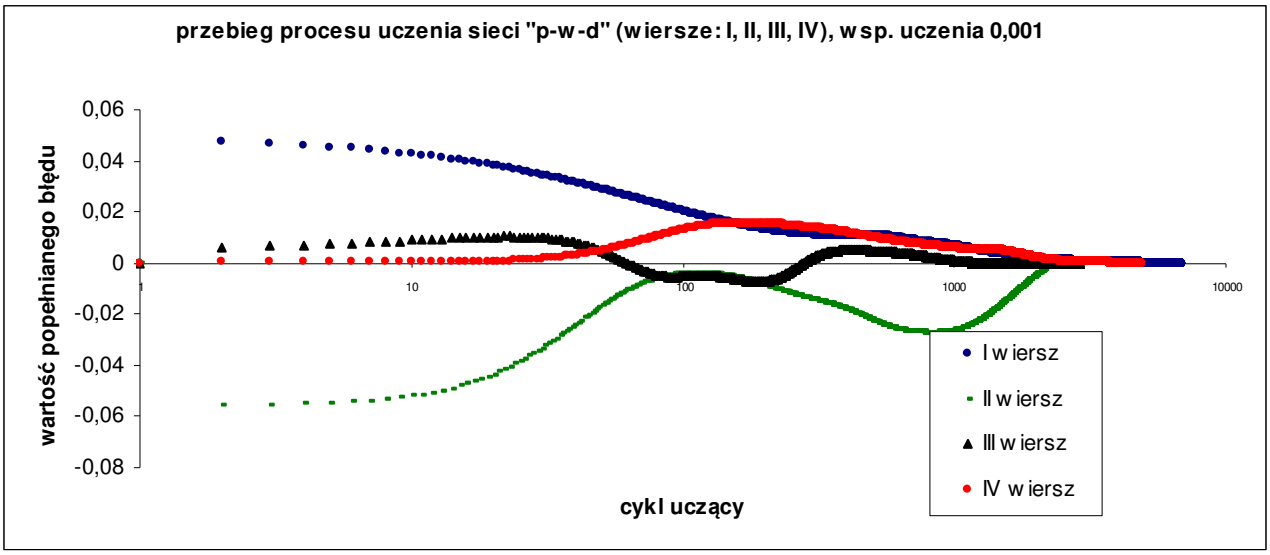
Na wstępnie należy wspomnieć jeszcze o koniecznej drobnej modyfikacji modułu podającego wartość zmiennoprzecinkową. W rozdziale 6.3.1 omówiony został układ, który na wyjściu podaje wartości od 0 do 15, gdzie pierwszym elementem wiersza jest 0. Jednak pewnym problemem jest to, że w pierwszej komórce wiersza może znajdować się dowolna wartość przy rzeczywistych realizacjach S-bloku. Na 16 wejściach sieci „p-w-d” w jednym przypadku na 16 pojawiają się same wartości 0, co oznacza adres pierwszej komórki w danym wierszu. Nie jest możliwe takie działanie sieci, aby przy samych zerach na wejściach na wyjściu pojawiła się wartość >0. Wobec tego do konstrukcji sieci dodany został jeden neuron, co pozwoliło poszerzyć liczbę wejść tego układu do 18. W praktyce wykorzystanych będzie 17 wejść sieci.

Wprowadzone zmiany wynikają z przeprowadzonych eksperymentów numerycznych. Proces uczenia dla poszczególnych wierszy S-bloku przeprowadzony jest z użyciem zbiorów uczących przedstawionych w tabeli 6.14.

Tabela 6.14. Zbiory uczące dla modułów „p-w-d"; realizacja: S-blok-u nr 1 z algorytmu DES.

X1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	y (w I)	y (w II)	y (w III)	y (w IV)
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	14	0	4	15
0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	4	15	1	12
0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	13	7	14	8
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	1	4	8	2
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	2	14	13	4
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	15	2	6	9
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	11	13	2	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	8	1	11	7
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	10	15	5
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	10	6	12	11
0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	6	12	9	3
0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	12	11	7	14
0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	5	9	3	10
0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	9	5	10	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	3	5	6
0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	7	8	0	13

Charakterystyka procesu uczenia dla poszczególnych modułów „p-w-d" różni się pod względem tempa procesu uczenia i stosowanego współczynnika ucznia. Zobrazowane zostało to na rysunku 6.31. Proces uczenia dla poszczególnych wierszy realizowany był przy takim samym współczynniku uczenia dla wszystkich przypadków wynoszącym 0,001. liczba cykli uczących wyniosła odpowiednio dla: I-6789, II-2489, III-2914, IV-4973. Każdy proces uczenia rozpoczynał się od losowania wartości wag.



Rys. 6.31 Przebieg uczenia pierwszej warstwy sieci realizującej S-blok nr 1 algorytmu DES.

Po zakończonym procesie uczenia modułu podającego wartości dziesiętne, kolej na dostosowanie modułów „dec2dec". Tak jak to zostało przedstawione w rozdziale 6.3.3 koniecznym jest rozwiązanie problemu adresowania poszczególnych komórek S-bloku adresami czterobitowymi. W wyniku zastosowanych połączeń pomiędzy układem „1z16" a układami „p-w-d", rysunek 6.28, na wyjściu układu „p-w-d" pojawiać się będą inne wartości niż z przedziału 0-15. Wobec tego kolejny krok praktycznej realizacji S-bloku to identyfikacja wartości, jakie będę się pojawiać na wyjściu układów podających wartość zmiennoprzecinkową, niezbędna konstrukcja zbioru uczącego za pomocą którego w wyniku procesu uczenia nastąpi adaptacja sposobów pracy poszczególnych modułów „dec2dec". Tak, aby kolejne warstwy sieci mogły wykonać poprawnie swoje zadanie.

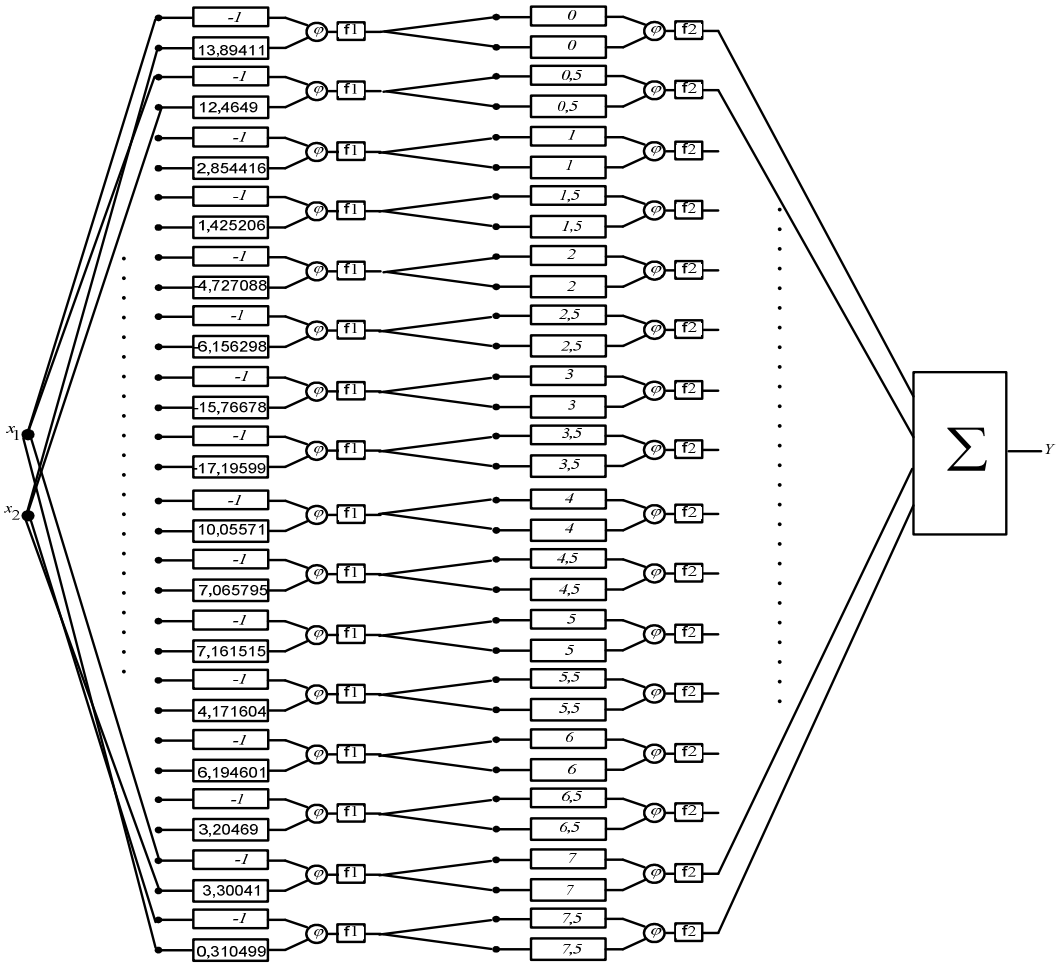
Proces uczenia modułów „dec2dec"

Zadaniem, które musi być zrealizowane jako pierwsze w procesie uczenia modułu „dec2dec", jest konstrukcja odpowiednich zbiorów uczących. W związku z tym dysponując gotowymi modułami „p-w-d" zaczynamy od ustalenia, jakie wartości zmiennoprzecinkowe będą się pojawiać na wyjściach poszczególnych modułów „p-w-d". Zostanie to ustalone poprzez podawanie na wejściu układu realizującego S-blok wszystkich 16 możliwych ciągów cztero-bitowych, efekty pokazano w tabeli 6.15.

Tabela 6.15 Działanie nauczonych modułów „p-w-d”.

x1	x2	x3	x4	Wiersz I	Wiersz II	Wiersz III	Wiersz IV
0	0	0	0	3,3004099	-0,144981	14,711072	-4,9176607
0	0	0	1	-1,795673	42,587631	-4,031827	-33,325734
0	0	1	0	-4,727088	11,49604	23,435342	-21,820849
0	0	1	1	10,055706	17,58562	26,858015	-15,941972
0	1	0	0	12,022716	26,049361	4,113998	-1,620071
0	1	0	1	6,1946012	-2,601904	24,713089	-2,0598093
0	1	1	0	7,1615148	-8,693116	9,5631632	-19,721743
0	1	1	1	0,3104988	30,94661	6,8695371	-38,280054
1	0	0	0	-20,41687	10,543292	1,694526	-16,866529
1	0	0	1	3,2046901	4,4404236	16,871555	-6,5743912
1	0	1	0	7,065795	-1,650789	16,855998	-5,8395807
1	0	1	1	-18,54548	4,4520796	15,289516	-1,1378893
1	1	0	0	4,1716037	11,494407	9,0144632	9,88481355
1	1	0	1	0,0757157	47,484879	17,708988	-18,799823
1	1	1	0	13,894105	16,393289	9,8403512	-15,020052
1	1	1	1	-6,598477	-5,04223	19,016481	14,8391338

Moduł „dec2dec” zaproponowany we wcześniejszych rozdziałach jest konstrukcją, która sprowadza się do zamiany różnych wartości zmiennoprzecinkowych na odpowiednie wartości z przedziału 0 do 15. Stosowanie tej konstrukcji wymaga procesu uczenia, który jak się okazało w wyniku eksperymentów przeprowadzonych na potrzeby praktycznej realizacji S-bloku, wymaga ponad 10000 cykli uczących i bieżących zmian współczynnika uczenia. Jest to więc rozwiązanie czasochłonne i trudne do automatyzacji. Dla praktycznych realizacji przyjęte zostało nieco inne rozwiązanie, które nie będzie wymagało realizacji procesu uczenia. Nowy układ „dec2dec” to sieć neuronowa, która składa się z dwóch warstw. Na każdą z warstw składa się po 16 neuronów posiadających po dwa wejścia. Na potrzeby realizacji konkretnego wiersza S-bloku jako każda co druga waga neuronów z pierwszej warstwy wpisywana jest wartość pojawiająca się na wyjściu układu „p-w-d”, rysunku 6.32. Pozostałe wagi w pierwszej warstwie przyjmują wartość „-1”. Wartości co drugiej wagi w warstwie pierwszej ulegają zmianie w zależności od tego, który wiersz ma być realizowany. Wartości wag w drugiej warstwie pokazane są na rysunku 6.32.

