INSTYTUT ŁĄCZNOŚCI
PAŃSTWOWY INSTYTUT BADAWCZY

Badania i analiza problemu bezpiecznej transmisji danych w paśmie krótkofalowym

mgr inż. Krzysztof Bronk

Rozprawa doktorska

Promotor:
dr hab. inż. Ryszard Katulski prof. nadzw. PG

Gdańsk, 2010

*Niniejszą pracę dedykuję mojej córce **Marcie** oraz żonie **Agnieszce.***

Pragnę podziękować wszystkim,
którzy przyczynili się do powstania tej rozprawy,
a przede wszystkim jej promotorowi
dr. hab. inż. Ryszardowi Katulskiemu
za inspirację oraz motywację
do pracy badawczej.

Spis treści

Wykaz skrótów

A	*Availability impact*
AC	*Access Complexity*
AES	*Advanced Encryption Standard*
AES-CTR	*AES CounTeR mode*
AR	*Availability Requirement*
ARQ	*Automatic Repeat reQuest*
ASCII	*American Standard Code for Information Interchange*
Au	*Authentication*
AV	*Access Vector*
BBS	*Blum-Blum-Shub*
C	*Confidentiality impact*
CB	*Centrum Bezpieczeństwa*
CBC	*Cipher Block Chaining*
CDP	*Collateral Damage Potential*
CFB	*Cipher FeedBack*
CMAC	*Cipher-based Message Authentication Code*
CPU	*Central Processing Unit*
CR	*Confidentiality Requirement*
CVSS	*Common Vulnerability Scoring System*
DES	*Data Encryption Standard*
DSP	*Digital Signal Processor*
DVB	*Digital Video Broadcasting*
E	*Exploitability*
ECB	*Electronic Code Book*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
EOM	*End Of Message*
FPGA	*Field Programmable Gate Array*
GSM	*Global System for Mobile communications*
HF	*High Frequency*
I	*Integrity impact*
IP	*Internet Protocol*
IR	*Integrity Requirement*
IT	*Information Technology*
LSB	*Least Significant Bit*
MAC	*Message Authentication Code*
MK0	*Master Key 0*
MSB	*Most Significant Bit*
ND	*Not Defined*
NIST	*National Institute of Standards and Technology*
OFB	*Output FeedBack*
OTAR	*Over The Air Re-keying*
PC	*Personal Computer*
PIN	*Personal Identification Number*

PSK	*Phase Shift Keying*
QAM	*Quadrature Amplitude Modulation*
QPSK	*Quadrature Phase Shift Keying*
RAM	*Random Access Memory*
RC	*Report Confidence*
RL	*Remediation Level*
ROM	*Read-Only Memory*
RSA	*Rivest-Shamir-Aldeman*
SHA	*Secure Hash Algorithm*
SMM	*Security Management Message*
TCP	*Transmission Control Protocol*
TD	*Target Distribution*
UDM	*User Data Message*
UMTS	*Universal Mobile Telecommunications System*
UTC	*Universal Time Clock*
WLAN	*Wireless Local Area Network*
XOR	*eXclusive OR*

Wstęp

W dobie społeczeństwa informacyjnego problem bezpiecznej transmisji danych staje się zagadnieniem o coraz większym znaczeniu. Jest on często rozważany w literaturze naukowej oraz ma swe odzwierciedlenie w istniejących jak i dopiero powstających systemach łączności.

Bezpieczna transmisja danych rozumiana jest tu w szerokim zakresie i nie sprowadza się jej definicji jedynie do zapewniania poufności przesyłanych informacji, choć jest to w sposób oczywisty element istotny. Nie mniej jednak ważnymi składnikami są zarówno mechanizmy uwierzytelniania, kontroli integralności przesyłanych wiadomości oraz ograniczania dostępu do urządzeń i tajnych parametrów systemowych.

W systemach łączności bezpieczna transmisja danych zapewniana jest z wykorzystaniem kryptosystemów, które poza realizacją powyższych mechanizmów, umożliwiają również zarządzanie parametrami użytkowników oraz udostępniają odpowiednie protokoły komunikacyjne jak i algorytmy generowania, dystrybucji i aktualizacji kluczy szyfrujących.

Problem zapewniania bezpiecznej transmisji danych występuje również w sieciach łączności bezprzewodowej, a zwłaszcza tych o charakterze globalnym. Medium bezprzewodowe jest bowiem dużo bardziej podatne na podsłuch niż rozwiązania przewodowe, przez co jest ono bardziej wrażliwe na niepożądane działania osób do tego nieupoważnionych, których celem jest nieautoryzowane zdobycie informacji, a często również wprowadzenie szkodliwych danych do sieci. W dobie terroryzmu, gdy bardzo często poufne przesyłanie wiadomości ma znaczenie krytyczne, zagadnienie to zasługuje na tym większą uwagę.

W sytuacji globalnego zagrożenia, szczególnie ważne pod względem użytecznym są satelitarne systemy bezprzewodowe z natury rzeczy zapewniające łączność globalną. Niestety, z oczywistych względów, systemy te są mało odporne na działania destrukcyjne. W takiej sytuacji należy szukać innego sposobu bezprzewodowej, globalnej transmisji danych. W ostatnim czasie rozwiązania tego problemu upatruje się w wykorzystaniu łączności krótkofalowej (HF), która między innymi dzięki odbiciom fal radiowych od warstw jonosfery umożliwia transmisję dalekosiężną. Obecnie systemy radiowe działające na falach krótkich zyskują

ponownie na znaczeniu, zwłaszcza do zastosowań specjalnych i w niedalekiej przyszłości mogą stać się bezpiecznym rozwiązaniem, służącym między innymi do przesyłania ważnych, a często także tajnych informacji, w sytuacjach zagrożenia bezpieczeństwa i nie tylko.

Istotnym problemem naukowym, który musi być pilnie rozwiązany, staje się zatem kwestia zapewnienia bezpiecznej transmisji informacji w paśmie krótkofalowym. Większość systemów łączności bezprzewodowej, działających współcześnie, posiada pewne mechanizmy gwarantujące zarówno poufność przesyłanych danych, jak również ich integralność i autentyczność. Nie ma jednak możliwości bezpośredniego ich wykorzystania w łączu krótkofalowym, ponieważ rozwiązania te przeznaczone są do współpracy z konkretnymi warstwami fizycznymi oraz protokołami komunikacyjnymi, stosowanymi w tych systemach. W przypadku łącza krótkofalowego brak jest jednak tego typu rozwiązań, co na dzień dzisiejszy utrudnia zastosowanie modemów HF w sposób uniemożliwiający podsłuch i ingerencję w przesyłaną informację.

Ponadto bezprzewodowe sieci lokalne czy komórkowe dostosowane są do częstotliwości z zakresu UHF a przez to do stosunkowo szerokich kanałów radiowych i dużych przepływności. Rozwiązania te projektowano i implementowano z myślą o zapewnieniu możliwości szybkiej, bezpiecznej oraz w dużym stopniu bezbłędnej i gwarantowanej transmisji danych. Łącze krótkofalowe jednak nie pozwala na uzyskanie podobnych parametrów jakościowych i z tego powodu wymaga nowego kryptosystemu, umożliwiającego efektywne wykorzystanie niewielkich zasobów oferowanych przez kanał radiokomunikacyjny w paśmie HF oraz warstwę fizyczną dostępnych modemów wykorzystywanych w tym zakresie częstotliwości.

Celem niniejszej rozprawy doktorskiej jest zatem dobór i modyfikacja, dla potrzeb bezpiecznej transmisji danych w łączu krótkofalowym, efektywnych algorytmów, zapewniających poufność, integralność i autentyczność danych, jak również mechanizmów zarządzania kluczami szyfrującymi, parametrami użytkowników oraz całym zaproponowanym w ten sposób kryptosystemem. Działania te będą podejmowane pod kątem oceny poziomu bezpieczeństwa oraz przydatności dla potrzeb transmisji w łączu HF, a dotychczasowe doświadczenia zdobyte dzięki

analizie tego typu rozwiązań w istniejących systemach telekomunikacyjnych pozwolą na ich zaadoptowanie dla potrzeb łączności krótkofalowej.

Teza pracy jest następująca: podczas transmisji danych w paśmie HF można zapewnić bezpieczeństwo tej transmisji poprzez zastosowanie mechanizmów poufności i kontroli integralności informacji oraz uwierzytelniania i ograniczania dostępu do urządzeń i sieci. Ponadto odpowiednie zarządzanie bezpieczeństwem umożliwi integrację wszystkich tych funkcji i stanie się środkiem do opracowania całościowej koncepcji kryptosystemu dla pasma krótkofalowego.

Zaprojektowany w ten sposób system bezpieczeństwa poddany zostanie testom, mającym na celu określenie między innymi możliwości nieautoryzowanego do niego dostępu, naruszenia integralności przesyłanych informacji, utraty poufności transmisji jak i ryzyka związanego z dezaktualizacją oraz przechwyceniem kluczy szyfrujących. Badania te staną się podstawą do oceny poziomu bezpieczeństwa oferowanego przez zaprojektowany system.

Przeprowadzone zostaną również pewne dodatkowe testy i analizy. Określony zostanie między innymi wpływ zaproponowanych mechanizmów bezpieczeństwa na ograniczenie szerokości pasma użytecznego. Przeanalizowane zostaną także: metoda dystrybucji kluczy szyfrujących, schematy zarządzania kryptosystemem oraz mechanizmy zaradcze, wykorzystywane w przypadku ryzyka nieautoryzowanego dostępu do systemu. Powyższe elementy są krytyczne dla bezpiecznej łączności krótkofalowej i dlatego w tym przypadku pozwolą na określenie stopnia przydatności zaproponowanego rozwiązania.

W niniejszej rozprawie doktorskiej zawarto (rozdział pierwszy) między innymi podstawowe informacje związane z ogólnie rozumianą kryptografią i takie, które pozwolą na późniejszą realizację koncepcji nowego kryptosystemu, umożliwiającego bezpieczną transmisję danych w paśmie HF oraz jego implementację. Przedstawiono więc podstawy teoretyczne wykorzystywanych algorytmów kryptograficznych, związanych zarówno z szyfrowaniem jak i uwierzytelnianiem nadawcy oraz zapewnianiem integralności przesyłanych wiadomości.

W pracy tej scharakteryzowano również (rozdział drugi) warstwę fizyczną klasycznych modemów krótkofalowych, które mogą wykorzystywać zaproponowany kryptosystem. Zaprezentowano również sposób jego realizacji w tego typu

urządzeniach przy założeniu łatwości jego implementacji oraz odpowiednich wymogów zawartych w najpopularniejszym standardzie modemów HF.

Jedną z kluczowych części niniejszej rozprawy jest rozdział trzeci, który zawiera ogólną specyfikację koncepcji zaproponowanego kryptosystemu. Ujęto w niej prezentację założeń odnośnie samego systemu oraz przedstawiono jego strukturę. Scharakteryzowano ponadto szczegółowo dwie podstawowe jednostki funkcjonalne kryptosystemu (centrum bezpieczeństwa oraz modułu kryptograficznego) oraz zaproponowano protokół transmisyjny umożliwiający komunikację między tymi elementami. Zaprezentowano typy i struktury przesyłanych wiadomości oraz wymagania odnośnie ich kodowania. Zaproponowano również zasady zarządzania systemem bezpieczeństwa, wprowadzając hierarchizację kluczy oraz identyfikatorów jak również określono zasady ich dystrybucji, aktualizacji oraz wykorzystania w systemie.

W rozdziale trzecim zaproponowano również autorskie modyfikacje powszechnie stosowanych algorytmów kryptograficznych, które zostały opracowane dla potrzeb nowego kryptosystemu. Dzięki takiemu podejściu realizowne będą wszystkie mechanizmy kryptograficzne i umożliwiona będzie współpraca z istniejącymi modemami krótkofalowymi, co pozwali na w pełni bezpieczną transmisję danych w łączu HF.

Na podstawie zaproponowanej i przedstawionej w rozdziale trzecim koncepcji, zrealizowany został symulator kryptosystemu. Uwzględnia on między innymi trzy główne mechanizmy kryptograficzne (zapewnianie poufności, uwierzytelnianie i kontrola integralność), ale również zaproponowany protokół komunikacyjny jak i schematy zarządzania systemem. W pracy przedstawiono (rozdział czwarty) sposób implementacji poszczególnych elementów symulatora, który umożliwia wykorzystanie wirtualnych modułów kryptograficznych oraz centrów bezpieczeństwa poszczególnych realizacji zaproponowanego kryptosystemu. Zaprojektowana aplikacja opiera się na wielu rozwiązaniach autorskich i pozwala na właściwie dowolną konfigurację elementów kryptosystemu oraz wykorzystanie wszystkich zdefiniowanych w nowym kryptosystemie schematów transmisji i zarządzania. Dodatkowo możliwe jest symulowanie prób ingerencji i nieautoryzowanego dostępu, ale również realizowanie mechanizmów zaradczych adekwatnych do zaistniałych sytuacji niepożądanych.

W rozdziale czwartym przeanalizowano ponadto szereg zdarzeń, mogących wystąpić w praktycznej implementacji zaproponowanego rozwiązania. Analizie poddano kryptosystem jako całość poprzez symulację rzeczywistych sytuacji, mogących mieć miejsce w trakcie działania takiego systemu. W szczególności przedstawiono tu efektywny sposób aktualizacji kluczy sesji z jednoczesną dezaktywacją niektórych modułów kryptograficznych przy użyciu pojedynczej tylko wiadomości zarządzającej typu punkt-wielopunkt.

W rozdziale czwartym zbadano także odporność systemu na różne próby ingerencji w przesyłane za jego pośrednictwem wiadomości oraz możliwość nieautoryzowanego do niego dostępu z użyciem ataku powtarzania i filtrowania wiadomości, a także z wykorzystaniem skradzionego modułu kryptograficznego. Przeanalizowano również czysto hipotetyczne sytuacje wycieku kluczy i innych parametrów systemowych na ryzyko naruszenia bezpieczeństwa transmisji.

Ostatecznie z wykorzystaniem standaryzowanego narzędzia oceny dokonano ewaluacji oferowanego poziomu bezpieczeństwa zaproponowanego rozwiązania (rozdział piąty). Wzięto tu pod uwagę zarówno uwarunkowania czasowe, mogące wpłynąć na zmiany sytuacji w systemie oraz środowiskowe, które uwzględniają użytkownika systemu i pozwalają określić z jego perspektywy praktyczne znaczenie pewnych potencjalnych słabości kryptosystemu.

W podsumowaniu niniejszej rozprawy doktorskiej przedstawiono wnioski i spostrzeżenia, wynikające z przeprowadzonych badań i analiz.

Rozdział I

Podstawy teoretyczne wybranych aspektów związanych z kryptografią

Niniejszy rozdział zawiera prezentację podstawowych pojęć i algorytmów związanych z ogólnie rozumianą kryptografią. Dobór jednak elementów tu przedstawionych nie jest przypadkowy i ma ścisły związek z zaproponowaną koncepcją systemu bezpiecznej transmisji danych w łączu krótkofalowym, którą przedstawiono w rozdziale trzecim.

1.1. Definicje najważniejszych pojęć

Jak wiadomo zagadnienia związane z ochroną informacji należą do dziedziny wiedzy zwanej kryptologią. Jest to nauka o bezpiecznych sposobach przekazywania informacji. Bezpiecznych, to znaczy uniemożliwiających ich odczytanie, zmodyfikowanie lub podszycie się pod ich nadawcę przez osoby niepowołane, które nie posiadają odpowiedniego poziomu autoryzacji i nie znają wymaganych algorytmów i kluczy. Kryptologia z kolei dzieli się na kryptografię i kryptoanalizę – dwie przeciwstawne dziedziny wiedzy. Kryptografia zajmuje się zabezpieczaniem informacji a kryptoanaliza próbami obejścia tych zabezpieczeń lub ich złamania. Kryptologia zawdzięcza swoje powstanie i rozwój głównie dzięki rywalizacji tych dwóch przeciwstawnych dziedzin.

Bezpieczeństwo systemów kryptograficznych (kryptosystemów) w dużej mierze opiera się na tzw. mechanizmach kryptografii, zwanych również funkcjami bezpieczeństwa. Zanim jednak zostaną one przedstawione, należy wyjaśnić kilka podstawowych pojęć [3, 63, 89, 90]:

- Algorytm kryptograficzny (szyfr) – funkcja matematyczna, stosowana do realizacji szyfrowania i deszyfrowania i najczęściej różna dla każdego z tych dwóch procesów;
- Wiadomość jawna (tekst jawny) – niezaszyfrowane dane, które mogą być odczytane przez każdego;

- Wiadomość tajna (szyfrogram, tekst zaszyfrowany) – dane zaszyfrowane, których odczytanie jest niemożliwe bez znajomości algorytmu szyfrującego i klucza;
- Szyfrowanie – proces zamiany wiadomosci jawnej na tajną przy użyciu odpowiedniego algorytmu kryptograficznego oraz klucza;
- Deszyfrowanie – proces odwrotny do szyfrowania, polegający na zamianie wiadomości tajnej na jawną przy użyciu odpowiedniego algorytmu kryptograficznego oraz klucza;
- Klucz – najczęściej tajna sekwencja stosowana w algorytmie kryptograficznym do szyfrowania i deszyfrowania; znana zwykle tylko nadawcy i uprawnionemu odbiorcy danej zaszyfrowanej wiadomości. Wyróżniamy klucze tajne, prywatne i publiczne;
- Algorytm z kluczem tajnym (szyfrowanie symetryczne z kluczem tajnym) – rozwiązanie, w którym zarówno dla szyfrowania jak i deszyfrowania wiadomości konieczna jest znajomość tego samego tajnego klucza;
- Algorytm z kluczem publicznym (szyfrowanie asymetryczne z kluczem publicznym) – rozwiązanie, w którym do szyfrowania używany jest klucz publiczny (jawny), a do deszyfrowania inny i tajny klucz prywatny;
- Szyfr strumieniowy (potokowy) – symetryczny algorytm kryptograficzny, w którym szyfrowaniu podlega strumień znaków tekstu jawnego element po elemencie (bit po bicie lub znak po znaku);
- Szyfr blokowy – algorytm kryptograficzny, w którym szyfrowaniu podlegają jednocześnie całe bloki danych a nie pojedyncze znaki czy bity;
- Kryptosystem (system kryptograficzny)[1] – zespół mechanizmów i algorytmów, w ramach których realizuje się szyfrowanie i deszyfrowanie informacji, wraz z zapewnieniem wszystkich funkcji bezpieczeństwa (patrz paragraf 1.1.1) oraz rozwiązań zarządzania kryptosystemem jak i parametrami jego użytkowników. Zdefiniowaniu podlegają tu również między innymi wykorzystywane

[1] W znaczeniu ściśle matematycznym (wąskim) kryptosystem może być rozumiany również jako zbiór reguł szyfrowania i deszyfrowania dla skończonego zbioru kluczy, możliwych tekstów jawnych oraz tajnych [89]. W niniejszej rozprawie doktorskiej jednak autor wykorzystuje definicję w ujęciu szerszym [89].

urządzenia, protokoły komunikacyjne czy algorytmy generowania i dystrybucji kluczy;

- Funkcja skrótu (funkcja haszująca) – funkcja jednokierunkowa, dostarczająca jednoznacznego i nieodwracalnego skrótu wiadomości jawnej. Sekwencja taka jest zwykle używana do kontroli integralności wiadomości. Charakteryzuje się tym, że na jej podstawie nie można wyznaczyć oryginału wiadomości, z którego został sporządzony skrót, ale dana wiadomość może zostać jednoznacznie powiązana z konkretnym skrótem (danej wiadomości odpowiada zawsze ten sam skrót). Bezpieczna funkcja haszująca uniemożliwia zatem (przynajmniej teoretycznie) wygenerowanie dwóch różnych wiadomości o identycznym skrócie;
- Podpis cyfrowy – jawna sekwencja pozwalająca na uwierzytelnienie nadawcy wiadomości oraz kontrolę jej integralności. Podpis cyfrowy generuje się najczęściej dla skrótu danej wiadomości z wykorzystaniem asymetrycznego algorytmu kryptograficznego oraz klucza prywatnego nadawcy. Weryfikacja takiej sekwencji, dokonywana przez odbiorcę, odbywa się w takiej sytuacji poprzez zastosowanie klucza publicznego.

1.1.1. Mechanizmy kryptograficzne

Aby system ochrony przesyłanych informacji był w pełni bezpieczny należy zapewnić w nim główne mechanizmy kryptografii (funkcje bezpieczeństwa) [60, 89, 90]:

- uwierzytelnianie wraz z autoryzacją,
- poufność,
- integralność,
- dostępność.

1.1.1.1. Uwierzytelnianie

Informacja jest uwierzytelniona, jeżeli pochodzi z wiarygodnego źródła. Prostym przykładem może tu być rozmowa telefoniczna, w której obie strony naturalnie identyfikują się wzajemnie za pomocą barwy i tonu głosu. W sytuacji, gdy osoby się nie znają lub jakość dźwięku przesyłanego przez kanał jest zła i utrudnia identyfikację, jak również w przypadku przesyłania danych należy, zastosować specjalne mechanizmy umożliwiające uwierzytelnianie [30].

Jednym ze sposobów uwiarygodniania wiadomości jest zastosowanie mechanizmu MAC (*Message Authentication Code*) [27, 56, 89, 90]. W metodzie tej do tekstu jawnego dołącza się niewielką ilość danych zaszyfrowanych za pomocą algorytmu MAC i klucza tajnego. Po stronie odbiorczej możliwa jest następnie weryfikacja sekwencji MAC przez adresata, posiadającego ten sam klucz tajny co nadawca (przykład algorytmu symetrycznego). Odbiorca zna ponadto algorytm uwierzytelniania, będący konkretnym typem algorytmu MAC. Dzięki zgodności ciągów uwierzytelniających po obu stronach łącza, możliwe jest zatem uwiarygodnienie nadawcy. Rozwiązanie to ma jednak pewne wady, ponieważ każdy nadawca i odbiorca musi przechowywać w swoim urządzeniu nadawczo-odbiorczym klucze wszystkich osób, z którymi chciałby się komunikować lub wymieniać dane. Prowadzi to często do nadużyć, ponieważ wszyscy posiadają klucze wszystkich, co może być niewygodnym i mało efektywnym rozwiązaniem przy dużej liczbie użytkowników, a dodatkowo istnieje tu ryzyko podszywania się przy użyciu posiadanych kluczy innych osób. Rozwiązanie to sprawdzi się zatem tylko w niewielkiej i zaufanej sieci. Warto tu dodatkowo zauważyć, że jeden klucz przydzielić można pewnej grupie osób. Wtedy sekwencja MAC identyfikuje nie jednostkę, a całą grupę.

Alternatywą dla powyższego jest użycie podpisów cyfrowych. Wykorzystuje się tu algorytm z kluczem publicznym (przykład szyfrowania asymetrycznego) oraz funkcję skrótu (haszującą) [91]. Podpis cyfrowy, wygenerowany przy użyciu klucza prywatnego nadawcy, jest dołączany do oryginalnej wiadomości. Dzięki niemu i kluczowi publicznemu odbiorca wiadomości jest w stanie zweryfikować nadawcę. Nie posiada jednak jego klucza prywatnego (a jedynie klucz publiczny) i dzięki temu nie ma możliwości podszycia się pod niego. Za pomocą klucza publicznego może jedynie zweryfikować podpis nadawcy.

W ogólności klucz publiczny wykorzystać można również do szyfrowania przesyłanych danych. W takim przypadku zapewniona zostaje poufność transmisji (patrz paragraf 1.1.1.2) z gwarancją możliwości odtworzenia wiadomości jawnej jedynie przez posiadacza klucza prywatnego.

Innym sposobem umożliwienia wiarygodnej transmisji jest uwierzytelnianie czasowe. Wymaga się wtedy jednak, aby wszystkie komunikujące się ze sobą

urządzenia miały jednakowe czasy systemowe, jednak najczęściej dopuszczalny jest tu pewien margines błędu [78].

1.1.1.2. Poufność

Informacja jest poufna[2], jeżeli bez znajomości klucza i algorytmu szyfrującego teoretycznie niemożliwe jest jej odczytanie (odszyfrowanie). Zwykle mechanizm ten realizuje się za pomocą algorytmu szyfrującego z użyciem klucza tajnego. Do zapewnienia poufności można wykorzystać jednak zarówno szyfrowanie symetryczne jak i asymetryczne. Zaletą tego pierwszego jest jednak to, iż z reguły jest ono wydajniejsze [41, 67] pod względem szybkości jego realizacji. Z tego powodu duże ilości danych powinno się szyfrować algorytmami z kluczem tajnym.

1.1.1.3. Integralność

Integralność wiadomości oznacza pewność, że nie została ona zmieniona od momentu jej wysłania przez wiarygodne źródło do chwili dotarcia do odbiorcy. Integralność może być kontrolowana przez zastosowanie podpisu cyfrowego. Zmiana jakiegokolwiek elementu wiadomości spowoduje bowiem, że wygenerowany po stronie odbiorczej *hash* wiadomości będzie różnił się od tego zaszyfrowanego w sekwencji podpisu i dołączonego do wiadomości. Po stronie odbiorczej możliwe jest zatem przeprowadzenie weryfikacji integralności z użyciem klucza publicznego. Analiza taka daje wynik negatywny, gdy odebrana wiadomość została w jakikolwiek sposób zmodyfikowana w kanale telekomunikacyjnym, w którym była transmitowana.

Warto tu także dodać, że realizacja mechanizmu kontroli integralności może być również przeprowadzona z wykorzystaniem sekwencji uwierzytelniających MAC (patrz paragraf 1.1.1.1).

1.1.1.4. Dostępność

Dostępność, a właściwie jej ograniczanie i kontrolowanie, polega na fizycznej ochronie tajnych danych, urządzeń je przechowujących oraz umożliwiających ich szyfrowanie i deszyfrowanie przed osobami do tego niepowołanymi. Najprostszym

[2] Informacja poufna – rozumiana jest tu zgodnie z ogólnie przyjętą nomenklaturą kryptograficzną. Należy jednak pamiętać, że w świetle ustawy [93] należałoby używać określenia „informacja niejawna", które jest pojęciem bardziej ogólnym. Autor jednak w niniejszej rozprawie doktorskiej stosować będzie określenie „informacja poufna", ponieważ jest to najczęściej spotykane w literaturze tłumaczenie angielskiego wyrażenia *confidential information*.

przykładem kontroli dostępności jest wykorzystanie tajnego hasła podczas logowania się do konta e-mail – tylko znający tą tajna sekwencję ma możliwość uzyskania dostępu do poczty.

W systemach bezpieczeństwa dostępność do usługi, urządzenia, czy danych jest zwykle realizowana za pomocą osobistego hasła czy numeru PIN. Inną metodą ograniczania dostępu może być jednak wykorzystanie właściwości biometrycznych osoby – cechy niepowtarzalne, takie jak: odcisk palca, skan tęczówki oka, czy identyfikacja za pomocą głosu lub nawet DNA.

Warto w tym miejscu dodać, że bardzo często mechanizm dostępności jest krytyczny dla systemów bezpieczeństwa, ponieważ jest najbardziej wrażliwy i narażony na błędy lub celowe i niepożądane działania człowieka.

1.1.1.5. Autoryzacja

Autoryzacja jest procesem, w trakcie którego ustala się, czy dana osoba ma wystarczające uprawnienia do korzystania z danych lub usług, których żąda. Proces ten jest często mylony z kontrolą wiarygodności, a w rzeczywistości jest on elementem składowym mechanizmu uwierzytelniania. Należy mieć bowiem na uwadze, iż samo potwierdzenie tożsamości użytkownika nie oznacza automatycznie, że ma on prawo do otrzymania danych, do których próbuje uzyskać dostęp. Mechanizm uwierzytelniania sprawdza bowiem, kim jest osoba próbująca uzyskać dostęp do systemu, a proces autoryzacji kontroluje, czy ma ona prawo korzystać z określonego zasobu. W praktyce stosuje się najczęściej różne poziomy autoryzacji, wykorzystywane do hierarchizacji systemów bezpieczeństwa. Jako przykład można tu podać hierarchię kont użytkowników w komputerach PC – podział na administratora i użytkownika z ograniczonymi uprawnieniami. Administrator ma możliwość zakładania nowych kont użytkowników i zmieniania poziomu ich uprawnień (poziomu autoryzacji), podczas gdy użytkownik z ograniczeniami może np. korzystać tylko z programów zainstalowanych na komputerze.

W dalszej części niniejszego rozdziału przedstawione zostaną przykładowe algorytmy kryptograficzne i funkcje skrótu ze szczególnym uwzględnieniem jednak tych, które wykorzystywane będą w zaproponowanym systemie bezpiecznej transmisji danych.

1.2. Algorytmy kryptograficzne i funkcje skrótu

Do najprostszych algorytmów kryptograficznych należą szyfry podstawieniowe. W takim rozwiązaniu każda litera lub grupa liter tekstu jawnego jest zastąpiona inną literą lub grupą liter. W klasycznej kryptografii wyróżnia się cztery grupy szyfrów podstawieniowych [44, 89, 90]:

- monoalfabetowe (jednoalfabetowe);
- homofoniczne;
- wieloalfabetowe;
- poligramowe.

Szyfry monoalfabetowe zawierają takie same znaki zarówno w tekście jawnym, jak i w szyfrogramie. Istnieje stałe przypisanie dla każdej litery tekstu jawnego konkretnego odpowiednika tajnego.

W szyfrach homofonicznych z kolei każdemu znakowi tekstu jawnego przyporządkowuje się po kilka znaków (homofonów) kryptogramu. W procesie szyfrowania dla każdej jawnej litery wybiera się losowy i przypisany do niej homofon, stający się od tej chwili znakiem tekstu tajnego. Takie podejście pozwala na częściowe ograniczenie możliwości wykorzystania ataków statystycznych opartych o analizę częstotliwości występowania znaków w tekście zaszyfrowanym

Szyfry wieloalfabetowe stanowią kombinacje prostych szyfrów monoalfabetowych. Szyfrowanie danego znaku odbywa się z zastosowaniem jednego z kilu używanych w danym przypadku szyfrów jednoalfabetowych. Wybór konkretnego z nich uzależniony jest od aktualnie wykorzystywanego klucza.

W szyfrach poligramowych szyfruje się jednocześnie pewne grupy znaków. Algorytmy te nie zostaną tu jednak szczegółowo opisane, ze względu na charakter niniejszego rozdziału, który ma na celu prezentację rozwiązań, które zostaną wykorzystane w projektowanym kryptosystemie.

Kolejną grupą algorytmów kryptograficznych są szyfry przestawieniowe (permutacyjne). Zmieniają one kolejność znaków w tekście zaszyfrowanym. Szczególnie jednak istotne z perspektywy niniejszej pracy są algorytmy kaskadowe, w których zestawiono podstawowe metody szyfrowania, jakimi są pojedyncze podstawienia i przestawienia. Szyfry kaskadowe są zatem połączeniem szyfrów podstawieniowych i przestawieniowych, dzięki czemu mają one lepsze właściwości kryptograficzne niż ich poszczególne elementy składowe osobno [89]. Szyfry te były

realizowane w maszynach rotorowych (np. w Enigmie [15, 77]) i są implementowane w postaci nowoczesnych algorytmów komputerowych [29, 44, 49].

W kolejnym paragrafie niniejszego rozdziału przedstawione zostaną zagadnienia związane ze sposobami wykorzystywania blokowych algorytmów kryptograficznych.

1.2.1. Tryby pracy blokowych algorytmów kryptograficznych

Istnieją dwa typy algorytmów kryptograficznych: szyfry blokowe i szyfry strumieniowe [55, 81]. Te pierwsze realizują operacje na blokach tekstu jawnego lub szyfrogramu. Bloki te mają zwykle 64 lub więcej bitów długości (zwykle wielokrotność tej wartości). Pojedynczy blok tekstu jawnego jest zamieniany w blok szyfrogramu o tej samej długości. Takie rozwiązanie posiada jednak zasadniczą wadę – dla danego klucza i wielu identycznych bloków tekstu jawnego uzyskuje się zawsze te same postacie szyfrogramów.

Szyfry strumieniowe natomiast wykorzystują sumowanie modulo 2 kolejnych bitów strumienia, wygenerowanego w oparciu o konkretny algorytm i sekwencję klucza, z kolejnymi bitami danych. Pojedynczy bit tekstu jawnego jest zatem zamieniany w odpowiedni bit szyfrogramu. Dzięki takiemu rozwiązaniu szyfry strumieniowe cechują się właściwością ukrywania struktury wiadomości i nadawania jej charakteru pseudolosowego[3]. W praktyce, ze względu na duże bezpieczeństwo [44, 81], wykorzystuje się jednak najczęściej blokowe algorytmy kryptograficzne, które w swej podstawowej formie nie posiadają wspomnianych tu zalet. Z tego powodu, w celu nadania im często cech zbliżonych do właściwości szyfrów strumieniowych [73], stosuje się różne tryby ich pracy.

W dalszej części niniejszego paragrafu przedstawione zostaną najważniejsze tryby pracy szyfrów blokowych [55] ze szczególnym uwzględnieniem cech charakterystycznych tych rozwiązań.

Jednym z trybów pracy blokowych algorytmów kryptograficznych jest metoda elektronicznej książki kodowej (ECB – *Electronic Code Book*). Jest to rozwiązanie najbardziej podstawowe i nie pozwala ono na uzyskanie zalet szyfrów strumieniowych. Każdy blok tekstu jawnego zamieniany jest bowiem w odpowiadający mu blok szyfrogramu. Możliwe jest zatem wygenerowanie tzw. książki

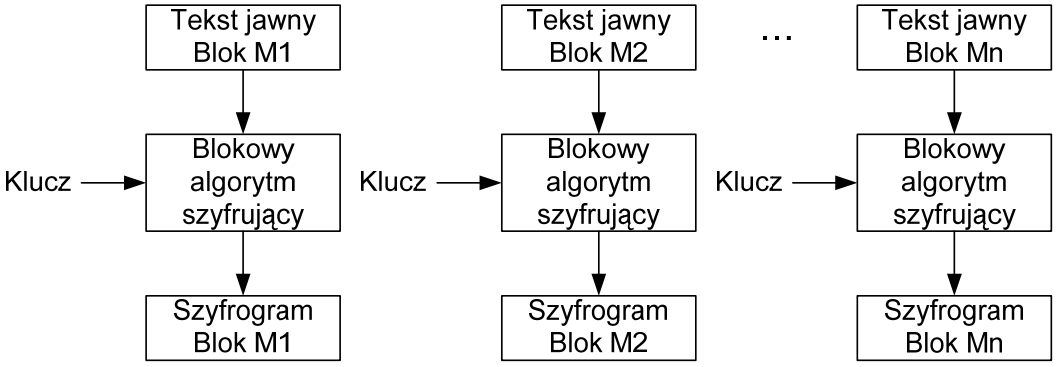
[3] Dzięki wykorzystaniu pseudolosowej i długiej sekwencji strumienia sumowanego modulo 2 z kolejnymi bitami danych możliwe staje się uzyskanie losowego charakteru szyfrogramu, a ponadto identyczne elementy tekstu jawnego uzyskają za każdym razem inne postacie zaszyfrowane.

kodowej, czyli tablicy ze wszystkimi przekształceniami bloku tekstu jawnego w blok szyfrogramu. Dla każdego klucza tablica kodowa ma inną postać. Przykładowo, jeśli długość bloku ma 64 bity, to książka kodowa ma 2^{64} możliwych par sekwencji wejściowych i wyjściowych dla danego klucza. Zaletą takiego rozwiązania jest jego zdolność do niezależnego szyfrowania każdego bloku tekstu jawnego. Można na przykład zaszyfrować tylko część bloków, bez konieczności szyfrowania pozostałych. Jest to bardzo istotna zaleta w przypadku danych, gdzie wymagany jest dostęp losowy do dowolnego ich fragmentu – na przykład w bazach danych każdy rekord może być w takiej sytuacji szyfrowany niezależnie. Podczas modyfikowania, wstawiania i kasowania rekordu nie trzeba deszyfrować poprzednich elementów. Rozwiązanie to ma jednak sporą wadę, ponieważ identyczne bloki tekstu jawnego posiadać będą zawsze identyczną postać szyfrogramu dla danego klucza, co ułatwiać może nieautoryzowaną deszyfrację informacji z wykorzystaniem ataków statystycznych [89, 90].