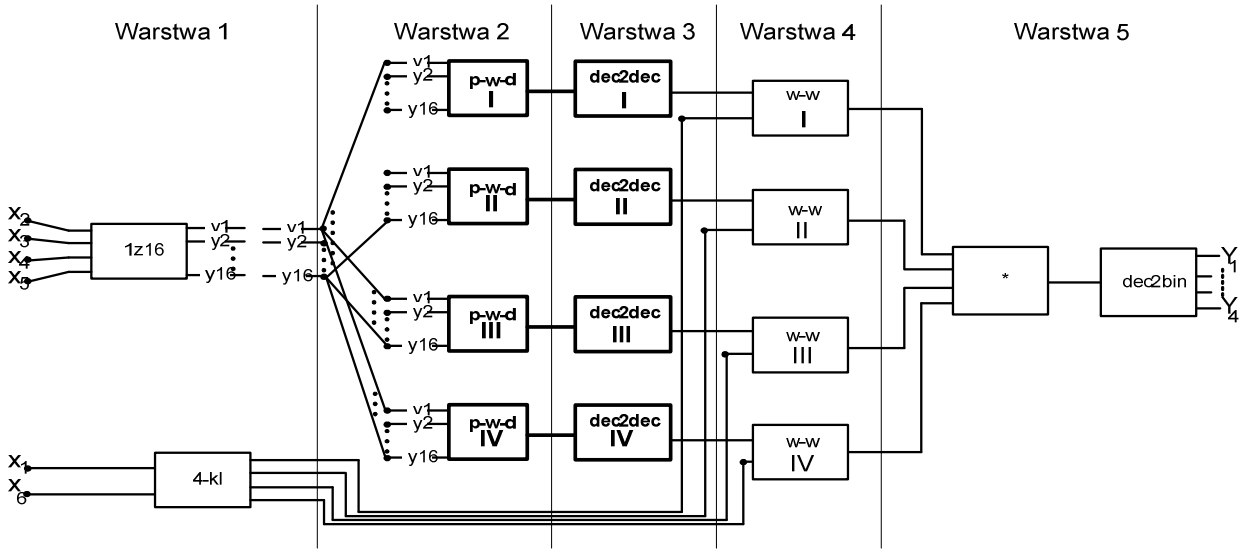


Rys. 6.32 Układ „dec2dec" wersja 2.

Na powyższym rysunku moduł $\sum$ posiada 16 wejść i jedno wyjście, pełni rolę zwykłego sumatora zmiennoprzecinkowego. Na jednym z 16 wyjść drugiej warstwy może pojawić się tylko jedna wartość różna od 0, w przypadku S-bloku algorytmu DES jest to wartość od 0-15. Neurony, z których zbudowane są I i II warstwa sieci działają wykorzystując funkcje aktywacji, które zapisano poniżej:

$$f_1 = \begin{cases} 1, & \varphi = 0 \\ 0, & \varphi <> 0 \end{cases}, \; f_2 = \varphi .$$

Wyższość rozwiązania układu „dec2dec" w wersji pokazanej na rysunku 6.32 nad rozwiązaniem zaproponowanym we wcześniejszych rozdziałach polega na tym, że przy zmianie zawartości S-bloku układ nie wymaga przeprowadzania procesu uczenia, a jedynie zmiany 16 wartości wag w I warstwie. Ostatecznie więc realizacja I S-bloku algorytmu DES z wykorzystaniem sieci neuronowych, proponowana w tej pracy do realizacji praktycznych, pokazana jest na rysunku 6.33.



Rys. 6.33 Sieć neuronowa realizująca S-blok algorytmu DES (układ dec2dec wersja 2).

Konstrukcja przedstawiona powyżej to układ neuronowy, za pomocą którego możliwa jest realizacja dowolnej funkcji S-blok. Mowa tu oczywiście o S-bloku składającym się z czterech wierszy i 16 kolumn. Dzięki przyjętej już od samego początku konstrukcji modułowej rozwiązań proponowanych w tej pracy, nic nie stoi na przeszkodzie aby realizować S-bloki o innych wymiarach. Wymagałoby to jedynie wymiany poszczególnych modułów; w zależności od potrzeb przeprowadzona może być rozbudowa liczby modułów lub rezygnacja z wybranych elementów. Wszystko zależy od tego jaki S-blok ma być realizowany.

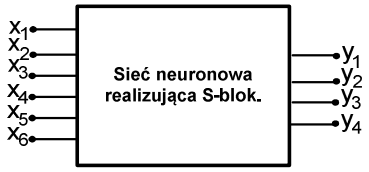
Na rysunku 6.33 „grubą kreską" zaznaczone zostały te elementy, których parametry (wagi neuronów) podlegają zmianie, jeśli ma nastąpić zmiana realizowanej funkcji podstawienia na inną. Liczbami rzymskimi oznaczone zostały te elementy, które muszą być powtórzone dla każdego wiersza S-bloku, pozostałe elementy są wspólne dla całego układu. Zastosowanie modułu „dec2dec" w wersji drugiej, o czym mowa była powyżej, czyni proces zmiany realizacji funkcji przez układ bardziej wydajnym. Procesowi uczenia poddawane są przy takim rozwiązaniu jedynie elementy „p-w-d" dla każdego wiersza osobno. W całym układzie neuronowym pokazanym na rysunku 6.33 z punktu widzenia funkcjonalnego wyróżnić można 5 warstw, które zbudowane są z różnej liczby neuronów. Przedstawione to zostało w tabeli 6.16

Tabela 6.16 Charakterystyka ilościowa neuronowej realizacji S-bloku.

	Warstwa 1	Warstwa 2	Warstwa 3	Warstwa 4	Warstwa 5	Suma
Ilość neuronów	100 + 25 = 125	15 * 4 = 60	32 * 4 = 128	1 * 4 = 4	1 + 14 = 15	332
Ilość wag	200 + 50 = 250	30 * 4 = 120	64 * 4 = 256	2 * 4 = 8	4 + 34 = 38	672

Rozmiary sieci neuronowych realizujące poszczególne bloki funkcjonalne to wartości jakie wynikają z dotychczas przeprowadzonych badań i eksperymentów. Z całą pewnością możliwa jest ich optymalizacja pod względem rozmiarów, jak i wykorzystywanych funkcji aktywacji w neuronach, jednak stanie się to przedmiotem badań i poszukiwań w przyszłości. Dla dopełnienia przykładu, w tabeli 6.17, przedstawiona została tablica prawdy dla układu realizującego pierwszy S-blok.

Tabela 6.17. Tablica prawdy funkcji S-blok nr 1 algorytmu DES.

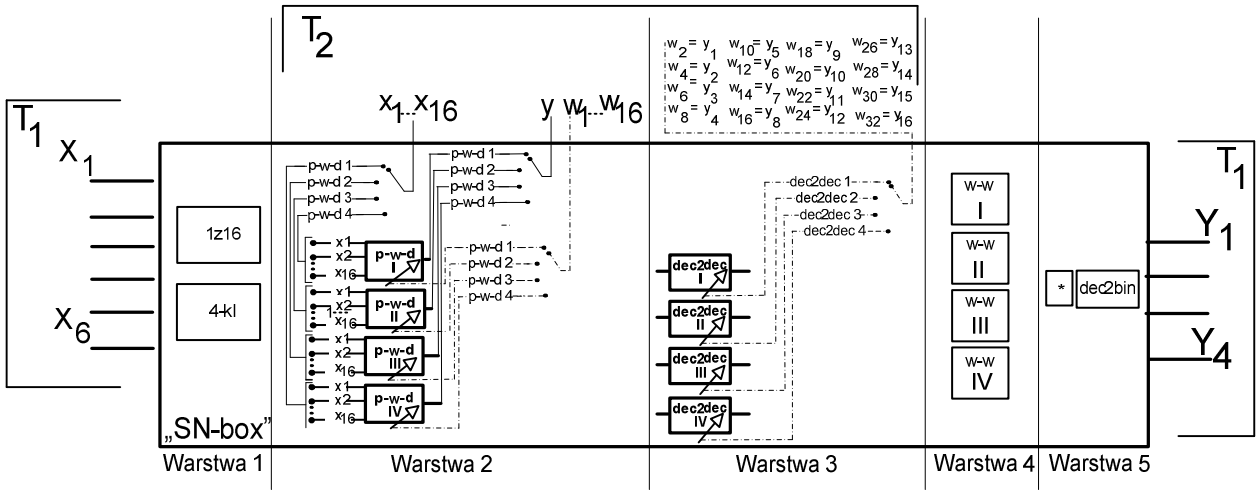


x1	x2	x3	x4	x5	x6	y1	y2	y3	y4
0	0	0	0	0	0	1	1	1	0
0	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	1	1	0	1
0	0	0	1	1	0	0	0	0	1
0	0	1	0	0	0	0	0	1	0
0	0	1	0	1	0	1	1	1	1
0	0	1	1	0	0	1	0	1	1
0	0	1	1	1	0	1	0	0	0
0	1	0	0	0	0	0	0	1	1
0	1	0	0	1	0	1	0	1	0
0	1	0	1	0	0	0	1	1	0
0	1	0	1	1	0	1	1	0	0
0	1	1	0	0	0	0	1	0	1
0	1	1	0	1	0	1	0	0	1
0	1	1	1	0	0	0	0	0	0
0	1	1	1	1	0	0	1	1	1
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1
0	0	0	1	0	1	0	1	1	1
0	0	0	1	1	1	0	1	0	0
0	0	1	0	0	1	1	1	1	0
0	0	1	0	1	1	0	0	1	0
0	0	1	1	0	1	1	1	0	1
0	0	1	1	1	1	0	0	0	1
0	1	0	0	0	1	1	0	1	0
0	1	0	0	1	1	0	1	1	0
0	1	0	1	0	1	1	1	0	0
0	1	0	1	1	1	1	0	1	1
0	1	1	0	0	1	1	0	0	1
0	1	1	0	1	1	0	1	0	1
0	1	1	1	0	1	0	0	1	1
0	1	1	1	1	1	1	0	0	0

x1	x2	x3	x4	x5	x6	y1	y2	y3	y4
1	0	0	0	0	0	0	1	0	0
1	0	0	0	1	0	0	0	0	1
1	0	0	1	0	0	1	1	1	0
1	0	0	1	1	0	1	0	0	0
1	0	1	0	0	0	1	1	0	1
1	0	1	0	1	0	0	1	1	0
1	0	1	1	0	0	0	0	1	0
1	0	1	1	1	0	1	0	1	1
1	1	0	0	0	0	1	1	1	1
1	1	0	0	1	0	1	1	0	0
1	1	0	1	0	0	1	0	0	1
1	1	0	1	1	0	0	1	1	1
1	1	1	0	0	0	0	0	1	1
1	1	1	0	1	0	1	0	1	0
1	1	1	1	0	0	0	1	0	1
1	1	1	1	1	0	0	0	0	0
1	0	0	0	0	1	1	1	1	1
1	0	0	0	1	1	1	1	0	0
1	0	0	1	0	1	1	0	0	0
1	0	0	1	1	1	0	0	1	0
1	0	1	0	0	1	0	1	0	0
1	0	1	0	1	1	1	0	0	1
1	0	1	1	0	1	0	0	0	1
1	0	1	1	1	1	0	1	1	1
1	1	0	0	0	1	0	1	0	1
1	1	0	0	1	1	0	1	0	1
1	1	0	1	0	1	0	0	1	1
1	1	0	1	1	1	1	0	1	0
1	1	1	0	0	1	1	0	0	0
1	1	1	0	1	1	0	0	0	0
1	1	1	1	0	1	0	1	1	0
1	1	1	1	1	1	1	1	0	1

Jeśli strukturę neuronowego układu pokazanego na rysunku 6.33, który od tego miejsca nazywany będzie „SN-box”, uznamy jako stałą, to zmiana realizowanych przekształceń musi odbyć się jedynie poprzez zmianę parametrów sieci neuronowych jakimi są wagi neuronów.

Wobec powyższego „SN-blok” może pracować w dwóch trybach. Pierwszy $T1$ to tryb szyfrowania tekstu jawnego, a $T2$ to tryb uczenia, pokazano to na rysunku 6.34.



Rys. 6.34 Schemat ideowy modułu „SN-blok”

Powyższy rysunek pokazuje ideowy schemat funkcjonowania omawianego układu w dwóch trybach pracy. Strzałki oznaczają moduł poddawany procesowi uczenia/adaptacji. W trybie uczenia ($T2$) wykorzystywane są wejścia i wyjścia elementów „p-w-d” oraz „dec2dec”. Proces uczenia, choć z pewnością w tym przypadku bardziej pasowałoby określenie adaptacji, przeprowadzony jest przez aplikację uczącą. Szczegóły tego procesu zostały omówione we wcześniejszej części tego rozdziału. W momencie normalnej pracy układu, „SN-box” znajduje się w trybie $T1$ i wykorzystywane są wejścia i wyjścia oznaczone na rysunku 6.34. Jak zostanie opisane w kolejnych rozdziałach, nie jest to jedyna możliwa konfiguracja, zaproponowane zostanie rozwiązanie alternatywne.