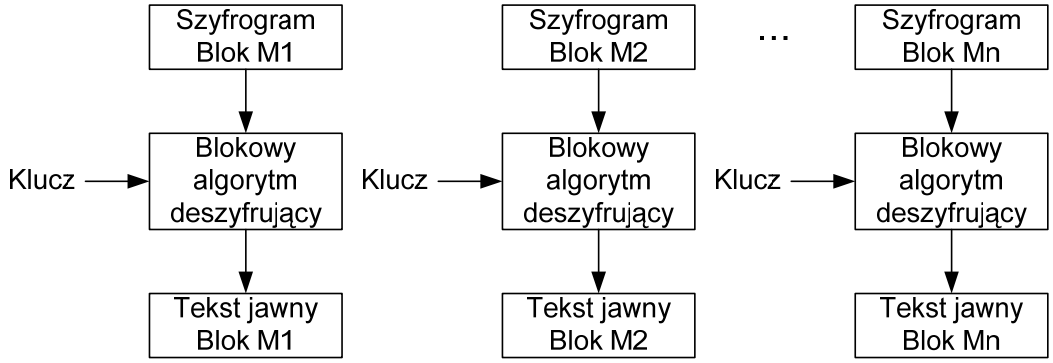
Tryb ECB jest jednak wydajny, ponieważ szybkość szyfrowania jest w tym przypadku taka jak samego szyfru blokowego, a szyfrogram jest dłuższy od tekstu jawnego co najwyżej o długość dopełnienia ostatniego bloku i dodatkowo przetwarzanie kolejnych bloków może być zrównoleglone. Rozwiązanie to nie powoduje ponadto propagacji błędów, ale nawet niepoprawny pojedynczy bit szyfrogramu wpływa na cały blok tekstu jawnego. Dodatkowo synchronizacja nie jest odtwarzalna – dodanie lub zagubienie fragmentu tekstu tajnego, nie będącego wielokrotnością długości bloku, całkowicie zrywa synchronizację procesu szyfrowania i deszyfrowania. Schematy blokowe, prezentujące sposób przetwarzania danych z wykorzystaniem metody ECB, przedstawiono na rysunkach 1.1 oraz 1.2.

Innym trybem jest metoda wiązania bloków zaszyfrowanych (CBC – *Cipher Block Chaining)*. Wiązanie to polega na wykorzystaniu mechanizmu sprzężenia zwrotnego. Blok tekstu jawnego przed zaszyfrowaniem jest sumowany modulo 2 z poprzednim blokiem tekstu tajnego. Wynik szyfrowania danego bloku jest zatem wykorzystywany w operacji szyfrowania kolejnego fragmentu. W efekcie każdy blok szyfrogramu zależy od bieżącego tekstu jawnego i poprzedniego bloku zaszyfrowanego. Wykorzystywany jest tu również wektor inicjalizujący, który zapewnia pseudolosową postać pierwszego bloku wejściowego, co daje możliwość uzyskania różnych szyfrogramów dla tej samej treści wiadomości jawnej. Należy tylko pamiętać, że

wektor ten musi być znany odbiorcy i dlatego powinien być mu wcześniej dostarczony w formie niezaszyfrowanej.

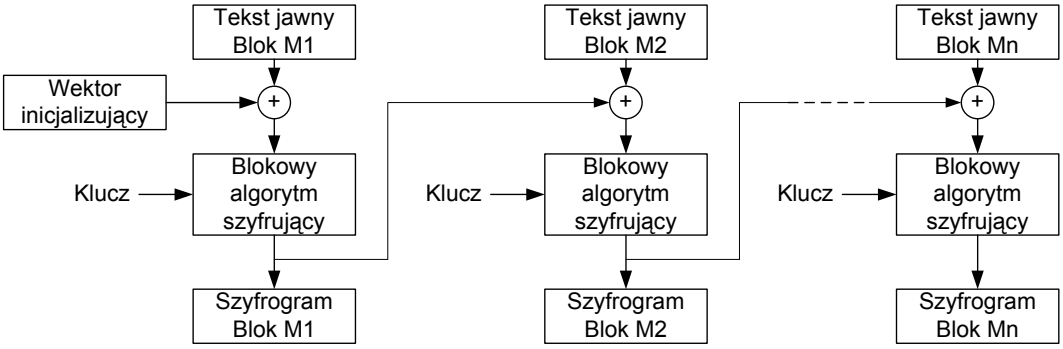


Rys. 1.1. Szyfrowanie z wykorzystaniem szyfrów blokowych w trybie ECB.

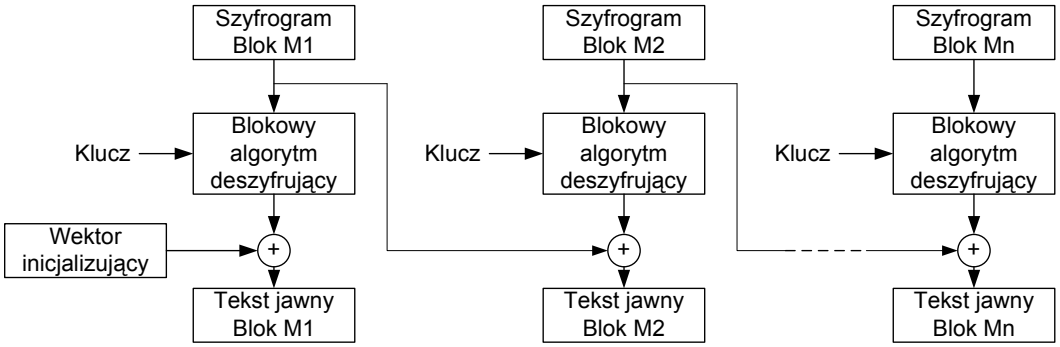


Rys. 1.2. Deszyfracja z wykorzystaniem szyfrów blokowych w trybie ECB.

Rozwiązanie CBC jest lepsze od trybu ECB, bo powtarzane fragmenty tekstu jawnego są ukrywane poprzez wykorzystanie sprzężenia zwrotnego oraz losowego wektora inicjalizującego. Ponadto więcej niż jedna wiadomość może być zaszyfrowana tym samym kluczem. Szyfrogram, podobnie jak w trybie ECB, jest większy od tekstu jawnego również co najwyżej o długość dopełnienia ostatniego bloku. Szyfrowanie nie może być tu jednak zrównoleglone ze względu na to, że każdy kolejny szyfrowany blok zależy od poprzednich. Błąd w szyfrogramie wpływa na jeden cały blok tekstu jawnego i odpowiadający mu bit w bloku następnym. Synchronizacja (podobnie jak w trybie ECB) nie jest odtwarzalna. Proces deszyfrowanie może być jednak zrównoleglony, ponieważ kolejne szyfrogramy potrzebne do odszyfrowania danych są najczęściej dostępne o ile już je odebrano. Schematy blokowe, prezentujące sposób szyfrowania i deszyfracji z wykorzystaniem omawianej metody, przedstawiono na rysunkach odpowiednio 1.3 oraz 1.4.



Rys. 1.3. Szyfrowanie z wykorzystaniem szyfrów blokowych w trybie CBC.



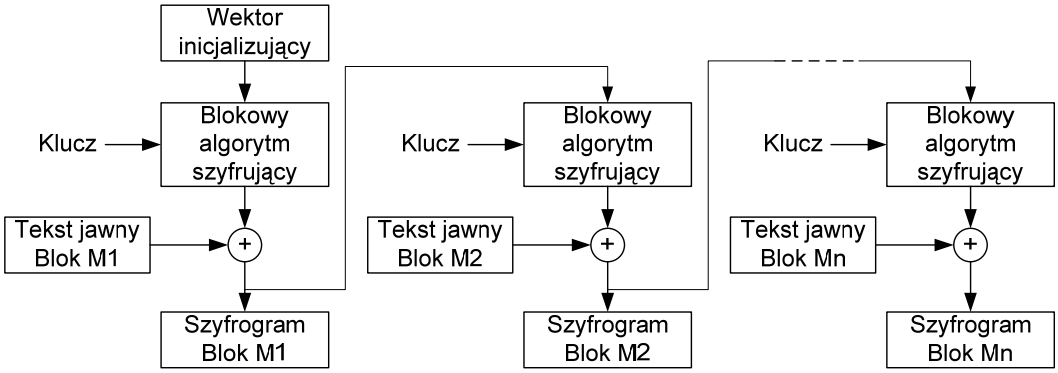
Rys. 1.4. Deszyfracja z wykorzystaniem szyfrów blokowych w trybie CBC.

Kolejnym trybem szyfrowania jest metoda sprzężenia zwrotnego szyfrogramu (CFB – *Cipher FeedBack*). Podobnie jak metoda CBC, rozwiązanie to również działa w ten sposób, że dany blok szyfrogramu zależy od całego poprzedzającego go tekstu jawnego. Ponownie stosowany jest tu wektor inicjalizujący oraz dodatkowo parametr *s*, będący liczbą naturalną, która określa wielkość (w bitach) bloku szyfrogramu i tekstu jawnego. Jest to podstawowa cecha odróżniająca ten tryb od wszystkich innych omawianych w tym paragrafie. Dzięki takiemu rozwiązaniu, które szczegółowo omówiono w [55], możliwe staje się uzyskanie szyfru samosynchronizującego się. Cecha ta jednak może zostać uzyskana tylko wówczas, gdy w trakcie deszyfracji zostanie utraconych lub dodanych *n*s* bitów (gdzie n jest dowolną liczą naturalną).

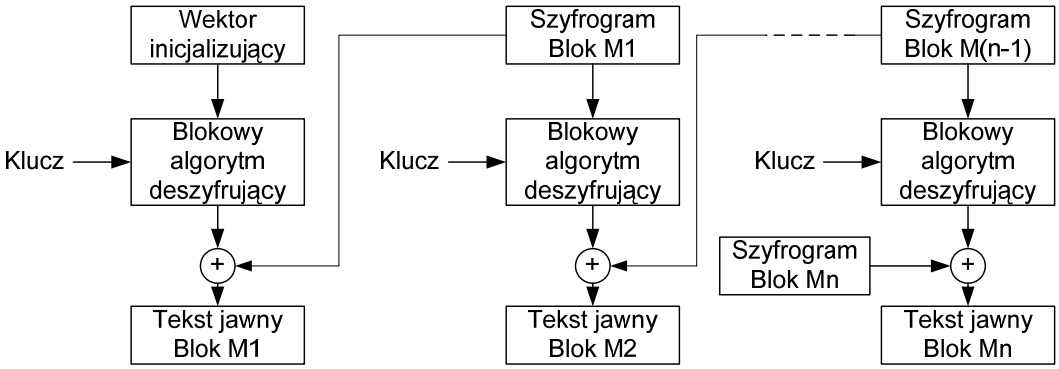
Dla omawianego trybu charakterystyczne fragmenty tekstu jawnego są ukrywane, a ciąg wyjściowy ma charakter losowy. Ponadto tym samym kluczem może być zaszyfrowana więcej niż jedna wiadomość. Szybkość szyfrowania w trybie CFB natomiast jest taka sama, jak szybkość samego szyfru blokowego jedynie wtedy, gdy rozmiar *s* bloku wejściowego jest równy długości ciągu wejściowego dla bazowego szyfru blokowego. Podobnie jak poprzednio, szyfrogram jest tej samej długości, co tekst jawny plus dopełnienie ostatniego bloku. W tym jednak wypadku fakt ten może

mieć mniejsze znaczenie, ponieważ długość bloku może być mniejsza niż w innych trybach, a nawet tak dopasowana, by dopełnienie nie było konieczne.

Błąd w szyfrogramie wpływa na jeden bit odpowiadającego mu tekstu jawnego i cały następny blok. Szyfrowanie nie może być zrównoleglone, natomiast deszyfrowanie już tak. Schematy blokowe, prezentujące sposób szyfrowania i deszyfracji z wykorzystaniem tej metody, przedstawiono na rysunkach odpowiednio 1.5 oraz 1.6. Założono tu jednak, że długość pojedynczego bloku tekstu jawnego i szyfrogramu jest równa długości sekwencji wejściowej dla blokowego algorytmu szyfrującego.



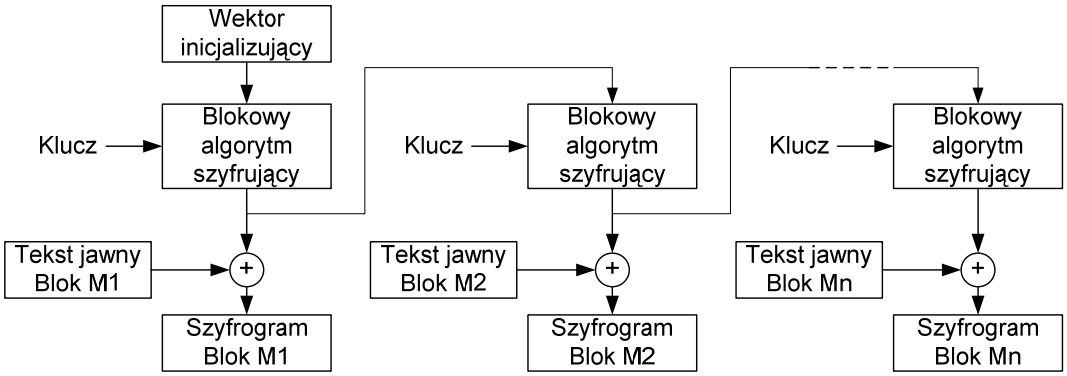
Rys. 1.5. Szyfrowanie z wykorzystaniem szyfrów blokowych w trybie CFB.



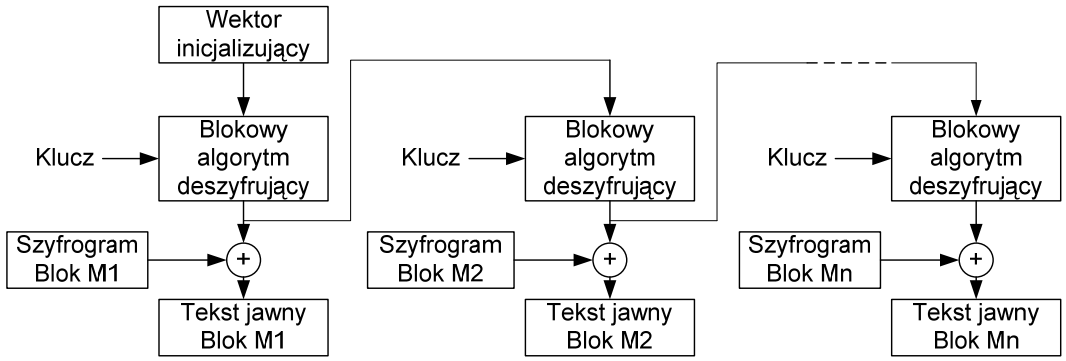
Rys. 1.6. Deszyfracja z wykorzystaniem szyfrów blokowych w trybie CFB.

Kolejnym z trybów szyfrowania jest rozwiązanie wykorzystujące sprzężenie zwrotne wyjścia (OFB – *Output FeedBack*). Przetwarzanie tutaj nie może być zrównoleglone, ale błąd w szyfrogramie wpływa tylko na odpowiadający mu bit tekstu jawnego. Synchronizacja nie jest odtwarzalna i podobnie jak w poprzednich trybach stosuje się tu wektor inicjalizujący, który musi być unikalny dla każdego użycia szyfru z tym samym tajnym kluczem. Te same wiadomości jawne uzyskają zatem różne i pseudolosowe postacie tajne. Schematy blokowe, prezentujące sposób szyfrowania i

deszyfracji z wykorzystaniem tej metody, przedstawiono na rysunkach odpowiednio 1.7 oraz 1.8.



Rys. 1.7. Szyfrowanie z wykorzystaniem szyfrów blokowych w trybie OFB.



Rys. 1.8. Deszyfracja z wykorzystaniem szyfrów blokowych w trybie OFB.

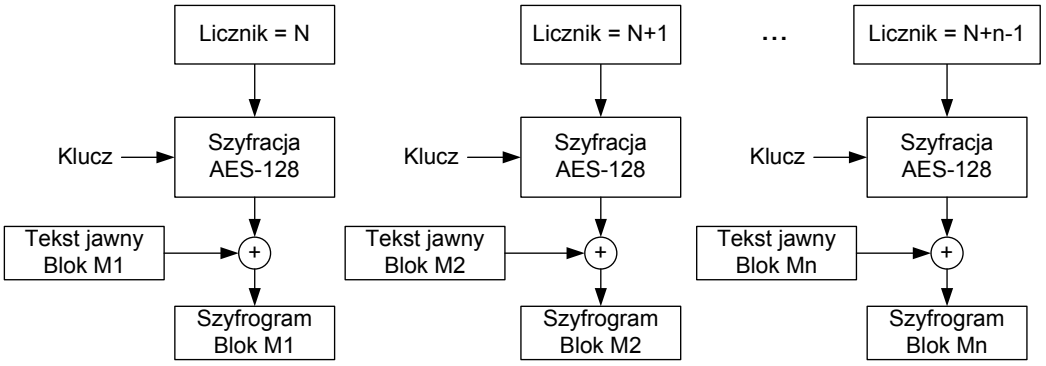
Ostatnim omawianym trybem szyfrowania jest rozwiązanie licznikowe (*CounTeR Mode* – CTR). Jest ono bardzo podobne do szyfrowania w trybie ECB, ponieważ nie występują tu żadne sprzężenia zwrotne. Wejściem do funkcji szyfrującej jest tu jednak tzw. licznik, którego wartość przy szyfrowaniu każdego kolejnego bloku jest zwiększana (a w ogólności po prostu zmieniana). Blok tekstu jawnego jest dodawany modulo 2 do zaszyfrowanej sekwencji licznika.

Tryb ten ma podobne zalety jak rozwiązania wykorzystujące sprzężenia zwrotne i wektor inicjalizujący. Każda zaszyfrowana wiadomość ma bowiem charakter pseudolosowy a dzięki temu, że wartości licznika są za każdym razem inne, otrzymuje się różne wartości szyfrogramów dla identycznych postaci tekstów jawnych i tych samych kluczy. Zaletą prezentowanego tu trybu pracy jest jednak również fakt, że zarówno operacje szyfrowania jak i deszyfrowania można zrównoleglić, a błędy nie propagują się. Niepoprawny bit w szyfrogramie daje w efekcie tylko błąd, odpowiadającego mu bitu tekstu jawnego. Ponadto proces szyfrowania i deszyfracji jest identyczny – łatwiejsza implementacja. Synchronizacja w trybie CTR nie jest

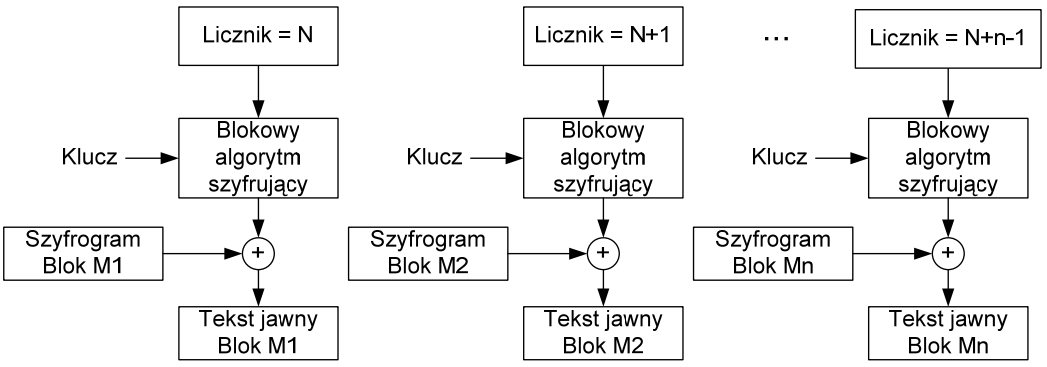
jednak automatycznie odzyskiwana i może być ponadto konieczne dopełnianie ostatniego bloku tekstu szyfrowanego.

Przykładowe sposoby ustalania wartości licznika w prezentowanym trybie przedstawiono w [55]. Standard ten daje między innymi możliwość wykorzystania fragmentu sekwencji licznika jako pewnego ciągu, niepowtarzalnego dla kolejnych wiadomości, gdy szyfrowane są one tym samym kluczem. Najprostszym przykładem jest tu rozwiązanie, w którym część bitów sekwencji licznika odpowiada za zliczanie bloków w danej wiadomości, a pozostałe zmieniają się dla kolejnych wiadomości.

Bardzo istotny jest w tym miejscu jednak fakt, że poprawnie zaimplementowany tryb CTR musi spełniać zasadę unikalności. Oznacza to, że sposób ustalania kolejnych sekwencji licznika jest właściwie dowolny, ale ciągi te muszą być różne dla każdego bloku wiadomości. Ponadto przy stosowaniu tego samego klucza sekwencje te muszą się również różnić dla bloków wszystkich wiadomości. Warto tu dodatkowo podkreślić, że sposób ustalania sekwencji licznika oraz same jego wartości nie muszą być tajne, ale muszą (!) być unikalne. Schematy blokowe, prezentujące sposób szyfrowania i deszyfracji z wykorzystaniem omawianej metody, przedstawiono na rysunkach odpowiednio 1.9 oraz 1.10.



Rys. 1.9. Szyfrowanie z wykorzystaniem szyfrów blokowych w trybie CTR.



Rys. 1.10. Deszyfracja z wykorzystaniem szyfrów blokowych w trybie CTR.

Podsumowując ten paragraf należy dodać, że, ze względu na swe zalety, tryb CTR został wybrany przez autora jako podstawowa metoda szyfrowania w nowym kryptosystemie dla łącza HF. Szczegóły odnośnie sposobu jej wykorzystania oraz implementacji przedstawione zostaną w rozdziale trzecim oraz czwartym niniejszej rozprawy.

W kolejnym paragrafie z kolei przedstawiony zostanie jeden z najbezpieczniejszych [37, 44, 81], stosowanych obecnie, blokowych algorytmów symetrycznych, który może zostać wykorzystany w każdym z omówionych tu trybów pracy.

1.2.2. Standard AES (*Advanced Encryption Standard*)

26 maja 2002 roku, w odpowiedzi na niedostateczne bezpieczeństwo algorytmu DES [4 – 6, 16, 45, 54], wprowadzono w życie nowy standard szyfrowania – AES [26, 81]. Rozwiązanie to wykorzystuje algorytm *Rijndael* [18, 44] – szyfr symetryczny blokowy o blokach danych 128 bitowych (wejściem i wyjściem dla algorytmu są sekwencje bitów zgrupowane w bloki po 128 bitów) i długościach klucza *K*: 128, 192 lub 256 bitów. *Rijndael* może wykorzystywać również dłuższe klucze, lecz nie są one ujęte w standardzie AES.

Poszczególne operacje w algorytmie AES realizowane są na dwuwymiarowej tablicy bajtów. Tablica ta nazywana jest stanem algorytmu. Składa się ona z czterech wierszy, z których każdy zawiera *Nb* bajtów, gdzie *Nb* jest długością bloku wejściowego podzieloną przez 32. Ze względu na to, że wejście algorytmu jest 128 bitowe, to *Nb* = *4* i oznacza liczbę kolumn (słów) tablicy stanu.

Po cztery bajty w czterech kolumnach tworzą zatem stan algorytmu. Może być on interpretowany jako jednowymiarowa tablica czterech 32 bitowych słów (kolumn).

Długość klucza z kolei, reprezentowana zmienną *Nk*, może być równa 4, 6 lub 8. Wartość ta oznacza liczbę kolumn (32 bitowych słów) w tablicy klucza szyfrującego.

Dla algorytmu AES liczba rund (*Nr*), wykonywanych cyklicznie, koniecznych do zaszyfrowania wiadomości zależy od długości wybranego klucza. Związek pomiędzy długością klucza, rozmiarem bloku i liczbą rund przedstawia tabela 1.1.

W każdej rundzie algorytmu przeprowadzane są cztery transformacje (etapy), które zostały przedstawione w dalszej części tego paragrafu. Są to:

- Podstawianie bajtów macierzy stanów;
- Przesuwanie cykliczne wierszy macierzy stanu;

- Przestawianie kolumn macierzy stanu według przekształceń na ciałach Galois;
- Dodawanie (XOR) klucza rundy do stanu.

Tab. 1.1. Zależności pomiędzy długością klucza, rozmiarem bloku i liczbą rund w algorytmie AES.

	Długość klucza (*Nk* słów)	**Długość bloku (*Nb* słów)**	**Liczba rund (*Nr* rund)**
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

W celu opisania algorytmu posłużono się pseudokodem zaczerpniętym ze specyfikacji algorytmu AES [26]. Na potrzeby pseudokodu należy jednak zdefiniować pseudofunkcje, odpowiadające poszczególnym etapom każdej rundy. I tak za: podstawianie bajtów macierzy stanów odpowiada funkcja *SubBytes()*, przesuwanie cykliczne – *ShiftRows()*, przestawianie kolumn – *MixColumns()* i dodawanie klucza rundy – *AddRoundKey()*.

W pseudokodzie szyfrowanie algorytmem AES przedstawić można w następujący sposób:

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]
  state = in
  AddRoundKey(state, w[0, Nb-1])//dodanie klucza rundy przed rundą pierwszą
  for round = 1 step 1 to Nr-1
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
  end for
  SubBytes(state)
  ShiftRows(state)      //ostatnia runda bez operacji MixColumns()
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  out = state
end
```

Transformacja *SubBytes()* zależy od bieżącego stanu algorytmu. Dla każdego wejściowego bajtu z tablicy stanu obliczana jest jego wartość wyjściowa według poniższego równania:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \tag{1.1}$$

gdzie:

b_0 (LSB) - b_7 (MSB) – bity bajtu wejściowego stanu;

b'_0 (LSB) - b'_7 (MSB) – bity bajtu wyjściowego funkcji *SubBytes().*

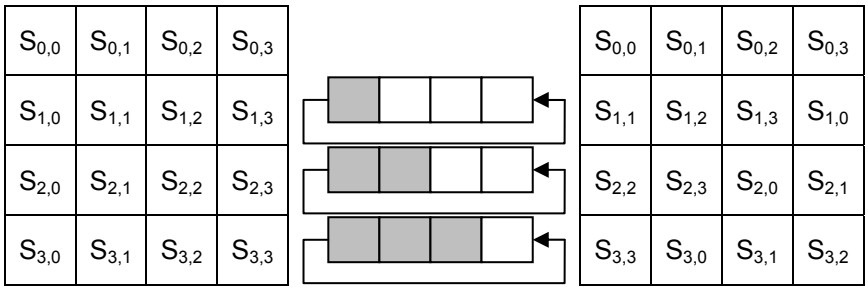
Prościej można przedstawić tą transformację, jako tzw. tablicę S-box, dla której bajtowi wejściowemu przyporządkowywany jest bajt wyjściowy według tabeli 1.2.

Tab. 1.2. Tablica S-box dla algorytmu AES

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Jeśli na przykład na wejściu znajduje się *0xAF*, to na wyjściu otrzyma się *0x79*.

Transformacja *ShiftRows()* polega na cyklicznej zamianie bajtów trzech ostatnich wierszy stanu algorytmu. Obrazuje to rysunek 1.11.



Rys. 1.11. Transformacja *Shift Rows* w AES

Pierwszy bajt drugiego wiersza tablicy stanu jest przesuwany na koniec tego wiersza, dwa pierwsze bajty trzeciego wiersza są przesuwane na jego koniec i analogicznie postępuje się z trzema pierwszymi bajtami czwartego wiersza.

Transformacja *MixColumns()* realizuje z kolei operację:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad dla\ 0 \le c < Nb, \tag{1.2}$$

gdzie $S_{i,j}$ to bajt stanu z *i*-tego wiersza i *j*-tej kolumny tablicy stanu.

Przykładowo, dla pierwszego wiersza tablicy stanów, przeprowadzane jest następujące przekształcenie:

$$S'_{0,c} = (\{02\} \bullet S_{0,c}) \oplus (\{03\} \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \quad dla\ 0 \le c < Nb, \tag{1.3}$$

gdzie operacja mnożenia • zdefiniowana została w sposób specyficzny dla algorytmu AES [26].

Podczas transformacji *AddRoundKey()* klucz rundy dodawany jest do stanu poprzez prostą operację XOR. Każdy klucz rundy składa się z *Nb* słów, a nie z *Nk* słów jak klucz główny (K) algorytmu AES. Wykorzystywany klucz zmienia się z każdą rundą w sposób opisany poniżej (funkcja *KeyExpansion()*).

Funkcja *KeyExpansion()* w algorytmie AES przy użyciu klucza szyfrującego K wyznacza klucz tymczasowy dla każdej rundy. Generowanych jest *Nb*(Nr+1)* słów, a więc *Nr+1* kluczy. Tworzy się zatem o jeden klucz więcej niż liczba rund, ponieważ pierwsza transformacja *AddRoundKey()* występuje jeszcze przed pierwszą rundą algorytmu AES.

Funkcja *KeyExpansion()* wykorzystuje początkowy zestaw *Nk* słów klucza K i dla każdej rundy tworzony jest na jego podstawie nowy zbiór *Nb* słów klucza rundy. Rozszerzenie wejściowego klucza przebiega w następujący sposób:

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
  word temp
  i = 0
  while (i < Nk)
    w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
    i = i+1
  end while
  i = Nk
  while (i < Nb * (Nr+1)]
    temp = w[i-1]
    if (i mod Nk = 0)
      temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
    else if (Nk > 6 and i mod Nk = 4)
      temp = SubWord(temp)
    end if
    w[i] = w[i-Nk] xor temp
    i = i + 1
  end while
end
```

Funkcja *SubWord()* pobiera czterobajtowe słowo wejściowe i dokonuje na nim operacji jak funkcja *SubBytes()* dla poszczególnych bajtów. *RotWord()* dokonuje cyklicznego przesunięcia bitowego słów o jeden bajt w lewo. *Rcon[i/Nk]* to stała czterobajtowa tablica postaci:

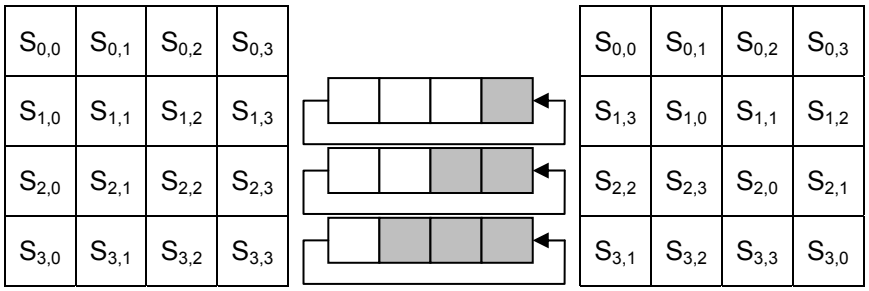
$$\left(\left\{02^{(i/Nk)-1}\right\},\{00\},\{00\},\{00\}\right), \tag{1.4}$$

którą wyznacza się zgodnie z zasadami mnożenia dla algorytmu AES.

Deszyfrowanie AES polega na wykonaniu operacji szyfrowania w odwrotnej kolejności. Transformacje stosowane przez deszyfrator AES to *InvShiftRows()*, *InvSubBytes()*, *InvMixColumns()* i *AddRoundKey()*. Deszyfrowanie można przedstawić za pomocą następującego pseudokodu:

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]
  state = in
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  for round = Nr-1 step -1 downto 1
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    InvMixColumns(state)
  end for
  InvShiftRows(state)
  InvSubBytes(state)
  AddRoundKey(state, w[0, Nb-1])
  out = state
end
```

Transformacja *InvShiftRows()* zrealizowana jest podobnie jak *ShiftRows()*, co zobrazowano na rysunku 1.12.



Rys. 1.12. Transformacja *Inverted Shift Rows* w AES

Transformacja *InvSubBytes()* wykonuje zadania odwrotne do *SubBytes()*. Tablica S-box ma tu postać przedstawioną w tabeli 1.3.

Tab. 1.3. Tablica S-box dla deszyfrowania AES

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Transformacja *InvMixColumns()* wykonuje zadania odwrotne do *MixColumns()*:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad \textit{dla } 0 \le c < Nb. \tag{1.5}$$

Przykładowo, dla pierwszego wiersza tablicy stanów, przeprowadzona jest następująca operacja:

$$S'_{0,c} = (\{0e\} \bullet S_{0,c}) \oplus (\{0b\} \bullet S_{1,c}) \oplus (\{0d\} \bullet S_{2,c}) \oplus (\{09\} \bullet S_{3,c}) \quad \textit{dla } 0 \le c < Nb. \tag{1.6}$$

Należy w tym miejscu ponownie pamiętać o specyficznym sposobie mnożenia ($\bullet$) zdefiniowanym a standardzie AES [26], gdzie można znaleźć również wszelkie szczegóły odnośnie sposobu implementacji tego algorytmu oraz szczegółowe przykłady szyfrowania oraz deszyfrowania.

Warto w tym miejscu zauważyć jeszcze kilka aspektów. Zaletą algorytmu AES jest to, że nie posiada on słabych kluczy. Nie ma więc żadnych ograniczeń co do ich wyboru. Ponadto omawiany tu szyfr pozostaje jak dotąd niezłamany [81]. Posiada on także bardzo wygodną algebraicznie reprezentację, co ułatwia jego implementację, ale również w pewnym sensie stawia poziom jego bezpieczeństwa pod znakiem zapytania. Istnieje bowiem ryzyko, że w przyszłości znajdzie się równie łatwa algebraicznie metoda, służącą złamaniu tego algorytmu. Nie jest to jednak wystarczający argument, który mógłby w jakikolwiek sposób przemawiać za tezą, że algorytm AES nie jest szyfrem bezpiecznym.

Dodatkowo nie są znane [44, 81] jakiekolwiek skuteczne i szybkie rozwiązania, pozwalające na wykorzystanie ataków kryptoanalitycznych, umożliwiające złamanie tego szyfru w jego pełnej formie[4]. Atak „pełnego przeglądu" [89, 90] z kolei, dla długości kluczy stosowanych w algorytmie AES, nie ma przy dzisiejszym stanie technologicznym większego sensu [81].

Ze względu na powyższe fakty standard AES został wybrany przez autora jako podstawowy algorytm szyfrowania w nowym kryptosystemie (patrz rozdział trzeci) dla transmisji danych w łączu krótkofalowym.

5.1.1. Szyfrowanie z kluczem publicznym – Algorytm RSA (*Rivest-Shamir-Aldeman*)

Algorytm RSA [61, 89, 90] opiera swoje bezpieczeństwo na trudności rozkładu bardzo dużych liczb na czynniki pierwsze [75]. W rozwiązaniu tym stosowane są dwa typy kluczy – klucz publiczny RSA oraz klucz prywatny RSA. Razem oba te klucze tworzą tzw. parę kluczy RSA. Obecna specyfikacja wspiera tzw. *multi-prime* RSA, który polega na tym, że dzielnik RSA może mieć więcej niż dwa dzielniki w postaci liczb pierwszych. Zaletą *multi-prime* jest mniejsza złożoność obliczeniowa algorytmu – lepsza wydajność na platformach jednoprocesorowych. Zastosowanie jednak tego rozwiązania wpływa niekorzystnie na bezpieczeństwo algorytmu RSA.

W podstawowej realizacji RSA klucz publiczny składa się z dwóch komponentów:

n – dzielnik RSA (liczba naturalna);

e – publiczny eksponent RSA (liczba naturalna).

W prawidłowym kluczu RSA, dzielnik RSA *n* jest produktem mnożenia dwóch (lub większej liczby – *multi-prime*) dużych liczb pierwszych *p* oraz *q*:

$$n = p \cdot q \,. \tag{1.7}$$

Jego długość określna ponadto długość klucza RSA.

Eksponent *e* jest wartością naturalną z przedziału od *1* do *n-1*. Jest to liczba względnie pierwsza z funkcją Eulera $\varphi(n)$:

$$\varphi(n) = (p-1)\cdot(q-1). \tag{1.8}$$

Klucz prywatny RSA składa się z pary liczb (*n,d*):

[4] Istnieją pewne rozwiązania [37, 81] skutecznej kryptoanalizy algorytmu AES, ale jedynie dla przypadków wykorzystania zmniejszonej liczby rund szyfrowania – standard AES nie dopuszcza jednak takich sposobów zastosowania algorytmu *Rijndael*.

n – dzielnik RSA (liczba naturalna);

d – prywatny eksponent RSA (liczba naturalna).

W prawidłowej reprezentacji klucza RSA wartość *n* jest taka sama jak dla klucza publicznego. Prywatny eksponent RSA z kolei jest liczbą naturalną mniejszą od *n* i spełniającą równanie:

$$e \cdot d \equiv 1 (\mathrm{mod}\, \varphi(n)) . \tag{1.9}$$

Szyfrowanie w algorytmie RSA odbywa się z wykorzystaniem jawnej wiadomości *m* oraz klucza publicznego RSA *(n, e)*. Wiadomość, która podlega zaszyfrowaniu, musi być z przedziału od *0* do *n-1*. Proces szyfrowania przedstawia poniższa zależność:

$$c = m^e \bmod n , \tag{1.10}$$

gdzie *c* oznacza tekst zaszyfrowany.

Deszyfracja z kolei odbywa się z wykorzystaniem zaszyfrowane wiadomości *c* oraz klucza tajnego RSA *(n, d)*. Wiadomość zaszyfrowana musi być z przedziału od 0 do *n-1*. Proces deszyfracji przedstawia poniższa zależność:

$$m = c^d \bmod n . \tag{1.11}$$

Warto w tym momencie dodać, że algorytm RSA może być również stosowany dla realizacji uwierzytelniania. W takim przypadku pozwala on na wygenerowanie podpisu *s* dla wiadomości *m* (*m* z przedziału od 0 do *n-1*) lub dla jej skrótu (np. z wykorzystaniem algorytmu SHA-256) przy wykorzystaniu klucza prywatnego *(n, d)*:

$$s = m^d \bmod n . \tag{1.12}$$

Weryfikacji tego podpisu (*s* z przedziału od 0 do *n-1*) dokonuje się z wykorzystaniem klucza publicznego *(n, e)*:

$$m = s^e \bmod n . \tag{1.13}$$

W przypadku szyfrowania nadawca jest pewny, że jego wiadomość odebrać może tylko osoba, która posiada klucz prywatny, tworzący parę z kluczem publicznym użytym do szyfrowania.

W sytuacji uwierzytelniania z kolei odbiorca jest pewny, że odebrana przez niego wiadomość pochodzić może tylko od osoby, która posiada klucz prywatny, związany z kluczem odbiorczym, użytym do weryfikacji.

Idealna sytuacja występuje zatem, gdy każda z komunikujących się ze sobą osób posiada swój klucz prywatny, a wszystkie klucze publiczne są ogólnie dostępne. W takim przypadku możliwa jest jednoczesna realizacja mechanizmów poufności oraz

uwierzytelniania. Zastosowanie dodatkowo bezpiecznej funkcji skrótu (np. SHA-256) pozwala również efektywnie realizować kontrolę integralności, przesyłanych danych.

Na zakończenie tego paragrafu warto dodać, że w literaturze nie ma rozwiązań, pozwalających sądzić, że wykorzystanie algorytmu RSA z kluczem o długości co najmniej 1024 bitów mogłoby nieść ze sobą jakiekolwiek praktyczne ryzyko [81, 89].

Ponadto rozwiązanie w tym paragrafie zaprezentowane jest wykorzystywane w zaproponowanym kryptosystemie jako jeden z algorytmów pozwalających na implementację bezpiecznych podpisów cyfrowych, generowanych dla wiadomości zarządzających – patrz rozdział trzeci.

1.2.3. Funkcja skrótu – Algorytm SHA-256 (*Secure Hash Algorithm*)

Funkcja skrótu, jak już wcześniej wspomniano, jest wykorzystywana najczęściej do realizacji mechanizmu kontroli integralności przesyłanych danych, a w połączeniu z asymetrycznymi algorytmami szyfrującymi również dla celów uwierzytelniania.

Za pomocą konkretnego algorytmu (np. SHA [24], MD4 [68], MD5 [69, 70] lub inne [62, 96]) dla danej wiadomości, wyznacza się jej tzw. skrót (*hash*) o z góry ustalonej długości. Funkcja haszująca przetwarza w charakterystyczny dla niej i najczęściej standaryzowany sposób dane wejściowe i generuje na wyjściu pewien ciąg pseudolosowy. Sekwencja ta to jakby odcisk palca danej wiadomości (danych).

Jedną z charakterystycznych właściwości prezentowanej tu grupy algorytmów jest to, iż niewielka modyfikacja wiadomości całkowicie zmienia postać jej skrótu. Z tego właśnie powodu funkcje haszujące są tak przydatne do kontroli integralności danych. Ponadto, jak już wcześniej wspomniano, bezpieczna funkcja skrótu nie pozwala na wytworzenie dwóch wiadomości o identycznym skrócie jak również wygenerowanie sekwencji o skrócie identycznym jak *hash* wiadomości zadanej. Nie możliwe jest również odzyskanie danych (czy choćby ich fragmentu) na podstawie samego tylko ich skrótu. Funkcje skrótu powinny być bowiem z definicji jednoznaczne i jednokierunkowe [97]. Udowodnienie takich właściwości nie jest jednak na dzień dzisiejszy najczęściej możliwe. W przypadku, gdy nie istnieją jednak metody, pozwalające stwierdzić, że dane rozwiązanie nie jest jednoznaczne czy jednokierunkowe, zakłada się najczęściej, że jest ono bezpieczne. Założenie takie jest tym pewniejsze, im bardziej rozpowszechniony jest dany algorytm.

Najbardziej popularnym i uważanym za bezpieczny typem funkcji skrótu jest SHA-2. Wyróżnia się tu algorytm SHA-256, generujący skrót o długości 256 bitów, SHA-

384 (skrót 384 bitowy) oraz SHA-512 (skrót 512 bitowy). W literaturze nie są bowiem znane żadne rozwiązania, pozwalające na wytworzenie kolizji[5] dla funkcji ze wspomnianej grupy. Nie ma również żadnych innych przesłanek, mogących podważyć domniemaną jednoznaczność i jednokierunkowość wspomnianych tu algorytmów.

Ciągiem wejściowym dla algorytmów SHA-256 są wiadomości o długości mniejszej niż 2^{64} bitów. Dla SHA-384 oraz SHA-512 natomiast ograniczenie to wynosi 2^{128} bitów. W dalszej części tego paragrafu przedstawiono zagadnienia związane z SHA-256, ponieważ tylko ten algorytm wykorzystywany będzie w zaproponowanym w rozdziale trzecim niniejszej pracy kryptosystemie dla transmisji danych w łączu krótkofalowej. Dodatkowe informacje odnośnie pozostałych odmian algorytmu SHA-2 można znaleźć w [24].

1.2.3.1. Podstawowe funkcje i stałe wykorzystywane przez SHA-256

Algorytm ten używa sześciu funkcji logicznych, z której każda operuje na 32 bitowych słowach (*x*, *y*, *z*). Na wyjściu każdej z nich otrzymuje się również słowo 32 bitowe. Wspomniane funkcje są następującej postaci:

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z), \tag{1.14}$$

$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), \tag{1.15}$$

$$\sum\nolimits_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x), \tag{1.16}$$

$$\sum\nolimits_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x), \tag{1.17}$$

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x), \tag{1.18}$$

$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x). \tag{1.19}$$

Prezentowany algorytm stosuje również funkcję $ROTL^n(x)$, zdefiniowaną jako przesunięcie bitowe cykliczne słowa *x* o *n* bitów w lewo oraz funkcję $ROTR^n(x)$ – cykliczne przesunięcie bitowe w prawo o *n* bitów.

SHA-256 wykorzystuje również 64 stałych 32 bitowych słów $K_0^{\{256\}}$, $K_1^{\{256\}}$, … $K_{63}^{\{256\}}$. Zapisane od lewej do prawej przedstawiają się one następująco:

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

[5] Wygenerowanie dwóch różnych wiadomości o identycznym skrócie.

Dodatkowo, używana jest też funkcja $SHR^n(x)$, zdefiniowana jako przesunięcie bitowe (niecykliczne) słowa x o n bitów w prawo.

1.2.3.2. Przetwarzanie wstępne

Zanim algorytm rozpocznie wyznaczanie skrótu danej wiadomości, należy tą wiadomość odpowiednio przygotować. Po pierwsze, dla potrzeb algorytmu SHA-256 długość wiadomości musi być wielokrotnością 512 bitów.

W celu zapewnienia zatem odpowiedniej długości wiadomości, przeprowadzane są przedstawione poniżej operacje.

Zakłada się, że długość wiadomości M jest równa l bitów. Do jej końca należy dopisać bit o wartości 1 a następnie k zerowych bitów, tak aby k było najmniejszą liczbą naturalną spełniającą równanie:

$$(l + 1 + k) \bmod 512 = 448. \tag{1.20}$$

Na końcu tak powstałej wiadomości, należy dopisać jej długość (w bitach), wyrażoną poprzez 64 bitową sekwencję.

Dzięki przedstawionym tu operacjom uzyskuje się wiadomości o odpowiednim rozmiarze.

Kolejnym krokiem przetwarzania wstępnego jest podzielenie wiadomości na bloki o określonej długości. Dla algorytmu SHA-256 poprzednio sformatowana wiadomość jest dzielona na N 512 bitowych bloków $M^{(1)}, M^{(2)}, \ldots M^{(N)}$. Dzięki temu, że 512 bitów wejściowego bloku może być wyrażone za pomocą szesnastu 32 bitowych słów, pierwsze 32 bity i-tego bloku wiadomości jest oznaczone jako $M_0^{(i)}$, następne 32 bity jako $M_1^{(i)}$, aż do ostatnich 32 bitów – $M_{15}^{(i)}$.

Finalnym krokiem przetwarzania wstępnego jest ustalenie początkowej wartości funkcji haszującej $H^{(0)}$. Dla SHA-256 $H^{(0)}$ składa się z następujących ośmiu 32 bitowych słów:

$$H_0^{(0)} = \texttt{6a09e667}$$
$$H_1^{(0)} = \texttt{bb67ae85}$$
$$H_2^{(0)} = \texttt{3c6ef372}$$
$$H_3^{(0)} = \texttt{a54ff53a}$$
$$H_4^{(0)} = \texttt{510e527f}$$
$$H_5^{(0)} = \texttt{9b05688c}$$
$$H_6^{(0)} = \texttt{1f83d9ab}$$
$$H_7^{(0)} = \texttt{5be0cd19}$$

Na tym kończy się proces przetwarzania wstępnego.

1.2.3.3. Wyznaczanie sekwencji skrótu

Poniżej przedstawiono (pseudokod) sposób wyznaczenia skrótu SHA-256.

Pętla od *i* = *1* do *N*:
{
//Przygotowanie wiadomości

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

//Zainicjowanie zmiennych roboczych
a = $H_0^{(i-1)}$
b = $H_1^{(i-1)}$
c = $H_2^{(i-1)}$
d = $H_3^{(i-1)}$
e = $H_4^{(i-1)}$
f = $H_5^{(i-1)}$
g = $H_6^{(i-1)}$
h = $H_7^{(i-1)}$

Pętla od *t* = *0* do *63*
{

$$T_1 = h + \sum\nolimits_1^{\{256\}}(e) + Ch(e, f, g) + K_t^{\{256\}} + W_t$$
$$T_2 = \sum\nolimits_0^{\{256\}}(a) + Maj(a, b, c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

}
//Obliczenie wartości tymczasowej funkcji haszującej $H^{(i)}$
$H_0^{(i)}$ = a + $H_0^{(i-1)}$
$H_1^{(i)}$ = b + $H_1^{(i-1)}$
$H_2^{(i)}$ = c + $H_2^{(i-1)}$
$H_3^{(i)}$ = d + $H_3^{(i-1)}$
$H_4^{(i)}$ = e + $H_4^{(i-1)}$
$H_5^{(i)}$ = f + $H_5^{(i-1)}$
$H_6^{(i)}$ = g + $H_6^{(i-1)}$
$H_7^{(i)}$ = h + $H_7^{(i-1)}$
}

Po wykonaniu *N* powyższych operacji (dla całej wiadomości), wyjściową 256 bitową wartością funkcji haszującej SHA-256 jest:

$$H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)} || H_4^{(N)} || H_5^{(N)} || H_6^{(N)} || H_7^{(N)}. \quad (1.21)$$

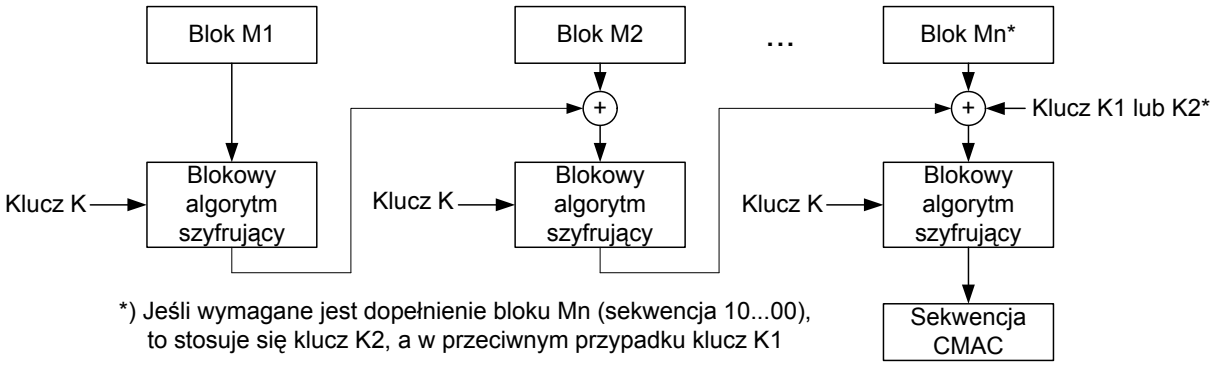
1.2.4. Algorytm CMAC (*Cipher-based Message Authentication Code*)

Algorytm CMAC (*Cypher-based MAC*) pozwala na realizację mechanizmu integralności oraz uwierzytelniania z wykorzystaniem sekwencji uwierzytelniających

MAC. Zgodnie z jego specyfikacją [56] bazuje on na blokowych szyfrach symetrycznych (np. AES). CMAC może być w ogólności rozumiany jako pewien dodatkowy tryb pracy algorytmów blokowych (patrz punkt 1.2.1).

CMAC stanowi silne zabezpieczenie przed przypadkowym czy celowym naruszeniem integralności wiadomości. Może również zostać wykorzystany dla realizacji uwierzytelniania jej nadawcy.

Sposób generacji ciągu CMAC przedstawiono na rysunku 1.13.



Rys. 1.13. Algorytm CMAC.

W pierwszym etapie wyznaczania sekwencji CMAC należy podzielić całą wiadomość *M* na *n* bloków tej samej długości. Jeśli jest to konieczne, to ostatnią sekwencję należy dopełnić ciągiem postaci *10…00*. W przypadku, gdy dopełnienie miało miejsce wykorzystuje się *Klucz K2*, a gdy długość wiadomość jest całkowitą wielokrotnością długości bloku *Mi* stosuje się *Klucz K1*.

Długość pojedynczego bloku *Mi* (*i = 1, 2, …, n*) zależy od zastosowanego szyfru blokowego. W przypadku algorytmu AES-128 sekwencje: *Mi, Klucz K, Klucz K1, Klucz K2* oraz ciąg CMAC[6] mają po 128 bitów.

Procedura generacji kluczy *K1* oraz *K2* jest następująca [56]:

- Niech *L* będzie wynikiem szyfrowania (kluczem *K* i danym szyfrem blokowym) pojedynczego bloku składającego się z samych zer.
- Jeśli najbardziej znaczącym bitem *L* będzie 0, to *K1 = L << 1*, a w przeciwnym wypadku *K1 = (L << 1) ⊕ Rb*, gdzie przykładowo *Rb* dla szyfrów 128-bitowych to sekwencja 128-bitowa postaci 00…0010000111.
- Jeśli najbardziej znaczącym bitem *K1* będzie 0, to *K2 = K1 << 1*, a w przeciwnym wypadku *K2 = (K1 << 1) ⊕ Rb*.

[6] Sekwencja CMAC może być w ogólności krótsza niż długość kluczy – patrz [56].

Sekwencja CMAC jest dołączana przez nadawcę do przesyłanej wiadomości. Jej weryfikacja polega na ponownym wyznaczeniu tego ciągu przez odbiorcę z wykorzystaniem tajnego klucza *K* i porównaniem wyniku z sekwencją zawartą w odebranej wiadomości. Pozytywna weryfikacja oznacza, że wiadomość jest integralna i autentyczna, a w ogólności oznacza, że sekwencja CMAC zawarta w odebranej wiadomości została wyznaczona na podstawie klucza *K* oraz tej właśnie wiadomości.

Warto w tym miejscu dodać, że ze względu na duże bezpieczeństwo zaprezentowanego tu algorytmu [44, 81] oraz niewielki rozmiar wyjściowej sekwencji CMAC rozwiązanie to, w formie zmodyfikowanej, wykorzystane zostanie dla realizacja mechanizmu uwierzytelniania i kontroli integralności danych przesyłanych pomiędzy terminalami użytkowników w nowym kryptosystemie dla łącza HF.

W rozdziale tym zaprezentowano podstawy teoretyczne związane z ogólnie rozumianą kryptografią. Informacje tu zawarte, wykorzystane zostały dla realizacji koncepcji systemu bezpiecznej transmisji danych w paśmie HF. Projekt, wspomnianego kryptosystemu, przedstawiono w rozdziale trzecim niniejszej rozprawy doktorskiej. Rozdział drugi z kolei poświęcono zagadnieniu istniejącej standaryzacji modemów krótkofalowych oraz analizie możliwości implementacji zaproponowanego w tej pracy rozwiązania w istniejących już urządzeniach.

Rozdział II

Modemy krótkofalowe a możliwość bezpiecznej transmisji danych w łączu HF

W niniejszej pracy podjęto między innymi próbę realizacji koncepcji kryptosystemu dla potrzeb transmisji danych w łączu krótkofalowym, który może zostać wykorzystany w modemach HF opartych o technologię SDR [2, 53, 88] – na przykład rozwiązania przedstawione w [10 – 12, 38, 85 – 87], ale i w klasycznych (sprzętowych) urządzeniach tego typu – jak choćby [65, 66, 72, 79].

Rozdział ten jest zatem poświecony między innymi opisowi warstwy fizycznej modemów krótkofalowych, ze szczególnym uwzględnieniem standardów wojskowych [51, 52, 83], będących wyznacznikiem dla tego typu urządzeń na rynku telekomunikacyjnym.

Wspomnieć należy tu tylko, że autor ogranicza się do specyfikacji modemów krótkofalowych, których szybkości transmisji są z zakresu od 3200 do 12800 bit/s przy wykorzystaniu pojedynczego kanału HF o szerokości 3 kHz.

Drugą część niniejszego rozdziału stanowi analiza możliwości implementacji zaproponowanego w rozdziale następnym kryptosystemu w modemach zgodnych ze wspomnianymi już standardami.

2.1. Charakterystyka warstwy fizycznej modemów krótkofalowych

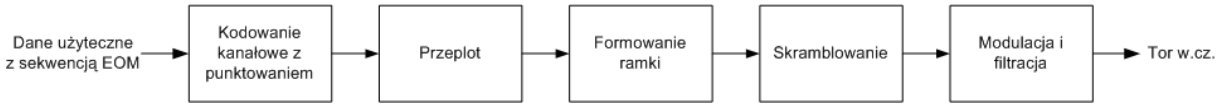
Modemy tu prezentowane przeznaczone są do pracy w kanale radiowym HF z pojedynczą częstotliwością nośną. Umożliwiają one transmisję danych z przepływnościami: 3200, 4800, 6400, 8000 i 9600 bit/s. W trybie bez kodowania kanałowego modem może osiągać przepływność 12800 bit/s, nie jest to jednak zalecany typ transmisji. Ze względu na wykorzystanie stosunkowo wysokich (jak na pasmo 3 kHz) przepływności informacyjnych, stosuje się tu bardziej złożone techniki modulacyjne i dłuższe bloki danych niż w przypadku transmisji o przepływnościach do 2400 bit/s [52].

W omawianej podgrupie modemów krótkofalowych dostępnych jest sześć różnych długości przeplotu w zakresie od 0,12 s do 8,64 s oraz jeden rodzaj kodowania splotowego o sprawności ½ i stałej ograniczającej równej 7 (liczba

komórek rejestru przesuwnego, z którego zbudowany jest koder). Kodowanie w powiązaniu z operacją punktowania umożliwia tu jednak uzyskanie sprawności ¾.

Informacje o wybranej przepływności informacyjnej i parametrach przeplotu są przesyłane do modemu odbiorczego jako część preambuły synchronizującej a następnie okresowo ponawiane w preambule powtarzalnej (*reinserted*).

W kolejnych paragrafach przedstawione zostaną poszczególne bloki toru nadawczego (modemu krótkofalowego zgodnego z [52]) przedstawionego na rysunku 2.1.

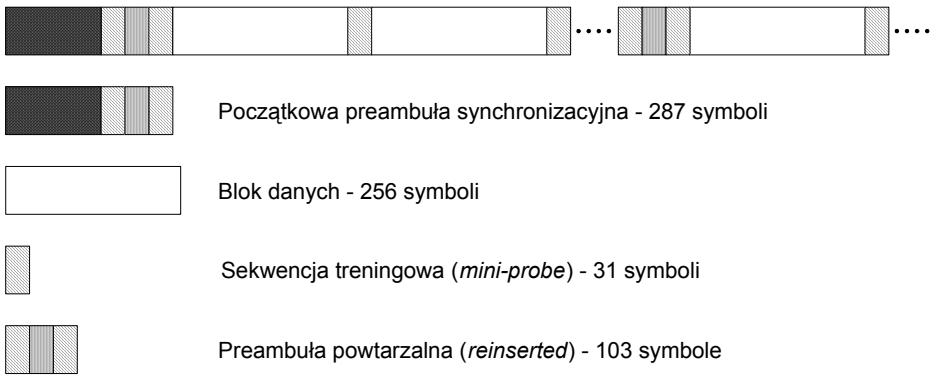


Rys. 2.1. Uproszczony schemat blokowy toru nadawczego (w paśmie podstawowym) modemu krótkofalowego [52] dla przepływności powyżej 2400 bit/s w pojedynczym kanale o szerokości 3 kHz.

2.1.1. Struktura ramek

Struktura ramek, wykorzystywana przez modem, została przedstawiona na rys. 2.2. Rozpoczyna się ona od preambuły synchronizującej o długości 287 symboli, po której następują 72 bloki danych o długości 256 symboli, transmitowane pomiędzy dwoma znanymi sekwencjami treningowymi (*mini-probe*) o długości 31 symboli. Blok danych z występującym po nim ciągiem treningowym (mini preambułą) zwany jest ramką.

Każda 72-ga sekwencja mini preambuły zastępowana jest przez preambułę powtarzalną, określaną jako *reinserted*, której zadaniem jest ułatwienie właściwego odbioru danych (między innymi, umożliwienie kompensacji przesunięcia Dopplera oraz zapewnienie synchronizacji).



Rys. 2.2. Struktura ramek dla modemów o przepływnościach powyżej 2400 bit/s.

Warto w tym miejscu nadmienić, iż ze względów oczywistych żadna z preambuł nie podlega procesom: kodowania kanałowego, punktowania, przeplotu czy

skramblowania. Poszczególne symbole zawarte we wspomnianych sekwencjach są już ostatecznymi symbolami modulacyjnymi 8PSK – tylko ta modulacja wykorzystywana jest w przypadku preambuł.

2.1.1.1. Preambuła synchronizująca

Preambuła synchronizująca składa się z dwóch części. Część pierwsza zawiera N bloków, składających się z 184 symboli modulacyjnych 8PSK (niezależnie od tego, jaki rodzaj modulacji stosuje modem podczas transmisji danych), wykorzystywanych wyłącznie przez układ automatycznej regulacji poziomu wzmocnienia w torze odbiorczym. Są one zespolonym sprzężeniem pierwszych 184 symboli drugiej części preambuły. Wartość N może przyjmować wartość całkowitą z zakresu od 0 do 7 (N=0 oznacza, że pierwsza część preambuły nie jest nadawana).

Część druga składa się z 287 symboli (tab. 2.1), z których pierwsze 184 są przeznaczone wyłącznie do synchronizacji i kompensacji przesunięcia Dopplera, natomiast pozostałe 103 symbole to tzw. preambuła powtarzalna. Niesie ona informacje o przepływności i długości przeplotu. Symbole D_i (i=0,1 lub 2), sumowane modulo 8 z ciągami Barkera (0, 4, 0, 4, 0, 0, 4, 4, 0, 0, 0, 0, 0), ustawiane są zgodnie z tab. 2.2. W tej części preambuły zawarto również dwie sekwencje treningowe.

Tab. 2.1. Preambuła synchronizacyjna.

1, 5, 1, 3, 6, 1, 3, 1, 1, 6, 3, 7, 7, 3, 5, 4, 3, 6, 6, 4, 5, 4, 0, 2, 2, 2, 6, 0, 7, 5, 7, 4, 0, 7, 5, 7, 1, 6, 1, 0, 5, 2, 2, 6, 2, 3, 6, 0, 0, 5, 1, 4, 2, 2, 2, 3, 4, 0, 6, 2, 7, 4, 3, 3, 7, 2, 0, 2, 6, 4, 4, 1, 7, 6, 2, 0, 6, 2, 3, 6, 7, 4, 3, 6, 1, 3, 7, 4, 6, 5, 7, 2, 0, 1, 1, 1, 4, 4, 0, 0, 5, 7, 7, 4, 7, 3, 5, 4, 1, 6, 5, 6, 6, 4, 6, 3, 4, 3, 0, 7, 1, 3, 4, 7, 0, 1, 4, 3, 3, 3, 5, 1, 1, 1, 4, 6, 1, 0, 6, 0, 1, 3, 1, 4, 1, 7, 7, 6, 3, 0, 0, 7, 2, 7, 2, 0, 2, 6, 1, 1, 1, 2, 7, 7, 5, 3, 3, 6, 0, 5, 3, 3, 1, 0, 7, 1, 1, 0, 3, 0, 4, 0, 7, 3,
0, 0, 0, 0, 0, 2, 4, 6, 0, 4, 0, 4, 0, 6, 4, 2, 0, 0, 0, 0, 0, 2, 4, 6, 0, 4, 0, 4, 0, 6, 4, 2, (D_0, D_0, D_0, D_0, D_0, D_0, D_0, D_0, D_0, D_0, D_0, D_0 + 0, 4, 0, 4, 0, 0, 4, 4, 0, 0, 0, 0, 0) mod 8 (D_1, D_1, D_1, D_1, D_1, D_1, D_1, D_1, D_1, D_1, D_1, D_1 + 0, 4, 0, 4, 0, 0, 4, 4, 0, 0, 0, 0, 0) mod 8 (D_2, D_2, D_2, D_2, D_2, D_2, D_2, D_2, D_2, D_2, D_2, D_2 + 0, 4, 0, 4, 0, 0, 4, 4, 0, 0, 0, 0, 0) mod 8 6, 4, 4, 4, 4, 4, 6, 0, 2, 4, 0, 4, 0, 4, 2, 0, 6, 4, 4, 4, 4, 4, 6, 0, 2, 4, 0, 4, 0, 4, 2, 0

Odwzorowanie zastosowane przy tworzeniu tab. 2.2 wykorzystuje po 3 bity dla zdefiniowania przepływności i długości przeplotu (jako liczbę ramek danych), zgodnie z tabl. 2.3. Przepływność określają trzy starsze bity (MSB) wybrane z trzech duobitów, trzy bity młodsze (LSB) tych duobitów definiują długość przeplotu. Wspomniane tu duobity zdefiniowano w tab. 2.4.

Tab. 2.2. Wartości symboli D0, D1, D2 w funkcji przepływności informacyjnej i długości przeplotu.

Przepływność informacyjna [bit/s]	**Długość przeplotu (liczba ramek)**					
	1	**3**	**9**	**18**	**36**	**72**
3200	0,0,4	0,2,6	0,2,4	2,0,6	2,0,4	2,2,6
4800	0,6,2	0,4,0	0,4,2	2,6,0	2,6,2	2,4,0
6400	0,6,4	0,4,6	0,4,4	2,6,6	2,6,4	2,4,6
8000	6,0,2	6,2,0	6,2,2	4,0,0	4,0,2	4,2,0
9600	6,0,4	6,2,6	6,2,4	4,0,6	4,0,4	4,2,6
12800	6,6,2*)	Rezerwa	Rezerwa	Rezerwa	Rezerwa	Rezerwa
*) Przy przepływności 12800 bit/s długość przeplotu odpowiadającą 1 ramce należy interpretować jako brak przeplotu.						

Faza ciągu Barkera jest określana na podstawie wartości duobitów uzyskanych w wyniku transkodowania symboli D_0, D_1, D_2, zgodnie z tab. 2.2. W efekcie takiego transkodowania 3 bity definiujące długość przeplotu są umieszczane w kwadraturze w stosunku do 3 bitów określających przepływność informacyjną.

Tab. 2.3. Sekwencje bitów definiujące określone przepływności i długości przeplotu.

Przepływność informacyjna [bit/s]	**Odwzorowanie 3-bitowe**
Rezerwa	000
3200	001
4800	010
6400	011
8000	100
9600	101
12800	110
Rezerwa	111

Długość przeplotu	**Odwzorowanie 3-bitowe**	**Nazwa**
Nielegal.	000	
1 ramka	001	Ultra krótki (US)
3 ramki	010	Bardzo krótki (VS)
9 ramek	011	Krótki (S)
18 ramek	100	Średni (M)
36 ramek	101	Długi (L)
72 ramki	110	Bardzo długi (VL)
Nielegal.	111	

Przykład

Zgodnie z tab. 2.2, przepływności 3200 bit/s i długości przeplotu 3 ramki odpowiada sekwencja symboli 0, 2, 6. Po przekształceniu symboli na duobity (tab. 2.4) sekwencja ta odpowiada trzem parom bitów 00, 01, 10. Bity starsze (MSB) tworzą ciąg 0, 0, 1; natomiast bity młodsze (LSB) – 0, 1, 0. Wartości te pozwalają określić na podstawie tab. 2.3 przepływność informacyjną i długość przeplotu.

Tabl. 2.4. Przekształcanie symboli na duobity.

Symbol	**Duobit**
0	00
2	01
4	11
6	10

2.1.1.2. Preambuła powtarzalna (*reinserted*)

Preambuła powtarzalna (tab. 2.5) jest identyczna z końcowymi stu trzema symbolami preambuły synchronizującej. Symbole te są zatem wspólne dla obu tych preambuł. Wartości D_0, D_1 i D_2 mają zatem takie samo znaczenie jak w przypadku preambuły synchronizacyjnej (tab. 2.2).

Tab. 2.5. Preambuła powtarzalna (*reinserted*).

0, 0, 0, 0, 0, 2, 4, 6, 0, 4, 0, 4, 0, 6, 4, 2, 0, 0, 0, 0, 0, 2, 4, 6, 0, 4, 0, 4, 0, 6, 4, 2, (D_0, D_0, D_0, D_0, D_0, D_0, D_0, D_0, D_0, D_0, D_0, D_0 + 0, 4, 0, 4, 0, 0, 4, 4, 0, 0, 0, 0, 0) Modulo 8 (D_1, D_1, D_1, D_1, D_1, D_1, D_1, D_1, D_1, D_1, D_1, D_1 + 0, 4, 0, 4, 0, 0, 4, 4, 0, 0, 0, 0, 0) Modulo 8 (D_2, D_2, D_2, D_2, D_2, D_2, D_2, D_2, D_2, D_2, D_2, D_2 + 0, 4, 0, 4, 0, 0, 4, 4, 0, 0, 0, 0, 0) Modulo 8 6, 4, 4, 4, 4, 4, 6, 0, 2, 4, 0, 4, 0, 4, 2, 0, 6, 4, 4, 4, 4, 4, 6, 0, 2, 4, 0, 4, 0, 4, 2, 0

2.1.1.3. Sekwencja treningowa (*mini-probe*)

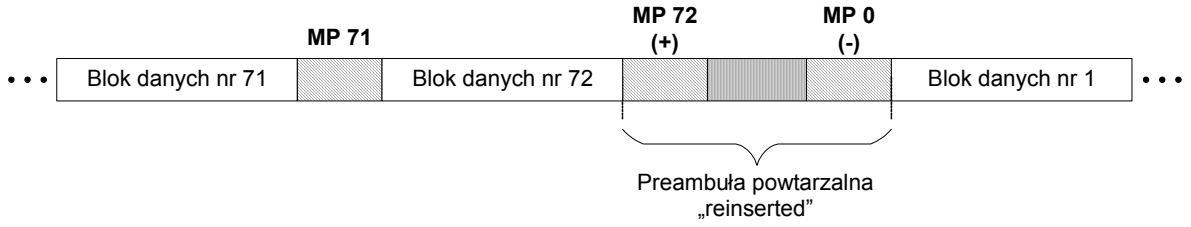
Mini preambuła o długości 31 symboli dodawana jest po każdym 256-cio symbolowym bloku danych. Oparta jest ona na odwzorowaniu symbolowym 8PSK – tak jak i preambuła synchronizująca czy *reinserted*. Wyróżniamy dwa typy ciągów *mini-probe* oznaczanych przez „+” i „-”. Sekwencja „+” przedstawia się następująco (cyfry odpowiadają numerom symboli modulacji 8PSK):

0, 0, 0, 0, 0, 2, 4, 6, 0, 4, 0, 4, 0, 6, 4, 2, 0, 0, 0, 0, 0, 2, 4, 6, 0, 4, 0, 4, 0, 6, 4.

Jej wersją odwróconą w fazie o 180 stopni jest sekwencja oznaczona przez „-”:

4, 4, 4, 4, 4, 6, 0, 2, 4, 0, 4, 0, 4, 2, 0, 6, 4, 4, 4, 4, 4, 6, 0, 2, 4, 0, 4, 0, 4, 2, 0.

W ramce występują 73 sekwencje *mini-probe* dla 72 bloków danych. Sekwencję nr 0 stanowi ostatnie 31 symboli preambuły powtarzalnej (*reinserted*) lub synchronizującej. Sekwencja nr 1 następuje po pierwszym bloku danych. Sekwencja nr 72 następuje po 72 bloku danych i stanowi pierwsze 31 symboli kolejnej preambuły *reinserted*. Sekwencje *mini-probe* o numerach 0 i 72 zostały zdefiniowane w preambule *reinserted* odpowiednio jako: ‘-‘ i ‘+’. Kolejność występowania sekwencji *mini-probe* (oznaczonych jako MP n) w ramce została pokazana na rys. 2.3.



Rys. 2.3. Kolejność występowania sekwencji „mini-probe” w ramce.

Informacje o przepływności informacyjnej i długości przeplotu, zakodowane w preambułach synchronizacyjnej i *reinserted*, są przesyłane również przy wykorzystaniu sekwencji *mini-probe* (MP). Sekwencje MP o numerach od 1 do 72 zostały zgrupowane w 4 zestawach: 1 do 18; 19 do 36; 37 do 54 oraz 55 do 72. Są to zestawy po 18 bloków danych z mini preambułami (po 18 ramek). Liczba 18 jest wielokrotnością, bądź podwielokrotnością liczby bloków danych podlegających przeplotowi. Blok danych (256 symboli), który następuje bezpośrednio po każdej 18-tej MP lub po preambule synchronizującej (lub *reinserted*) jest również pierwszym elementem bloku przeplotu dla długości 1, 3, 9 oraz 18 ramek. 36-ramkowy blok przeplotu rozpoczyna się po preambule (synchronizującej lub reinserted), a kolejny po drugim zestawie sekwencji treningowych. Po czwartym zestawie z kolei następuje preambuła powtarzalna (*reinserted*).

Wykorzystując oznaczenia sekwencji *mini-probe* ('-' i '+') można zakodować informację o przepływności i długości przeplotu oraz numerze aktualnego zestawu. Jeden zestaw 18-tu sekwencji *mini-probe* ma następującą postać: - - - - - - - + S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 +.

Początkowa sekwencja „- - - - - - - +" jednoznacznie określa początek odczytu dziewięciu wartości S_i. Wartości oznaczone S_0 do S_5 przenoszą informację o przepływności i parametrach przeplotu (tab. 2.6), a wartości S_6 do S_8 o aktualnym numerze zestawu (tab. 2.7). Wartości S_0, S_1, S_2 odpowiadają wartościom przepływności, a S_3, S_4, S_5 długościom przeplotu.

Tab. 2.6. Wartości S_0 S_1,S_2, S_3, S_4, S_5 w funkcji przepływności i długości przeplotu.

Przepływność informacyjna [bit/s]	**Długość przeplotu (liczba ramek)**					
	1	**3**	**9**	**18**	**36**	**72**
3200	+ + - + + -	+ + - + - +	+ + - + - -	+ + - - + +	+ + - - + -	+ + - - - +
4800	+ - + + + -	+ - + + - +	+ - + + - -	+ - + - + +	+ - + - + -	+ - + - - +
6400	+ - - + + -	+ - - + - +	+ - - + - -	+ - - - + +	+ - - - + -	+ - - - - +
8000	- + + + + -	- + + + - +	- + + + - -	- + + - + +	- + + - + -	- + + - - +
9600	- + - + + -	- + - + - +	- + - + - -	- + - - + +	- + - - + -	- + - - - +
12800	- - + + + -					

W tabeli 2.6 oraz 2.7 „-" oznacza binarne „1", natomiast „+" binarne „0". Zgodnie więc z tą regułą np. drugi zestaw ciągów treningowych zakodowano jako „+ - +".

Tabl. 2.7. Wartości S6, S7, S8 w zależności od numeru zestawu mini-probe.

Zestaw „mini probe"			
1-18	19-36	37-54	55-72
+ + -	+ - +	+ - -	- + +

Cała sekwencja *mini-probe* ma zatem postać następującą:

[rp] - - - - - - - + S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 + - - - - - - - + S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8+- - - - - - - + S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8+ - - - - - - - + S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 [rp],

gdzie [rp] reprezentuje 103 symbole preambuły *reinserted*.

2.1.2. Kodowanie kanałowe i przeplot

Dane wejściowe poddawane są procesowi kodowania kanałowego z wykorzystaniem kodera splotowego. Rozmiar bloku bitów wejściowych musi odpowiadać blokowi przeplotu. Tabela 2.8 prezentuje liczbę bitów w bloku danych wejściowych kodera kanałowego w funkcji przepływności informacyjnej i długości przeplotu. Określenia „blok danych wejściowych" nie należy tu mylić z 256-symbolowym blokiem danych, który jest częścią ramki danych. Bity z bloku danych wejściowych są poddawane procesowi kodowania i przeplotu przed wprowadzeniem do odpowiedniej liczby (oznaczającej długość przeplotu) 256-symbolowych bloków danych w ramce. W tabeli 2.8 kolorem zielonym zaznaczono długości bloku danych wejściowych najbardziej optymalne z perspektywy, zaproponowanego w rozdziale trzecim niniejszej pracy, kryptosystemu dla transmisji danych w łączu krótkofalowym. Kryterium oraz sposób wyboru tych właśnie wartości omówiony zostanie w paragrafie 4.2.2 niniejszej pracy.

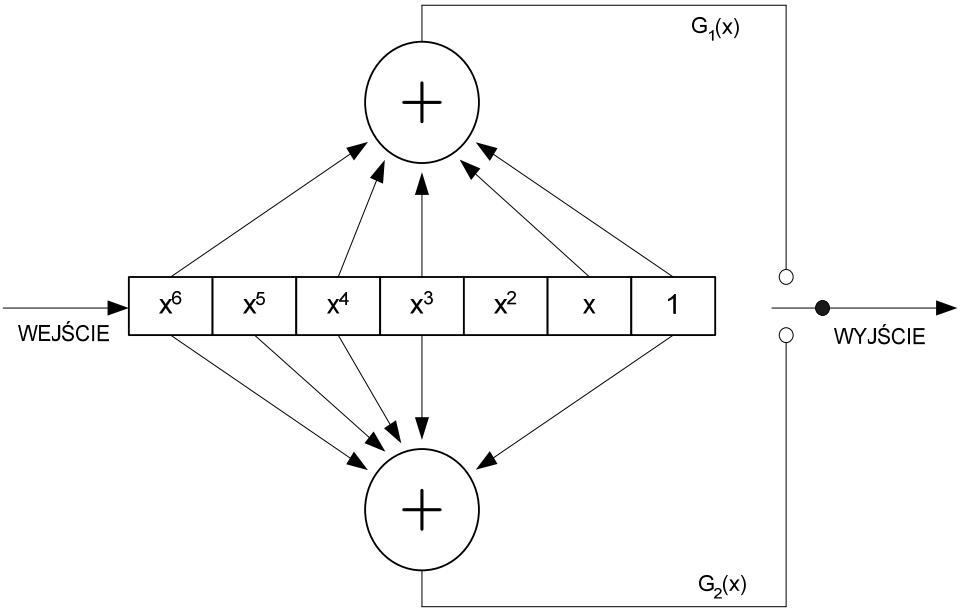
Tab. 2.8. Liczba bitów w bloku danych wejściowych.

Przepływność informacyjna [bit/s]	**Długość przeplotu (liczba ramek)**					
	1	**3**	**9**	**18**	**36**	**72**
3200	384	1152	3456	6912	13824	27648