6.3.6. Aspekty bezpieczeństwa układu „SN-blok”

Na temat bezpieczeństwa oraz korzyści wynikających z podejścia do konstrukcji układu „SN-box”, zaproponowanego w rozdziale 6.3.5, mowa będzie szerzej w końcowych rozdziałach pracy. Jednak w tym miejscu warto wspomnieć o konstrukcji układu „dec2dec” pokazanej na rysunku 6.32. Biorąc pod uwagę, że w praktycznym rozwiązaniu, wprowadzenie zmian w funkcji realizowanej przez układ „SN-box” mogłoby się odbywać na odległość. Ponadto, jeśli zależałoby nam na zachowaniu tych zmian w tajemnicy przed potencjalnym intruzem to informacja wysyłana nie powinna ułatwiać krytpoanalizy.

W praktyce przesyłane są jedynie wartości wag układu „p-w-d”. Parametry układu „dec2dec” ustalane są po stronie odbiorcy. Pozostałe moduły „dec2dec” dla pozostałych trzech wierszy pierwszego S-bloku przygotowywane są w analogicznych sposób, jak dla wiersza nr 1.

Wartości wag w warstwie pierwszej są nadpisywane wartościami przedstawionymi w tabeli 6.15. Wykorzystanie wartości pośrednich, jakimi są liczby zapisane w tabeli 6.15 rozszerza możliwości projektowanego układu do zastosowań wymagających utajnienia wartości docelowych. Każdorazowo proces uczenia zaczyna się od losowania wartości wag. Jeśli wzięlibyśmy pod uwagę ten sam S-blok i różne warunki początkowe procesu uczenia to ten sam S-blok może być realizowany przez sieci posiadające różne parametry (wartości wag).

6.3.7. Dyskusja wydajność układu „SN-blok"

Szacowanie wydajności proponowanego wyżej układu poprzedzone zostanie informacją, że to, iż składa się on z 332 neuronów a z tego wynika 672 wag, nie oznacza, że układu tego nie można zrealizować w sposób mniej złożony. Z całą pewnością można, w wyniku prowadzonych eksperymentów obecna liczba neuronów została zredukowana z początkowych 1219 neuronów w pierwszej wersji do wspomnianych 332 w wersji obecnej. W obecnej wersji „SN-box" tkwią jeszcze możliwości do dalszej optymalizacji bez utraty funkcjonalności, co będzie przedmiotem dalszym badań. Zasadnym jest też rozważenie innej konstrukcji układu „SN-box", o czym będzie mowa w tym rozdziale. Ważna jest jeszcze jedna uwaga, wszystkie pomiary prezentowane w tym rozdziale dokonane zostały na komputerze z procesorem 1,2 GHz i 512 MB pamięci RAM. Całość była implementowana w środowisku Visual Studio NET 6.0. Wszystkie eksperymenty, również na potrzeby wcześniejszych rozdziałów pracy, były wykonywane w stworzonym oprogramowaniu do konstrukcji sieci neuronowych. Aplikacja ta daje dowolne możliwości tworzenia struktur neuronowych, jednak jej uniwersalność jest też w pewnym stopniu mankamentem z punktu widzenia wydajności budowanych sieci. Tak więc, również i na etapie implementacji samych sieci stosowanych do realizacji funkcji S-blok, skorzystanie z wydajniejszego środowiska programistycznego i stworzenia implementacji dedykowanej konkretnemu rozwiązaniu, można jeszcze wiele uzyskać w materii poprawy wydajności układu „SN-blok".

Czasochłonność procesu uczenia/adaptacji

W momencie podjęcia decyzji o zmianie realizowanej funkcji przez omawiany układ, koniecznym jest przeprowadzenie korekty 184 wag. Ustalenie nowych wartości dla 120 wag wymaga przeprowadzenie procesu uczenia (moduły „p-w-d"), pozostałe 64 wagi (moduły „dec2dec") wymagają jedynie nadpisania nowych wartości, rysunek 6.32, które uzyskiwane

są w procesie uczenia modułu „p-w-d". Czas potrzebny na ich modyfikację jest na pomijalnym poziomie. Dla S-bloku numer 1 algorytmu DES, modyfikację wag należy przeprowadzić dla 4 modułów „p-w-d", co wynika z rysunku 6.34. Zbiory uczące przedstawione zostały w tabeli 6.14. W tabeli 6.18 przedstawione zostały wybrane parametry procesu uczenia.

Tabela 6.18 Statystyka procesu uczenia modułu „SN-box".

moduł "p-w-d"	**wsp. uczenia**	**liczba cykli**	**czas [s]**
1	0,001	6789	96,11
2	0,001	2489	37,8
3	0,001	2914	43,2
4	0,001	4976	69,31
	suma	**17168**	**246,42**

Czas potrzebny na przeprowadzenie procesu uczenia można nieco zmniejszyć, przeprowadzając proces uczenia wszystkich czterech modłów „p-w-d" równolegle. Wtedy łączny czas jest nieco krótszy i wynosi 235,12 sekund. Czasochłonność procesu uczenia ma jednak mniejsze znaczenie niż sprawność działania już gotowego (nauczonego) układu.

Czasochłonność procesu szyfrowania

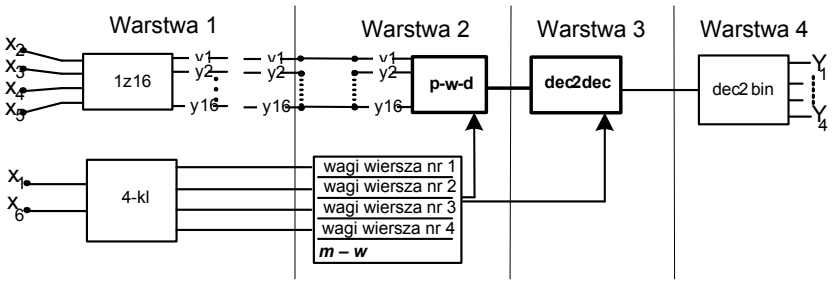
Normalnym trybem pracy S-bloku jest szyfrowanie, czyli zamiana sześciobitowego ciągu wejściowego na czterobitowy szyfrogram. W tabeli 6.19 przedstawione są wyniki pomiarów wydajności dla różnej długości ciągów jawnych. Oczywiście, wcześniej ciąg tekstu jawnego dzielony jest na sześciobitowe bloki, jeśli któryś z bloków nie jest 6 bitowy, to dopełniany jest wartościami losowymi.

Tabela 6.19 Statystyka szyfrowania modułu „SN-box.

dł. tekstu jawnego	Czas szyfrowania [s]	procent wzrostu czasu
4	0,000775	
16	0,0031	400,00%
64	0,0121	390%
256	0,0391	323%
1024	0,1235	316%

Powyższe pomiary dokonane zostały dla układu „SN-box" w konfiguracji pokazanej na rysunku 6.34. Zasadnym jednak wydaje się zaproponowanie nieco innego podejścia. Poprzez zastosowanie rozwiązania opisanego poniżej zmniejsza się złożoność całego układu blisko o 1/3 liczby potrzebnych neuronów. Wzrasta jednak czasochłonność układu pracującego

w trybie szyfrowania. Założenie jest następujące, z gotowego układu pokazanego na rysunku 6.34 w warstwie 2 i 3, pozostaje jeden moduł „p-w-d" i jeden „dec2dec", ginie również warstwa 4, a z warstwy piątej pozostaje układ „dec2bin". Dochodzi natomiast macierz wag nazywana dalej „m-w". Schemat ideowy pokazany jest na rysunku 6.35.



Rys. 6.35 Neuronowy układ realizujący S-blok algorytmu DES.

W tablicy „m-w" przechowywane są zestawy wartości wag dla elementu „p-w-d" oraz „dec2dec" dla wszystkich czterech wierszy S-bloku. Gdy na wejściach x1 i x6 pojawia się sygnał wejściowy wskazujący na jeden z czterech wierszy na jednym z wyjść układu „4-kl" pojawia się wartość 1. Powoduje to, że do układów zaznaczonych „pogrubioną kreską" wysyłany jest odpowiedni zestaw wag. Następnie na wejściach $x_2 - x_5$ pojawiający się zestaw czterech bitów powoduje poprzez układy „p-w-d", „dec2dec" oraz „dec2bin" wygenerowanie ciągu wyjściowego na Y_1 do Y_4. Przy kolejnym zestawie wartości wejściowych na $x_1 - x_6$ sytuacja się powtarza. Przy każdorazowej zmianie wskazywanego wiersza następuje aktualizacja wartości wag w modułach „p-w-d", „dec2dec". Liczba wag, jaka decyduje o zawartości S-bloku to 30+16=46. Wynika z tego, że tablica „m-w" przechowuje 4*46=184 wartości, więc tablica ma wymiar 4 wiersze na 184 kolumny. Oczywiście dochodzi to tego jeszcze prosta aplikacja obsługująca nadpisanie wag. W stosunku do pełnego układu „SN-box" charakterystykę ilościową układu przedstawia tabela 6.20.

Tabela 6.20 Statystyka modułu „SN-blok – m-w".

	Warstwa 1	Warstwa 2	Warstwa 3	Warstwa 4	Suma
Liczba neuronów	100 + 25 = 125	15	32	14	186
liczba wag	200 + 50 = 250	30	64	34	378

Powyższa tabela nie odzwierciedla informacji na temat połączeń pomiędzy neuronami. Można przyjąć, że jeśli łączna liczba neuronów zmniejszyła się blisko o połowę, to podobny „uzysk" będzie miał miejsce w przypadku tablicy połączeń sieci. Widać więc tu, że układ ma mniejszą złożoność. W tabeli 6.21 zostały przedstawione czasy działania układu pokazanego

na rysunku 6.35 dla różnych ciągów wejściowych. Jeśli któryś z bloków nie jest 6 bitowy, to dopełniany jest wartościami losowymi.

Tabela 6.21 Statystyka szyfrowania modułu „SN-blok – m-w”.

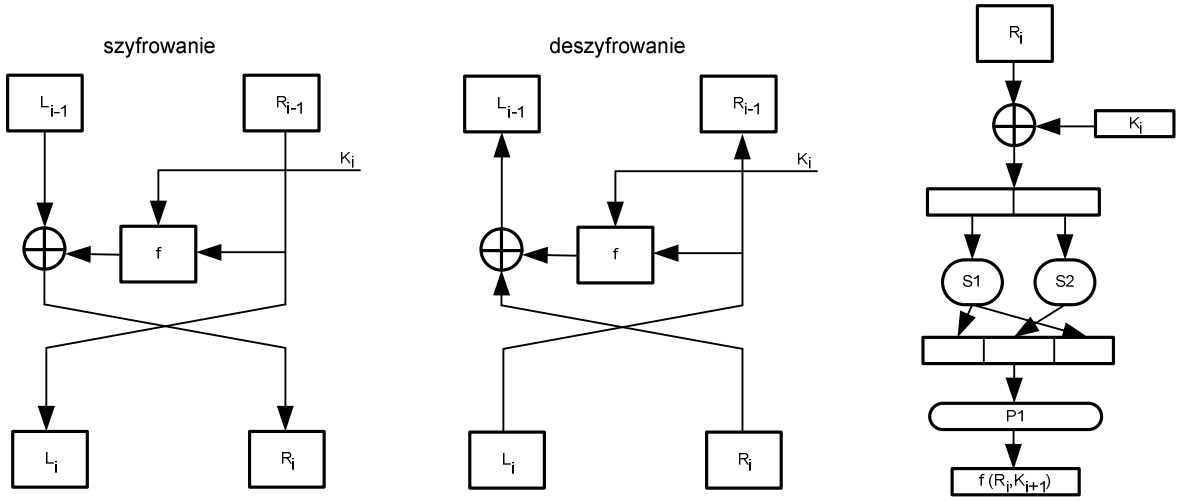
dł. tekstu jawnego	czas szyfrowania [s]	wzrost w stosunku do układu „SN-box”
4	0,000933	20%
16	0,00399	29%
64	0,0189	56%
256	0,0456	17%
1024	0,1369	11%

Jak wynika z danych przedstawionych w tabelach 6.16 i 6.20, zmniejszyła się złożoność układu, jednak odnotowane zostały wzrosty czasów szyfrowania, co można zaobserwować w trzeciej kolumnie tabeli 6.21. Tam, gdzie w ciągu bitów wejściowych konieczność zmiany wiersza S-bloku występowała rzadziej uzysk czasu jest większy. Wynika to z faktu, że należało wykonać mniej powtórzeń zmian wartość wag. Decyzja, którą konstrukcję wybrać do realizacji konkretnego szyfru, zależy od tego czy ważniejsza w danej sytuacji jest prostsza budowa czy czas działania układu.

Rozdział 7. Neuronowa realizacja szyfru blokowego

7.1. Realizacja sieci S-P

W poprzednich rozdziałach przedstawione zostały propozycje rozwiązań elementarnych przekształceń kryptograficznych z wykorzystaniem sieci neuronowych. Celem tej pracy jest jednak pokazanie, że możliwe jest zrealizowanie kompletnego algorytmu szyfrującego za pomocą sieci neuronowej. Wobec tego w tym rozdziale przedstawiony zostanie przykład realizacji koncepcji Shannon-a [2],[3] szyfru kaskadowego, czyli sieci podstawień i permutacji (S-P sieć). Nie będzie to jeszcze realizacja konkretnego, stosowanego w praktyce algorytmu. Opisany tu zostanie prosty algorytm wymyślony na potrzeby tego rozdziału, nazwany „crypt3". Nadmienić należy, że chodzi tu o przykład, więc nie należy traktować tego algorytmu jako bezpiecznego, z punktu widzenia współczesnych metody kryptoanalizy. Szyfr ten składać się będzie z 2 S-bloków o wymiarach 4x16, oraz jednej permutacji. Algorytm ten operować będzie na 24 bitowym ciągu wejściowym oraz wykorzystywać będzie 12 bitowy klucz, w każdej z trzech rund inny. Najpierw przedstawiona zostanie ogólna koncepcja algorytmu; pomijamy tutaj opis zasady działania sieci Feistela. Została ona opisana we wstępnych rozdziałach tej pracy. Na rysunku 7.1 został przedstawiony jedynie ogólny schemat szyfrowania i deszyfrowania.



Rys. 7.1 Budowa szyfru „crypt3".

S-bloki S_1 i S_2 są takie same, jak wykorzystywane w algorytmie DES, zasady permutacji P_1 oraz S-bloki pokazano w tabeli 7.1.

Tabela 7.1 Permutacja i podstawienia wykorzystywane przez algorytm „crypt3”.

$$P_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 2 & 1 & 12 & 4 & 5 & 11 & 10 & 9 & 9 & 7 & 6 & 3 \end{pmatrix}$$

S1:

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

S2:

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

Powyższy szyfr jest trzyrundowym algorytmem opartym na sieci S-P, więc ta sama funkcja wykorzystywana jest do szyfrowania i deszyfrowania. Oczywiście, przy szyfrowaniu podklucze poszczególnych rund są używane w odwrotnej kolejności. W każdej rundzie używany jest 16 bitowy klucz, trzy podklucze losowane są jako ciąg 36 bitowy, który następnie dzielony jest na trzy części. Poza operacjami permutacji (P_1) i podstawienia (S_1 i S_2) algorytm wykorzystuje jeszcze sumę modulo 2 ($\oplus$) zarówno w funkcji szyfrującej f jak i schemacie Feistela. Ciąg bitów R_i tekstu jawnego w pierwszym kroku jest poddawany działaniu funkcji $\oplus$ wraz z kolejnymi bitami klucza. Na wejścia S-bloków S_1 i S_2 trafia po 6 bitów, wynik działania każdego z S-bloków to 4 bity. Permutacji P_1 jest poddawany ciąg 12 bitów, wynik działania S_1 jest umieszczany na początku i końcu ciągu bitów. W ten sposób uzyskiwany jest ciąg 12 bitowy. Wynik działania rundy pierwszej trafia jako wejściowy ciąg bitów dla rundy kolejnej. Każda runda używa innego z trzech kluczy, o czym mowa była wcześniej. Do deszyfrowania używany jest ten sam algorytm, co zobrazowane zostało na rysunku 7.1. Podklucze w procesie deszyfrowania używane są w odwrotnej kolejności. W dalszej części rozdziału pokazany zostanie proces przygotowania neuronowej realizacji algorytmu „crypt3”.

7.2. Neuronowa realizacja permutacji

Sieć neuronowa wykorzystująca neurony z wagami zmienno przecinkowymi.

W rozdziale 6.2. tej pracy przedstawiona została koncepcja neuronowego bloku elementarnego zdolnego do wykonania permutacji dwóch bitów. W tym rozdziale do zrealizowania jest permutacja 12 bitów. Sieć neuronowa, która zostanie do tego użyta zapewnić musi możliwość realizacji wszystkich możliwych permutacji pomiędzy 12 wejściami a 12 wyjściami układu P_1. Można to rozwiązać na kilka sposobów (wykorzystać różne bloki elementarne, rozdział 6.2). W tym miejscu zastosowana zostanie sieć zbudowana z 12 bloków elementarnych przedstawionych na rysunku 6.3. Algorytm „Crypt3" stosuje konkretną permutację, jednak założeniem tego rozdziału jest możliwość konstrukcji neuronowego układu szyfrującego, który pozwoliłby dokonywać zmian w działaniu elementów P_1, S_1 i S_2. Konstrukcja do realizacji permutacji 12 bitów składać się będzie z trzech warstw bloków elementarnych. Dla pełnej realizacji tego zadania we wspomnianym bloku wprowadzona zostanie drobna zmiana. Funkcja aktywacji neuronów wykorzystanych do budowy bloków w warstwach A i C (rysunek 7.2) przedstawia się następująco:

$$f = \begin{cases} 1, & \varphi > p \\ 2, & \varphi < p \end{cases}$$

Zmiana ta jest konieczna, jeśli sieć permutująca, która jest w tym rozdziale konstruowana ma faktycznie umożliwiać realizowanie dowolnych permutacji. Warstwy sieci połączone są ze sobą na zasadzie: każde wyjście nieparzyste „a" połączone jest z każdym wejściem nieparzystym „b", a każde „c" z „d" poprzez neuron, który posiada 6 wag. Tylko jedna z tych wag może przyjmować wartość 1, pozostałe przyjmują wartość 0. Dzięki temu uzyskuje się możliwość wyboru połączenia. Każde parzyste wyjście „a" jest połączone tylko z odpowiadającym mu wejściem „b", analogicznie „c" z „d". Taka konfiguracja warstwy pośredniej zapewnia szerokie możliwości konstrukcji konkretnych permutacji. Na rysunku 7.2 pokazany jest ogólny schemat sieci nauczonej, realizującej permutację P_1.

Permutacja realizowana przez całą sieć:

$$P_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 2 & 1 & 12 & 4 & 5 & 11 & 10 & 8 & 9 & 7 & 6 & 3 \end{pmatrix} =$$

$$= A \circ B \circ C \circ D \circ E =$$

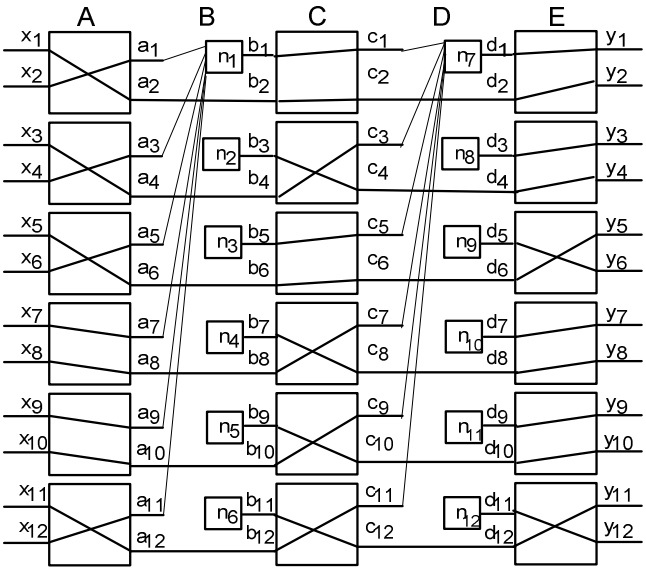
$$= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 2 & 1 & 4 & 3 & 6 & 5 & 7 & 8 & 9 & 10 & 12 & 11 \end{pmatrix} \circ$$

$$\circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 1 & 2 & 3 & 4 & 11 & 6 & 9 & 8 & 7 & 10 & 5 & 12 \end{pmatrix} \circ$$

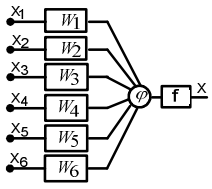
$$\circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 1 & 2 & 4 & 3 & 5 & 6 & 8 & 7 & 10 & 9 & 12 & 11 \end{pmatrix} \circ$$

$$\circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 1 & 2 & 11 & 4 & 3 & 6 & 9 & 8 & 7 & 10 & 9 & 12 \end{pmatrix} \circ$$

$$\circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 1 & 2 & 3 & 4 & 6 & 5 & 7 & 8 & 9 & 10 & 12 & 11 \end{pmatrix}$$



Neuron n_i :



$$, \varphi_n = \sum w_i^n x_k \ , \ f = \begin{cases} 0, & \varphi > p \\ 1, & \varphi < p \end{cases} , y_l = f(\varphi_n)$$

Rys. 7.2 Sieć realizująca permutację P1 z algorytmu „Crypt3”.

Zapis permutacji zamieszczony na rysunku 7.2 oddaje zasadę działania sieci. Oczywiście tę samą permutację (P_1) można zrealizować na wiele innych sposobów. Cała sieć składa się z 36 dwuwejściowych neuronów, których wagi przyjmują wartości zmiennoprzecinkowe w procesie uczenia oraz 12 neuronów, przyjmujących wartości wag 0 lub 1. Bloki z warstw A, C, E poddawane są procesowi uczenia niezależnie (opis procesu uczenia przedstawiony

został w rozdziale 6). Wagi neuronów z warstwy B i D mogą być ustalone na drodze procesu uczenia lub po prostu mogą być nadpisywane arbitralnie na zasadzie wstawienia wartości „1" dla wagi odpowiadającej wejściu, z którego sygnał ma być przekazywany dalej, do warstwy C lub E. Przygotowanie przebiegu procesu uczenia rozpocząć należy od rozpisania permutacji P_1 na 5 permutacji elementarnych. Następnie należy ustalić wartości losowe na wagach warstwy A, D, E i korzystając z odpowiednich zbiorów uczących przeprowadzić proces uczenia. Natomiast wagi neuronów warstwy B i D wyzerować, po czym ustalić wartości „1" na odpowiednich wagach kierując się ustalonymi odpowiednio permutacjami B i D. Jeśli założyć, że przy zmianie permutacji realizowanej przez sieć dokonywane zmiany maja dotyczyć wszystkich neuronów, to wiedza sieci „jak szyfrować" zawiera się w 36 wagach przyjmujących wartości zmiennoprzecinkowe oraz 72 wagach przyjmujących wartość 0 lub 1. Oczywiście sporo możliwości realizacji różnych permutacji daje jedynie zmiany w warstwach A, D, E bez ingerowania w warstwy B i D. Wtedy liczbę wag decydujących o zmianie realizowanej permutacji można minimalizować.

Sieć neuronowa wykorzystująca neurony boolowskie

Alternatywą rozwiązania zaproponowanego powyżej jest realizacja permutacji $P1$ z wykorzystaniem modułów elementarnych opartych na neuronach boolowskich, przedstawionych w rozdziale 6.2 tej pracy. We wspomnianym rozdziale pokazane zostały neuronowe realizacje permutacji 2 lub 4 bitów. Jednym z możliwych rozwiązań jest zamiana modułów elementarnych operujących na dwóch bitach w strukturze zaproponowanej na rysunku 7.2. Inną możliwością jest wykorzystanie bloku elementarnego dla permutacji 4 bitów, rysunek 6.7. W końcu trzecią możliwością jest konstrukcja dedykowanej sieci neuronowej jako modułu z 12 wejściami i 12 wyjściami. Najprostszym do porównania rozwiązaniem jest wykorzystanie boolowskiego bloku neuronowego do realizacji permutacji dwóch bitów, korzystając dokładnie ze struktury opisanej na rysunku 7.2. Ilościowo oba rozwiązania wykorzystują taką samą liczbę neuronów i wag. Z tą jednak zasadniczą różnicą, że przy rozwiązaniu drugim wszystkie wagi mogą przyjmować wartości 0 lub 1. Daje to oszczędność w zapotrzebowaniu na ilość pamięci potrzebnej do przechowywania informacji o wartościach wag. Przy stosowaniu neuronów z wagami przyjmującymi wartości pomiędzy 0 a 1 na potrzeby realizacji sieci realizującej permutacje istotne są wartości do trzech miejsc po przecinku. Średnio na jedną wagę przypada 10 bitów. Natomiast w przypadku korzystania z sieci boolowskich na jedną wagę przypada 1 bit. Pokazuje to tabela 7.2.