4800	576	1728	5184	10368	20736	41472
6400	768	2304	6912	13824	27648	55296
8000	960	2880	8640	17280	34560	69120
9600	1152	3456	10368	20736	41472	82944

Każdy zakodowany blok danych jest poddawany operacji przeplotu w ramach bloku przeplotu o takich samych rozmiarach. Granice tych bloków powinny być ustawiane w taki sposób, aby początek pierwszej ramki danych, następującej po każdej preambule powtarzalnej (*reinserted*), pokrywał się z granicą zarówno bloku przeplotu jak i bloku danych wejściowych kodera. Tak więc, w przypadku przeplotu o długości 3 ramek, pierwsze trzy bloki danych następujące po preambule *reinserted* powinny zawierać wszystkie zakodowane bity jednego bloku danych wejściowych.

2.1.2.1. Kodowanie kanałowe

Modem wykorzystuje kodowanie splotowe o sprawności ½ i stałej ograniczającej 7. W połączeniu z operacją punktowania uzyskuje się sprawność ¾. Schemat blokowy kodera przedstawiono na rysunku 2.4.



Rys. 2.4. Schemat blokowy kodera kanałowego.

Wykorzystywane tu wielomiany generujące są zatem następujące:

$$G_1(x) = x^6 + x^4 + x^3 + x + 1, \quad (2.1)$$

$$G_2(x) = x^6 + x^5 + x^4 + x^3 + 1. \quad (2.2)$$

Każdemu bitowi na wejściu kodera odpowiadają dwa bity wyjściowe, przy czym bit pochodzący z górnej gałęzi kodera [$G_1(x)$] powinien występować jako pierwszy.

W koderze została zastosowana technika określana angielską nazwą *full-tail-biting*. Polega ona na tym, że na początku kodowania każdego bloku, rejestr przesuwny kodera zostaje napełniony pierwszymi sześcioma bitami danych wejściowych, bez wyznaczania w tym czasie bitów wyjściowych. Te 6 bitów musi być zapamiętane, ponieważ będą wykorzystane do „płukania” kodera. Pierwsze dwa bity wyjściowe kodera są wytwarzane po wprowadzeniu do rejestru siódmego bitu wejściowego. Są to dwa pierwsze bity sekwencji wynikowej. Po zakodowaniu ostatniego bitu wejściowego, następuje kodowanie sześciu pierwszych bitów przechowywanych w pamięci. Przed tą operacją rejestr przesuwny powinien być napełniony ostatnimi siedmioma bitami danych wejściowych. Sześć zapamiętanych bitów wejściowych wpisuje się do rejestru pojedynczo, począwszy od bitu

najwcześniejszego. W ten sposób proces kodowania jest kontynuowany i na wyjściu pojawiają się dwa bity zakodowane dla każdego z sześciu bitów wejściowych.

Ponieważ sprawność kodowania wynosi ½, blok bitów zakodowanych jest dokładnie dwa razy większy od bloku bitów wejściowych. Procedura „punktowania", poprzedzająca przekazanie zakodowanego wektora danych do układu przeplotu, ma na celu uzyskanie sprawności kodowania ¾.

Operacja punktowania polega na wykluczaniu z transmisji jednej trzeciej liczby bitów, poprzez zastosowanie maski 111001 nakładanej na ciąg bitów wyjściowych kodera splotowego. Sekwencja bitów, wygenerowana przez koder:

$G_1(k)$, $G_2(k)$, $G_1(k+1)$, $G_2(k+1)$, $G_1(k+2)$, $G_2(k+2)$ ....,

zostaje w wyniku punktowania, przekształcona na następującą:

$G_1(k)$, $G_2(k)$, $G_1(k+1)$, $G_2(k+2)$ .....

2.1.2.2. Operacja przeplotu

Operacja przeplotu wykonywana jest przy pomocy jednowymiarowej tablicy o rozmiarach równych liczbie bitów w bloku danych na wyjściu kodera (po „punktowaniu"). Rozmiary tej tablicy zależą zarówno od długości przeplotu jak i przepływności informacyjnej, co przedstawia tab. 2.9.

Tab. 2.91. Rozmiary tablicy przeplotu.

Przepływność informacyjna [bit/s]	**Długość przeplotu (liczba ramek)**					
	1	**3**	**9**	**18**	**36**	**72**
	Rozmiar tablicy przeplotu (bity)					
3200	512	1536	4608	9216	18432	36864
4800	768	2304	6912	13824	27648	55296
6400	1024	3072	9216	18432	36864	73728
8000	1280	3840	11520	23040	46080	92160
9600	1536	4608	13824	27648	55296	110592

Zakodowany blok bitów (po operacji punktowania) jest wprowadzany do tablicy przeplotu począwszy od pozycji 0. Pozycje dla kolejnych bitów są wyznaczane przez dodanie do numeru poprzedniej pozycji tzw. wskaźnika inkrementacji, którego wartości są określone na podstawie tab. 2.10. Oznacza to, że jeżeli pierwszy bit bloku oznaczymy *B(0)*, to pozycję zapisu bitu *n*-tego *B(n)* można wyznaczyć z następującej zależności:

$$B(n) = (n * \text{"wskaźnik inkrementacji"}) \text{ modulo (rozmiar tablicy w bitach).} \quad (2.3)$$

Przykładowo, dla przepływności 3200 bit/s i długości przeplotu 1 (rozmiar tablicy 512 bitów), pierwszych 8 pozycji zapisu przedstawia się następująco: 0, 97, 194, 291, 388, 485, 70 i 167.

Tab. 2.10. Wartości wskaźnika inkrementacji.

Przepływność informacyjna [bit/s]	**Długość przeplotu (liczba ramek)**					
	1	**3**	**9**	18	36	72
	Wskaźnik inkrementacji					
3200	97	229	805	1393	3281	6985
4800	145	361	1045	2089	5137	10273
6400	189	481	1393	3281	6985	11141
8000	201	601	1741	3481	8561	14441
9600	229	805	2089	5137	10273	17329

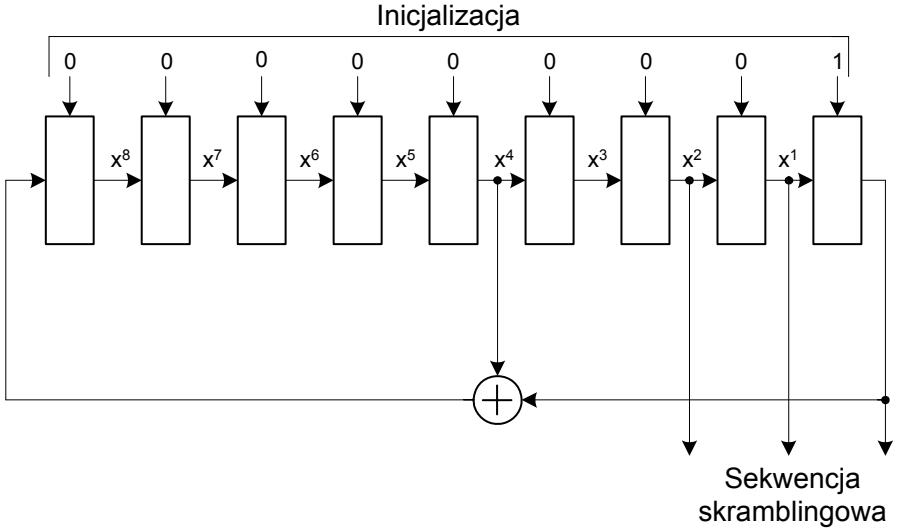
Wyprowadzanie bitów z tablicy przeplotu, dla wszystkich przepływności i długości przeplotu, odbywa się w sposób liniowy od początku (pozycja 0) do końca tablicy.

2.1.3. Skrambler

Procedura skramblingu w modemie jest uzależniona od rodzaju modulacji zastosowanej do transmisji danych o określonej przepływności informacyjnej (tab. 2.11). We wszystkich przypadkach wykorzystywany jest jednak ten sam generator sekwencji skramblingowej opisany wielomianem:

$$x^9 + x^4 + 1, \tag{2.4}$$

który inicjalizowany jest wartością 1 na początku każdej ramki danych (rys. 2.5).



Rys. 2.5. Generator sekwencji skramblingowej (przykład dla 8PSK).

Dla modulacji QPSK oraz 8PSK (3200 bit/s i 4800 bit/s) skramblowanie polega na sumowaniu modulo 8 wartości symbolu z wartością wytworzoną przez generator sekwencji skramblingowej na jego trzech młodszych bitach.

Przykładowo, jeżeli wartość wygenerowana przez rejestr przesuwny wynosi 010 a numer symbolu przed skramblingiem wynosił 6 to nadany zostanie symbol 0, ponieważ: (6+2) modulo 8 = 0.

W przypadku modulacji 16QAM realizowana jest operacja XOR 4-bitowego symbolu z czterema młodszymi bitami wygenerowanymi przez rejestr. Przykładowo, jeżeli wartość wygenerowana przez rejestr przesuwny wynosi 0101 a numer symbolu przed skramblingiem wynosił 3 (0011) to nadany zostanie symbol 6 (0110).

Odpowiednio dla modulacji 32QAM operacja dokonywana jest na 5-bitowych symbolach i pięciu młodszych bitach rejestru przesuwnego, a dla 64QAM, na 6-bitowych symbolach i sześciu młodszych bitach rejestru. Po każdej operacji skramblingu pojedynczego symbolu musi nastąpić przesunięcie zawartości rejestru: 3 razy dla 8PSK, 4 razy dla 16QAM, 5 razy dla 32QAM i 6 razy dla 64QAM. Pierwszy symbol danych w każdej ramce jest poddawany operacji skramblingu przy wykorzystaniu odpowiedniej liczby bitów sekwencji inicjalizującej: 00000001 (inicjalizacja po każdym 256-symbolowym bloku danych).

Długość sekwencji skramblingowej wynosi 511 bitów (2^9-1). Nie dokonuje się operacji skramblingu na preambułach.

2.1.4. Modulacja

W modemie powinna być stosowana jedna szybkość modulacji dla wszystkich symboli transmisyjnych, wynosząca 2400 symboli/s. W przypadku generowania zegara nadawczego przez modem, szybkość ta powinna być utrzymywana z dokładnością nie gorszą niż ±0,24 symboli/s (10 ppm). Symbole transmisyjne powinny modulować przebieg sinusoidalny (lub parę takich przebiegów pozostających względem siebie w kwadraturze, w przypadku QAM) o częstotliwości nominalnej 1800 Hz, generowanej z dokładnością 10 ppm (±0,018 Hz).

W przypadku symboli preambuł modem stosuje zawsze modulację 8PSK, zgodnie z tab. 2.12 i rys. 2.6. Symbole te przed procesem modulacji nie podlegają operacji skramblingu.

Rodzaj modulacji zastosowanej do transmisji symboli danych użytkownika, zależy od przepływności informacyjnej (tab. 2.11).

Tab. 2.11. Rodzaje modulacji obowiązujące przy poszczególnych przepływnościach informacyjnych.

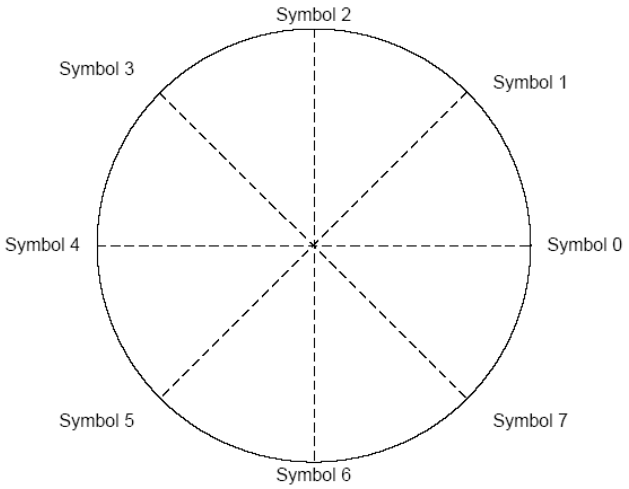
Przepływność informacyjna [bit/s]	**Rodzaj modulacji**
3200	QPSK
4800	8PSK
6400	16QAM
8000	32QAM
9600	64QAM
12800	64QAM

2.1.4.1. Modulacja PSK

Odwzorowanie symboli modulacji 8PSK zostało przedstawione w tab. 2.12 i na rys. 2.6. W przypadku przepływności 3200 bit/s (modulacja QPSK), dwa kolejne bity (duobity) danych użytkownika są przyporządkowywane symbolom modulacji 8PSK, zgodnie z tab. 2.4.

Tab. 2.12. Składowe synfazowa i kwadraturowa symboli modulacji 8 PSK.

Symbol	**Faza**	**Składowa synfazowa**	**Składowa kwadraturowa**
0	0	1,000000	0,000000
1	π/4	0,707107	0,707107
2	π/2	0,000000	1,000000
3	3π/4	-0,707107	0,707107
4	π	-1,000000	0,000000
5	5π/4	-0,707107	-0,707107
6	3π/2	0,000000	-1,000000
7	7π/4	0,707107	-0,707107



Rys. 2.6. Konstelacja symboli modulacji 8PSK.

Tab. 2.13 pokazuje przyporządkowanie kolejnych trzech bitów (tribitów) danych poszczególnym symbolom modulacji 8PSK, stosowane przy przepływności informacyjnej 4800 bit/s oraz dla wszystkich preambuł.

Tab. 2.13. Przyporządkowanie symboli 8PSK tribitom (4800 bit/s).

Tribit	Symbol
000	1
001	0
010	2
011	3
100	6
101	7
110	5
111	4

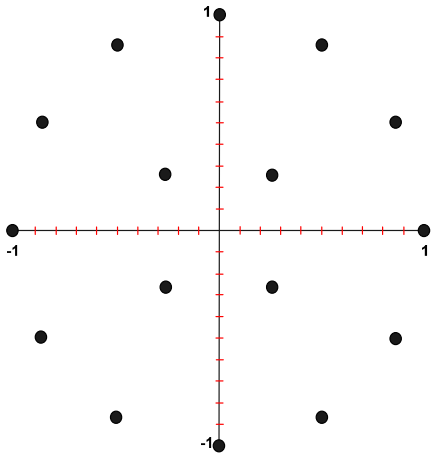
2.1.4.2. Modulacja QAM

Numery symboli modulacji QAM odpowiadają sekwencjom bitów danych. Sekwencja czterech (16QAM), pięciu (32QAM) lub sześciu (64QAM) kolejnych bitów mapowana jest bezpośrednio na symbol modulacji QAM. Przykład takiego odwzorowania przedstawia się następująco:

4 bity 0111 – symbol 7 (16QAM);

6 bitów 100011 – symbol 35 (64QAM).

Konstelacja modulacji 16QAM została pokazana na rys. 2.7, a w tab. 2.14 określone zostały wartości składowych synfazowej i kwadraturowej poszczególnych symboli tej modulacji.

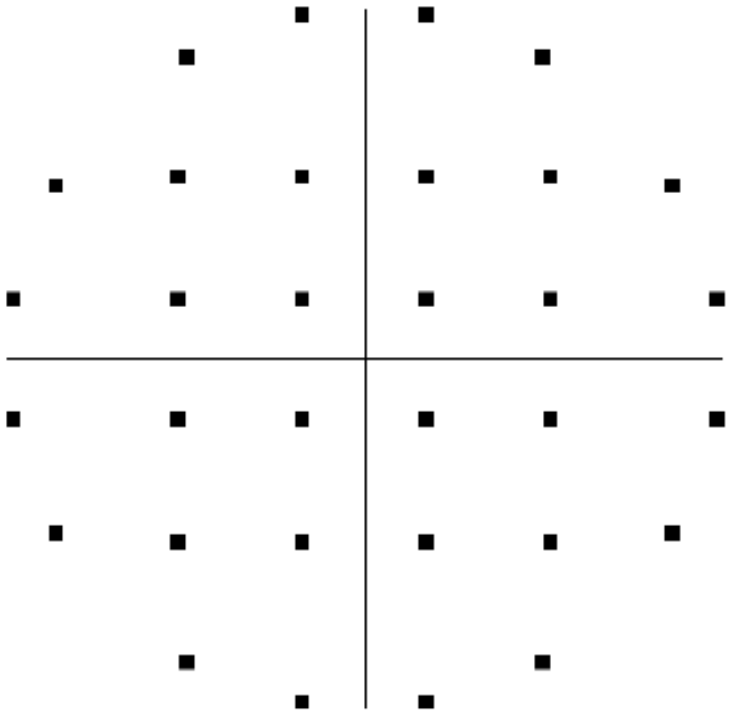


Rys. 2.7. Konstelacja symboli modulacji 16QAM.

Tab. 2.14. Składowe synfazowa i kwadraturowa symboli modulacji 16QAM.

Symbol	Składowa synfazowa	Składowa kwadraturowa
0	0,866025	0,500000
1	0,500000	0,866025
2	1,000000	0,000000
3	0,258819	0,258819
4	-0,500000	0,866025
5	0,000000	1,000000
6	-0,866025	0,500000
7	-0,258819	0,258819
8	0,500000	-0,866025
9	0,000000	-1,000000
10	0,866025	-0,500000
11	0,258819	-0,258819
12	-0,866025	-0,500000
13	-0,500000	-0,866025
14	-1,000000	0,000000
15	-0,258819	-0,258819

Konstelacja modulacji 32QAM została pokazana na rys. 2.8, a w tab. 2.15 zostały określone wartości składowych synfazowej i kwadraturowej poszczególnych symboli tej modulacji.

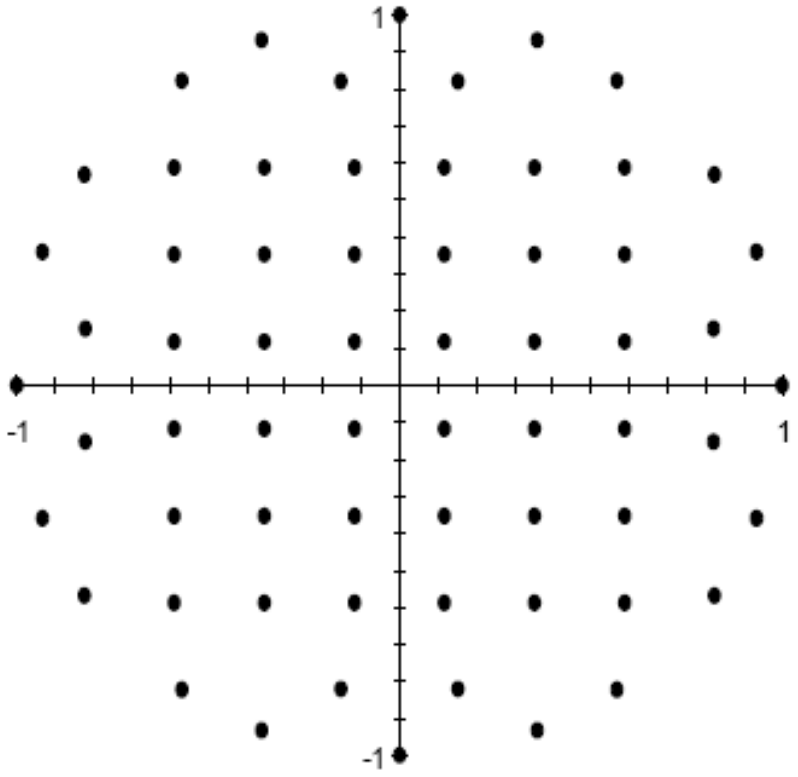


Rys. 2.8. Konstelacja symboli modulacji 32QAM.

Tab. 2.15. Składowe synfazowa i kwadraturowa symboli modulacji 32QAM.

Symbol	Składowa synfazowa	Składowa kwadraturowa	Symbol	Składowa synfazowa	Składowa kwadraturowa
0	0,866380	0,499386	16	0,866380	-0,499386
1	0,984849	0,173415	17	0,984849	-0,173415
2	0,499386	0,866380	18	0,499386	-0,866380
3	0,173415	0,984849	19	0,173415	-0,984849
4	0,520246	0,520246	20	0,520246	-0,520246
5	0,520246	0,173415	21	0,520246	-0,173415
6	0,173415	0,520246	22	0,173415	-0,520246
7	0,173415	0,173415	23	0,173415	-0,173415
8	-0,866380	0,499386	24	-0,866380	-0,499386
9	-0,984849	0,173415	25	-0,984849	-0,173415
10	-0,499386	0,866380	26	-0,499386	-0,866380
11	-0,173415	0,984849	27	-0,173415	-0,984849
12	-0,520246	0,520246	28	-0,520246	-0,520246
13	-0,520246	0,173415	29	-0,520246	-0,173415
14	-0,173415	0,520246	30	-0,173415	-0,520246
15	-0,173415	0,173415	31	-0,173415	-0,173415

Konstelacja modulacji 64QAM została pokazana na rys. 2.9, a w tab. 2.16 określone zostały wartości składowych synfazowej i kwadraturowej poszczególnych symboli tej modulacji.



Rys. 2.9. Konstelacja symboli modulacji 64QAM.

Tab. 2.16. Składowe synfazowa i kwadraturowa symboli modulacji 64QAM.

Symbol	Składowa synfazowa	Składowa kwadraturowa	Symbol	Składowa synfazowa	Składowa kwadraturowa
0	1,000000	0,000000	32	0,000000	1,000000
1	0,822878	0,568218	33	-0,822878	0,568218
2	0,821137	0,152996	34	-0,821137	0,152996
3	0,932897	0,360142	35	-0,932897	0,360142
4	0,000000	-1,000000	36	-1,000000	0,000000
5	0,822878	-0,568218	37	-0,822878	-0,568218
6	0,821137	-0,152996	38	-0,821137	-0,152996
7	0,932897	-0,360142	39	-0,932897	-0,360142
8	0,568218	0,822878	40	-0,568218	0,822878
9	0,588429	0,588429	41	-0,588429	0,588429
10	0,588429	0,117686	42	-0,588429	0,117686
11	0,588429	0,353057	43	-0,588429	0,353057
12	0,568218	-0,822878	44	-0,568218	-0,822878
13	0,588429	-0,588429	45	-0,588429	-0,588429
14	0,588429	-0,117686	46	-0,588429	-0,117686
15	0,588429	-0,353057	47	-0,588429	-0,353057
16	0,152996	0,821137	48	-0,152996	0,821137
17	0,117686	0,588429	49	-0,117686	0,588429
18	0,117686	0,117686	50	-0,117686	0,117686
19	0,117686	0,353057	51	-0,117686	0,353057
20	0,152996	-0,821137	52	-0,152996	-0,821137
21	0,117686	-0,588429	53	-0,117686	-0,588429
22	0,117686	-0,117686	54	-0,117686	-0,117686
23	0,117686	-0,353057	55	-0,117686	-0,353057
24	0,360142	0,932897	56	-0,360142	0,932897
25	0,353057	0,588429	57	-0,353057	0,588429
26	0,353057	0,117686	58	-0,353057	0,117686
27	0,353057	0,353057	59	-0,353057	0,353057
28	0,360142	-0,932897	60	-0,360142	-0,932897
29	0,353057	-0,588429	61	-0,353057	-0,588429
30	0,353057	-0,117686	62	-0,353057	-0,117686
31	0,353057	-0,353057	63	-0,353057	-0,353057

2.1.5. Tor odbiorczy

Standard [52] nie definiuje części odbiorczej modemu. Zakłada się tu wykorzystanie standardowych rozwiązań stosowanych w odbiornikach telekomunikacyjnych takich jak:

- Synchronizacja i korekta charakterystyki kanału w oparciu o znane sekwencje preambuł;

- Zastosowanie dekodera Viterbi'ego (najlepiej miękko-decyzyjnego) dla realizacji dekodowania splotowego;
- Próby detekcji sekwencji EOM [52] – realizowane w sposób ciągły.

Szczegółowe rozwiązania wykorzystywane w torze odbiorczym modemu HF zależą jednak od konkretnej implementacji i nie będą tu prezentowane.

W niniejszym paragrafie zaprezentowano wszystkie bloki funkcjonalne warstwy fizycznej toru nadawczego standardowych modemów krótkofalowych, ponieważ elementy te nie ulegną zmianie w przypadku zastosowania zaproponowanej w rozdziale następnym koncepcji kryptosystemu. Ponadto pozwolą one na poprawne jego funkcjonowanie poprzez zapewnienie mechanizmu transmisji zabezpieczonych danych w łączu krótkofalowym.

Paragraf następny z kolei ma na celu przedstawienie sposobu implementacji zaproponowanego rozwiązania w torze nadawczym i odbiorczym modemów zgodnych ze standardem [52].

2.2. Możliwość implementacji kryptosystemu w modemach krótkofalowych

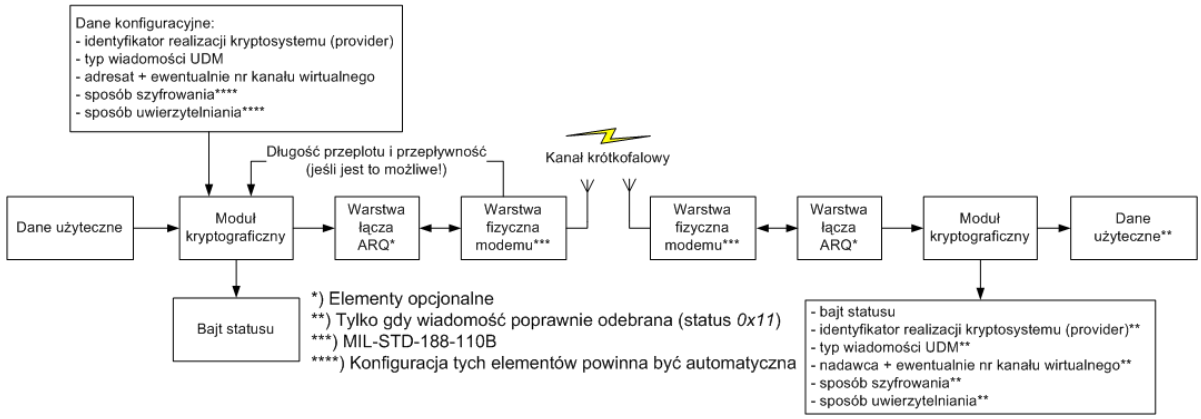
Twórcy standardu [52] przewidzieli możliwość wykorzystania rozwiązań kryptograficznych w modemach krótkofalowych, choć nie zdefiniowali sposobu ich realizacji. Specyfikacja ta nie zawiera bowiem wytycznych odnośnie bloku modułu kryptograficznego, a jedynie pozwala na określenie jego lokalizacji w torze nadawczym, a przez to również w torze odbiorczym urządzenia.

Na rysunku 2.10 przedstawiono zatem sposób implementacji (zgodny ze standardem [52]), zaproponowanego w rozdziale trzecim niniejszej pracy rozwiązania, w torze transmisyjnym klasycznego modemu krótkofalowego.

Na podstawie poniższego rysunku łatwo stwierdzić, że kryptosystem jest zupełnie niezależny od warstwy fizycznej modemu. Zabezpieczone dane wyjściowe z modułu kryptograficznego podawane są bezpośrednio na pierwszy blok warstwy fizycznej (patrz rysunek 2.1) lub opcjonalnie wykorzystywany jest najpierw protokół warstwy łącza (ARQ). Jest on właściwie dowolnego typu, ale najczęściej wykorzystuje się tu standard [82].

Jego zadaniem jest między innymi automatyczne retransmitowanie danych, które nie zostały poprawnie odebrane przez odbiorcę oraz dobieranie parametrów

transmisji (przepływność i długość przeplotu – patrz paragraf 2.1) w zależności od liczby retransmisji w stosunku do całkowitej ilości danych przesłanych.



Rys. 2.10. Sposób implementacji systemu bezpiecznej transmisji danych w łączu HF w modemie opartym o standard MIL-STD-188-110B.

Standard modemów HF [52] dopuszcza opcjonalnie wykorzystanie modułu kryptograficznego po bloku ARQ w torze nadawczym. Autor jednak zdecydował się na rozwiązanie przedstawione na rysunku 2.10, ze względu na:

- Moduł kryptograficzny w torze odbiorczym uzyska dane, które przeszły już pozytywnie weryfikację ich poprawności – jeśli bowiem protokół ARQ stwierdzi wystąpienie błędów niekorygowalnych w odebranych danych, to zażąda ponownej ich transmisji przez nadawcę (jest to proces automatyczny). Moduł będzie zatem wykorzystywany tylko, gdy dane będą uznane za poprawne przez ARQ – dane z błędami i tak nie przejdą pomyślnie procesu kontroli ich integralności (patrz rozdział kolejny). Implementacja zatem modułu kryptograficznego przed blokiem ARQ w odbiorniku spowodowałaby często bezcelowe wykorzystywanie modułu.
- Moduł kryptograficzny nie wpłynie w żaden sposób na funkcjonowanie protokołu ARQ w nadajniku – na wejście bloku ARQ podawane będą bowiem dane, które potraktowane zostaną tak samo jak w przypadku braku zabezpieczenia kryptograficznego.
- Brak konieczności ingerowania w samo urządzenie modemowe w celu implementacji zaproponowanego kryptosystemu – wystarczy zewnętrzna aplikacja, pełniąca rolę modułu kryptograficznego, uruchomiona na komputerze podłączonym do modemu.

Należy tu ponadto nadmienić, że powyższe rozważania staną się w dużej mierze nieistotne w momencie, gdy opcjonalny blok ARQ nie będzie wykorzystywany. W

takiej sytuacji dane uzyskane na wyjściu modułu zostaną tylko rozszerzone o sekwencję EOM (patrz rysunek 2.1) i następnie podane na wejście warstwy fizycznej modemu.

Niezależnie od faktu wykorzystania protokołu ARQ ważne jest to, aby umożliwić konfigurację (przez użytkownika modemu) samego modułu kryptograficznego w celu możliwości jego poprawnego wykorzystania dla realizacji bezpiecznej transmisji danych. Nadawca wiadomości musi ustalić (patrz rozdział następny):

- Identyfikator wykorzystywanego provider'a (identyfikator realizacji kryptosystemu) – o ile moduł obsługuje więcej niż jeden;
- Typ wiadomości UDM;
- Adresata wiadomości – o ile nie wybrano wiadomości typu broadcast;
- Ewentualnie numer kanału wirtualnego – o ile wybrano wiadomość UDM w obrębie grupy w kanale wirtualnym;
- Opcjonalnie (nie zaleca się, ponieważ sam moduł może dobrać te parametry w sposób optymalny – maksymalnie bezpieczny i możliwy do realizacji):
    - Sposób szyfrowania;
    - Sposób uwierzytelniania.

Moduł Kryptograficzny po uzyskaniu danych konfiguracyjnych i danych użytkowych generuje kolejne wiadomości UDM dla jednej z optymalnych długości wiadomości ustalonych określonych w paragrafie 4.2.2 oraz tabeli 2.8 w niniejszej pracy.

Ponadto, jeśli istnieje możliwość przekazania do modułu parametrów transmisji (przepływność i długość przeplotu), możliwe staje się dobranie długości wiadomości w sposób najbardziej optymalny.

Wiadomości wygenerowane przez moduł kryptograficzny podawane są następnie na blok ARQ (o ile występuje) i dalsze przetwarzanie odbywa się już w sposób zgodny ze standardem [52].

W części odbiorczej zabezpieczone dane z wyjścia opcjonalnego bloku ARQ podawane są na wejście modułu, który realizując odpowiednie algorytmy (patrz rozdział czwarty) generuje zawsze bajt statusu, a w przypadku poprawnie zdekodowanej wiadomości zwraca na wyjściu również dane użyteczne oraz udostępnia dodatkowe informacje odnośnie odebranej wiadomości (patrz rysunek 2.10).

W tym miejscu warto zauważyć, że ze względu na stosowany znacznik czasu (patrz rozdział kolejny) w przesyłanych wiadomościach konieczny jest pewien poziom synchronizacji zegarów pomiędzy terminalami. Zgodność czasu powinna być na poziomie kilkunastu a maksymalnie kilkudziesięciu sekund (mniej niż minuta).

Stosowany znacznik czasu jest zawsze ważny przez jedną minutę i dzięki temu nawet w przypadku automatycznych retransmisji, zabezpieczonych już wiadomości, przez blok ARQ, do odbiorcy dotrzeć powinna nadal ważna wiadomość (jeśli poziom synchronizacji czasu pomiędzy terminalami jest wystarczający). Więcej szczegółów odnośnie powyższych rozważań znaleźć można w dalszej części niniejszej rozprawy doktorskiej.

Sama fizyczna implementacja kryptosystemu może zostać zrealizowana z wykorzystaniem modułu kryptograficznego wbudowanego w modem lub poprzez odpowiednią aplikację uruchamianą np. na komputerze PC, do którego podłączony jest modem krótkofalowy. O ile to pierwsze rozwiązanie może być dość trudne w realizacji, to w drugiej sytuacji praktyczna implementacja, zaproponowanej w rozdziale kolejnym koncepcji kryptosystemu, nie będzie wymagać ingerencji w samo urządzenie modemowe. Dane użytkownika w tej sytuacji będą przetworzone przez wirtualny moduł kryptograficzny, będący aplikacją programową, a następnie zostaną poddane procesom przedstawionym w [52], realizowanym przez istniejące i wykorzystywane obecnie modemy HF.

Warto w tym miejscu zauważyć, że przechowywane na komputerze PC tajne parametry i klucze systemowe muszą być oczywiście dodatkowo zabezpieczone przed niautoryzowanym do nich dostępem.

Podsumowując ten paragraf można stwierdzić, że implementacja kryptosystemu w istniejących urządzeniach nie będzie skomplikowana, ponieważ:

- Standard [52] przewiduje taką możliwość;
- Moduł kryptograficzny może zostać zrealizowany w postaci aplikacji komputerowej uruchomionej na komputerze PC.

W kolejnym rozdziale niniejszej pracy zawarto ogólną specyfikację zaproponowanej koncepcji systemu bezpiecznej transmisji danych w łączu krótkofalowym.

Rozdział III

Koncepcja nowego kryptosystemu dla potrzeb transmisji danych w łączu HF

Koncepcja systemu zakłada przede wszystkim realizację wszystkich mechanizmów kryptograficznych niezbędnych dla realizacji w pełni bezpiecznego systemu transmisji danych w łączu krótkofalowym. Opracowany kryptosystem, przedstawiony już częściowo w [7 – 9, 13], opiera się na istniejących algorytmach kryptograficznych takich jak: AES, RSA, CMAC i SHA-256 oraz ich modyfikacjach. Bezpieczeństwo zaproponowanego rozwiązania zależy zatem w znacznym stopniu od skuteczności i odporności na ataki kryptoanalityczne wspomnianych algorytmów. Przy projektowaniu niniejszego systemu założono więc, że szyfrowanie przy pomocy AES jest w pełni poufne, funkcja RSA (z kluczem 1024 bitowym) natomiast gwarantuje uwierzytelnienie a wraz z SHA-256 integralność przesyłanych danych. Założenie takie na dzień dzisiejszy jest w pełni uzasadnione (patrz rozdział pierwszy).

Przy projektowaniu tego typu systemu należy wziąć pod uwagę pewne dodatkowe elementy, które w znaczny sposób wpłyną na postać końcową proponowanego rozwiązania. Te dodatkowe składniki to przede wszystkim:

- Specyfika łączności krótkofalowej [39, 64];
- Istniejąca standaryzacja modemów HF (patrz rozdział drugi niniejszej pracy) oraz [52];
- Minimalne ograniczenie pasma użytecznego spowodowane wprowadzeniem systemu.

W kolejnych paragrafach niniejszego rozdziału przedstawiona zostanie podstawowy projekt zaproponowanego systemu z uwzględnieniem powyższych aspektów.

3.1. Informacje wstępne

W celu realizacji autorskiego rozwiązania systemu bezpiecznej transmisji danych w paśmie HF, zebrano i przeanalizowano dokumentację istniejących standardów związanych z kryptografią:

- Standardy symetrycznych szyfrów blokowych: DES [54], 3DES [54], AES [26];
- Asymetryczny algorytm szyfrowania z kluczem publicznym: RSA [61];
- Funkcje skrótu: SHA [24], MD4 [68], MD5 [69, 70];
- Algorytmy uwierzytelniające MAC [27, 56];
- Standard podpisu cyfrowego [17, 25, 59];
- I inne algorytmy i rozwiązania kryptograficzne [40, 42, 46, 48, 57, 84].

W dalszej kolejności przeanalizowano systemy bezpieczeństwa w wybranych znanych systemach łączności bezprzewodowej (IEEE 802.11 [36], Bluetooth [80], GSM/UMTS [1, 95, 43]). Szczególną uwagę zwrócono tu na odmiennie realizowane poszczególne mechanizmy kryptograficzne oraz sposoby zarządzania całym systemem bezpieczeństwa jak i kluczami szyfrującymi czy innymi parametrami systemowymi.

Na podstawie zdobytej wiedzy opracowano projekt systemu bezpiecznej transmisji danych w paśmie HF. Projekt ten uwzględnia i zakłada między innymi:

- Wybór oraz modyfikację odpowiednich algorytmów do realizacji poszczególnych mechanizmów kryptograficznych;
    - Wykorzystanie zmodyfikowanego blokowego algorytmu szyfrującego AES w trybie pracy AES-CTR;
    - Modyfikacja algorytmu CMAC dla realizacji mechanizmu integralności i uwierzytelniania w transmisji pomiędzy terminalami[7];
    - Wykorzystanie standardu podpisu cyfrowego, opartego o funkcję skrótu SHA-256 oraz algorytm RSA-1024, dla realizacji mechanizmów uwierzytelniania oraz integralności w komunikacji centrum bezpieczeństwa z terminalami;
- Zaproponowanie protokołu komunikacyjnego dla transmisji pomiędzy terminalami oraz pomiędzy centrum bezpieczeństwa a terminalami;
- Określenie struktur podstawowych wiadomości;
- Zdefiniowanie sposobu adresowania oraz „podpisywania” wiadomości;

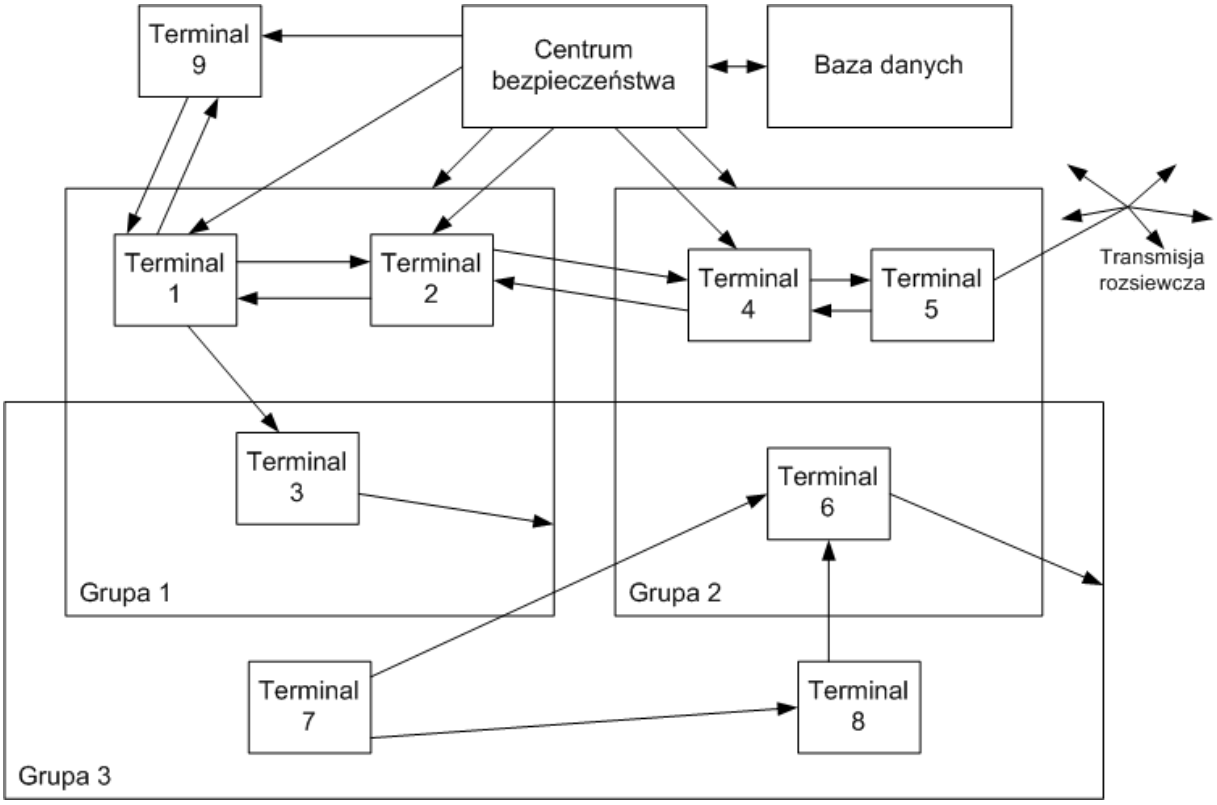
[7] Dla potrzeb projektu niniejszego kryptosystemu w terminalu użytkownika założono istnienie modułu kryptograficznego (możliwość jego implementacji w modemie rozważana była w rozdziale poprzednim). W pracy zatem zawsze, gdy mowa o łączności pomiędzy terminalami lub pomiędzy centrum bezpieczeństwa a terminalami, zakłada się wykorzystanie modułu (czy modułów) w takiej komunikacji.

- Określenie sposobu tworzenia elementów składowych wiadomości sterujących oraz zawierających dane użyteczne z wykorzystaniem nanoinstrukcji (pikoinstrukcji);
- Opracowanie sposobu dystrybucji i zmiany kluczy szyfrujących, jak również mechanizmu ich generowania;
- Opracowanie schematu hierarchizacji kluczy szyfrujących wraz z określeniem warunków, kiedy tracą one ważność i podlegają aktualizacji;
- Określenie sposobu wykorzystania oraz generacji sekwencji skramblujących dla algorytmów AES-CTR oraz zmodyfikowanego CMAC, jak również do bezpieczniejszego przesyłania kluczy szyfrujących;
- Opracowanie autorskiego mechanizmu, pozwalającego na łatwe określanie poziomu autoryzacji danego terminala oraz wykorzystywanie tylko niewielkiej liczby kluczy do realizacji transmisji typu terminal-terminal w całym systemie;
- Opracowanie struktury modułu kryptograficznego oraz centrum bezpieczeństwa (CB);
- Zdefiniowanie parametrów (klucze, maski, identyfikatory) opisujących i wykorzystywanych przez te dwa elementy;
- Określenie sposobów definiowania tych parametrów możliwości ich zmian oraz znaczenia w całym systemie;
- Zdefiniowanie zakresu komend sterujących wysyłanych przez CB oraz sposobu reagowania na nie przez terminal;
- Określenie sposobu współpracy terminala z modułem kryptograficznym (patrz paragraf 2.2);
- Dopasowanie systemu do dostępnych standardów modemów HF oraz zastosowania go w trudnym, jonosferycznym kanale radiowym;
- Opracowanie sposobu zarządzania całym systemem poprzez generowanie instrukcji sterujących przez centrum bezpieczeństwa;
- Określenie sposobu tworzenia wirtualnych kanałów transmisyjnych, pozwalających na transmisję o określonym poziomie autoryzacji;
- Zapewnienie możliwości współistnienia zaproponowanego rozwiązania z ewentualnymi innymi rozwiązaniami tego typu;
- Umożliwienie tworzenia niezależnych realizacji zaproponowanego kryptosystemu;

- Oraz wiele innych istotnych szczegółów, pozwalających na implementację zaprojektowanego systemu.

3.2. Struktura zaproponowanego kryptosystemu

Przykładowa struktura zaproponowanego rozwiązania przedstawiona została na rysunku 3.1.



Rys. 3.1. Przykładowa struktura zaproponowanego systemu bezpieczeństwa.

Na podstawie powyższego rysunku widać, że prezentowany system bezpieczeństwa bazuje na łączach jednokierunkowych. Jest to jedna z najistotniejszych cech odróżniająca to rozwiązanie od wielu systemów tego typu, które bazują na łączach dwukierunkowych (GSM, UMTS, WLAN itp.).

Realizacja bezpiecznego systemu, bazującego tylko na transmisji informacji i nieposiadającego możliwości uzyskania informacji zwrotnej o poprawności przeprowadzonej transmisji jest dużo trudniejsza niż w przypadku systemów typu pytanie-odpowiedź.

Należy w tym miejscu oczywiście wyraźnie zaznaczyć, że wspomniana tu jednokierunkowość tyczy się tylko wymiany informacji dotyczących opracowanego protokołu transmisji informacji, związanych z systemem kryptograficznym.

Najczęściej bowiem modemy HF posiadają wbudowane protokoły (np. ARQ [52, 82]) umożliwiające uzyskanie informacji zwrotnych. Nie jest to jednak część systemu bezpieczeństwa i nie ma nic wspólnego z rysunkiem 3.1. Aspekt ten omówiono nieco szerzej w rozdziale drugim.

Przedstawiona tu przykładowa struktura systemu została wybrana ze względu na wykorzystanie zaproponowanego kryptosystemu w bardzo destruktywnym kanale krótkofalowym [39, 64]. Trudno jest bowiem w łączu HF wymagać jakości i dostępności systemu na poziomie spotykanym na przykład w sieciach komórkowych. Autor założył zatem, że zaproponowane rozwiązanie musi funkcjonować nawet w warunkach, w których łączność z CB jest ograniczona przez nawet bardzo długi czas, a komunikacja zwrotna może często nie być możliwa.

Każda odebrana wiadomość z centrum bezpieczeństwa (adresowana do danego terminala lub grupy) powinna zatem zawierać już informacje, które pozwalają na dokonanie aktualizacji danych w module kryptograficznym. A dodatkowo wiadomość taka musi być wiarygodna, poufna oraz integralna.

Komunikacja między terminalami podlega podobnym zasadom. Pojedyncza wiadomość przesłana od jednego terminala do drugiego musi pozwalać na jej deszyfrację, sprawdzenie jej integralności i uwiarygodnienie nadawcy. Określony może w niej być również poziom autoryzacji wymagany do jej odczytania.

Przy takim podejściu pojawiają się jednak problemy. Po pierwsze, terminal nadawczy nie wie czy jego wiadomość dotarła do adresata. Zagadnienie to jest jednak nieistotne z punktu widzenia bezpieczeństwa systemu. Nie jest zatem definiowane w zaproponowanej specyfikacji tego systemu. Warto tu jednak przypomnieć, że problem ten jest zazwyczaj rozwiązany przez inne protokoły komunikacyjne (np. ARQ [82]) wykorzystywane w przypadku transmisji danych w łączu HF.

Kolejny problem stanowi jednak fakt, że również centrum bezpieczeństwa nie posiada informacji, czy terminal (a właściwie moduł kryptograficzny terminala) otrzymał i wykonał odpowiednie instrukcje od CB. Z tą sprawą wiąże się kilka aspektów:

- Centrum bezpieczeństwa powinno transmitować dane aktualizacyjne do terminali w sposób względnie ciągły. Może być do tego przystosowany np. pewien konkretny kanał HF.

- CB powinno wykorzystywać wiadomości typu multicast w celu uzyskania większej wydajności wysyłanych aktualizacji.
- Czas ważności kluczy w module zapewnia margines bezpieczeństwa. Jest zatem pewien przedział czasu, w którym terminal wystarczy, że odbierze właściwie tylko jedną wiadomość aktualizacyjną od CB, przeznaczoną np. dla jego grupy.
- W przypadku nieautoryzowanego dostępu do systemu z wykorzystaniem na przykład skradzionego modułu i braku możliwości komunikacji CB z terminalami (w celu na przykład ich dezaktywacji), zmniejszenie bezpieczeństwa jest w dużym stopniu ograniczone. Dany moduł ma bowiem możliwość transmisji tylko w obrębie konkretnych grup terminali oraz do mało bezpiecznej i niezalecanej metody komunikacji poza grupami. Ponadto nieautoryzowany dostęp ograniczony jest również czasowo, ponieważ klucze skradzionego terminala z czasem same utracą ważność i nie zostaną zaktualizowane o ile CB uzyska informacje o kradzieży modułu. Warto tu również dodać, że czasowo ograniczona jest również autoryzacja w grupach oraz aktywacja modułu.

Na podstawie rysunku 3.1 widać, że CB może wysyłać wiadomości nie tylko do konkretnego terminala, ale również do całej grupy terminali. Wiadomości wysyłane przez CB będą w ogólności nazywane SMM (*Security Management Messages*).

Terminale mogą przesyłać aż pięć typów wiadomości zwanych ogólnie UDM (*User Data Messages*):

- Wiadomość rozsiewcza (broadcast) – mało bezpieczna wiadomość możliwa do odebrania przez wszystkie terminale w zasięgu stacji nadawczej i posiadające odpowiednie klucze oraz obsługujące daną realizację systemu;
- Wiadomość multicast typ 1 – wiadomość do całej grupy nadawcy wiadomości;
- Wiadomość multicast typ 2 (w kanale wirtualnym) – wiadomość do tej części grupy nadawcy wiadomości, która posiada odpowiednie uprawnienia (odpowiedni poziom autoryzacji) do odbioru tej wiadomości;
- Wiadomość unicast typ 1 (w grupie) – wiadomość do jednego terminala będącego członkiem grupy nadawcy;
- Wiadomość unicast typ 2 (poza grupę) – mało bezpieczna wiadomość do jednego terminala nienależącego do grupy nadawcy.