Tabela 7.2 Statystyka modułu realizującego permutację P1.

	Sieć z wagami dziesiętnymi	Sieć z wagami binarnymi
Liczba neuronów	36 + 12 = 48	36 + 12 = 48
Liczba wag	72 + 72 = 144	72 + 72 = 144
Liczba bitów na potrzeby zapisu wartości wag	72 * 10 + 72 * 1 = 792	72 * 1 + 72 * 1 = 144

7.3. Neuronowa realizacja podstawienia

Sieć realizująca S-blok S_1, jak i proces uczenia opisane zostały już w rozdziale 6.3 W tym rozdziale omówiony zostanie więc proces przygotowania modułu „SN-box" do realizacji S-bloku S_2.

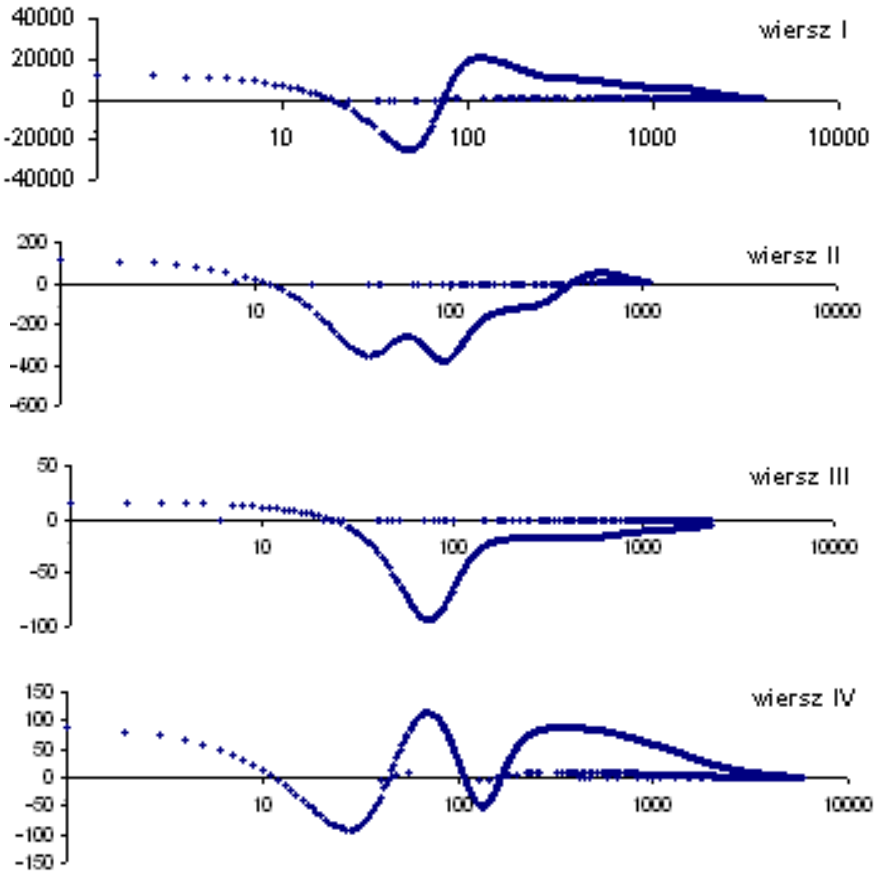
Przygotowanie modułu „p-w-d"

Zacząć należy od przypisania wagom sieci, pokazanej na rysunku 6.13, wartości losowych. Kolejny etap to przygotowanie zbioru uczącego dla poszczególnych wierszy tablicy S2, co obrazuje tabela 7.3.

Tabela 7.3 Zbiór uczący dla modułu „p-w-d" dla poszczególnych wierszy S-bloku S2 algorytmu „crypt3".

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	Y (w I)	y (w II)	y (w III)	y (w IV)
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	15	3	0	13
0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	13	14	8
0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	8	4	7	10
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	14	7	11	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	6	15	10	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	11	2	4	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	3	8	13	4
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	14	1	2
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	9	12	5	11
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	7	0	8	6
0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	2	1	12	7
0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	13	10	6	12
0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	12	6	9	0
0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	9	3	5
0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	5	11	2	14
0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	10	5	15	9

Proces uczenia dla poszczególnych wierszy realizowany był dla wierszy 1,2 i 4 przy współczynniku uczenia 0,001, natomiast wiersz 3 wymagał użycia współczynnika uczenia 0,0001 co wpłynęło na zwiększoną liczbę cykli uczących. Liczba cykli uczących wyniosła odpowiednio dla : I-3211, II-1913 III-5014, IV 2367. Charakterystyka przebiegu uczenia modułu „p-w-d" na potrzeby realizacji S-bloku S_2 pokazana jest na rysunku 7.3.



Rys. 7.3 Przebieg procesu uczenia dla S-bloku 2.

Następny krok to ustalenie wartości wyjściowych, jakie pojawiać się będą na wyjściu nauczonych układów „p-w-d” po połączeniu z układem „1z16”. Wartości te zapisano w tabeli 7.4.

Tabela 7.4 Działanie nauczonych modułów „p-w-d”.

x1	x2	x3	x4	Wiersz I	Wiersz II	Wiersz III	Wiersz IV
0	0	0	0	-7,187752	1,2156844	-0,044494	-5,977422
0	0	0	1	6,7515129	7,1083753	44,312048	2,02288374
0	0	1	0	-6,974683	13,154383	20,370625	-28,901592
0	0	1	1	-31,50673	2,8013859	4,4010615	-17,95242
0	1	0	0	-39,60884	15,235315	19,387444	-33,89438
0	1	0	1	-14,85882	3,7926926	-0,741464	-3,0106231
0	1	1	0	-6,805321	-0,743851	9,3803892	3,96206542
0	1	1	1	0,6963806	13,801523	24,88011	-7,9814173
1	0	0	0	-9,847779	27,022104	-3,665319	17,9648434
1	0	0	1	4,9192717	-1,421925	7,3249168	-11,987307
1	0	1	0	-19,08171	6,4303015	18,268664	-2,9699041
1	0	1	1	-14,90744	4,4707664	25,99889	0,99526646
1	1	0	0	-2,964861	1,8937581	12,304244	-5,0146183
1	1	0	1	-11,19757	1,3675942	43,193268	12,9720556
1	1	1	0	-17,94989	25,588312	1,0742861	-14,954106
1	1	1	1	14,853629	14,588175	17,44677	-12,959632

Następnie przygotowane zostają moduły „dec2dec" wersji 2, według zasady opisanej w rozdziale 6.3. Po wykonaniu tych czynności, przygotowane w realizacji neuronowej zostały przekształcenia permutacji i podstawienia, wykorzystywane w algorytmie „Crypt3". Możliwym jest, więc złożenie funkcji f, rysunek 7.1. Można, więc przystąpić do podsumowania ilościowego, którego wynika prezentuje tabela 7.5.

Tabela 7.5 Statystyka neuronowej realizacji szyfru „Crypt3".

	Permutacja P1	S-blok S1 i S2	Suma
Liczba neuronów	38 + 12 = 46	332 * 2 = 664	710
Liczba wag	76 + 72 = 148	672 * 2 = 1344	1492

W tym miejscu warto przypomnieć o tym, że szczególnie w przypadku S-bloków, w tych sieciach są możliwości ich optymalizacji pod względem liczby wykorzystywanych neuronów. W kolejnym rozdziale przeprowadzona zostanie analiza możliwych korzyści wynikających z realizacji algorytmów szyfrujących za pomocą sieci neuronowych. Analiza ta przeprowadzona będzie na przykładzie omawianego w tym rozdziale szyfru „Crypt3".

Rozdział 8. Możliwości wykorzystania neuronowego układu szyfrującego

8.1. Sieć neuronowa jako uniwersalny układ szyfrujący

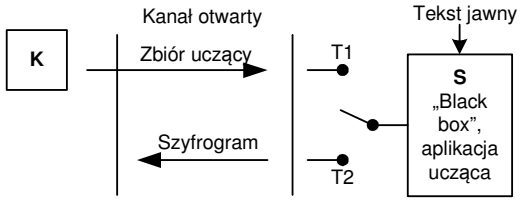
Wyjaśnienia wymaga jeszcze jedno zagadnienie. Sieci neuronowe podają zawsze wynik przybliżony, tak jest również w przypadku sieci stosowanych do szyfrowania w tej pracy. Jednak, jeśli neuronowa konstrukcja permutacji oraz S-bloków zostaną przygotowane poprzez odpowiednio przeprowadzone procesy uczenia i adaptacji, według zasad opisanych w rozdziale 6, to całość będzie udzielała na wyjściu odpowiedzi dokładnych. Kluczowe znaczenie mają tu odpowiednio dobrane funkcje aktywacji.

Posiadając przygotowaną sieć realizującą konkretną funkcję szyfrującą f, np. szyfr „Crypt3", w prosty sposób można doprowadzić do zmiany realizowanych przekształceń, permutacji lub podstawienia. Zakładamy, że struktura sieci nie podlega zmianie, ponieważ zmiany można dokonać poprzez odpowiednie dokonanie zmian związanych z wartościami wag poszczególnych neuronów. Tak więc zmiana algorytmu – szyfru realizowanego przez sieć dokonuje się poprzez prowadzony w określonych warunkach proces uczenia. We wspomnianym procesie uczenia wykorzystywana jest metoda uczenia z nauczycielem. Przyjmując założenie, że skonstruowaną odpowiednio sieć traktujemy jako czarną skrzynkę umieszczoną na przykład gdzieś na odległym serwerze, w jakiś sposób konieczna jest możliwość zdalnego wpływania na realizowane przez nią przekształcenie. Zakładamy, że rozważany układ pracuje w układzie klient-serwer. Rolę klienta będzie pełnił „właściciel" czarnej skrzynki, który będzie chciał decydować jakie przekształcenie szyfrujące ma realizować owa czarna skrzynka, czyli serwer.

Jak już zostało to wspomniane wcześniej zmiana algorytmu odbywa się poprzez przeprowadzenie procesu uczenia. Do realizacji tego procesu można podejść na dwa sposoby. Po pierwsze, cały proces uczenie może zostać przeprowadzony po stronie sieci neuronowej (czyli na serwerze). Drugą możliwością jest przeprowadzenie procesu uczenia po stronie klienta na takiej samej sieci, która jest umieszczona na serwerze. Dalej następuje dostarczenie do sieci na serwerze nowych wartości wag. Obydwie metody wymagają przeanalizowania i oceny pod dwoma aspektami. Jest to kwestia związana z wydajnością i „obciążalnością", przez proces uczenia, zasobów klienta i serwera oraz analiza pod kątem bezpieczeństwa jednego i drugiego rozwiązania.

8.2. Uczenie po stronie serwera (Przesyłanie zbioru uczącego)

Decydując się na przeprowadzenie procesu uczenia przez serwer (S), niezbędne jest dostarczenie zbioru uczącego skonstruowanego po stronie klienta (K) do aplikacji odpowiedzialnej za przeprowadzenie procesu uczenia. Na rysunku 8.1 przedstawiony jest schematycznie wspomniany wcześnie układ $K-S$. Układ S może pracować w dwóch trybach. Pierwszym jest tryb uczenia (tryb $T1$), gdy dostarczany jest zbiór uczący i przeprowadzony przez aplikację uczącą proces zmiany realizowanego szyfru. Drugim trybem ($T2$) jest praca czarnej skrzynki realizującej szyfrowanie tekstu jawnego. Sposób realizacja dwóch trybów w neuronowym układzie „SN-box" został pokazany na rysunku 6.34. Przez kanał otwarty rozumiemy sieć, która nie jest chroniona, np. Internet.



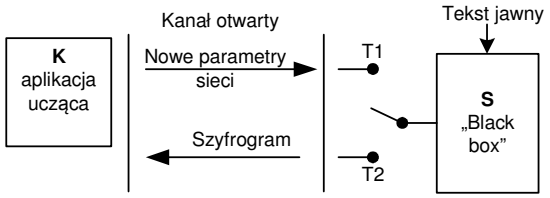
Rys. 8.1 Uczenie po stronie serwera.