Szczegóły odnośnie podstawowych struktur wiadomości SMM oraz UDM przedstawione zostaną w dalszej części niniejszego rozdziału.

Protokół transmisji pomiędzy CB oraz bazą danych (patrz rysunek 3.1) nie podlega specyfikacji i zależy od konkretnej realizacji kryptosystemu. Zakłada się tylko, że musi on być bezpieczny i ograniczać dostęp tylko dla osób upoważnionych. Dotyczy to również samej bazy danych [34, 35], zawierającej dane użytkowników systemu oraz historię przesyłanych do nich instrukcji.

Poszczególne realizacje systemu będą w pełni niezależne od siebie i nie będzie możliwości interakcji pomiędzy należącymi do nich CB. Występować będzie jednak pewien sposób komunikacji pomiędzy terminalami należącymi do różnych realizacji (patrz punkt 3.3).

W celu zapewnienia możliwości współistnienia wielu systemów, a więc i takich nie określonych w tej pracy, wprowadzono niezależny identyfikator systemu. Jest to parametr jednobajtowy i w przypadku prezentowanego systemu ma on wartość *0x86.* W dalszej części pracy *System ID* będzie rozumiane zawsze jako właśnie ta wartość, a zaprojektowany system bezpiecznej transmisji danych będzie często nazywany kryptosystemem *0x86*.

Innym parametrem stosowanym w dalszym opisie będzie *Provider ID*. Określa on poszczególne i niezależne realizacje systemu bezpieczeństwa (np. zaproponowanego systemu o identyfikatorze *0x86*). Dana wartość *Provider ID* będzie odnosić się do konkretnego centrum bezpieczeństwa (czyli providera systemu *0x86* – a właściwie jednej jego realizacji).

3.3. Moduł kryptograficzny

Stanowi on niezależną część terminala użytkownika, a w szczególności modemu HF. Projekt modułu dla potrzeb kryptosystemu *0x86* powstał (w dużym stopniu) w oparciu o wymogi zawarte w standardzie [23]. Moduł kryptograficzny pełnić będzie w przypadku nadawania pierwszy blok przetwarzania danych użytkownika, po którym występować będą kolejne zdefiniowane w [52] bloki funkcjonalne nadajnika (w tym również opcjonalny blok protokołu ARQ – patrz rozdział drugi). W przypadku odbiornika blok ten stanowić będzie ostatni element toru odbiorczego.

Warto tu zwrócić uwagę na dwa aspekty. Po pierwsze: moduł kryptograficzny powinien umożliwiać działanie modemu HF z wyłączonym systemem bezpieczeństwa. Po drugie: szyfrowaniu nie mogą podlegać:

- Wszelkie preambuły i ciągi treningowe (a jedynie dane użytkownika);
- Sekwencja EOM (*End Of Message*) [52] – konieczna do wykrycia końca transmisji.

Oczywistym jest fakt, że powyższe dwa warunki będą zawsze spełnione, gdy moduł kryptograficzny będzie pierwszym blokiem funkcjonalnym w torze nadawczym. Zostały one jednak tu przedstawione w celu uwypuklenia tego faktu.

Moduł kryptograficzny to element, który posiadać musi pewien zasób pamięci, w którym przechowywane będą klucze kryptograficzne oraz wiele innych identyfikatorów czy sekwencji skramblujących. Wymogi co do pamięci nie są duże. Pamięć wymagana do przechowywania danych (wraz ze sporym zapasem) to 64 kB (kilobajty). Taka ilość wystarczy na przechowanie wszystkich parametrów modułu nawet przy wykorzystywaniu w większej liczby realizacji kryptosystemu czy grup użytkowników. Warto tu jednak zauważyć, że część z tych 64 kB pamięci powinna być typu ROM, ponieważ niektóre parametry nie będą mogły być zmieniane. Sposób jednak organizacji tej pamięci nie podlega specyfikacji i jest dowolny w zależności od realizacji. Ważne jest jednak, aby uniemożliwić zewnętrzny dostęp do tych zasobów, ponieważ informacje tam zawarte mogą przyczynić się do złamania danej realizacji systemu.

Wymagana jest również pewna dodatkowa pamięć ROM (np. EEPROM), w której przechowywany będzie kod programu, realizowanego przez moduł. Ilość tej pamięci nie jest tu zdefiniowana, bo zależy od konkretnej realizacji zaproponowanej koncepcji, a dostęp do niej musi być ściśle ograniczony tylko jeśli chodzi o jej zapis.

Zasoby pamięci wymagane z kolei do realizacji obliczeń są niewielkie, ale trudne w tej chwili do sprecyzowania. Potrzeba bowiem tyle pamięci RAM, aby wystarczyło do zapisu zmiennych tymczasowych oraz nawet maksymalnej długości wiadomości (4 kB) przed i po danej operacji realizowanej w module. Wydaje się zatem, że 16 kB będzie ilością wystarczającą, ale wartość ta może być nieco inna w praktycznych realizacjach. Ze względów oczywistych dostęp do pamięci RAM musi być ograniczony.

Do realizacji obliczeń wymagany będzie układ o niezbyt dużych wymogach odnośnie szybkości przetwarzania. Jest to związane z niewielkimi przepływnościami osiąganymi w modemach krótkofalowych [52, 83]. Autor zaleca stosowanie układów typu FPGA a nie procesorów DSP, ponieważ reprogramowalność i elastyczność nie jest tu cechą krytyczną. Raz zaprogramowany układ FPGA może pełnić swa rolę w każdej realizacji systemu *0x86*. Przykłady tego typu rozwiązań zaprezentowano między innymi w [14, 21, 71].

Ponadto należy tu wspomnieć, że implementacja modułu kryptograficznego może zostać zrealizowana w postaci aplikacji uruchamianej na kompouterze PC (patrz punkt 2.2). W takim przypadku w sposób oczywisty jednostką przetwarzania będzie tu standardowy procesor CPU.

Do podstawowych zadań[8] modułu kryptograficznego należą między innymi:

- Deszyfracja informacji przesyłanych przez CB oraz aktualizacja konfiguracji w zależności od otrzymanych instrukcji;
- Uwierzytelnianie centrum bezpieczeństwa oraz kontrola integralności danych otrzymywanych od CB;
- Szyfrowanie i deszyfracja danych w trakcie transmisji pomiędzy terminalami;
- Uwierzytelnianie terminala nadawczego oraz kontrola integralności danych otrzymanych od niego;
- Deszyfracja kluczy otrzymywanych od CB;
- Kontrola aktualności uprawnień, kluczy szyfrujących oraz czasu pozostałego do końca aktywacji modułu;
- Rozpoznawanie nadawcy, adresata oraz typu otrzymywanych wiadomości (a także ich długości i innych parametrów);
- Dekodowanie wiadomości SMM oraz UDM;
- Tworzenie wiadomości UDM;
- Odrzucanie wiadomości niespełniających odpowiednich kryteriów;
- Wyznaczanie sekwencji skramblujących
- Wyznaczanie właściwych wartości licznika CTR przy realizacji zmodyfikowanego algorytmu AES-CTR;
- Zwracanie bajtu statusu.

[8] Większość z tych zadań omówiona będzie w dalszej części niniejszej rozprawie doktorskiej.

Algorytmy kryptograficzne wykorzystywane przez moduł (patrz rozdział pierwszy i dalsza część niniejszego rozdziału):

- Szyfrowanie i deszyfracja zmodyfikowanym AES-CTR;
- Wyznaczanie i weryfikacja zmodyfikowanej sekwencji CMAC;
- Uwierzytelnianie CB z wykorzystaniem RSA oraz SHA-256;
- Inne – autorskie rozwiązania implementacyjne.

Moduł kryptograficzny odrzuci wiadomość, która:

- Jest zakodowana z wykorzystaniem innego kryptosystemu niż *0x86*;
- Została zakodowana z wykorzystaniem nieobsługiwanej realizacji systemu (nieobsługiwany *provider*) *0x86*;
- Jest adresowana do kogoś innego;
- Wymaga wyższego poziomu autoryzacji;
- Nie można jej poprawnie zdeszyfrować;
- Posiada błędne instrukcje lub błędy w strukturze;
- Nie pozwala na uwierzytelnienie nadawcy;
- Jest już nieważna;
- Nie jest integralna.

Każdy moduł kryptograficzny musi zawierać dane zdefiniowane w tabeli 3.1.

W dalszej części niniejszego paragrafu powyższe parametry zostaną przedstawione bardziej szczegółowo.

W tym miejscu należy wyraźnie zaznaczyć, że niektóre parametry z powyższej tabeli wiążą się ze sobą jednoznacznie. Identyfikator modułu (*Module ID*) jest jednoznacznie powiązany z kluczem głównym (*MasterKey 0*). Dodatkowo oba te parametry są niepowtarzalne w całym kryptosystemie *0x86*. Nie może istnieć sytuacja, w której istnieć będą dwa moduły o takim samym *Module ID* (jest to jednoznaczny identyfikator). Nie powinno być też sytuacji, kiedy dwa lub więcej modułów posiadają ten sam klucz główny, który (podobnie jak klucze F danych realizacji) nie jest nigdy aktualizowany czy zmieniany (nie jest nawet przesyłany).

Wszystkie parametry z tabeli 3.1, których nie można modyfikować są na stałe zapisane w module jeszcze przed jego udostępnieniem danemu użytkownikowi.

Jednoznaczne powiązanie występuje również pomiędzy kluczami grup (*GroupKey*) oraz ich identyfikatorami (*Group ID*). Parametry te nie mogą się

powtarzać w obrębie jednej realizacji systemu, ale nie tyczy się to przypadku różnych *Provider ID*.

Parametr *Module ID* rozumiany może być jako globalny adres użytkownika w systemie. Może on być wykorzystywany do adresowania połączeń w trybie punkt-punkt (*unicast*) poza grupą.

Tab. 3.1. Wartości przechowywane w module kryptograficznym.

Nazwa		Długość [bity]	Możliwość zmiany	Opis
System ID		8	Brak	Identyfikator prezentowanego kryptosystemu (wartość stała *0x86*)
Module ID		32	Brak	Stały identyfikator modułu kryptograficznego
RSAPublicKey		1024	Brak	Publiczny klucz RSA kryptosystemu *0x86*
MasterKey 0		128	Brak	Klucz główny modułu kryptograficznego
ScramblingTable A1 – A4		4 x 256 x 128	Brak	Tablice wykorzystywane do generacji sekwencji skramblujących
ScramblingTable B1 – B4		4 x 256 x 128	Brak	
Parametry niezależne dla każdej z maksymalnie czterech realizacji kryptosystemu				
Provider ID		8	Brak	Identyfikator jednej z maksymalnie czterech konkretnych realizacji kryptosystemu obsługiwanych jednocześnie przez pojedynczy moduł kryptograficzny
SessionBasicKey F		128	Brak	Klucz podstawowy sesji (wartość stała w danej realizacji systemu)
ActivationDate		16	Jest	Data wygaśnięcia aktywacji danej realizacji kryptosystemu w module kryptograficznym
User ID 1 – 8	*Group ID 1 – 8*	8 x 24	Jest	Identyfikatory maksymalnie ośmiu grup, do których może należeć użytkownik
	InGroup ID 1 – 8	8 x 8	Jest	Identyfikatory użytkownika w maksymalnie ośmiu grupach, do których może on należeć
GroupKey 1 – 8		8 x 128	Jest	Klucze grup dla maksymalnie ośmiu grup, do których może należeć użytkownik
GroupAuthorizationDate 1 – 8		8 x 16	Jest	Data wygaśnięcia uprawnień w maksymalnie ośmiu grupach, do których może należeć użytkownik
AuthorizationMask 1 – 8		8 x 128	Jest	Maska uprawnień dla maksymalnie ośmiu grup, do których może należeć użytkownik
SessionKey A, B		2 x 128	Jest	Klucze sesji
KeyValidityDate A, B		2 x 16	Jest	Daty ważności poszczególnych kluczy sesji

Group ID z kolei oznacza właściwie adres pewnej grupy użytkowników i wykorzystywany może być przy realizacji transmisji multicastowych.

Group ID wraz z *InGroup ID* tworzą łącznie pewien identyfikator użytkownika (*User ID*), jednoznaczny jednak tylko w obrębie danej realizacji systemu. Może on służyć do adresowania połączeń unicastowych. Sam parametr *InGroup ID* jest

lokalnym adresem użytkownika w danej grupie. W jej obrębie może znajdować się maksymalnie 255 użytkowników (*InGroup* o wartości *0x00* nie jest poprawne).

Maska uprawnień (*AuthorizationMask*) to 128-bitowy ciąg bitowy, w którym każda jedynka oznacza uprawnienie do odbioru/nadawania a zero jego brak. Numer pozycji, na której znajduje się dana jedynka lub zero nazywany będzie identyfikatorem tak zwanego wirtualnego kanału transmisyjnego w danej grupie. Kanały te wykorzystywane będą tylko w trakcie transmisji w obrębie danej grupy. Można więc zrealizować do 128 niezależnych kanałów w każdej z grup. Dany użytkownik może nadawać w danym wirtualnym kanale tylko, gdy posiada do tego uprawnienia (jedynka na odpowiedniej pozycji). Analogicznie terminal może odbierać informacje tylko, gdy posiada odpowiedni poziom autoryzacji.

Klucz publiczny RSA (*RSAPublicKey*) konieczny jest dla uwierzytelniania wiadomości pochodzących z CB oraz kontroli ich integralności. Parametr ten jest jawny w całym kryptosystemie.

Klucz główny oraz klucze grup wykorzystywane są przez CB w trakcie przesyłania informacji do terminali. Klucze grup stosowane są również przez moduły do realizacji transmisji w obrębie grup. Wykorzystuje się je do uzyskania jawnej postaci klucza zarówno dla zmodyfikowanego algorytmu AES-CTR jak i CMAC. Szczegółowe zadania kluczy grup omówione zostaną w dalszej części niniejszego rozdziału.

Klucze sesji (*SessionKey*) wykorzystywane są przez moduły kryptograficzne do transmisji informacji użytkowych. Z założenia do nadawania powinien być wykorzystywany zawsze klucz aktualny – nigdy przyszły. Przy próbie deszyfracji wykorzystuje się również klucz aktualny. W momencie jednak jej niepowodzenia zastosować można również klucz przyszły. Klucze sesji, które utraciły swą ważność (*KeyValidityDate*), nie mogą być wykorzystane do tworzenia czy deszyfrowania wiadomości.

W związku z kluczami sesji obowiązuje zasada, że podczas ich aktualizacji centrum bezpieczeństwa przesyła zawsze klucz aktualny i przyszły (wraz z kluczami CB powinno dostarczać również datę ich ważności). Dzięki temu, gdy ten pierwszy utraci ważność, to drugi staje się kluczem aktualnym. Centrum bezpieczeństwa ma w tym momencie czas na przesłanie kolejnej aktualizacji. Przesyłanie samego tylko klucza przyszłego nie jest pożądane, ponieważ moduły nowe lub takie, które nie

odebrały poprzedniej aktualizacji, nie będą posiadać aktualnego klucza sesji (jedynie przyszły[9]).

Moduł aktywny do transmisji wykorzystać może również klucz F (*SessionBasicKey*), ale jest to rozwiązanie mało bezpieczne, ponieważ klucz ten jest stały w danej realizacji systemu. Podejście takie stosować może moduł niemający ważnych kluczy sesji.

Dodatkowe informacje związane ze sposobem używania odpowiednich kluczy w danych sytuacjach będą jeszcze przedstawiane w dalszych paragrafach niniejszej pracy.

Data, do której ważna jest aktywacja danej realizacji systemu (*ActivationDate*), wykorzystywana jest w celu automatycznej dezaktywacji po jej upłynięciu. W tym miejscu warto zdefiniować pojęcie modułu aktywnego (a właściwie aktywnej realizacji). Jest to taki moduł, który posiada jakikolwiek klucz grupy wraz z identyfikatorem *User ID* oraz ważną datą autoryzacji oraz aktywacji.

Po upłynięciu daty aktywacji w module pozostają tylko parametry niezmienialne (patrz tabela 3.1) a reszta zostaje wyzerowana (skasowana). Moduł nieaktywny, który nie posiada choćby jednej aktywnej realizacji kryptosystemu, nie może aktualizować kluczy sesji, ponieważ nie posiada kluczy grup. Moduł taki nie może ponadto odbierać i generować jakichkolwiek wiadomości UDM dla nieaktywnej realizacji kryptosystemu. Możliwy staje się tylko odbiór wiadomości SMM typu punkt-punkt. Szczegóły dotyczące aktywacji i dezaktywacji modułu znaleźć można w rozdziale następnym niniejszej pracy.

Data autoryzacji w grupie (*GroupAuthorizationDate*) to data, po której nie możliwy staje się odbiór i nadawanie wiadomości w danej grupie. Po jej upłynięciu moduł nie kasuje żadnych parametrów, ale grupa taka staje się niemożliwa do wykorzystania dla użytkownika do momentu ewentualnej aktualizacji przez CB parametru *GroupAuthorizationDate*.

Znaczenie tablic skramblujących (*ScramblingTable A* oraz *B*) zostanie szerzej przedstawione w dalszej części niniejszego rozdziału.

[9] Klucz przyszły będzie oczywiście w tym momencie również aktualny, ale inne terminale nie będą go używać, ponieważ posiadać będą wcześniejszy i nadal ważny klucz sesji.

Podsumowując należy stwierdzić, że pojedynczy moduł obsługiwać może tylko jeden typ systemu (w tym przypadku *0x86*), ale kilka (maksymalnie cztery) jego realizacji.

Nie ma możliwości transmisji pomiędzy dwoma różnymi realizacjami (różne parametry *Provider ID*) systemu (nawet w trybie punkt-punkt). Jeśli moduł jednak obsługuje więcej niż jedną realizację, to transmisja może odbywać się niezależnie w jej obrębie.

Moduł po odebraniu i zdekodowaniu wiadomości zwraca na wyjściu bajt statusu i ewentualnie zdeszyfrowane dane użytkownika (tylko dla poprawnie zdekodowanej wiadomości UDM – patrz punkt 3.6). Dla wiadomości SMM ponadto wykonywane są odpowiednie instrukcje w niej zawarte (tylko po poprawnym jej zdekodowaniu).

W tabeli 3.2 przestawiono w formie dziesiętnej możliwe postacie bajtu statusu.

Tab. 3.2. Status modułu kryptograficznego.

Status	Opis
00	Brak transmisji
10	Wiadomość SMM poprawnie zdekodowana (Wszelkie instrukcje od CB zostały wykonane)
11	Wiadomość UDM poprawnie zdekodowana/utworzona
20	System inny niż *0x86*
30	Nieobsługiwana realizacja systemu
40	Wiadomość adresowana do kogoś innego
41	Brak uprawnień do odbioru/nadania wiadomości
50	Błąd deszyfracji (Błędne nano *0x2F* – patrz punkt 3.6)
60	Błędna nanoinstrukcja
61	Błędna treść nanoistrukcji
62	Błędna nanoinstrukcja w nagłówku
63	Błędna treść nanoistrukcji w nagłówku
70	Błąd uwierzytelniania/kontroli integralności
80	Dana realizacja kryptosystemu jest nieaktywna
90	Brak aktualnego klucza sesji
91	Nieaktywna tablica skramblująca A
92	Nieaktywna tablica skramblująca B
99	Błędne funkcjonowanie modułu

W powyższej tabeli kolor zielony oznacza poprawne zdekodowanie/utworzenie wiadomości, a kolor czerwony błędne. Kolor szary z kolei to status neutralny.

3.4. Centrum Bezpieczeństwa (CB)

Centrum bezpieczeństwa jest jednym z najistotniejszych elementów większości kryptosystemów. Pełni ono zasadniczą rolę w procesie zarządzania systemem bezpieczeństwa. W zaproponowanym rozwiązaniu do zadań CB należą między innymi:

- Generowanie kluczy;

Problem generowania kluczy jest wydawałoby się sprawą oczywistą. W rzeczywistości jednak element ten stanowi słabe ogniwo projektowanych systemów. Manualne generowanie jest obarczone schematycznym myśleniem człowieka. Tak zwany czynnik ludzki jest tu krytyczny. Do realizacji tego procesu stosuje się zatem najczęściej generatory pseudolosowe. Dla potrzeb niniejszego projektu wykorzystano generator deterministyczny BBS (*Bluma-Bluma-Shuba*) [60, 89] uważany za jeden z najbezpieczniejszych obecnie stosowanych algorytmów tego typu [89].

W przypadku generowania kluczy istotna jest również ich długość. Dzisiaj uważa się, że klucze o długości rzędu 64 bitów nie zapewniają wystarczającego poziomu bezpieczeństwa i metodą całkowitego przeglądu można złamać je w stosunkowo niedługim okresie czasu [81, 89]. Na potrzeby proponowanego rozwiązania wybrano zatem klucze długości 128 bitów (dla algorytmu AES oraz zmodyfikowanego CMAC). Długość ta uważana jest dzisiaj za wartość bezpieczną [81, 89]. W przypadku algorytmu RSA zdecydowano się na bardzo bezpieczną długość kluczy – 1024 bity.

- Przechowywanie kluczy i danych użytkowników;

Przechowywanie jest istotną sprawą z punktu widzenia systemu bezpieczeństwa, ponieważ nieautoryzowany dostęp do bazy, zawierającej klucze np. wszystkich użytkowników systemu, oznaczałby krytyczne naruszenie bezpieczeństwa systemu. W proponowanym rozwiązaniu przyjęto, że wszystkie dane przechowywane są w bezpiecznej bazie danych [34, 35, 44], która niekoniecznie musi stanowić integralną część CB (patrz rysunek 3.1). Zakłada się tu jednak, że komunikacja centrum z bazą danych jest w pełni bezpieczna, chociaż jej sposób nie jest tu zdefiniowany.

Warto tu również dodać, że w przypadku przechowywania kluczy i danych użytkowników istotne jest ograniczenie dostępu do tych informacji tylko dla osób do tego upoważnionych. Ważna jest tu zatem ścisła realizacja mechanizmu dostępności.

- Dystrybucja i zmiany kluczy;

Jest to jedno z ważniejszych zadań CB i kluczowe zagadnienie projektowanego kryptosystemu. Mechanizm dystrybucji kluczy jest bowiem

krytyczny jeśli chodzi o nieautoryzowany dostęp. W dalszej części niniejszej pracy przedstawiony zostanie szczegółowo zaproponowany bezpieczny protokół komunikacyjny pomiędzy CB a użytkownikami systemu, który umożliwiać będzie między innymi bezpieczną dystrybucję kluczy w trybie OTAR (*Over The Air Re-keying*). Mechanizm dystrybucji kluczy opracowano, opierając się o standard [57] oraz rozwiązania zawarte także w [22, 74].

Określony musi zostać również schemat zmian (aktualizacji) kluczy. Trzeba w tym miejscu rozgraniczyć dwie odmienne sytuacje. Pierwsza to taka, w której klucze tracą swą ważność i muszą zostać zmienione, aby utrzymać funkcjonalność całego systemu. Częstość tych aktualizacji zależy od konkretnego systemu. Dla potrzeb niniejszego rozwiązania przyjęto, że klucze sesji powinny być ważne około miesiąca (parametr ten jest modyfikowalny). Druga sytuacja to taka, w której naruszone zostało bezpieczeństwo systemu i należy zaktualizować klucze, aby kryptosystem mógł dalej pełnić swą rolę. Szczegóły dystrybucji kluczy będą przedstawione w dalszej części pracy.

- Aktywacja i dezaktywacja modułów kryptograficznych

Centrum bezpieczeństwa musi posiadać możliwość zdalnego zarządzania użytkownikami systemu. Jednym z działań, które może podjąć CB jest właśnie aktywacja nowych lub nieaktywnych modułów kryptograficznych (a właściwie konkretnych realizacji kryptosystemu w modułach) oraz dezaktywacja tych które nie są dalej wykorzystywane lub zostały np. skradzione. Szczegóły tego procesu omówiono w dalszej części pracy.

- Przydzielanie do grup, zmiana poziomu autoryzacji i inne

CB musi posiadać inne narzędzia, pozwalające na bardziej szczegółową kontrolę nad całością systemu. W niniejszym rozwiązaniu zaproponowano, że możliwe będzie między innymi przydzielanie użytkowników do grup, posiadających pewne wspólne cechy. Wprowadzona zostanie również tak zwana maska uprawnień dla danego użytkownika, która określać będzie poziom jego autoryzacji w danej grupie. Możliwym będzie również dowolne dodawanie i usuwanie użytkowników z jednej lub wielu grup. Szczegóły tych działań przedstawione zostaną w dalszej części niniejszej pracy.

- Tworzenie poprawnych składniowo wiadomości SMM, zawierających instrukcje dla terminali;

- Wyznaczanie właściwych wartości licznika CTR przy realizacji zmodyfikowanego algorytmu AES-CTR;
- Wyznaczanie sekwencji skramblujących;
- Generowanie kluczy AES, przesyłanych modułom kryptograficznym;
- Algorytmy wykorzystywane przez CB:
    - Szyfrowanie zmodyfikowanym algorytmem AES-CTR;
    - Generowanie podpisu z wykorzystaniem RSA-1024 oraz SHA-256;
    - Inne – autorskie rozwiązania implementacyjne.

Każde centrum bezpieczeństwa posiada swój własny identyfikator wspomniany już wcześniej *Provider ID*. Jest to wielkość jednobajtowa i rozgranicza poszczególne realizacje systemu bezpieczeństwa tu prezentowanego. Należy tu jednak zauważyć, że konkretna wartość *Provider ID* nie musi odnosić się wcale do jednego tylko CB. Często występować będzie bowiem sytuacja, w której dla jednej realizacji systemu dostępnych będzie kilka CB w celu np. zapewnienia dostępu do informacji aktualizacyjnych na większym obszarze. Istotny jest tu tylko fakt, że niezależnie od tego ile CB w danej realizacji będzie przewidzianych, to muszą się one komunikować z jedną wspólną bazą danych. Sposób tej komunikacji nie jest określany. Ważne tylko, aby w bazie znajdowały się zawsze aktualne informacje dotyczące terminali.

Same centra bezpieczeństwa są zatem jednostkami zarządzającymi systemem, a wszelkie dane powinny być przechowywane w dedykowanej dla danej realizacji systemu bezpiecznej bazie danych, do której dostęp będą mieć tylko autoryzowane osoby.

W bazie danych muszą znajdować się zatem dane poszczególnych użytkowników danej realizacji systemu.

Dodatkowo centrum bezpieczeństwa musi posiadać prywatny klucz RSA (*RSAPrivateKey* – ten sam w całym kryptosystemie), klucz F swojej realizacji oraz systemowe tablice skramblujące A oraz B (*ScramblingTable A, B*).

3.5. Stosowane modyfikacje algorytmów kryptograficznych

Niniejszy projekt systemu opiera się na kilku podstawowych algorytmach kryptograficznych (AES-CTR, RSA-1024, SHA-256 oraz CMAC). Dla potrzeb zaproponowanego rozwiązania część z nich została zmodyfikowana w celu dopasowania ich właściwości do wymogów stawianych systemowi oraz dla

zwiększenia całościowego bezpieczeństwa systemu jak również umożliwienia sprawnego nim zarządzania. Ważne jest w tym miejscu jednak to, że wprowadzone zmiany nie naruszają standardów, definiujących wspomniane algorytmy, a są pewnymi mechanizmami je rozszerzającymi.

3.5.1. Sekwencje skramblujące

W celu zaprezentowania autorskich rozwiązań rozszerzeń znanych algorytmów kryptograficznych, konieczne jest najpierw przedstawienie aspektu sekwencji skramblujacych, które są jednym z podstawowych elementów tych modyfikacji.

Zaproponowano wykorzystanie czterech takich sekwencji, zdefiniowanych w tabeli 3.3. Każda z nich utworzona jest z wykorzystaniem operacji sumowania modulo 2 dwóch wektorów skramblujących pobranych z systemowych tablic odpowiednio A oraz B. Numery tych wektorów w trakcie dekodowania odebranych wiadomości są wyznaczane na podstawie treści nanoinstrukcji *0x15* oraz *0x16* zawartych w nagłówku. Podczas tworzenia wiadomości z kolei losowane są dwie liczby z zakresu od 0 do 255 każda. Są one następnie zapisywane jako treści wspomnianych nanoinstrukcji i wykorzystywane do wyznaczenia sekwencji skramblujących koniecznych do zakodowania danej wiadomości.

Tab. 3.3. Sposób wyznaczania i wykorzystania sekwencji skramblujących w kryptosystemie *0x86*.

Sekwencja skramblująca	Sposób wyznaczania	Wykorzystanie
S1	$S_{i,A_m}^{(128)} \oplus S_{j,B_n}^{(128)}$	Dla realizacji zmodyfikowanego algorytmu AES-CTR
S2	$S_{i,A_m}^{(128)} \oplus S_{255-j,B_n}^{(128)}$	Dla realizacji zmodyfikowanego algorytmu CMAC
S3	$S_{255-i,A_m}^{(128)} \oplus S_{j,B_n}^{(128)}$	Deszyfracja kluczy dla mechanizmu uwierzytelniania (dla algorytmu CMAC)
S4	$S_{255-i,A_m}^{(128)} \oplus S_{255-j,B_n}^{(128)}$	Deszyfracja kluczy dla mechanizmu poufności (dla algorytmu AES-CTR)

Oznaczenia:
$S_{k,X_l}^{(128)}$ – *k*-ty 128 bitowy wektor z *l*-tej aktywnej tablicy skramblującej X;
A_m – *m*-ta aktywna tablica skramblująca A (*m* z zakresu od 1 do 4);
B_n – *n*-ta aktywna tablica skramblująca B (n z zakresu od 1 do 4);
i – numer wektora z zakresu od 0 do 255, przy odbiorze: wartość liczbowa uzyskana z treści nanoinstrukcji *0x15* (z nagłówka), przy nadawaniu: wartość losowa zapisywana w treści nanoinstrukcji *0x15* (w nagłówku);
j – numer wektora z zakresu od 0 do 255, przy odbiorze: wartość liczbowa uzyskana z treści nanoinstrukcji *0x16* (z nagłówka), przy nadawaniu: wartość losowa zapisywana w treści nanoinstrukcji *0x16* (w nagłówku).

Sposób tworzenia wspomnianych ciągów skramblujących nie jest przypadkowy, a każdy z nich ma inne przeznaczenie (patrz tabela 3.3). Sekwencje tworzone są w ten

sposób, że każda z nich wymaga dwóch wektorów po jednym z każdej tablicy skramblującej. Dzięki temu ujawnienie tylko tablicy A lub tylko tablicy B nie stanowi zagrożenia, bo żadnej z sekwencji i tak nie można w tym momencie wyznaczyć.

Ponadto poszczególne numery wektorów nie mogą być dobierane w sposób przypadkowy. Do wygenerowania bowiem czterech sekwencji skramblujących wymagane są również cztery wektory z odpowiednich tablic (po dwa wektory z każdej). Z nanoinstrukcji *0x15* oraz *0x16* uzyskuje się jednak tylko dwie liczby, będące numerami dwóch z czterech wektorów skramblujących. Pozostałe dwa uzyskuje się poprzez proste operacje przedstawione w tabeli 3.3. Takie podejście jednak zabezpiecza przed możliwością nieautoryzowanego tworzenia i dekodowania wiadomości, gdy jawne są tylko pojedyncze wektory skramblujące. Przykładowo, jeśli atakujący posiada wektory o numerach 15 (z tablicy A) oraz 32 (z tablicy B), to potrzebne mu są jeszcze dwa inne, ale już o ściśle określonych numerach: 240 (255 – 15) z tablicy A oraz 223 (255 – 32) z tablicy B. Bez tych dodatkowych ciągów można wyznaczyć tylko jedną z sekwencji skramblujących, co nie wystarcza do poprawnej realizacji zaproponowanych rozwiązań algorytmów kryptograficznych (nawet przy znanych odpowiednich kluczach).

Przy doborze reguł tworzenia sekwencji skramblujących autor kierował się ponadto jedną jeszcze zasadą. Każdy z tych ciągów ma w sposób oczywisty jeden wspólny wektor z jednym innym ciągiem. Dla realizacji mechanizmu poufności jak i uwierzytelniania (z kontrolą integralności) stosuje się po dwie różne sekwencje. Jedna zawsze stosowana jest przy deszyfracji klucza, a druga dla realizacji już samego algorytmu. Autor założył tu, że para sekwencji, wykorzystywana w realizacji danego mechanizmu kryptograficznego, nie mogła być wygenerowana przy użyciu choćby jednego wspólnego wektora skramblującego. Chodzi tu oto, by zarówno w uwierzytelnianiu jak i szyfrowaniu stosować wszystkie cztery wektory, pobrane z tablic systemowych. Dzięki temu konieczna jest znajomość ich wszystkich, by nawet przy znanym kluczu móc zmniejszyć oferowany poziom bezpieczeństwa któregokolwiek z mechanizmów kryptograficznych tu wspomnianych.

3.5.2. Zmodyfikowany AES-CTR (*Advanced Encryption Standard – CounTeR mode*)

Klasyczne rozwiązanie AES przedstawiono w punkcie 1.2.2 niniejszej pracy, a tryb CTR jego działania w punkcie 1.2.1. Dla potrzeb projektowanego kryptosystemu

wybrano 128-bitowy algorytm AES, ponieważ zapewnia on bardzo wysoki poziom poufności przesyłanych wiadomości i na dzień dzisiejszy nie istnieją szybkie sposoby ataków na niego [81, 89]. Możliwy jest tylko atak pełnego przeglądu, ale dla jego realizacji należałoby dokonać prób deszyfracji dla maksymalnie 2^{128} = 340282366920938463463374607431768211456 (ponad 340 sekstylionów) postaci klucza, co nie jest realizowalne przy dzisiejszym stanie techniki w rozsądnym przedziale czasu [37, 81].

Tryb CTR wybrano z kolei ze względu na:

- Brak powtarzalności szyfrogramów (ten sam tekst jawny daje różne teksty zaszyfrowane);
- Możliwość zrównoleglenia przetwarzania (zarówno szyfrowania jak i deszyfracji);
- Szyfrowanie i deszyfracja realizowana identycznie (uproszczenie implementacji);
- Błędy nie propagują się (pojedynczy błąd szyfrogramu generuje również jeden tylko błąd tekstu jawnego);
- Możliwość modyfikacji licznika (przy zachowaniu zasady unikalności – patrz paragraf 1.2.1 oraz [55]) – dodatkowe zabezpieczenie oraz elastyczność algorytmu.

Modyfikacja klasycznego rozwiązania AES-CTR wiąże się wyłącznie ze zmodyfikowanym sposobem wyznaczania kolejnych wartości licznika. Do tego celu korzysta się z: sekwencji skramblującej S1, znacznika czasu, sekwencji adresu oraz numeru aktualnie szyfrowanego bloku.

Wartość licznika ustalana jest zgodnie ze wzorem:

$$CTR_i^{(128)} = S1 \oplus (\{adresat\}, \{rok\}, \{miesiąc\}, \{dzień\}, \{godzina\}, \{minuta\}, \{i\}, \{i\}, \dots, \{i\}), \quad (3.1)$$

gdzie:

$CTR_i^{(128)}$ – 128-bitowa sekwencja licznika dla *i*-tego 128-bitowego bloku danych,

S1 – sekwencja skramblująca utworzona na podstawie tabeli 3.3,

adresat – adresat wiadomości (od 0 do 4 bajtów),

rok – rok wysłania wiadomości (dwie ostatnie cyfry dziesiętne zapisana binarnie, jeden bajt),

miesiąc – miesiąc wysłania wiadomości (od 1 do 12 binarnie, jeden bajt),

dzień – dzień wysłania wiadomości (od 1 do 31 binarnie, jeden bajt),

godzina[10] – godzina wysłania wiadomości (od 0 do 23 binarnie, jeden bajt),

minuta – minuta wysłania wiadomości (od 0 do 59 binarnie, jeden bajt),

i – numer bloku wiadomości (jeden bajt), powtarzany tyle razy, aby cały ciąg, sumowany modulo 2 z sekwencją skramblującą, miał dokładnie 128 bitów.

Wartość *i* zmienia się w zakresie od *0x01* do *0xFF* (lub mniej jeśli wiadomość jest krótsza niż maksymalna długość – patrz punkt 3.6). Dla każdej nowej wiadomości *i* ustawiane jest na wartość początkową – *0x01*. Wartość *i* = *0x00* jest niepoprawna, ponieważ blokiem „zerowym" jest nagłówek, który nie podlega szyfrowaniu, więc ustalanie dla niego wartości licznika nie ma większego sensu.

Na podstawie wzoru 3.1 można zauważyć, że każda wiadomość wysyłana z wykorzystaniem zmodyfikowanego algorytmu AES-CTR jest ważna tylko przez jedną minutę – ze względu na wykorzystanie znacznika czasu [20, 33]. Należy zatem zapewnić poziom synchronizacji pomiędzy terminalami i pomiędzy terminalami a CB na poziomie (przynajmniej z dokładnością do kilkudziesięciu sekund), pozwalającym na realizację transmisji. Ten minutowy zapas konieczny jest również, gdy wykorzystywany jest protokół ARQ [52, 82], ze względu na możliwe retransmisje tych samych zaszyfrowanych wiadomości w przypadku wystąpienia niekorygowanych błędów transmisji (aspekt ten był już poruszany w rozdziale poprzednim i będzie o nim jeszcze mowa w dalszej części niniejszej pracy).

W tym miejscu ważna jest jeszcze jedna kwestia. Sekwencja wyznaczona na podstawie wzoru 3.1 musi (zgodnie ze standardem [55]) spełniać kryterium unikalności, aby zaproponowana modyfikacja nie obniżała oferowanego w trybie AES-CTR poziomu bezpieczeństwa. Wartość licznika $CTR_i^{(128)}$ musi być bowiem unikalna nie tylko dla każdego bloku danej wiadomości, ale również nie może ulec powtórzeniu w żadnych wiadomościach, dla których zaszyfrowania stosuje się ten sam klucz.

Wspomniana sekwencja $CTR_i^{(128)}$ spełnia wymagania unikalności, ponieważ:

1. Jest inna dla każdego bloku danej wiadomości – wykorzystanie parametru *i* (patrz wzór 3.1).
2. Ma różne postaci dla różnych adresatów – stosowanie pola *adresat* dla wytworzenia sekwencji licznika (parz zależność 3.1).
3. Zmienia się co minutę – wykorzystanie znacznika czasu (patrz wzór 3.1).

[10] Czas musi być zawsze przedstawiony jako UTC (*Universal Time Clock*).

4. Może mieć jedną z losowo wybranych $256^2 = 65536$ postaci nawet dla stałych parametrów z punktów 1, 2 oraz 3.

Wyjaśnieniu wymaga tu punkt czwarty. W celu wytworzenia sekwencji licznika zgodnie ze wzorem 3.1 konieczny jest ciąg skramblujący S1, który wyznaczony został z kolei na podstawie operacji sumy modulo 2 dwóch wektorów skramblujących, pobranych z tablic systemowych odpowiednio A oraz B (patrz tabela 3.3). Każda z nich zawiera po 256 takich ciągów, które wybierane są losowo dla każdej tworzonej wiadomości. Sekwencja skramblująca może mieć zatem 65536 różnych postaci. Ta liczba ciągów musi być wystarczająca dla zapewnienia unikalności licznika w algorytmie AES-CTR przez minutę transmisji do tego samego użytkownika.

Należy w tym momencie nadmienić, że pomimo faktu wykorzystywania losowej sekwencji S4 przy deszyfracji kluczy dla algorytmu AES-CTR (patrz punkt 3.5.5), kryterium unikalności musi zostać mimo wszystko spełnione w formie przedstawionej powyżej, ponieważ danej sekwencji S4 odpowiada zawsze ta sama sekwencja S1.

W praktyce oznacza to, że tym samym jawnym kluczom odpowiadają jednakowe sekwencje S1[11], a w przypadku tego samego adresata oraz znacznika czasu również identyczne są wartości licznika. W tej sytuacji konieczne jest zatem, by ciągi S1 były unikalne mimo faktu wykorzystywania właściwie różnych jawnych postaci klucza.

W tym miejscu rozważyć można pewien przypadek graniczny. Najkrótsza wiadomość przesyłana z wykorzystaniem zaproponowanego kryptosystemu ma długość zaledwie 384 bitów[12] (tak krótkie wiadomości nie są jednak zalecane – patrz rozdział kolejny). Największa przepływność wykorzystywana w jednokanałowych i standardowych modemach HF wynosi 12800 bit/s[13] (patrz rozdział poprzedni). Maksymalna zatem (choć mało praktyczna) liczba wiadomości przesłanych pojedynczemu adresatowi (przy założeniu również jednego kanału transmisji) w ciągu jednej minuty wynosi:

[11] Wszystkie sekwencje skramblujące powstają z wykorzystaniem tych samych danych z nanoinstrukcji *0x15* oraz *0x16* (patrz tabela 3.3). Jeśli tylko klucze tajne i deszyfrujące są identyczne, to dla danych wartości uzyskanych z nanoinstrukcji *0x15* oraz *0x16* uzyskuje się te same klucze jawne i te same sekwencje skramblujące.

[12] 384 bity = 128 bitów (jawny nagłówek) + 128 bitów (przynajmniej jeden blok zaszyfrowany) + 128 bitów (sekwencja CMAC)

[13] Nie jest to przepływność zalecana w standardzie [52].

$$ceil\left(\frac{12800\ [\frac{bity}{s}]}{384\ [\frac{bity}{wiadomość}]}\cdot 60\ [s]\right) = 2000\ [wiadomości], \tag{3.2}$$

gdzie *ceil()* oznacza zaokrąglenie w górę do najbliższej wartości całkowitej.

Teoretycznie można zatem w ciągu 60-ciu sekund przesłać 2 tysiące wiadomości. Należy tu zwrócić uwagę na fakt, że liczba ta może być jeszcze większa. Kiedy założymy bowiem, że wielu nadawców będzie przesyłać wiadomości do tego samego adresata i w tym samym czasie, ale na innych kanałach częstotliwościowych. Zważywszy jednak na fakt, że potencjalny atakujący nie ma możliwości ustalenia, że na dwóch kanałach radiowych przesyłane są dane do tego samego odbiorcy (tajne pole adresata), to założenie o wykorzystaniu jednego kanału transmisji można uznać za praktycznie spełnione.

Szczególnie groźne są jednak wiadomości typu *broadcast* (pole adresata nie istnieje), które mogą być stosunkowo często nadawane przez wiele terminali i nietrudno je rozpoznać (jawny nagłówek). Taki typ wiadomości jest jednak z góry uznawany za mało bezpieczny (patrz rozdział czwarty).

W tym miejscu interesujące może być wyznaczenie prawdopodobieństwa (*P(k,n)*) zdarzenia, że dla konkretnej liczby wiadomości (*k*), przesyłanych do jednego adresata w przeciągu jednej minuty, wystąpi jakiekolwiek powtórzenie sekwencji skramblującej S1 (a tak właściwie wszystkich sekwencji od S1 do S4). Odpowiednia zależność jest tu następująca:

$$P(k,n) = 1 - \frac{C_n^k}{\bar{C}_n^k} = 1 - \frac{\frac{n!}{k!\cdot(n-k)!}}{\frac{(k+n-1)!}{k!\cdot(n-1)!}} = 1 - \frac{n!\cdot(n-1)!}{(n-k)!\cdot(k+n-1)!}, \tag{3.3}$$

gdzie:

- *n* – liczba możliwych postaci sekwencji skramblującej S1 (wartość stała *n = 65536*);
- *k* – liczba wiadomości kryptosystemu *0x86* przesłanych do jednego adresata w czasie (jednej minuty) ważności jednego znacznika czasu;
- C_n^k - kombinacja bez powtórzeń z *n* po *k*;
- $\bar{C}_n^k$ - kombinacja z powtórzeniami z *n* po *k*.

Należy tu pamiętać, że powtórzenie sekwencji S1 będzie automatycznie oznaczać powtórzenie ciągu $CTR_i^{(128)}$ dla danego numeru bloku (*i*) w wiadomościach, przesyłanych do jednego adresata i w obrębie jednego znacznika czasu.

Dla przypadku przedstawionego we wzorze 3.2 prawdopodobieństwo *P(2000,65536)* jest praktycznie równie jedności. Oznacza to, że wartość licznika dla omawianej sytuacji powtórzy się przynajmniej raz. Nie jest to cecha korzystna, ale jak już wspominano transmisja tak krótkich wiadomości (384 bity) z tak dużą jak na łącze HF szybkością (12800 bit/s) jest pozbawiona większego sensu, ponieważ zysk ze sporej przepływności zostanie zmarnowany przez ogromną nadmiarowość protokolarną kryptosystemu w takiej konfiguracji. Ponadto dla przepływności 12,8 kbit/s nie stosuje się kodowania kanałowego (patrz rozdział drugi), co dodatkowo ogranicza praktyczną użyteczność prezentowanego tu przypadku.

Przedstawione zostaną tu zatem dwa przykładowe i dużo bardziej praktyczne scenariusze (zależność 3.4 oraz 3.5). Wartości przepływności oraz długości wiadomości (a przez to i przeplotu) dobrane zostały jako optymalne dla zaproponowanego krytposystemu na podstawie tabeli 2.8 oraz paragrafu 4.2.2.

Prawdopodobieństwo P(56,65536) dla sytuacji:

$$ceil\left(\frac{6400\ [\frac{bity}{s}]}{6912\ [\frac{bity}{wiadomość}]}\cdot 60\ [s]\right) = 56\ [wiadomości] \tag{3.4}$$

wynosi około 0,046. Oznacza to, że przy wielokrotnym generowaniu serii po 56 wiadomości każda wystąpi co najmniej jedno powtórzenie sekwencji licznika dla 4,6 % z tych grup.

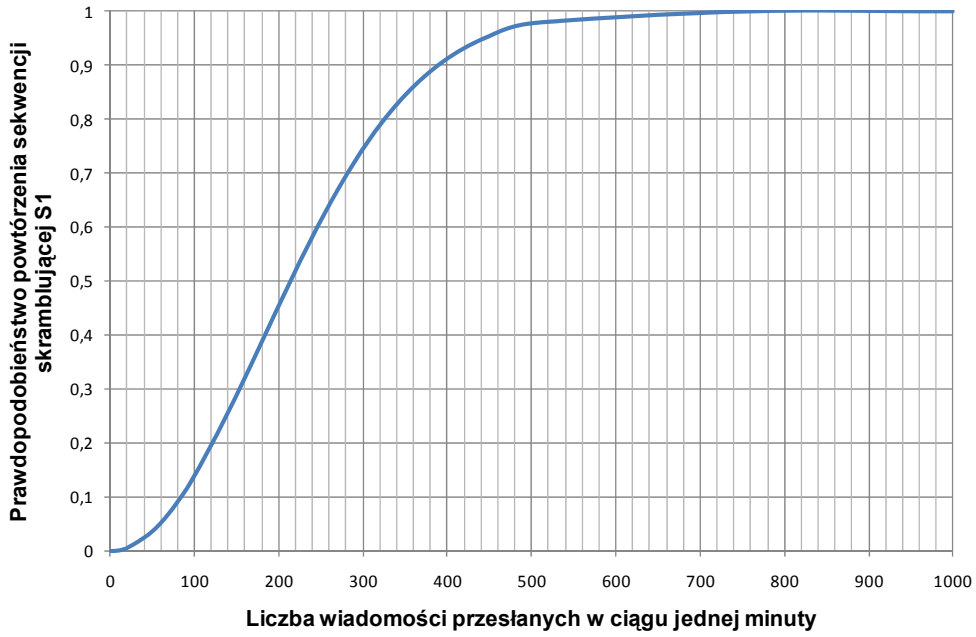
Dla dłuższych wiadomości sytuacja jest jeszcze korzystniejsza:

$$ceil\left(\frac{4800\ [\frac{bity}{s}]}{20736\ [\frac{bity}{wiadomość}]}\cdot 60\ [s]\right) = 14\ [wiadomości]. \tag{3.5}$$

Prawdopodobieństwo P(14,65536) wynosi tu zaledwie 0,3%.

Dla optymalnych długości wiadomości i przeplotu (patrz tabela 2.8 w rozdziale drugim oraz paragraf 4.2.2) wartość uzyskana na podstawie zależności 3.3 nigdy nie przekracza 5 %.

Dla lepszego zrozumienia, omawianego w tym paragrafie zagadnienia, na rysunku 3.2 przedstawiono wykres, prezentujący prawdopodobieństwo *P(k,65536)* dla różnej liczby wiadomości *k*.



Rys. 3.2. Prawdopodobieństwo powtórzenia sekwencji skramblującej S1 w czasie trwania pojedynczego znacznika czasu.

Podsumowując ten paragraf można stwierdzić, że dla praktycznie realizowalnych sytuacji ryzyko utraty unikalności licznika w zmodyfikowanym algorytmie AES-CTR jest znikome.

5.1.2. Zmodyfikowany CMAC (*Cipher-based Message Authentication Code*)

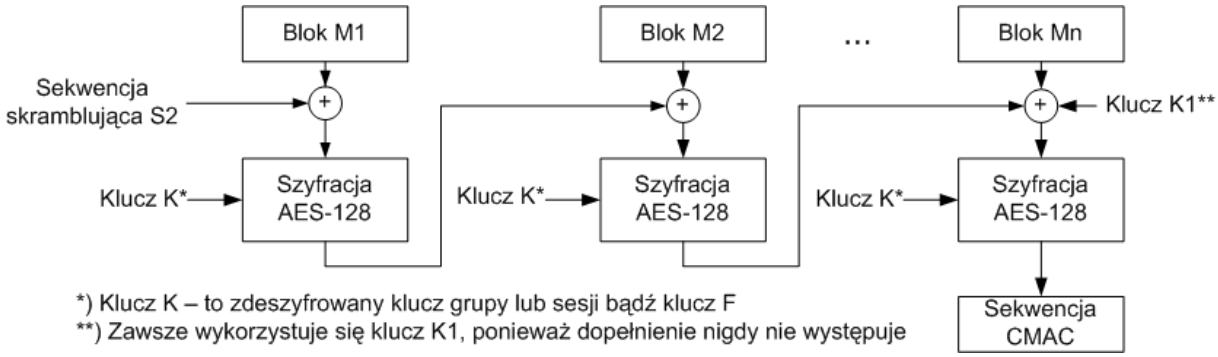
Podstawową wersję algorytmu CMAC przedstawiono w punkcie 1.2.5 niniejszej pracy. Dla celów zaproponowanego rozwiązania zmodyfikowano ten algorytm. Działania te podjęto ze względu na poniższe fakty.

W klasycznym rozwiązaniu CMAC czasami stosuje się dopełnienie ostatniego bloku danych w celu umożliwienia wyznaczenia sekwencji CMAC. W prezentowanym systemie taka sytuacja nie będzie miała miejsca (klucz K2 nie będzie nigdy wykorzystywany – patrz punkt 1.2.5), ponieważ długość wiadomości jest zawsze wielokrotnością 128 bitów (patrz punkt 3.6). Pierwszą zatem zmianą jest uproszczenie algorytmu CMAC poprzez usunięcie konieczności kontroli długości sekwencji wiadomości oraz mechanizmu generowania klucza K2.

Dla poprawy bezpieczeństwa oraz powiązania sekwencji uwierzytelniającej z parametrami systemowymi postanowiono skorzystać przy realizacji algorytmu ze 128-bitowej sekwencji skramblującej S2. Modyfikacja ta nie wpływa (czyli nie zmniejsza oferowanego poziomu bezpieczeństwa) jednak na samo standardowe

rozwiązanie CMAC, ponieważ wprowadzona jest ona jakby dodatkowo i jeszcze przed realizacją wspomnianego algorytmu.

Na rysunku 3.3 przedstawiono schemat realizacji zmodyfikowanego algorytmu CMAC.



Rys. 3.3. Zmodyfikowany algorytm CMAC.

Jak łatwo zaobserwować zaproponowana modyfikacja sprowadza się do dodania modulo 2 sekwencji skramblującej S2 do pierwszego bloku wiadomości, dla której wyznaczona ma zostać sekwencja CMAC. Efektem takiego działania będą możliwe różne sekwencje uwierzytelniające nawet w przypadku tego samego klucza i tej samej postaci wiadomości.

Należy w tym miejscu ponadto pamiętać, że do wyznaczenia klucza K (patrz rysunek 3.3) konieczna jest również sekwencja S3 (patrz paragraf 3.5.5 oraz 3.5.1). Dzięki temu dla realizacji omawianego tu algorytmu konieczne są cztery różne wektory z tablic skramblujących (dla wyznaczenia ciągów S2 oraz S3). Co więcej wektory te nie mogą być przypadkowe, ale zgodne z wymaganiami przedstawionymi w tabeli 3.3. Takie rozwiązanie jest zatem zabezpieczeniem w sytuacji nieautoryzowanej próby ustalenia sekwencji CMAC, gdy ewentualnemu atakującemu znane są tylko pojedyncze wektory skramblujące z tablicy A i B lub jedna cała tablica oraz na przykład klucz F danej realizacji kryptosystemu.

3.5.3. Algorytm podpisu cyfrowego

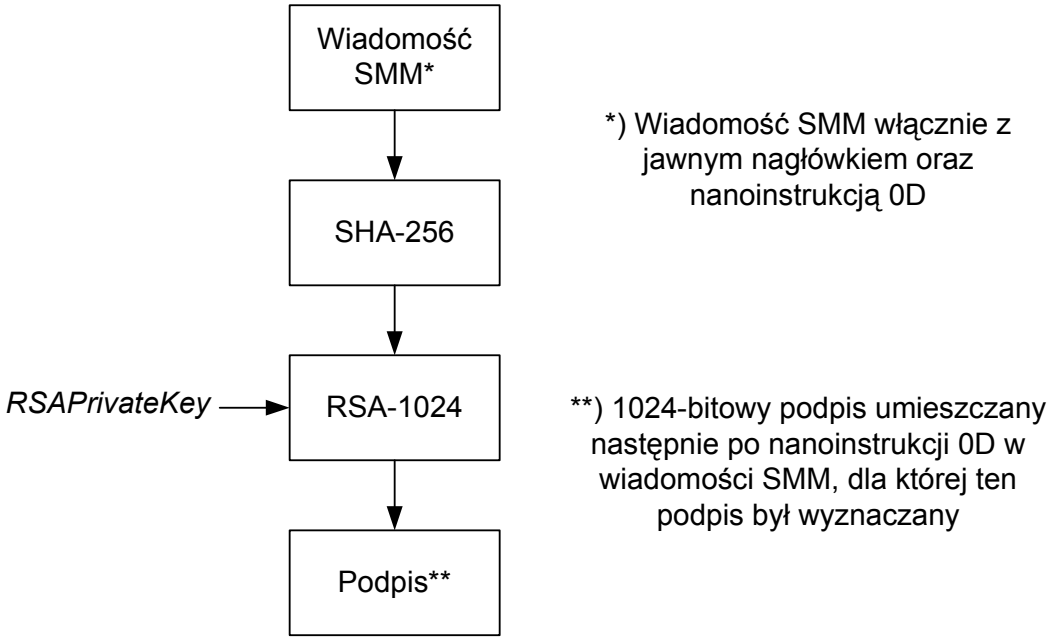
Podpis cyfrowy generowany jest tylko (nigdy przez moduły kryptograficzne) przez centrum bezpieczeństwa. Dołączany jest on do wszystkich wiadomości wysyłanych do terminali. Do jego implementacji wykorzystywany jest 1024-bitowy algorytm RSA (patrz punkt 1.2.3) oraz funkcja skrótu SHA-256 (patrz punkt 1.2.4). Sposób jego wyznaczania jest zgodny ze standardem podpisu cyfrowego [25].

W pierwszym etapie wyznaczania podpisu obliczany jest skrót jawnej wiadomości (wraz z nagłówkiem oraz nanoinstrukcją *0x0D* – patrz punkt 3.6). W efekcie uzyskuje się 256-bitową sekwencję, która w dalszej kolejności szyfrowana jest algorytmem RSA z wykorzystaniem prywatnego klucza RSA (*RSAPrivateKey*) kryptosystemu. W rezultacie otrzymuje się 1024-bitową sekwencję, która umieszczana jest na końcu wiadomości SMM (po nanoinstrukcji *0x0D*).

Cała procedura wyznaczania podpisu cyfrowego przedstawiona została na rysunku 3.4.

Weryfikacja podpisu odbywa się w modułach kryptograficznych adresatów wiadomości zarządzających i realizowana jest z wykorzystaniem publicznego klucza RSA (*RSAPublicKey*), który pozwala na deszyfrację sekwencji umieszczonej po nano *0x0D* w wiadomości SMM. Po wyznaczeniu z kolei skrótu (SHA-256) odebranej (i zdeszyfrowanej) wiadomości oraz porównaniu go ze zdekodowaną sekwencją RSA możliwe staje się zweryfikowanie wiarygodności danej wiadomości oraz kontrola jej integralności.

W tym miejscu warto tylko dodać, że w przypadku wiadomości UDM kontrolę integralności oraz uwierzytelnianie realizuje się z wykorzystaniem zmodyfikowanego algorytmu CMAC (patrz punkt 3.5.3). Sekwencję CMAC wyznacza się również dla całej wiadomości włącznie z nagłówkiem oraz nanem *0x0D*. Nie jest tu już ponadto wymagana funkcja skrótu. Wyznaczony ciąg uwierzytelniający dopisuje się również na końcu wiadomości (po nano *0x0D*).



Rys. 3.4. Procedura wyznaczania podpisu RSA.

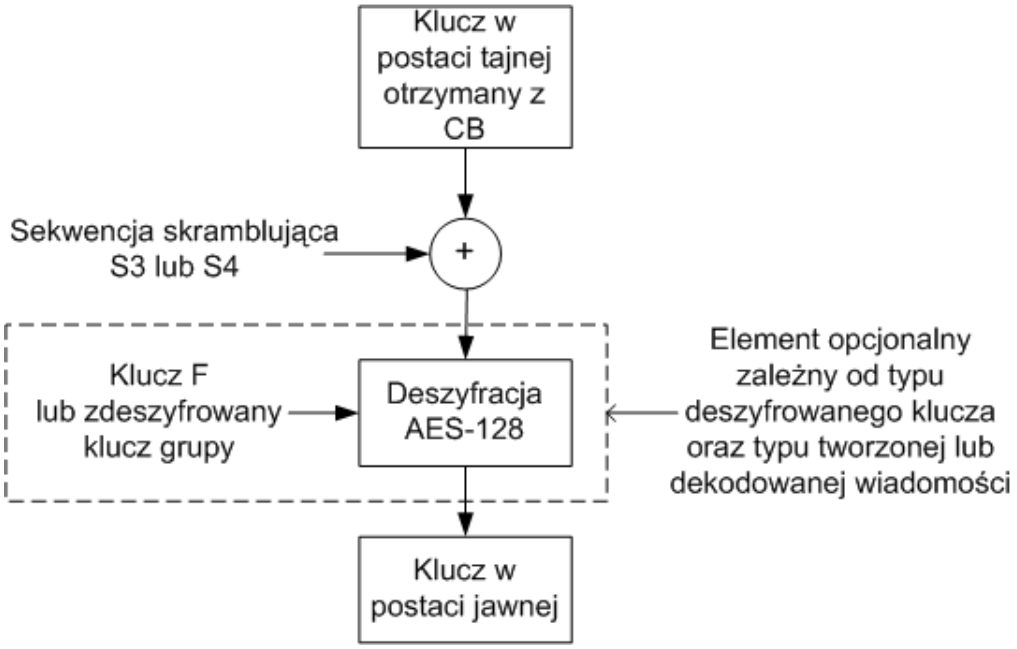
Weryfikacja wiarygodności i integralności odbywa się poprzez ponowne wyznaczeni w module odbiorczym sekwencji CMAC i porównanie jej z ciągiem dołączonym do wiadomości.

3.5.4. Deszyfracja kluczy

W zaproponowanym rozwiązaniu nigdy nie może wystąpić sytuacja, w której jakikolwiek klucz przesyłany przez CB w wiadomości SMM jest w niej zawarty w postaci jawnej. Poza faktem szyfrowania wiadomości jako całości, dla kluczy stosuje się dodatkowe zabezpieczenie. Rozwiązanie to nie ma na celu tylko dodatkowej ochrony przesyłanych kluczy, ale również pozwala na pewną ich hierarchizację.

Klucz, odebrany od CB w formie tajnej, w celu jego wykorzystania do tworzenia i dekodowania wiadomości musi zostać zsumowany modulo 2 z odpowiednią sekwencją skramblującą (S3 lub S4 – patrz tabela 3.3), a następnie ewentualnie zdeszyfrowany (AES-128) przy użyciu odpowiedniego klucza (F lub zdeszyfrowanego klucza grupy z zakresu 1 – 8). Wybór klucza deszyfrującego jest ściśle określony i zależy od typu przesyłanej wiadomości oraz od wykorzystywanego i aktualnie deszyfrowanego klucza (szczegóły w paragrafie 3.5.5.1).

Na rysunku 3.5 przedstawiono schematycznie sposób deszyfracji kluczy otrzymanych od CB.

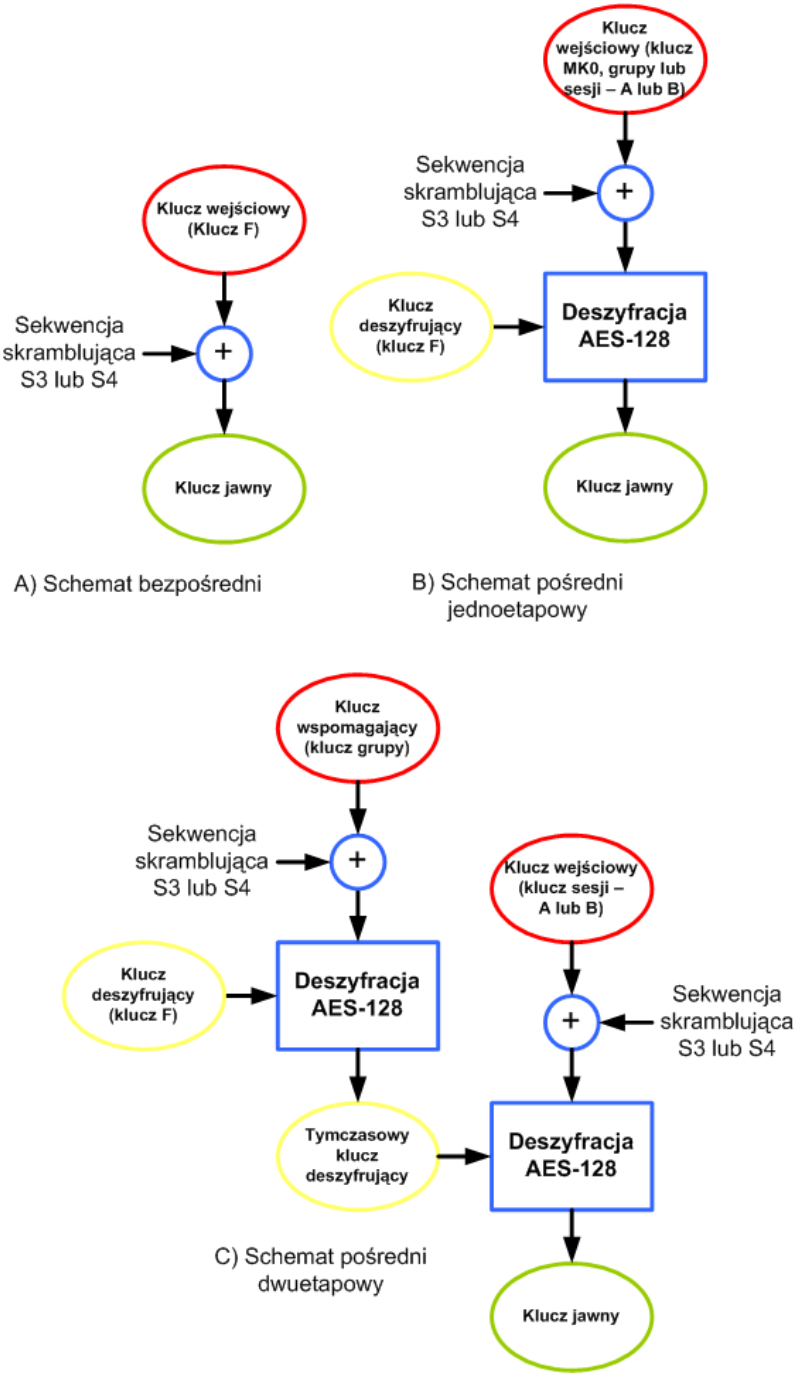


Rys. 3.5. Wyznaczanie jawnej postaci kluczy kryptograficznych otrzymywanych z centrum bezpieczeństwa.

W kolejnym paragrafie przedstawione zostaną bardziej szczegółowe informacje odnośnie stosowania różnych schematów deszyfracji kluczy w danych sytuacjach.

3.5.4.1. Schematy deszyfracji kluczy

Na rysunku 3.6 przedstawiono trzy stosowane w zaproponowanym kryptosystemie schematy uzyskiwania jawnych postaci kluczy szyfrujących/deszyfrujących (dla zmodyfikowanego algorytmu AES-CTR) oraz uwierzytelniających (dla zmodyfikowanego algorytmu CMAC).



Rys. 3.6. Schematy uzyskiwania klucza jawnego w kryptosystemie *0x86*.

Dla obu przypadków wykorzystuje się te same rozwiązania z jedną tylko różnicą. W przypadku deszyfracji kluczy dla realizacji poufności stosuje się bowiem sekwencję skramblującą S4, a dla uwierzytelniania S3 – patrz tabela 3.3.

W tabeli 3.4 z kolei zaprezentowano sposób wyboru danego schematu w zależności od konkretnej sytuacji.

Tab. 3.4. Schematy uzyskiwania jawnych postaci kluczy dla realizacji mechanizmu poufności i uwierzytelniania oraz kontroli integralności dla różnych typów wiadomości przesyłanych w kryptosystemie *0x86*.

<table>
<tr><th>Typ wiadomości</th><th>Realizacja mechanizmu poufności</th><th>Realizacja mechanizmu uwierzytelniania oraz integralności</th></tr>
<tr><td>1001
Unicast SMM</td><td>AES-CTR-128
(Schemat pośredni jednoetapowy:
Klucz wejściowy – klucz MK0,
klucz deszyfrujący – klucz F)</td><td rowspan="2">Podpis cyfrowy
(SHA-256 oraz RSA-1024)</td></tr>
<tr><td>1002
Multicast SMM</td><td>AES-CTR-128
(Schemat pośredni jednoetapowy:
klucz wejściowy – klucz grupy,
klucz deszyfrujący – klucz F)</td></tr>
<tr><td>1011
Broadcast UDM</td><td rowspan="2">AES-CTR-128
(Schemat bezpośredni: klucz F
lub schemat pośredni jednoetapowy:
klucz wejściowy – klucz sesji A lub B,
klucz deszyfrujący – klucz F)</td><td rowspan="2">CMAC-128
Schemat bezpośredni: klucz F
lub schemat pośredni jednoetapowy:
klucz wejściowy – klucz sesji A lub B,
klucz deszyfrujący – klucz F)</td></tr>
<tr><td>1015
Unicast UDM
poza grupę</td></tr>
<tr><td>1012
Multicast UDM</td><td rowspan="3">AES-CTR-128
(Schemat bezpośredni: klucz F
lub schemat pośredni dwuetapowy:
klucz wejściowy – klucz sesji A lub B,
klucz deszyfrujący – klucz F,
klucz wspomagający – klucz grupy)</td><td rowspan="3">CMAC-128
(Schemat bezpośredni: klucz F
lub schemat pośredni jednoetapowy:
klucz wejściowy – klucz grupy,
klucz deszyfrujący – klucz F
lub schemat pośredni dwuetapowy:
klucz wejściowy – klucz sesji A lub B,
klucz deszyfrujący – klucz F,
klucz wspomagający – klucz grupy)</td></tr>
<tr><td>1013
Multicast UDM w
kanale
wirtualnym</td></tr>
<tr><td>1014
Unicast UDM w
grupie</td></tr>
</table>

Metoda zaprezentowane w paragrafie 3.5.5.1 pozwala wykorzystać różne jawne postaci kluczy sesji dla transmisji w obrębie różnych grup. Ich formy tajne są bowiem identyczne w danej realizacji kryptosystemu. Dodatkowo technika tu przedstawiona umożliwia uzależnienie jawnych postaci kluczy stosowanych w danej realizacji kryptosystemu od jej klucza F. Pomimo faktu, że dane realizacje mają w ogólności różne tajne formy kluczy sesji, to ich ujawnienie nie spowoduje jeszcze naruszenia oferowanego poziomu bezpieczeństwa (o ile klucz F lub tablice skramblujące pozostają tajne).

Podobna sytuacja dotyczy uwierzytelniania z wykorzystaniem kluczy grup, które deszyfruje się najczęściej z użyciem klucza podstawowego sesji. W tym przypadku stanowi to również dodatkowe zabezpieczenie i uniemożliwia wyznaczenie

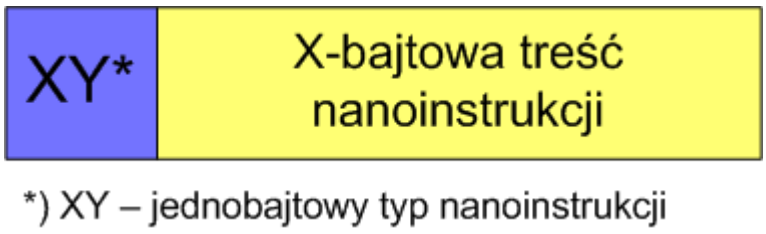
poprawnej sekwencji CMAC bez znajomości wszystkich wymaganych kluczy (najczęściej klucza grupy i F) oraz odpowiednich wektorów skramblujących (patrz tabela 3.3).

3.6. Typy wiadomości oraz ich struktura

Jak już wcześniej wspomniano w zaproponowanym kryptosystemie stosuje się dwa typy wiadomości:

- SMM (*Security Management Message*) – wiadomość wykorzystywana przy komunikacji centrum bezpieczeństwa z terminalami;
- UDM (*User Data Message*) – wiadomość wykorzystywana przy komunikacji pomiędzy terminalami.

W celu omówienia tych dwóch struktur należy w pierwszej kolejności przedstawić podstawową jednostkę, która służy do ich budowy. Jest nią tak zwana nanoinstrukcja. Na rysunku 3.7 przedstawiono schematycznie jej ogólną postać.



Rys. 3.7. Ogólna postać nanoinstrukcji.

Jak widać każda nanoinstrukacja (w skrócie nano) składa się z dwóch części. Pierwszy bajt to typ nano, a dalej wystąpić może od zera do piętnastu bajtów treści nanoinstrukcji. Długość tego drugiego pola zależy od pierwszych czterech bitów typu nano. Wiadomości UDM a przede wszystkim SMM w dużej mierze składają się właśnie z nanoinstrukcji. Jest to bowiem bardzo wygodna forma, pozwalająca na budowanie właściwie dowolnych instrukcji czy rozkazów oraz przesyłanie niewielkich ilości danych.

W tabeli 3.5 przedstawiono szczegółowo poszczególne nanoinstrukcje, wykorzystywane w zaproponowanym kryptosystemie.

Tab. 3.5. Nanoinstrukcje wykorzystywane w zaproponowanym kryptosystemie.

Typ nanoinstrukcji	**Opis**	**Sposób wykorzystania i Uwagi**	**Stosowane w UDM**
0x01 – 0x08	Anulowanie uprawnień w grupie 1 – 8	Skasowanie klucza grupy, maski uprawnień, identyfikatora *User ID* oraz daty końca autoryzacji w danej grupie.	Nie
0x0A, 0x0B	Kasowanie klucza sesji A, B	Również wyzerowanie daty ważności danego klucza sesji.	Nie
0x0C	Nanoinstrukcja wypełniająca	Pomiędzy nanem *0x0D* oraz *0x0E*. Minimalne dopełnienie do całkowitej długości wiadomości (patrz nano *0x11*).	Tak
0x0D	Znacznik początku podpisu lub sekwencji CMAC	Nana *0x0E* oraz *0x0D* mogą przylegać do siebie. Nie zawsze konieczne jest wypełnienie nanem *0x0C*.	Tak
0x0E	Znacznik końca nanoinstrukcji lub danych użytkownika		Tak
0x10	Typ wiadomości	*0x1001* – unicast SMM *0x1002* – multicast SMM *0x1011* – broadcast UDM *0x1012* – multicast UDM *0x1013* – multicast UDM w kanale wirtualnym *0x1014* – unicast UDM w grupie *0x1015* – unicast UDM poza grupę	Tak
0x11	Długość wiadomości	*0x11XX* (od *0x1100* do *0x11FF*): Długość$_{10}$ = (XX$_{10}$ + 1$_{10}$) * 128$_{10}$ [bitów] (max. 4 kB). Dobierana tak aby zminimalizować liczbę nan *0x0C*.	Tak
0x12	Typ klucza użytego do szyfrowania wiadomości	*0x1200* – szyfrowanie kluczem MK0 *0x1201* – szyfrowanie kluczem grupy *0x1202* – szyfrowanie kluczem sesji *0x120F* – szyfrowanie kluczem F	Tak
0x13	Sposób kontroli integralności wiadomości i uwierzytelniania nadawcy oraz typ użytego do tego celu klucza	*0x1300* – uwierzytelnianie RSA *0x1301* – uwierzytelnianie CMAC kluczem grupy *0x1302* – uwierzytelnianie CMAC kluczem sesji *0x130F* – uwierzytelnianie CMAC kluczem F	Tak
0x14	Numery tablic skramblujących	*0x14 XY*: *X* – numer tablicy A (od 1 do 4) *Y* – numer tablicy B (od 1 do 4) Numery otrzymane w ostatnim odebranym SMM od CB muszą być stosowane przez moduł do tworzenia i dekodowania wiadomości UDM.	Tak
0x15	Numery wykorzystywane do pobrania odpowiednich wektorów z tablicy A (nano *0x15*) oraz B (nano *0x16*) i wygenerowania na ich podstawie odpowiednich sekwencji skramblujących (patrz tabela 3.3)	Od 0 (nano *0x1500* lub *0x1600*) do 255 (nano *0x15FF* lub *0x16FF*), bo po 256 wektorów w każdej z tablic. Numery wybierane losowo dla każdej nowej wiadomości	Tak
0x16			Tak
0x1A	Identyfikator adresata w grupie	*0x1A* lub *0x1B* + Identyfikator *InGroup*	Tak (Tylko w UDM)
0x1B	Identyfikator nadawcy w grupie		Tak (Tylko w UDM)

0x1F	Numer wirtualnego kanału transmisji	Numer bitu maski uprawnień, na którym znajdować musi się jedynka, aby możliwy był odbiór tej wiadomości (od 1 do 128 zapisane binarnie)	Tak (Tylko w UDM)
0x20	Zmiana daty wygaśnięcia aktywacji	*0x2X YY ZZ*: *0xYY* – rok *0xZZ* – miesiąc Format zapisu jak we wzorze 3.1. Wygaśnięcie/Koniec ważności z ostatnim dniem miesiąca ZZ.	Nie
0x21 – 0x28	Zmiana daty wygaśnięcia uprawnień w grupie 1 – 8		Nie
0x2A, 0x2B	Zmiana daty ważności klucza sesji A lub B		Nie
0x2F	Pierwsze tajne Nano występujące po jawnym nagłówku wiadomości	*0x2F XX YY*: *0xXX – System ID* *0xYY – Provider ID* Kontrola poprawności deszyfracji.	Tak
0x3A	Identyfikator grupy adresata wiadomości	*0x3A* + Identyfikator *Group ID*	Tak
0x41 – 0x48	Zmiana identyfikatora użytkownika dla grupy 1 – 8	*0x4X*(1 – 8) + Identyfikator *User ID*	Nie
0x4A	Identyfikator modułu kryptograficznego adresata	*0x4A* lub *0x4B* + Identyfikator *Module ID*	Tak
0x4B	Identyfikator modułu kryptograficznego nadawcy		Tak (tylko w UDM)
0x90	Nadanie/Aktualizacja maski uprawnień	*0x90 XY* + (1/2 maski uprawnień): *X* – numer grupy *Y* (0 lub 1) – pierwsza (0) lub druga (1) część maski uprawnień	Nie
0x91	Nadanie/Aktualizacja klucza	*0x91 XY* + (1/2 klucza): *X* – typ klucza (1 – 8, A lub B) *Y* (0 lub 1) – pierwsza (0) lub druga (1) część zaszyfrowanego klucza *0x91F*, *0x910* - postacie błędne	Nie
0x30, 0x50, 0xF0	Anulowanie uprawnień użytkowników w grupie*	*0x30*, *0x50* lub *0xF0* + (3, 5 lub 15 nieaktywnych identyfikatorów *InGroup* w grupie określonej przez nano 3A). Ostatni identyfikator *InGroup* ewentualnie powtarzany w celu uzyskania 3, 5 lub F (15) bajtów treści nanoinstrukcji.	Nie (Tylko w multicast SMM)
0x31, 0x52, 0xF1	Dezaktywacja danej realizacji kryptosystemu w module**	*0x31*, *0x51* lub *0xF1* + (3, 5 lub 15 identyfikatorów *InGroup* w grupie określonej przez nano 3A, dla których wykonana będzie dezaktywacja danej realizacji systemu). Ostatni identyfikator *InGroup* ewentualnie powtarzany w celu uzyskania 3, 5 lub F (15) bajtów treści nanoinstrukcji.	Nie (Tylko w multicast SMM)

*) Użytkownik, który posiada uprawnienia w danej grupie i odbierze nano *0x30*, *0x50* lub *0xF0*, w którym znajdzie się jego identyfikator *InGroup*, utraci uprawnienia do nadawania i odbioru w obrębie tejże grupy (automatyczne wyzerowanie maski uprawnień, klucza grupy, identyfikatora *User ID* oraz daty końca autoryzacji w grupie).
**) Użytkownik danej realizacji kryptosystemu, który posiada uprawnienia w danej grupie i odbierze nano *0x31*, *0x51* lub *0xF1*, w którym znajdzie się jego identyfikator *InGroup*, utraci możliwość wykorzystywania wspomnianej realizacji systemu – dokonana zostanie automatyczna jej dezaktywacja.

Każda wiadomość (SMM oraz UDM) rozpoczyna się jawnym nagłówkiem. Jest on zawsze długości 128 bitów. W jego skład wchodzą następujące pola (w kolejności od pierwszego):

- *System ID* (zawsze *0x86*) – jeden bajt;
- *Provider ID* – jeden bajt;
- Typ wiadomości (nano *0x10*) – dwa bajty;
- Długość wiadomości (nano *0x11*) – dwa bajty;
- Rodzaj klucza użytego do szyfrowania (nano *0x12*) – dwa bajty;
- Rodzaj klucza oraz sposób kontroli integralności wiadomości jak i uwierzytelniania nadawcy (nano *0x13*) – dwa bajty;
- Numery tablic skramblujących (nano *0x14*) – dwa bajty;
- Numer wykorzystany do pobrania odpowiednich wektorów z tablicy A (nano *0x15*) – dwa bajty;
- Numer wykorzystany do pobrania odpowiednich wektorów z tablicy B (nano *0x16*) – dwa bajty.

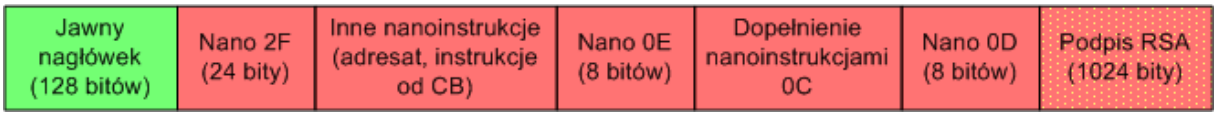
Niezależnie również od typu wiadomości szyfrowanie zrealizowane jest w sposób właściwie identyczny. Wykorzystuje się do tego celu wspomniany wcześniej 128-bitowy zmodyfikowany algorytm AES-CTR (patrz punkt 3.5.2). Szyfrowaniu podlega wiadomość bez nagłówka, ale wraz z wyznaczonym wcześniej i dodanym do niej podpisem lub sekwencją CMAC.

Ze względu na wykorzystywany 128-bitowy jawny nagłówek oraz 128-bitowy algorytm szyfrujący, długość wiadomości jest zawsze wielokrotnością 16-tu bajtów.

Poniżej przedstawiono sposób tworzenia oraz strukturę obu typów wykorzystywanych w kryptosystemie wiadomości.

3.6.1. Wiadomości SMM (*Security Management Message*)

Podstawowa struktura wiadomości SMM została przedstawiona na rysunku 3.8.



Rys. 3.8. Struktura wiadomości SMM kryptosystemu *0x86*.

Wiadomości SMM szyfrowane są z wykorzystaniem kluczy grup (multicast SMM) lub klucza głównego (unicast SMM) – patrz nano *0x12* w jawnym nagłówku. Sposób uzyskania postaci jawnych tych kluczy przedstawiono w paragrafie 3.5.5 oraz 3.5.5.1.

Uwierzytelnianie oraz kontrola integralności w wiadomościach SMM muszą być zawsze realizowane z wykorzystaniem podpisu cyfrowego (patrz punkt 3.5.4 oraz nano *0x13*).

Adresowanie wiadomości SMM zależne jest od jej typu. Dla wiadomości typu *0x1001* (unicast SMM) stosuje się nano *0x4A*, które musi wystąpić zaraz po nano *0x2F*.

W przypadku wiadomości typu *0x1002* (multicast SMM) stosuje się nano *0x3A*, występujące również zaraz po nano *0x2F*.

W wiadomościach SMM nie podaje się nadawcy (oczywisty – CB realizacji systemu o danym *Provider ID*).

Po nano *0x3A* lub *0x4A* wystąpić mogą już bardzo różne typy nanoinstrukacji (patrz tabela 3.5). Sposób formułowania instrukcji i rozkazów jest w dużej mierze dowolny i zależny od danej realizacji. Instrukcje od CB kończą się nanem *0x0E*, po którym występuje ewentualne dopełnieni nanoinstrukcjami *0x0C* (dla uzyskania długości całej wiadomości, będącej wielokrotnością 128 bitów). Ostatnim nanem jest *0x0D*, które oznacza początek 1024-bitowej sekwencji RSA.

Przy tworzeniu wiadomości SMM centrum bezpieczeństwa musi przestrzegać kilku podstawowych zasad:

- Całkowita aktywacja modułu nie powinna odbywać się z wykorzystaniem jednej wiadomości SMM;
- Z wykorzystaniem klucza *MasterKey 0* przesyłane powinny być tylko informacje, których nie można wysłać z wykorzystaniem kluczy grup (np. same właśnie klucze grup);
- Klucze sesji powinny być przesyłane z wykorzystaniem kluczy grup;
- Należy minimalizować liczbę wystąpień nano *0x0C* – trzeba dobierać odpowiednią długość wiadomości;
- Należy unikać zmian numerów tablic skramblujących (nano *0x14*);
- Każdy moduł, który odbierze nano *0x14* musi potem tworzyć wiadomości UDM z wykorzystaniem zdefiniowanych w tym nano numerów tablic;
- Dezaktywacja danego modułu powinna być realizowana z wykorzystaniem zarówno wiadomości typu unicast jak i multicast (patrz tabela 3.5 oraz rozdział czwarty);
- Podczas aktualizacji kluczy sesji muszą być wysyłane daty ich ważności;

- Aktualizacja kluczy musi zawierać zawsze klucz aktualny i przyszły.

Dodatkowo pewne praktyczne zasady tworzenia wiadomości SMM przedstawiono również w rozdziale czwartym niniejszej pracy.

3.6.2. Wiadomości UDM (*User Data Message*)

Podstawowa struktura wiadomości UDM została przedstawiona na rysunku 3.9.

Rys. 3.9. Struktura wiadomości UDM kryptosystemu *0x86*.

Wiadomości UDM szyfrowane są z wykorzystaniem kluczy sesji (SessionKey *A, B*) lub klucza F (*SessionBasicKey F*) – patrz nano *0x12* w jawnym nagłówku. Sposoby uzyskiwania ich postaci jawnych przedstawiono w paragrafie 3.5.5 oraz 3.5.5.1.

Uwierzytelnianie oraz kontrola integralności w wiadomościach UDM realizowane są z wykorzystaniem zmodyfikowanego algorytmu CMAC (patrz punkt 3.5.3). Stosuje się do tego celu odpowiednie sekwencje skramblujące oraz klucze grup, sesji lub klucz F (patrz nano *0x13* oraz paragraf 3.5.5 i 3.5.5.1).

Sposób adresowania oraz przedstawiania nadawcy w wiadomościach UDM zależny jest od ich typu. Adresat występuje zaraz po nano *0x2F*, a nadawca po adresacie. Możliwe schematy adresowania przedstawiono w tabeli 3.6.

Tab. 3.6. Schematy adresowania wiadomości UDM w kryptosystemie *0x86*.

Typ wiadomości UDM	Adresat	Nadawca
0x1011 – broadcast UDM	Brak	*0x4B + Module ID*
0x1012 – multicast UDM	*0x3A + Group ID*	*0x1B + InGroup ID*
0x1013 – multicast UDM w kanale wirtualnym	*0x3A + Group ID*	*0x1B + InGroup ID* *0x1F* + numer kanału wirt.
0x1014 – unicast UDM w grupie	*0x3A + Group ID* *0x1A + InGroup ID*	*0x1B + InGroup ID*
0x1015 – unicast UDM poza grupę	*0x4A + Module ID*	*0x4B + Module ID*

Po nadawcy występuje nano *0x0E* a następnie dane użytkownika. Po nich kolejne nano *0x0E* oraz wypełnienie nanami *0x0C* (o ile jest konieczne) dla zapewniania długości wiadomości, będącej wielokrotnością 128 bitów. Na końcu wiadomości występuje nano *0x0D* oraz 128-bitowa sekwencja CMAC.

Przy tworzeniu wiadomości UDM moduł kryptograficzny powinien:

- Korzystać przy szyfrowaniu z kluczy A oraz B (unikać klucza F);

- Nie mieć możliwości tworzenia wiadomości UDM z wykorzystaniem kluczy sesji, które utraciły swą ważność;
- Kiedy tylko to możliwe korzystać z kluczy grup do wyznaczania sekwencji CMAC (nie możliwe w wiadomościach typu *0x1011* oraz *0x1015*);
- Minimalizować liczbę wystąpień nano *0x0C*;
- Nie mieć możliwości odbierania i tworzenia wiadomości UDM w obrębie grupy, w której wygasła ważność autoryzacji;
- Nie mieć możliwości odbierania i tworzenia wiadomości UDM w kanale wirtualnym, do którego nie posiada uprawnień;
- Nie mieć możliwości tworzenia ani odbierania żadnych (w danej realizacji kryptosystemu) wiadomości UDM czy też odbierania wiadomości multicast SMM, gdy nie jest aktywna dana realizacja kryptosystemu.

Zarówno dekodowanie wiadomości SMM jak i tworzenie oraz dekodowanie wiadomości UDM powinno być automatycznie realizowane przez moduł kryptograficzny. Użytkownik powinien mieć tylko możliwość wyboru typu wysyłanych wiadomości UDM, adresata tych wiadomości oraz ewentualnie numeru kanału wirtualnego. Powinien on być również informowany (w postaci np. pewnego dziennika zdarzeń) o statusie modułu po otrzymaniu każdej wiadomości UDM czy SMM (patrz tabela 3.2).

Część praktycznych zagadnień, związanych z zaproponowanym rozwiązaniem, przedstawiona zostanie w rozdziale kolejnym niniejszej rozprawy doktorskiej. Zaprezentowano w nim bowiem między innymi sposób implementacji kryptosystemu oraz badania symulacyjne, potwierdzające jego funkcjonalność oraz wysoki oferowany przez niego poziom bezpieczeństwa transmisji.

Rozdział IV

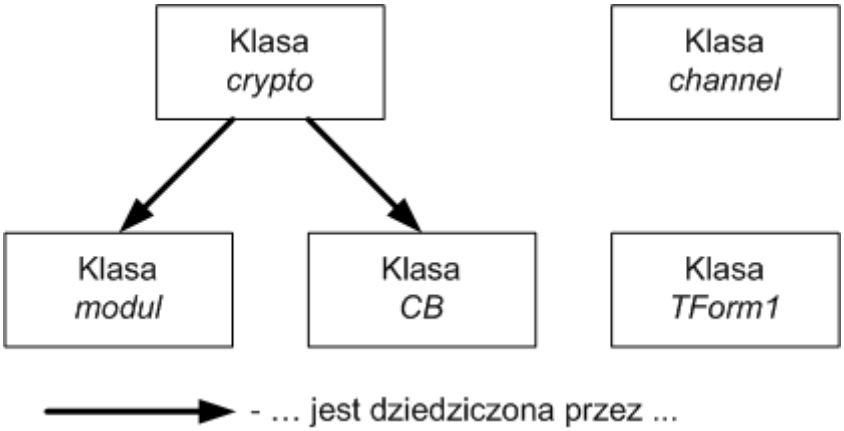
Implementacja i badania funkcjonalne zaproponowanego kryptosystemu

Dla celów niniejszej rozprawy doktorskiej w rozdziale tym zaprezentowany zostanie między innymi sposób implementacji autorskiego rozwiązania kryptosystemu, przedstawionego w rozdziale trzecim oraz częściowo w [7 – 9, 13].

W drugiej jego części znajdzie się zestawienie badań funkcjonalnych, zaproponowanego rozwiązania, które następnie zostaną wykorzystane (patrz rozdział piąty) do kompleksowej oceny realnego i w dużym stopniu obiektywnego poziomu jego bezpieczeństwa.

4.1. Implementacja kryptosystemu

Zaproponowane w rozdziale trzecim rozwiązanie zaimplementowano (patrz też [9]) z wykorzystaniem środowiska *C++ Builder* [19]. Powstała w ten sposób aplikacja została zrealizowana jako zorientowana obiektowo [32, 58] z obsługą sytuacji wyjątkowych [31] i składa się ona zasadniczo z pięciu klas. Cztery z nich to komponenty samego kryptosystemu, a ostatnia to tak zwana klasa „formatki”. Ogólna struktura klas aplikacji przedstawiona została na rysunku 4.1.



Rys. 4.1. Struktura klas w zaproponowanej implementacji kryptosystemu *0x86*.

Klasa *crypto* odpowiada za wszystkie algorytmy kryptograficzne, wykorzystywane przez system. Zawiera ona również pewne podstawowe i wspólne elementy zaproponowanego rozwiązania. Klasa ta jest właściwie najistotniejsza dla całego

kryptosystemu i dlatego jest ona dziedziczona przez klasy *modul* oraz *CB*. Struktura klasy c*rypto* przedstawiona została poniżej.