W rozważanym przypadku po stronie serwera układ w postaci sieci neuronowej realizuje opisany wyżej szyfr „Crypt3" używając S-bloków (S_1 i S_2) oraz permutacji (P_1). W momencie, gdy zapada decyzja o wprowadzeniu zmian w realizowanym algorytmie po stronie klienta (K), gdzie znajduje się taka sama sieć jak po stronie serwera (S), konstruowany jest zbiór uczący. Zbiór uczący musi zostać dostarczony do aplikacji serwera obsługującej proces uczenia sieci. Zakładając, że transmisja zbioru uczącego ma się odbyć przez ten sam niebezpieczny kanał wymiany informacji, postawić należy pytanie o bezpieczeństwo takiego procesu. Jeśli założymy, że intruz podsłuchuje kanał transmisji przyjąć należy, że wejdzie on w posiadanie zbioru uczącego. Jeśli przesłany zostanie kompletny zbiór uczący, to intruz na podstawie jego analizy będzie w stanie ustalić realizowane przekształcenia, ponieważ zbiór uczący taką informację zawiera. Istotną informacją dla atakującego jest tu struktura sieci. Można przyjąć, że struktura sieci jest tajemnicą, jednak byłoby to nie zawsze rozwiązanie pożądane. Realizując proces uczenia po stronie serwera, poza kwestiami bezpieczeństwa należy wziąć pod uwagę jeszcze inne elementy. Jednym z nich jest obciążenie procesem uczenia serwera oraz wyposażenie go dodatkowo w aplikację odpowiedzialną za przeprowadzenie tego procesu. Wspomniany koszt

ma tu znaczenie tym większe, jeśli przyjmiemy, że dysponowalibyśmy wieloma serwerami rozsianymi w sieci.

8.3. Uczenie po stronie klienta

Alternatywą dla tego rozwiązania jest przeprowadzenie procesu uczenia po stronie klienta. Do serwerów natomiast przesyłane są nowe parametry sieci. Wiedza sieci neuronowej, w tym przypadku dotycząca tego jak szyfrować, przechowywana jest w wartościach wag. Podczas trybu uczenia, który został tu zastosowany, zmianie ulegają wartości wag natomiast nie zmienia się struktura sieci. Tak więc rezultatem uczenia są zmienione wartości wag sieci. Do serwera należałoby więc dostarczyć jedynie nowe wartości wag.



Rys. 8.2 Uczenie po stronie serwera

Jak zostało to pokazane na rysunku 8.2, serwer nie jest już wyposażony w moduł uczący. Serwer nadal pracuje w dwóch trybach. T_1 to tryb, w którym następuje aktualizacja wartości wag, czyli zmiana realizowanego algorytmu szyfrującego. Tryb T_2 natomiast jest nazwijmy to normalnym trybem pracy, czyli szyfrowaniem.

Rozwiązanie jest zdecydowanie mniej kosztowne w sytuacji, gdy istnieje wiele serwerów, na których chcemy mieć możliwość zmiany realizowanego algorytmu szyfrującego. Moduł po stronie serwera będzie zdecydowanie mniej skomplikowany, co w zasadzie pozwala na całkowitą automatyzację jego działania. Realizację tego modelu zmiany działania układu szyfrującego można przeprowadzić na dwa sposoby.

a) Przesyłanie nowych wartości wag

Poprzez przesłanie nowych parametrów sieci, rysunek 8.2, rozumiemy dostarczenie nowych wartości wag. Wymogiem podstawowym przy takim podejściu jest oczywiście to, aby po stronie serwera i klienta znajdowała się sieć neuronowa o jednakowej architekturze. Po przeprowadzonym procesie uczenia po stronie klienta nowo ustalone wartości wag muszą zostać dostarczone do serwera, ten natomiast po przełączeniu w tryb T_1, dokonuje nadpisania wartości wag. Od tego momentu, po przełączeniu serwera w tryb T_2, rysunek 8.2, realizowany jest inny algorytm szyfrujący. Z punktu widzenia bezpieczeństwa całego procesu

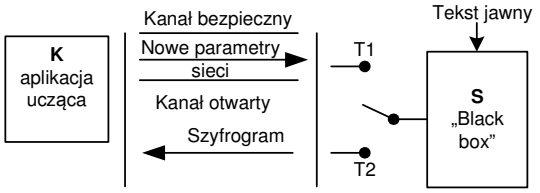
zmiany algorytmu realizowanego przez sieć, należy zastanowić się jaką faktycznie informację zawierają same wartości wag. Zakładając, że transmisja odbywa się kanałem otwartym pytanie można postawić inaczej: co intruz podsłuchujący ten kanał uzyska w wyniki przechwycenia przesyłanych informacji? Same wartości wag to wiedza sieci neuronowej na temat tego jak ma szyfrować. Jednak wiedza ta bez znajomości struktury sieci jest w zasadzie bezużyteczna. Tutaj pojawia się różnica, w stosunku do układów programowalnych, w przypadku których należałoby przesłać program, zawierający dokładny opis algorytmu. Natomiast w przypadku „sterowania" siecią neuronową przesyłany jest zbiór wartości, którego sama znajomość właściwie nic nie mówi intruzowi o realizowanym algorytmie przez sieć. Jest to bezpieczne pod jednym warunkiem, że intruz nie posiada informacji na temat struktury sieci.

b) Przesyłanie różnicy wag

Jeśli przyjmiemy, że pojawienie się w otwartym kanale wymiany informacji nowych wartości wag jest pewnym problemem z punktu widzenia bezpieczeństwa, to można utrudnić całą operację potencjalnemu intruzowi. Zmuszając go do prowadzenia nasłuchu ciągłego i gromadzenia przechwyconych informacji. Zakładamy, że nie będą przesyłane kompletne zestawy wartości nowych wag (W'), a jedynie ich różnice (R) w stosunku do wartości poprzednich (W). Aplikacja po stronie serwera posiada stan poprzedni sieci neuronowej, czyli wartości wag. Na podstawie otrzymanych różnic pomiędzy wartościami starymi a nowymi będzie w stanie przez zwykłą operację dodawania ustalić nowe wartości wag. Prowadzenie sterowania realizowanym algorytmem przez sieć neuronową w taki sposób wymaga oczywiście ustalenia stanu początkowego po stronie serwera i klienta. Każda kolejna zmiana algorytmu realizowanego przez sieć będzie wymagała znajomości stanu poprzedniego. Tak więc osoba podsłuchująca transmisję będzie musiała posiadać wiedzę na temat stanu początkowego. Ponadto konieczne jest w takim wypadku prowadzenie nasłuchu ciągłego w celu gromadzenia informacji na temat kolejnych stanów sieci. Jeżeli założymy, że stan początkowy sieci ustalany będzie w tajemnicy pomiędzy stroną klienta i serwera, to utrzymanie w tajemnicy struktury sieci nie będzie już warunkiem koniecznym dla zachowania bezpieczeństwa.

8.4. Wykorzystanie kanału bezpiecznego

Wyżej prowadzone rozważania zakładały, że transmisja zbioru uczącego czy wartości wag lub ich różnicy odbywała się w kanale otwartym (niebezpiecznym). Można jednak do transmisji danych, niezbędnych do zmiany algorytmu realizowanego przez sieć, użyć kanału bezpiecznego tak, jak to przedstawia rysunek 8.3.



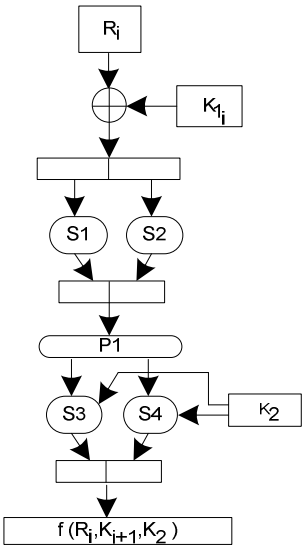
Rys. 8.3 Wykorzystanie kanału bezpiecznego.

Wykorzystanie kryptografii asymetrycznej rozwiązuje problem bezpiecznego przeprowadzenia zmiany w neuronowym układzie szyfrującym w sposób bezpieczny. Zwiększa to jednak obciążenie serwera, ponieważ kryptografia asymetryczna absorbuje duże zasoby systemowe. Sam serwer i klient natomiast muszą mieć dodatkowo zaimplementowane odpowiednie algorytmy niezbędne do realizacji kanału bezpiecznego.

Rozdział 9. Możliwości wykorzystania neuronowego układu szyfrującego

9.1. Wykorzystanie sieci neuronowej jako element klucza

W zastosowaniach praktycznych algorytmów szyfrujących bardzo często wykorzystuje się klucze sesji. Mechanizm polega na tym, że na potrzeby poszczególnych sesji (w rozumieniu wymiany informacji, np. podczas jednego połączenia z odległym serwerem) wykorzystywany jest ten sam algorytm, jednak zmieniają się stosowane klucze. W kryptografii symetrycznej problem dystrybucji klucza; przy kluczach zmieniających się co sesję problem występuje często. Istnieje kilka możliwych rozwiązań, kryptografia asymetryczna, szyfrowanie nowego klucza starym, protokoły uzgadniania i przesyłania klucza. Wydaje się, że zastosowanie realizacji neuronowej S-bloku stwarza inną możliwość. Z założenia, w dzisiejszej kryptografii (przynajmniej tej komercyjnej) jest tak, że algorytm powinien być jawny. Co ważne zawartość S-bloków (o ile szyfr je wykorzystuje) jako elementów nieliniowych, czyli w znacznym stopniu decydujących o bezpieczeństwie szyfrowania, również powinna być upubliczniona. Jednak jeśli potraktować jeden lub kilka z S-bloków spośród wielu stosowanych w algorytmie jako klucz, a dokładniej jego dopełnienie, to pojawia się tu pewna możliwość wykorzystania modułu „SN-blok". Znanym algorytmem, który korzysta z tajnych S-bloków jest symetryczny szyfr GOST [73]. W tym rosyjskim szyfrze (zaprojektowanym jeszcze za czasów ZSSR) tajne S-bloki wykorzystywane są jako dopełnienie klucza. Stosowanie w całym algorytmie tylko samych tajnych S-bloków może rodzić podejrzenia dotyczące bezpieczeństwa samego algorytmu. Przypomnieć można tu chociażby kontrowersje, jakie wzbudził DES i jego jawne S-bloki [5], [6]. Pokazuje to, jakie znacznie w szyfrowaniu dla bezpieczeństwa mają S-bloki. Można natomiast zaproponować rozwiązanie stanowiące rozszerzenie algorytmu „Crypt3" jakie przedstawione zostało na rysunku 9.1 („Crypt3a").



Rys. 9.1 Rozszerzenie „a" szyfry „Crypt3".

Zakładamy, że algorytm „Crypt3a" ma jawne, silne S-bloki S_1 i S_2, (o tym co to są silne S-bloki wspomniano w rozdziale 3.1.4). Zawartość S_3 i S_4 można potraktować jako klucz, oznaczony na rysunku 9.1 jako K_2. Jeśli natomiast S_3 i S_4 zostaną zrealizowane za pomocą modułu „SN-bolok" to pod postacią klucza K_2 kryją się wartości wag sieci „p-w-d", rysunek 6.33. Jeśli teraz przyjąć, że do przesłania nowego klucza K_2 wykorzystamy protokół zaproponowany w rozdziale 8.3 w części b, to dostarczenie klucza K_2 mogłoby odbyć się otwartym tekstem. Możliwości wykorzystania modułu „SN-blok" wydają się być pod tym względem dość duże. Dodatkową kluczową informacją może być struktura sieci „1z16", rozdział 6.3, lub stosowany zbiór uczący pokazany w tabeli 6.14. Zarówno w jednym, jak i w drugim przypadku możliwe jest zrealizowanie zadania tych modułów przy różnych strukturach sieci lub zbiorach uczących. W wyniku przeprowadzonych eksperymentów w toku badań okazało się, że zmiany, jakich można dokonywać w strukturze sieci (stosowanie różnych bloków elementarnych) czy zbiorach uczących posiadają spory zakres. Wynika to z pewnością z faktu pewnej „nadmiarowości" jeśli chodzi o strukturę zaproponowanego rozwiązania. Jednak być może ta nadmiarowość może się okazać przydatna w rozwiązaniu, które zostało opisane w tym rozdziale.