```
class crypto
{
private:    //pola prywatne klasy (elementy tajne klucza RSA)
LINT p;
LINT q;
LINT fi;
LINT a;
public:
LINT n;     //elementy publiczne klucza RSA
LINT b;
LINT tableA[4][256];    //tablice skramblujące
LINT tableB[4][256];
LINT SS[4];       //sekwencje skramblujące S1, S2, S3 oraz S4
LINT systemID;    //identyfikator systemu
LINT kluczAES;    //wygenerowany klucz AES
UINT genID;       //wygenerowany identyfikator przyszłego modułu

//deklaracje przyjaźni z elementami klasy TForm1 oraz klasą CB
friend void __fastcall TForm1::Button1Click(TObject *Sender);
friend void __fastcall TForm1::Button6Click(TObject *Sender);
friend class CB;

crypto();   //deklaracja konstruktora

void RSAKey(int len);   //metoda inicjalizująca klucze RSA
LINT EncDec(LINT data, LINT key);   //szyfrowanie/deszyfracja RSA
LINT SHA256(String data);     //funkcja skrótu SHA-256
LINT CMAC(String data, LINT key);   //zmodyfikowany algorytm CMAC
String AES(String data, LINT key, bool szyfr);//szyfrowanie/deszyfracja AES
//szyfrowanie/deszyfracja zmodyfikowanym AES-CTR
String AESCTR(String data, LINT key, String adres, int czas=0, bool
proba=false);
//deszyfracja kluczy AES dla realizacji poufności oraz uwierzytelniania
LINT DecAESKey(LINT Enc, LINT MK, bool szyfr=true);
void AESkey(void);      //inicjalizacja klucza AES
void IDgen(void);       //inicjalizacja identyfikatora
UINT gen(int ile);      //generowanie dowolnej sekwencji pseudoprzypadkowej
USHORT Data(void);      //aktualna data i czas w formie kryptosystemu HF
UINT rotr(UINT co, UINT ile);       //cykliczne przesunięcie bitowe w prawo

//metody pomocnicze (konwersje)
String HexToASCII(String Hex);
char* HexToASCII(String Hex, int &ile);
String ASCIIToHex(String ASCII);
String ASCIIToHex(char *ASCII, int ile);
String HexToBin(String Hex);
String BinToHex(String Bin);
void TableInit(void);   //inicjalizacja tablic skramblujących
void LoadTable(void);   //ładowanie tablic skramblujących

void Status(void);      //zapisywanie dziennika działania systemu
};
```

Klasa *CB* z kolei służy do definiowania centrów bezpieczeństwa. Posiada ona pola związane z parametrami unikalnymi dla danej i konkretnej instancji.

Najważniejszymi jednak elementami są tu metody, pozwalające na tworzenie oraz szyfrowanie poprawnych składniowo wiadomości SMM (patrz rozdział trzeci). Strukturę klasy *CB* przedstawiono poniżej.