Wykorzystanie wartości wag bloku „SN-blok" jako klucza wydaje się wygodne z powodu nie przysłania wprost wartości klucza. Przy założeniu, że warunki początkowe sieci modułu „p-w-d" są tajne i znane tylko uprawnionym osobom, przysłanie tylko różnicy powoduje konieczność znajomości stanu poprzedniego sieci. Kolejnym rozbudowaniem całego systemu

może też być zastosowanie modułu „dec2dec” w wersji pierwszej, rysunek 6.13, co zwiększy przestrzeń kluczy bez zwiększania rozmiaru S-bloku. Choć oczywiście zaproponowana koncepcja sieci modułów „SN-box” w układzie modułowym pozwala na łatwą rozbudowę rozmiarów S-bloku.

Zamiast wykorzystania modułu S_3 i S_4, rysunek 9.1, można oczywiście zastosować sieć realizującą permutacje. Jednak nieliniowe S-bloki wydają się do tego lepsze. Pewnym problemem jest oczywiście generowanie silnych S-bloków, jednak jest to problem możliwy do rozwiązania. Pewną pomocą w tym przypadku może być zwiększanie rozmiarów tajnych S-bloków.

9.2. Wykorzystanie sieci neuronowej jako elementu generowania klucza cyklu

Jeśli używamy algorytmu szyfrującego w architekturze sieci S-P to używane są różne algorytmy tworzenia kluczy dla poszczególnych cykli. Możliwym wydaje się wykorzystanie do realizacji generowania kluczy dla poszczególnych cykli sieci neuronowej. Jeśli założylibyśmy, że przed szyfrowaniem strony chcące wymieniać informacje w sposób bezpieczny ustalają wspólny tajny klucz, to kolejne klucze cyklu powinny być tworzone po jednej i drugiej stronie w taki sam sposób. Na potrzeby prowadzonych tu rozważań przyjmiemy, że algorytm generowania klucza wykorzystuje permutacje przedstawioną na rysunku 9.2. Na rysunku 9.3 pokazano również wykorzystanie proponowanego rozwiązania w schemacie sieci S-P.

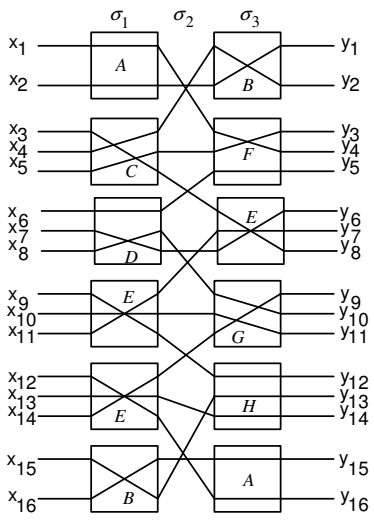
$$\sigma = (\sigma_1 \circ \sigma_2 \circ \sigma_3) =$$

$$= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 1 & 2 & 5 & 3 & 4 & 6 & 8 & 7 & 11 & 10 & 9 & 14 & 13 & 12 & 16 & 15 \end{pmatrix} \circ$$

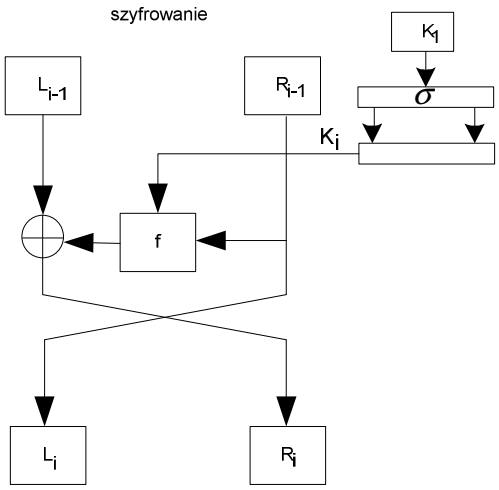
$$\circ \begin{pmatrix} 1 & 2 & 5 & 3 & 4 & 6 & 8 & 7 & 11 & 10 & 9 & 14 & 13 & 12 & 16 & 15 \\ 3 & 2 & 6 & 1 & 4 & 5 & 8 & 9 & 12 & 10 & 7 & 16 & 14 & 11 & 13 & 15 \end{pmatrix} \circ$$

$$\circ \begin{pmatrix} 3 & 2 & 6 & 1 & 4 & 5 & 8 & 9 & 12 & 10 & 7 & 16 & 14 & 11 & 13 & 15 \\ 4 & 1 & 8 & 2 & 3 & 5 & 6 & 10 & 12 & 11 & 7 & 16 & 14 & 9 & 13 & 15 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 4 & 1 & 8 & 2 & 3 & 5 & 6 & 10 & 12 & 11 & 7 & 16 & 14 & 9 & 13 & 15 \end{pmatrix}$$



Rys. 9.2 Permutacja użyta na potrzeby generowania klucza.



Rys. 9.3 Wykorzystanie sieci neuronowej do generowania klucza cyklu.

Używając wspomnianych w rozdziale 6.2 tej pracy układów elementarnych skonstruowano sieć realizującą określoną permutację 16 bitów. Jest to sieć składającą się z dwóch warstw. Każda warstwa realizuje inną permutację, co odpowiednio zostało oznaczone jako σ_1, σ_3. Permutacja σ_2 odzwierciedla połączenie pomiędzy warstwami sieci. Na rysunku 9.2 pokazana została permutacja realizowana przez sieć. Obie strony ustalają w tajemnicy warunki początkowe sieci, co oznacza początków wartości wag. Następnie jedna ze stron ustala 16 bitowy wektor inicjujący K_1 i wysyła go do strony drugiej jawnym tekstem. Dla ustalenia wspólnego klucza cyklu obie strony przeprowadzają proces uczenia, opisany w rozdziale 6.2, wykonując np. 10 cykli uczących, wartość również ustalona wcześniej. Odpowiedź sieci po wykonanych 10 cyklach, przy podanym na wejściu sieci wektorze inicjującym K_1, jest kluczem cyklu szyfrowania. Operacja powtarzana jest tyle razy, ile cykli ma być wykonanych przez dany algorytm. Tu uwaga, w kolejnych procesach uczenia jako wektor K_1 przyjmowany jest ostatnio użyty kluczy.

9.3. Wykorzystania sieci neuronowej do realizacji szyfrów tajnych

Jedną z możliwych dróg wykorzystania proponowanych w tej pracy rozwiązań w kryptografii, może być realizacja szyfrów tajnych. W kryptografii nie wszystkie algorytmy szyfrowania są jawne. Przykładem może być chociażby algorytm Skipjack, opracowany przez NSA (The National Security Agency/Central Security Service is America's cryptologic organization). Z założenia implementacje tego algorytmu miały być realizowane tylko w postaci układu scalonego zabezpieczonego przed demontażem. Realizacje neuronowe szyfrów mogą stwarzać tu nowe możliwości. Dysponując siecią neuronową nauczoną realizacji określonego algorytmu można korzystać z tego algorytmu jednak, na podstawie np. samych wartości wag, zadaniem trudnym jest odtworzenie stosowanego algorytmu. Istnieją tak zwane metody ekstrakcji reguł [74], jednak przy dużych sieciach i wartościach wag zmiennoprzecinkowych metody te nie dają wyników na poziomie zadawalającym z punktu widzenia kryptografii. Sieć neuronowa mogłaby więc pełnić rolę czegoś w rodzaju uniwersalnej „czarne skrzynki". Zastosowanie sieci neuronowej w takiej roli daje jeszcze jedną możliwość. Jest nią wprowadzanie zmian na odległość w tajnych algorytmach, z zachowaniem tajności zmienianego algorytmu. Zmiana realizowanego przez sieć neuronową algorytmu mogłaby się odbyć według protokołu zaproponowanego w rozdziale 8.4 tej pracy.

Podsumowanie i wnioski

W pracy tej przedstawiona została koncepcja budowy neuronowego układu szyfrującego. Postawiona na początku pracy teza została udowodniona. Wykazano, że alternatywą dla stosowanych obecnie sposobów implementacji algorytmów kryptograficznych mogą być implementacje neuronowe.

Najważniejszym osiągnięciem pracy jest realizacja za pomocą sieci neuronowych szyfru blokowego opartego na modelu sieci S-P. Kompletna realizacja szyfru blokowego opisana została w rozdziale 7. Na końcowy wynik pracy złożyła się realizacja szeregu zadań pośrednich. Pierwszym postawionym problemem była budowa sieci neuronowych zdolnych do realizacji permutacji bitów tekstu jawnego. Zaproponowane zostały dwa możliwe sposoby neuronowych implementacji operacji liniowych w szyfrach blokowych. Pierwsze rozwiązanie wykorzystuje sieć składającą się z neuronów posiadających wagi zmiennoprzecinkowe. Innym zaproponowanym podejściem jest implementacji permutacji za pomocą neuronów boolowskich. Obie koncepcje przedstawiono w rozdziale 6.2. Drugim ważnym etapem badań, zaprezentowanym w rozdziale 6.3, była neuronowa realizacja przekształceń nieliniowych, czyli S-bloków.

Ważnym efektem pracy, zaprezentowanym w rozdziale 8, jest opracowanie koncepcji protokołu kryptograficznego, który precyzuje możliwości wykorzystania proponowanych rozwiązań w sieciach informatycznych. W rozdziale 9 przedstawiono również propozycje wykorzystania neuronowych implementacji funkcji liniowych i nieliniowych jako rozbudowę możliwości znanych algorytmów szyfrujących.

Pośrednim osiągnięciem jest opracowanie modułowej koncepcji budowy neuronowego układu szyfrującego. Opracowano koncepcję budowy neuronowych modułów elementarnych między innymi neuronową implementację dekodera wartości dziesiętnych na binarne. Na potrzeby prowadzenia eksperymentów numerycznych stworzono oryginalne oprogramowania oraz sposób opisu struktur neuronowych. Pozwoliło to na sprawną konstrukcję sieci o dowolnej architekturze oraz na pełną kontrolę funkcjonowania projektowanych układów.

Kolejnym zrealizowanym celem, jaki został postawiony na początku badań, jest uniwersalność proponowanego rozwiązania. Przy niezmiennej strukturze układu neuronowego możliwe jest wprowadzanie zmian w realizowanym algorytmie szyfrującym. Elementem wpływającym na realizowany szyfr są odpowiednio ustalone wartości wag. Sposób modyfikacji szyfru realizowanego za pomocą układów neuronowych, opisano w rozdziale 6. Zmianie mogą podlegać permutacje, oraz funkcje S-blok. Takie rozwiązanie daje szerokie możliwości modyfikacji algorytmów, o czym napisano w rozdziale 8. Warto zauważyć, że wśród publicznych komentarzy [75] do wymagań konkursu na AES wspominano o potrzebie możliwości tworzenia indywidualnych wersji algorytmu (modyfikacje w pewnym zakresie). Implementacja w układzie neuronowym z możliwością wprowadzania zmian w funkcji S-blok czy permutacji, jest możliwą alternatywą w stosunku do obecnie stosowanych sposobów implementacji szyfrów.

Uzyskane wyniki teoretyczne i rezultaty badań eksperymentalnych (numerycznych), przedstawione w tej pracy są efektem kilkuletnich poszukiwań autora polegających na identyfikacji możliwości, jakie tkwią w sieciach neuronowych w przypadku ich zastosowania jako adaptacyjnych układów szyfrujących. Na potrzeby tych badań wykonano wiele eksperymentów numerycznych, przebadanych zostało wiele modeli sieci oraz różne modele neuronów i funkcje aktywacji. Sposób wykorzystania sieci neuronowej w tej pracy odbiega od ogólnie przyjętego modelu. Sieci neuronowe traktowane są tu jako metoda konstrukcji układu adaptacyjnego, a nie jako narzędzie sztucznej inteligencji.

Dalsze perspektywy badawcze wynikające z już uzyskanych wyników to identyfikacja uniwersalnego zestawu modułów neuronowych przydatnego do implementacji większości współcześnie stosowanych szyfrów. W zakresie konstrukcji samych modułów należy jeszcze zastanowić się nad ich optymalizacją pod względem złożoności konstrukcji, mającej wpływ na czas uczenia i szybkość szyfrowania. Kolejnym obszarem badań, który będzie kontynuowany jest udoskonalenie konstrukcji protokołów kryptograficznych, przystosowanych do wykorzystania implementacji neuronowych algorytmów szyfrujących.

Literatura

[1] D. Kahn, “Łamacze kodów”, WNT Warszawa, 2004

[2] W. F. Friedmana, „The Index of Coincidence and its Applications in Cryptography”, 1918

[3] C. E. Shannon, “Communication Theory of Secrecy Systems”, Bell System Technical Journal, vol.28-4, 1949

[4] H. Feistel, “Cryptography and Computer Privacy", Scientific American, 228(5), May 1973

[5] http://nvl.nist.gov/

[6] NBS FIPS PUB 46, “Data Encryption Standard”, 1977

[7] A.Shimizu and S.Miyaguchi, “Fast Data Encirpherment FEAL” “Transactions of IEICE of Japan”, 1987

[8] X.Lai and J.Massey, “A proposal for a New Block Encryption Standard”, EUROCRYPT’90, 1990

[9] W. Diffie and M.E. Hellman, “New directions in cryptography”, IEEE Transactions on Information Theory, vol. IT-22, 644-654, 1976

[10] R. Rivest, A. Shamir, and L.M. “Adleman. A method for obtaining digital signatures and public key cryptosystems.”, Communications of the ACM, 1978, Vol.21, Nr.2, S.120-126

[11] D. Bernstein, “Circuits for integer factorization: a proposal.”, 2001

[12] W. Geiselmann and R. Steinwandt, "A Dedicated Sieving Hardware", 6th International Workshop on Practice and Theory in Public Key Cryptography, 2003

[13] B. Schneier, “Kryptografia dla praktyków”, WNT, Warszawa 2002

[14] S. Axelsson, “A Preliminary Attempt to Apply Detection and Estimation Theory to Intrusion Detection”, Technical report, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, 2000

[15] J.Cannady, J.Mahaffey, “The Application of Artificial Neural Networks to Misuse Detection: Initial Results”, Proceedings of the 1998 National Information Systems Security Conference (NISSC’98) October 5-8 1998. Arlington, VA., pp. 443–456

[16] A.Ghosh i inni, “Learning Program Behaviour Profiles for Intrusion Detection”, USENIX Technical Program - Paper - 1st Workshop on Intrusion Detection and Network Monitoring, Pp. 51–62 of the Proceedings, 1999

[17] I. Kanter, W.Kinzel, E.Kanter, “Secure exchange of information by synchronization of neural networks” Institut für Theoretische Physik, Universität Würzburg, Am Hubland, 97074 Würzburg, Germany

[18] A. Ruttor and W. Kinzel, “Dynamics of neural cryptography”, Institut für Theoretische Physik, Universität Würzburg, Am Hubland, 97074 Würzburg, Germany, 2007

[19] A.Klimov, A.Mityaguine, A.Shamir, “Analysis of Neural Cryptography”, Computer Science department, The Weizmann Institute, Israel, ASIACRYPT 2002

[20] Mitrokotsa, Aikaterini; Komninos, Nikos; Douligeris, “Intrusion Detection with Neural Networks and Watermarking Techniques for MANET”, IEEE International Conference on Volume , Issue , 15-20 July 2007 pp:118 – 127

[21] E.Mosanya, C.Teuscher, “A Reconfigurable and Modular Cryptographic Coprocessor”, Lecture Notes in Computer Science , str. 246 - 256, Berlin 1999

[22]S.Goldstein, “A High-Perfomance Flexible Architecture for Cryptography”, Proceedings of the Workshop on Cryptographic Hardware, August 1999

[23] C.H Lim, “CRYPTON: A New 128-bit Block Cipher”, Proceedings of the First Advanced Encryption Standard Candidate Conference, California, August 1998

[24]W. Laskowski, „Układy programowalne jako narzędzia wspomagające kryptograficzną ochronę danych”, Przegląd telekomunikacyjny, nr 3/2001, str. 178-183

[25]L. R. Knudsen, “Correlations in RC6”, Department of Informatics, University of Bergen, 1999

[26] C. E. Shannon, „A mathematical theory of communication” Bell System Technical Journal, vol. 27, July and October, 1948

[27] C. E. Shannon, “Predication and entropy of printed English”, The Bell System Technical Journal, 30. 50-64., 1951

[28] H. Feistel, “Cryptography and Computer Privacy", Scientific American, May 1973, pp 15–23, 1973

[29] L. R. Knudsen, “Practically secure Feistel ciphers”, 1994

[30] R. Rivest, A. Shamir, L.M. Adleman, “A method for obtaining digital signatures and public key cryptosystems”, Communications of the ACM, Vol. 21, str. 120-126, 1978

[31] A. Kerkhoff, „La Cryptographie Militare“, 1883

[32] J. Seberry, J. Pieprzyk, T. Hardjono, „Teoria bezpieczeństwa systemów komputerowych” Helion 2005

[33] Kam, J.B. Davida, G.I, “Structured Design of Substitution-Permutation Encryption Networks”, Transactions on Computers, 1979

[34] H. Feistel, W.A. Notz, J.L. Smith, “Some cryptographic techniques for machine-to-machine data communications”, Proceedings of the IEEE, Vol.63, pp. 1545- 1554, 1975

[35] A.F. Webster and S.E. Tavares, “On the design of S-boxes.Advances in Cryptology” CRYOTO'85, 1986

[36] O.S. Rothaus, “On bent functions”, Journal of Combinatorial Theory, Series A, str. 2537 - 2543, 1976

[37] E.Biham, A.Shamir, “Differential Cryptanalysis of DES-like Cryptosystems”, Journal of Cryptology, Vol. 4, str. 3-72, 1991

[38] W. S. Mc Culloch, W. Pitts, “A Logical Calculus of the Ideas Immanent in Nervous Activity”, Neurocomputing: foundations of research, str. 15 – 27, 1988

[39] S. Osowski, "Sieci neuronowe do przetwarzania informacji", Warszawa 2000

[40] W. Kacalak, M. Majewski, “Selected Problems of Automatic Recognition and Evaluation of Voice Commands in Natural Language given by the Operator using Artificial Neural Networks”, 9th World Multi-Conference on Systemics, Cybernetics and Informatics, USA July 2005

[41] J.B. Alam, C.K. Sarkar, E.U. Ahmed, “Study on thickness of two-way lab by artificial neural network”. Asian journal of civil engineering (building and housing), 2007

[42] W. Armstrong, J. Gecsei, "Adaptation Algorithms for Binary Tree Networks", IEEE Trans. on Systems, Man and Cybernetics, str. 276-285, 1979

[43] P. S. Szczepaniak, „Obliczenia inteligentne szybkie przekształcenia i klasyfikatory”, Akademicka Oficyna Wydawnicza EXIT, Warszawa 2004

[44] S. Peleg, A. Rosenfeld, “Breaking substitution ciphers using a relaxation algorithm.” Communications of the ACM 22, 598 - 605, 1979

[45] D. Hunter, A. McKenzie “Experiments with relaxation algorithms for breaking simple substitution ciphers.”, The Computer Journal, Vol. 26, pp. 68-71, 1983

[46] B. Delman, “Genetic Algorithms in Cryptography”, Department of Computer Engineering Thesis (M.S.)--Rochester Institute of Technology, 2004

[47] R.A.J. Matthews, “The use of genetic algorithms in cryptanalysis.” Cryptologia, Vol. 17, str. 187-201, 1993

[48] D. Dascalu, C. Vertan, C. Geangala, “Breaking the Merkle-Hellman Cryptosystem by Genetic Algorithms: Locality versus Performance, Real World Applications of Intelligent Technologies”, Romanian Academy, 1996 , pp. 201-208

[49] T. Bagnall, G.P. McKeown, V.J. Rayward-Smith, “The cryptanalysis of a three rotor machine using a genetic algorithm.” In: B¨ack T (ed) Proceedings of the 7th International Conference on Genetic Algorithms (ICGA97), San Francisco, CA, Morgan Kaufmann, 1997

[50] L.A. Adleman, “Subexponential algorithm for discrete logarithm problem with applications to cryptography”, 20th FOCS, 1979

[51] A.M. Odlyzko, “Discrete logarithms: the past and the future. Designs, Codes and Cryptography”, str. 129-145, 2000

[52] G.C. Meletiou, D.K. Tasoulis, M.N. Vrahatis, “Cryptography through interpolation approximation and computational inteligence methods”, Bull. Greek Math. Soc., pp. 61–75, 2004

[53] E.C. Laskari, G.C. Meletiou, D.K. Tasoulis, M.N. Vrahatis, “Studying the performance of artificial neural networks on problems related to cryptography.” Series B: Real World Applications 7(5) :937–942, 2006

[54] P.Kotlarz, M.Piechowiak, I.Dunajewski, "The Security of The RSA Cryptosystem and Electronic Signature", Image processing & communications, ISSN 1425-140X, str. 33-39, 2006

[55] P.Kotlarz, P.Żmudziński, I.Dunajewski, "The Attack on the RSA Algorithm with the Application of Euler's Function", Image processing & communications ", str. 25-31, ISSN 1425-140X, 2006

[56] V.Mueller, S.Paulus, “On the generation of cryptographically strong elliptic curves”, Preprint str. 1-11, 1998

[57] E.C.Laskari, G.C.Meletiou, Y.C.Stamatiou, M.N.Vrahatis, “Cryptography and Cryptanalysis Through Computational Intelligence”, Cryptography and Cryptanalysis Through Computational Intelligence, Springer, str. 1-49, 2007

[58] P. Kotlarz, Z. Kotulski, “On application of neural networks for s-boxes design.”, Lecture Notes in Artificial Intelligence, Springer-Verlag Berlin Heidelberg 2005, ISBN: 3-540-26219-9

[59] P. Kotlarz, Z. Kotulski, “Neural network as a programmable block cipher – experimental results”, Advances in Information Processing and Protection, Springer-Verlag ISBN: 978-0-387-73136-0, 2007

[60] P. Kotlarz, „Znaczenie właściwości funkcji boolowskich w kryptografii.”, Przegląd Telekomunikacyjny, ISSN 1230-3496 str 609-616, wrzesień 2007

[61] Ch. J. Mendelsohn Isis,"De Cifris" of Leone Battista Alberti, Vol. 32, No. 1, Jul., 1940

[62] D. J. Bernstein, “Cache-timing attacks on AES”, ENIGMA 2005

[63] P. Nazimek, G.Wojtenko, „Karty Java – technologia, bezpieczeństwo, implementacje”, konferencja Enigma 2004

[64] A. Klapper, M. Goresky, “2-adic Shift Register. Fast Software Encryption”, Second International Workshop. Lecture Notes in Computer Science, vol. 950, Springer Verlag, N. Y., pp.174-178, 1994

[65] A. Klapper, J. Xu, “Algebraic Feedback Shift Registers”, (submitted to Elsevier Preprint), 2003

[66] F. Arnault, T.P. Berger, and C. Lauradoux. “Update on F-FCSR stream cipher.”, ECRYPT Network of Excellence in Cryptology, Call for stream Cipher Primitives Phase 2 2006

[67] F. Arnault, Thierry P. Berger, C. Lauradoux, “X-FCSR - A New Software Oriented Stream Cipher Based Upon FCSRs”, INDOCRYPT 2007

[68] W. Laskowski, „Układy programowalne jako narzędzia wspomagające kryptograficzną ochronę danych”, KST 2003, Bydgoszcz

[69] N. Yu and H.M. Heys, "A Hybrid Approach to Concurrent Error Detection for a Compact ASIC Implementation of the Advanced Encryption Standard", Proceedings of IASTED International Conference on Circuits, Signal, and Systems, 2007

[70] V. Fischer, “Realization of the round 2 AES candidates using Altera FPGA”, April 2000

[71] W. Laskowski, „Układy programowalne jako narzędzia wspomagające kryptograficzną ochronę danych”, Przegląd Telekomunikacyjny, Rocznik LXXIV , nr 3/2001

[72] T. Łuba, K. Jasiński, B. Zwierzchowski, „Programowalne układy przetwarzania sygnałów i informacji – technika cyfrowa w multimediach i kryptografii”, Przegląd Telekomunikacyjny, Rocznik LXXVI nr 8–9/2003

[73] B. Schneier. „The GOST encryption algorithm”. Dr. Dobb’s Journal, pages 143 – 144, January 1995

[74] M. Kordos, „Algorytmy przeszukiwania w zastosowaniu do perceptrona wielowarstwowego”, rozprawa doktorska, Politechnika Śląska Wydział Automatyki, Elektroniki i Informatyki, Gliwice 2005

[75] K. Gaj „Od ENIGMY do ENIGMY1997-2006”, Materiały konferencyjne, Enigma 2006

Spis Rysunków

Spis tabel