```
class CB : public crypto      //dziedziczenie po klasie crypto
{
public:
USHORT providerID;      //identyfikator danej realizacji systemu
LINT basicKey;    //klucz podstawowy sesji (unikalny dla danej realizacji)
LINT decKey;      //zdeszyfrowany klucz AES
static int ile;   //liczba zdefiniowanych obiektów CB
bool activ;       //stan aktywności danego obiektu
//aktualnie wykorzystywane tablice skaramblujące oraz wektory z tych tablic
USHORT actA, actB, vecA, vecB;
void vectors(int A, int B);   //inicjalizacja wektorów skramblujących

String SMM(String in);        //tworzenie wiadomości SMM
//szyfrowanie wiadomości SMM
String SMMEnc(String in, LINT key, String adres);
CB(int prov)      //deklaracja i definicja konstruktora
{
ile++;
providerID=prov;
AESkey();
basicKey=kluczAES;
activ=false;
}

~CB()       //deklaracja i definicja destruktora
{
ile--;
}

};
```

Kolejną bardzo istotną z punktu widzenia całego rozwiązania jest klasa *modul*, która odpowiada za definiowanie konkretnych modułów kryptograficznych. Posiada ona pola związane ze wszystkimi identyfikatorami oraz kluczami wykorzystywanymi przez moduł. Zawiera również metody, pozwalające na deszyfrację i dekodowanie zarówno wiadomości SMM jak i UDM.

Dodatkowo zawiera także metodę odpowiedzialną za wykonywanie instrukcji otrzymanych od centrum bezpieczeństwa oraz pozwalającą tworzyć oraz szyfrować wiadomości UDM, przesyłane do innych terminali. Struktura klasy *modul* przedstawiona została poniżej.

```
class modul : public krypto   //dziedziczenie po klasie crypto
{
private:
LINT MK0;   //Klucz główny
LINT groupKey[4][8];    //Klucze grupy
LINT keyA[4], keyB[4], keyF[4];     //Klucze sesji
```

```
public:
USHORT providerID[4];   //identyfikatory realizacji systemu
int moduleID;     //identyfikator modułu
int groupID[4][8];      //identyfikatory grup
int nadawca;      //aktualna forma adresu nadawcy
int status;       //status modułu
USHORT ingroupID[4][8];       //adresy w grupach
LINT mask[4][8];        //maski uprawnień w grupach
USHORT activDate[4];    //data końca aktywacji dla danej realizacji
USHORT authDate[4][8];        //daty ważności autoryzacji modułu w grupach
USHORT valDateA[4], valDateB[4];    //daty ważności kluczy sesji
USHORT actA[4], actB[4];      //numery aktywnych tablic skramblujących
USHORT vecA[4], vecB[4];      //numery wektorów skramblujących
USHORT len, typ;        //długość oraz typ wiadomości
USHORT szyfr, uwierz;   //sposób szyfrowanie oraz uwierzytelniania
USHORT actProv, actGroup;     //aktywna realizacja oraz grupa
USHORT kanV;      //aktualnie wykorzystywany kanał wirtualny
static int ile;   //liczba zdefiniowanych obiektów klasy modul

//zagnieżdżona struktura, zawierająca parametry modułu sprzed ostatniej
aktualizacji
struct old
{
int groupID[8];
USHORT ingroupID[8];
LINT mask[8];
USHORT activDate;
USHORT authDate[8];
USHORT valDateA, valDateB;
USHORT actA, actB;
LINT groupKey[8];
LINT keyA, keyB;
} org;
//deklaracje przyjaźni z elementami klasy TForm1
friend void __fastcall TForm1::Button10Click(TObject *Sender);
friend void __fastcall TForm1::ComboBox5Select(TObject *Sender);
friend void __fastcall TForm1::ComboBox4Select(TObject *Sender);
friend void __fastcall TForm1::ComboBox3Select(TObject *Sender);
friend void __fastcall TForm1::Button23Click(TObject *Sender);
//deklaracja konstruktora
modul(short *providerID, LINT *keyF, int ile);

~modul()    //deklaracja i definicja destruktora
{
ile--;
}
//wykonywanie instrukcji uzyskanych od CB
void instrExe(USHORT instrID, String dane);
void header(String nag);      //dekodowanie nagłówka wiadomości
//deszyfrowanie i dekodowanie wiadomości
int MessDec(String mess, String &dec);
void vectors(int prov);       //wybór wektorów skramblujących
//tworzenie wiadomości UDM
String UDM(String pocz, String dane, int prov, int group, int key, int
typ);
//szyfrowanie wiadomości UDM
String UDMEnc(String in, String adres, int prov, int group, int key, int
typ);
void check(void);       //kontrola statusu modułu
String Receive(String dane);        //odbiór danych z kanału
};
```

Klasa *channel* jest ostatnią klasą wykorzystywaną w aplikacji. Jest to prosta struktura, która umożliwia realizację kanału transmisyjnego pomiędzy centrum a modułami oraz pomiędzy samymi modułami. Jedynym właściwie zadaniem tego elementu jest umożliwienie przechwytywania przesyłanych wiadomości w celu ich modyfikacji, próby deszyfracji, późniejszego przesłania lub realizacji innej formy ingerencji w zaimplementowany kryptosystem.

Prezentowane rozwiązanie korzysta dodatkowo z dwóch bibliotek zewnętrznych: *LINT* [94] oraz *rijndael* [94].

Pierwsza z nich odpowiada między innymi za możliwość deklaracji bardzo dużych zmiennych stałoprzecinkowych, reprezentowanych nawet na 4096-ciu pozycjach binarnych (1233 cyfry dziesiętne). Co najważniejsze jednak biblioteka ta umożliwia wykonywanie wielu operacji na tychże zmiennych, a są to między innymi:

- Operacje wejścia/wyjścia (np. operacje na plikach);
- Konwersje (także do/z typów wbudowanych języka *C/C++*);
- Porównywanie liczb;
- Matematyczne działania podstawowe;
- Operacje arytmetyki modularnej;
- Działania bitowe;
- Operacje z zakresu teorii liczb.

Większość powyższych elementów wykorzystana została przy implementacji zaproponowanego kryptosystemu. Dodatkowo jedną z istotniejszych zalet biblioteki *LINT* jest możliwość generacji liczb pseudolosowych przy użyciu jednego z najbezpieczniejszych rozwiązań tego typu – generatora BBS (*Bluma-Bluma-Shuba*) [89].

Biblioteka *rijndael* użyta została z kolei dla realizacji podstawowego procesu szyfrowania i deszyfrowania z wykorzystaniem algorytmu AES z kluczem 128-bitowym. Zdecydowano się na takie rozwiązanie ze względu na wysoką optymalizację (pod względem szybkości działania) wspomnianej biblioteki, co znacznie usprawnia funkcjonowanie implementacji kryptosystemu. Należy tu bowiem pamiętać, że algorytm AES jest kluczowym elementem zaproponowanego rozwiązania (patrz rozdział trzeci).

Podczas implementacji innych standardowych algorytmów kryptograficznych wykorzystywano głównie ich specyfikacje oraz [77, 94].

Zrealizowana aplikacja korzysta również z plików zewnętrznych, w których przechowuje się następujące informacje:

- Tablice skramblujące A oraz B;
- Klucz prywatny oraz publiczny RSA;
- Dziennik działania aplikacji.

Warto w tym miejscu zauważyć, że na potrzeby symulatora systemu powyższe elementy są przechowywane w sposób jawny, ale w implementacji praktycznej dostęp do nich (tablice skramblujące oraz klucz tajny) powinien być ściśle ograniczony. Zgodnie jednak z założeniami początkowymi zagadnienia związane z fizycznym dostępem do zasobów systemowych nie będą tu brane pod uwagę. Zakłada się tylko, że praktyczna implementacja systemu musi zapewniać ochronę w tym zakresie (np. numery PIN, hasła, bezpieczne pamięci ROM, biometryczne narzędzia dostępu itp.) – patrz rozdział pierwszy niniejszej rozprawy doktorskiej.

Dodatkowo baza użytkowników systemu nie jest zaimplementowana, ponieważ w trakcie symulacji nowe moduły kryptograficzne „tworzone są na bieżąco", co wyklucza konieczność istnienia tego elementu w symulatorze. Bezpieczna baza danych [34, 35, 60], do której musi mieć oczywiście dostęp każde centrum bezpieczeństwa, to kolejny aspekt mechanizmu dostępności, który jak już wspomniano nie jest tu rozważany.

W narzędziu symulacyjnym skupiono się przede wszystkim na sprawach związanych z pozostałymi mechanizmami kryptograficznymi (poufność, integralność oraz uwierzytelnianie wraz z autoryzacją), które to zostały zaimplementowane wraz z metodami zarządzania modułami kryptograficznymi, generowania kluczy, tworzenia i dekodowania wiadomości oraz wieloma innymi wykorzystywanymi w zaproponowanym kryptosystemie. W kolejnym paragrafie przedstawione zostaną sposoby implementacji najważniejszych z nich.

4.1.1. Najistotniejsze metody

W pierwszej kolejności przedstawione zostaną tu rozwiązania, pozwalające generować klucze oraz inne identyfikatory systemowe. Elementy te w dużej mierze wykorzystują algorytm BBS ze względu na jego duże bezpieczeństwo oraz fakt, że jest on bardzo często polecany jako doskonałe narzędzie w tego typu rozwiązaniach [60, 81, 89].

W celu realizacji mechanizmu podpisu cyfrowego wykorzystuje się algorytm RSA z 1024-bitowym kluczem. Sposób generacji elementów klucza jawnego oraz tajnego przedstawia poniższy kod. Argument *len* w tej metodzie jest domyślnie przekazywany jako 1024 (długość stosowanego klucza RSA).

```
void crypto::RSAKey(int len)
{
LINT seed;
//wyznaczenie ziarna seed dla generatora BBS
seed=LINT(time(NULL))*LINT(random(pow(10.0,7.0)))+LINT(random(pow(10.0,10.0
)));
//inicjalizacja generatora BBS ziarnem seed
seedBBS(seed);

//generowanie dwóch różnych i silnych liczb pierwszych o zadanej długości
do
{
p=strongprime(len/2);
q=strongprime(len/2);
}while(p==q);

//wyznaczanie wartości parametrów algorytmu RSA
n=p*q;
fi=(p-1)*(q-1);

//wyznaczenie klucza publicznego
b=randBBS(len/2);

//sprawdzenie wymogów algorytmu RSA (gdzie gcd oznacza NWD)
while(b>=fi || gcd(fi,b)!=1)
{
b=randBBS(len/2);       //wyznaczenie klucza publicznego
}

//wyznaczenie klucza prywatnego (gdzie inv oznacza odwrotność b mod fi)
a=inv(b,fi);
}
```

W powyższej metodzie wykorzystuje się funkcje dostępne we wspomnianej wcześniej klasie *LINT*. Zaimplementowany tu algorytm generacji kluczy RSA omówiono w rozdziale pierwszym niniejszej pracy.

Generator BBS wykorzystywany jest również do generacji wszelkich kluczy AES oraz identyfikatorów modułów kryptograficznych. Poniższy fragment kodu prezentuje te rozwiązania.

```
void crypto::AESkey(void)     //generowanie kluczy AES
{
LINT seed;
seed=LINT(time(NULL))*LINT(random(pow(10.0,7.0)))+LINT(random(pow(10.0,10.0
)));
seedBBS(seed);
kluczAES=randBBS(128);
}
//-------------------------------------------------------------
```

```
void crypto::IDgen(void)      //generowanie identyfikatorów
{
LINT seed;
seed=LINT(time(NULL))*LINT(random(pow(10.0,7.0)))+LINT(random(pow(10.0,10.0
)));
seedBBS(seed);
genID=StrToInt("0x"+(String)lint2str(randBBS(32),16,0));
}
```

Warto tu wspomnieć, że w trzech powyższych metodach użyto inicjalizacji ziarna (*seed*) dla generatora BBS z wykorzystaniem aktualnego czasu systemowego (*time(NULL)*) oraz wbudowanego (biblioteka standardowa) generatora liczb pseudolosowych (funkcja *random(float)*). Jest to rozwiązanie przykładowe.

Należy również przypomnieć, że w zaproponowanym kryptosystemie nie wykorzystuje się algorytmu szyfrowania kluczy AES, a jedynie generuje się je w postaci 128-bitowych sekwencji pseudolosowych. Zadaniem modułów z kolei jest potem odpowiednie ich przekształcenie (deszyfracja) w celu uzyskania postaci jawnej. Sposób realizacji tego zadania w modułach przedstawiono poniżej.

```
LINT crypto::DecAESKey(LINT Enc, LINT MK, bool szyfr)
{
LINT tmp;
String out, tmp2;
//suma modulo 2 niejawnej postaci klucza AES z odpowiednią sekwencją
skramblującą
if(szyfr) tmp=Enc^SS[3];      //dla poufności sekwencja S4
else tmp=Enc^SS[2];           //dla uwierzytelniania sekwencja S3

tmp2=lint2str(tmp,16,0);
while(tmp2.Length()%32) tmp2="0"+tmp2;
//deszyfracja klucza algorytmem AES z wykorzystaniem klucza MK (klucz F lub
zdeszyfrowany klucz grupy)
out=AES(tmp2,MK,false); //false oznacza deszyfrację
//funkcja zwraca zdeszyfrowany klucz w postaci liczby typu LINT
return LINT(out.c_str(),16);
}
```

Elementami wejściowymi dla tej metody są:

- Wygenerowana i niejawna postać klucza AES (*Enc*);
- Klucz deszyfrujący (*MK*);
- Zmienna logiczna *szyfr* (wartość *true* oznacza deszyfrację klucza dla realizacji poufności, a wartość *false* dla realiacji uwierzytelniania).

Procedura zaprezentowana powyżej jest czasami tylko elementem składowym całego algorytmu uzyskiwania klucza w postaci jawnej, a w niektórych przypadkach rozwiązanie tu przedstawione nie jest wcale stosowane. Całościowy i uogólniony

algorytm deszyfracji kluczy oraz trzy odmienne tryby jego funkcjonowania przedstawiono w rozdziale poprzednim.

Podobnie jak w przypadku kluczy AES generuje się również tablice skramblujące. Zawierają one bowiem wektory 128-bitowe, a więc ich długość jest taka sama jak w przypadku wspomnianych kluczy. Mechanizm generacji tablic jest zatem podobny (wielokrotna inicjalizacja pseudolosowego wektora 128-bitowego) do tworzenia kluczy AES i dlatego nie będzie on tu przedstawiany.

W paragrafie tym nie zostaną zaprezentowane również sposoby realizacji algorytmów: RSA, AES oraz SHA-256, ponieważ są to albo rozwiązania powszechnie znane, albo wykorzystuje się do ich realizacji funkcje biblioteczne.

Ważnym jednak aspektem związanym z prezentowanym rozwiązaniem są modyfikacje (patrz rozdział trzeci) algorytmów AES-CTR oraz CMAC. Ten pierwszy jest jak wiadomo głównym mechanizmem, zapewniającym poufność w zaproponowanym kryptosystemie. Sposób jego realizacji w środowisku symulacyjnym przedstawiono poniżej.

```
String crypto::AESCTR(String data, LINT key, String adres, int czas, bool
proba)
{
//elementy związane z uzyskiwaniem aktualnej daty oraz czasu i zapisem ich
w odpowiednim formacie
int godz, min, rok, mies, dzien, krot;
time_t t;
struct tm *gmt;
struct tm *local;
LINT licznik;     //jawna postać licznika dla danego bloku w formie LINT
char out[17]={NULL};    //licznik zaszyfrowany
char tmp[17]={NULL};    //jawny blok danych
char *tmp2;       //jawna postać licznika dla danego bloku
AESclass pier;    //obiekt potrzebny do szyfrowania AES

//zmienne wspomagające
String klucz, wyj, licz, blok, tmp3, tmp4, tmp5;
int len, i, j, ile;
char *keyy, *dane;

//liczba powtórzeń numeru bloku w liczniku AES-CTR
krot=11-adres.Length()/2;

//ustalenie czasu i daty
t = time(NULL);
local = localtime(&t);

if(czas>0) local->tm_min+=5;  //następny znacznik czasu
else if(czas<0) local->tm_min-=5;   //poprzedni znacznik czasu

t=mktime(local);
gmt = gmtime(&t);
```

```
//zmiana formatu czasu i daty do postaci wymaganej w kryptosystemie
godz=gmt->tm_hour;
min=gmt->tm_min;
rok=gmt->tm_year-100;
dzien=gmt->tm_mday;
mies=gmt->tm_mon+1;

//zmiana formatu klucza szyfrującego AES na ASCII
klucz=lint2str(key,16,0);
while(klucz.Length()%32) klucz="0"+klucz;
keyy=HexToASCII(klucz,ile);
keyy[ile]=NULL;

//zmiana formatu danych wejściowych na ASCII
dane=HexToASCII(data,ile);
dane[ile]=NULL;

//wyznaczenie liczby bloków do szyfrowania
len=ile;
len=ceil((float)len/16.0);

if(proba) len=1;  //jeśli próba deszyfracji (deszyfracja jednego bloku)
wyj="";
for(i=0;i<len;i++)      //pętla kolejnych bloków danych do szyfrowania
{
for(j=0;j<16;j++) tmp[j]=dane[i*16+j];    //podział danych na bloki

pier.enc_key(keyy,16);  //ustalenie klucza szyfrującego

//tworzenie licznika
licz="";

for(j=0;j<krot;j++)
{
licz=licz+IntToHex(i+1,2);    //dołączanie numeru bloku
}

//dołączanie adresu i znacznika czasu do numeru bloku
licz=adres+IntToHex(rok,2)+IntToHex(mies,2)+IntToHex(dzien,2)+IntToHex(godz
,2)+IntToHex(min,2)+licz;
//sumowanie modulo 2 licznika z sekwencją skramblującą S1
licznik=SS[0]^LINT(licz.c_str(),16);
//zmiana formatu licznika na ASCII
tmp3=lint2str(licznik,16,0);
while(tmp3.Length()%32) tmp3="0"+tmp3;
tmp2=HexToASCII(tmp3,ile);
tmp2[ile]=NULL;
pier.enc_blk(tmp2,out);       //szyfrowanie licznika

tmp5=ASCIIToHex(out,16);
tmp4=ASCIIToHex(tmp,16);

blok="";
//suma modulo 2 zaszyfrowanego licznika z pojedynczym blokiem danych
for(j=1;j<=32;j++)
{
blok+=IntToHex(StrToInt("0x"+(String)tmp5.operator[](j))^StrToInt("0x"+(Str
ing)tmp4.operator [](j)),1);
}
```

```
//kontrola długości bloku w formacie String
while(blok.Length()%32!=0) blok="0"+blok;

//dołączanie danego bloku do końcowej sekwencji wyjściowej
wyj=wyj+blok;
}
return wyj;       //funkcja zwraca zaszyfrowaną sekwencję danych
}
```

Argumentami powyższej metody są między innymi: dane do szyfrowania/deszyfrowania (*data*), klucz (*key*) oraz adresat wiadomości (*adres*). Dodatkowo na wejściu podaje się również zmienne *proba* oraz *czas*. Jeśli pierwsza z nich przyjmuje wartość *true*, to algorytm zrealizowany zostanie dla jednego tylko bloku (128-bitowego) sekwencji *data* w celu realizacji próby deszyfracji. Parametr *czas* z kolei, gdy ma wartość zero, powoduje realizację algorytmu dla aktualnego znacznika czasu. Wartość ujemna tejże zmiennej oznacza znacznik poprzedni, a dodatnia następny.

Warto tu przypomnieć, że autorską modyfikacją niniejszego algorytmu jest dopuszczona przez standard [55] modyfikacja licznika (*licz*), przedstawiona w poprzednim rozdziale niniejszej pracy.

Kolejnym algorytmem istotnym z punktu widzenia zaproponowanego kryptosystemu jest CMAC, który został zmodyfikowany i dopasowany do potrzeb prezentowanego rozwiązania (patrz rozdział trzeci). Sposób jego implementacji przedstawiono poniżej.

```
LINT crypto::CMAC(String data, LINT key)
{
char out[16]={NULL};    //wyjściowa sekwencja CMAC
char tmp[16];     //pojedynczy blok danych
char *tmp2;       //tymczasowa sekwencja CMAC
//zmienne pomocnicze
AESclass pier;
String klucz, wyj;
char *dane, *keyy;
int len, i, ile, j;
char Lin[16]={0};
char Lout[16]={0};
char R[128]={0};
LINT Rb, K1, L;
//formatowanie klucza do odpowiedniej postaci
klucz=lint2str(key,16,0);
keyy=HexToASCII(klucz,ile);
keyy[ile]=NULL;

//procedura wyznaczenia klucza K1 dla algorytmu CMAC
R[127]=1;
R[126]=1;
R[125]=1;
R[120]=1;
```

```
Rb=LINT(R,2);
pier.enc_key(keyy,16);
pier.enc_blk(Lin,Lout);

wyj=ASCIIToHex(Lout,16);
L=LINT(wyj.c_str(),16);
wyj=HexToBin(wyj);
wyj=wyj.SubString(1,1);
if(wyj==”0”) K1=L<<1;
else K1=(L<<1)^Rb;

//formatowanie danych do odpowiedniej postaci
dane=HexToASCII(data,ile);
dane[ile]=NULL;

//wyznaczenie liczby bloków danych
len=ile;
len=ceil((float)len/16.0);

for(i=0;i<len;i++)      //pętla po kolejnych blokach
{
for(j=0;j<16;j++) tmp[j]=dane[i*16+j];    //pojedynczy blok

pier.enc_key(keyy,16);        //ustalenie klucza szyfrującego

if(i==0)    //realizacja algorytmu CMAC dla pierwszego bloku danych
{
//suma modulo 2 pierwszego bloku z sekwencją skramblującą S2
wyj=lint2str(LINT(ASCIIToHex(tmp,16).c_str(),16)^SS[1],16,0);
while(wyj.Length()%32) wyj=”0”+wyj;
tmp2=HexToASCII(wyj,ile);
tmp2[ile]=NULL;
pier.enc_blk(tmp,out);        //szyfrowanie
}
else if(i!=len-1)       //realizacja algorytmu CMAC dla bloków kolejnych
{
//suma modulo 2 aktualnego bloku z dodychczasową sekwencją
wyj=lint2str(LINT(ASCIIToHex(tmp,16).c_str(),16)
^LINT(ASCIIToHex(out,16).c_str(),16),16,0);

while(wyj.Length()%32) wyj=”0”+wyj;
tmp2=HexToASCII(wyj,ile);
tmp2[ile]=NULL;
pier.enc_blk(tmp2,out);       //szyfrowanie
}
else        //realizacja algorytmu CMAC dla ostatniego bloku danych
{
//suma modulo 2 aktualnego bloku z dodychczasową sekwencją oraz kluczem K1
wyj=lint2str(LINT(ASCIIToHex(tmp,16).c_str(),16)^LINT(ASCIIToHex(out,16).c_
str(),16)^K1,16,0);
while(wyj.Length()%32) wyj=”0”+wyj;
tmp2=HexToASCII(wyj,ile);
tmp2[ile]=NULL;
pier.enc_blk(tmp2,out);       //szyfrowanie
}
}
//ostateczne formatowanie sekwencji końcowej i zwrócenie wyniku
wyj=ASCIIToHex(out,16);

return LINT(wyj.c_str(),16);
}
```

Na wejście powyższej metody należy podać klucz oraz dane, dla których wyznaczona ma być sekwencja CMAC. Warto w tym miejscu nadmienić, że autorską modyfikacją niniejszego algorytmu jest głównie odpowiednie wykorzystanie do jego realizacji sekwencji skramblującej (szczegóły – rozdział trzeci).

Zaprezentowane tu implementacje algorytmów, wykorzystywane są przede wszystkim do realizacji mechanizmów bezpieczeństwa, ale również w dużej mierze do zarządzania całym systemem i odpowiedniego formatowania danych użytkowników. Procedury, zaprezentowane zatem w dalszej części niniejszego paragrafu, służą tworzeniu wiadomości zarówno SMM jak i UDM, a więc metody te wykorzystuje się w komunikacji między terminalami oraz pomiędzy odpowiednim centrum bezpieczeństwa a terminalami (właściwie modułami kryptograficznymi). Sposób zatem tworzenia wiadomości SMM, jako istotnego elementu zaproponowanego rozwiązania, przedstawiono poniżej.

```
String CB::SMM(String in)
{
//wykorzystywane zmienne
String out, ttt;
LINT RSA;
int tmp;
String err="Za długa wiadomość SMM!";
//dodanie do wejściowej sekwencji, zawierającej już jawny nagłówek oraz
inne nanoinstrukcje, nanoinstrukcji 0x0E
In+="0E0C";
while(in.Length()%32!=0) in+="0C"; //dopełnienie nanoinstrukcją 0x0C
//dodanie nanoinstrukcji oznaczającej początek podpisu
in.operator [](in.Length())='D';

//ustalenie długości wiadomości i zmodyfikowanie jawnego nagłówka
tmp=(in.Length()/32)+7;
if(tmp>255) throw err;

//wyznaczanie sekwencji S1 oraz S4
SS[0]=tableA[actA][vecA]^tableB[actB][vecB];
SS[3]=tableA[actA][255-vecA]^tableB[actB][255-vecB];
in.Delete(11,2);
in.Insert(IntToHex(tmp,2),11);

//procedura ustalenia podpisu cyfrowego
RSA=SHA256(in);
RSA=EncDec(RSA,a);
ttt=lint2str(RSA,16,0);
while(ttt.Length()%256!=0) ttt="0"+ttt;

out=in+ttt;       //dodanie podpisu na końcu wiadomości SMM

return out;       //zwrócenie jawnej postaci wiadomości SMM
}
```

Argumentem prezentowanej tu metody jest jawna postać nagłówka wiadomości oraz nanoinstrukcje (nadawca oraz instrukcje wykonawcze od centrum bezpieczeństwa). Sposób formowania nagłówka i adresowania wiadomości oraz możliwe postacie instrukcji od CB nie będą tu prezentowane, ponieważ zagadnienia te omówiono już w poprzednim rozdziale niniejszej pracy.

Zaprezentowana powyżej metoda zwraca jawną postać wiadomości SMM wraz z dołączoną sekwencją podpisu cyfrowego. W dalszej kolejności należy zatem zaszyfrować taką wiadomość. Sposób implementacji tego zadania zaprezentowano poniżej.

```
String CB::SMMEnc(String in, LINT key, String adres)
{
String out;       //zaszyfrowana wiadomość SMM
//zmienne pomocnicze
String tmp;
int len;
len=in.Length();        //ustalenie długości całkowitej wiadomości

tmp=in.SubString(33,len-32);        //odseparowanie jawnego nagłówka

decKey=DecAESKey(key, basicKey);    //deszyfracja klucza AES
out=AESCTR(tmp,decKey,adres);       //szyfrowanie AES-CTR

//dodanie jawnego nagłówka
tmp=in.SubString(1,32);
out=tmp+out;

return out;       //zwrócenie gotowej wiadomości SMM
}
```

Na wejście niniejszej metody podawana jest jawna postać wiadomości SMM wraz z podpisem, klucz szyfrujący oraz adresat.

W przypadku wiadomości UDM proces jest również dwuetapowy. W pierwszej kolejności realizowane jest tworzenie jawnej wiadomości wraz z dołączoną sekwencją CMAC. Jest to istotny element systemu i sposób jego implementacji przedstawiono poniżej.

```
String modul::UDM(String pocz, String dane, int prov, int group, int key,
int typ)
{
String out;       //sekwencja wyjściowa
//zmienne wspomagające
int tmp, i;
String err="Za długa wiadomość UDM!";
LINT skrot, klucz;
String tmp2;
//kontrola aktywności modułu dla danej realizacji
if(activDate[prov]<Data())
{     //automatyczna dezaktywacja danej realizacji
keyA[actProv]=nul_l;
```

```
keyB[actProv]=nul_l;
valDateA[actProv]=0;
valDateB[actProv]=0;

for(i=0;i<8;i++)
{
      mask[actProv][i]=nul_l;
      groupKey[actProv][i]=nul_l;
      authDate[actProv][i]=0;
}

throw 80;
}
//kontrola daty wygaśnięcia autoryzacji w danej grupie
if(authDate[prov][group]<Data() && (typ==1 || typ==2 || typ==3)) throw 41;
//wyznaczanie sekwencji skramblujących S1, S2, S3 oraz S4
SS[0]=tableA[actA[prov]][vecA[prov]]^tableB[actB[prov]][vecB[prov]];
SS[1]=tableA[actA[prov]][vecA[prov]]^tableB[actB[prov]][255-vecB[prov]];
SS[2]=tableA[actA[prov]][255-vecA[prov]]^tableB[actB[prov]][vecB[prov]];
SS[3]=tableA[actA[prov]][255-vecA[prov]]^tableB[actB[prov]][255-
vecB[prov]];
//realizacja podstawowej struktury wiadomości UDM
dane=ASCIIToHex(dane);
out=pocz+"0E"+dane+"0E0C";
while(out.Length()%32!=0) out=out+"0C";
out.operator [](out.Length())='D';

//ustalenie i kontrola długości wiadomości
tmp=(out.Length()/32);
if(tmp>255) throw err;
out.Delete(11,2);
out.Insert(IntToHex(tmp,2),11);

//deszyfracja klucza uwierzytelniającego w zależności od jego typu
switch(key)
{
case 0:     //uwierzytelnianie kluczem grupy
{
klucz=DecAESKey(groupKey[prov][group], keyF[prov], false);
break;
}
case 1:     //uwierzytelnianie kluczem sesji
{
LINT decKey;
//dla przypadku broadcast oraz unicast poza grupę
if(typ==0 || typ==4) decKey=keyF[prov];
//dla przypadku transmisji w grupie
else decKey=DecAESKey(groupKey[prov][group], keyF[prov], false);

//kontrola ważności kluczy sesji
if(valDateA[prov]<Data() && valDateB[prov]<Data()) throw 90;
//optymalny wybór klucza sesji (A lub B)
else if(valDateA[prov]>=Data() && valDateB[prov]<Data())
{
klucz=DecAESKey(keyA[prov], decKey, false);
}
else if(valDateA[prov]<Data() && valDateB[prov]>=Data())
{
klucz=DecAESKey(keyB[prov], decKey, false);
}
```

```
else if(valDateA[prov]<valDateB[prov])
{
klucz=DecAESKey(keyA[prov], decKey, false);
}
else
{
klucz=DecAESKey(keyB[prov], decKey, false);
}
break;
}
case 2:     //uwierzytelnianie kluczem podstawowym F
{
klucz=keyF[prov]^SS[2];       //schemat bezpośredni uzyskiwania klucza
break;
}
}

//wyznaczenie sekwencji CMAC
skrot=CMAC(out, klucz);

tmp2=lint2str(skrot,16,0);
while(tmp2.Length()%32) tmp2="0"+tmp2;

//zwrócenie kompletnej i jawnej wiadomości UDM wraz z sekwencją CMAC
return out+tmp2;
}
```

Na wejściu powyższej metody znajdują się: sekwencja początkowa wiadomości (*pocz*), która zawiera jawne dane wiadomości UDM do pierwszej nanoinstrukcji *0x0E*; dane użytkownika (*dane*); numer realizacji systemu dostępnej w danym module i wykorzystywanej dla tworzenia aktualnej wiadomości (*prov*); numer wykorzystywanej i dostępnej grupy (*group*); typ klucza uwierzytelniającego (*key*) oraz typ wiadomości UDM (*typ*).

Szczególnie istotnym elementem w tej metodzie jest optymalny pod względem daty ważności wybór klucza uwierzytelniającego, gdy ma być to klucz sesji. W przypadku tylko jednego ważnego klucza sesji problem posiada trywialne rozwiązanie – wybierany jest ten właśnie aktualny klucz. Gdy jednak oba klucze są ważne, to obowiązuje zasada, że moduł wybiera zawsze ten, który wcześniej utraci swą ważność. Postępuje się tak w celu maksymalizacji szansy na odbiór takiej wiadomości przez adresata. Nadawca nie posiada bowiem informacji o tym, kiedy moduł odbiorczy otrzymał ostatnią aktualizację od centrum bezpieczeństwa. Jest zatem większa szansa, że posiadać będzie on ten starszy (lecz wciąż ważny) klucz.

Kolejny etap to szyfrowanie wiadomości UDM. Kod źródłowy tej metody przedstawiono poniżej.

```
String modul::UDMEnc(String in, String adres, int prov, int group, int key,
int typ)
{
String out;       //sekwencja wyjściowa
//zmienne wspomagające
String tmp;
int len;
LINT decKey;
//kontrola aktywności danej realizacji oraz daty wygaśnięcia autoryzacji
if(activDate[prov]<Data()) throw 80;
if(authDate[prov][group]<Data() && (typ==1 || typ==2 || typ==3)) throw 41;
//wydzielenie wiadomości bez nagłówka
len=in.Length();
tmp=in.SubString(33,len-32);

switch(key)       //szyfrowanie w zależności od typu klucza
{
case 0:     //szyfrowanie kluczem sesji
{
//dla przypadku broadcast oraz unicast poza grupę
if(typ==0 || typ==4) decKey=keyF[prov];
//dla przypadku transmisji w grupie
else decKey=DecAESKey(groupKey[prov][group], keyF[prov]);
//kontrola ważności kluczy sesji
if(valDateA[prov]<Data() && valDateB[prov]<Data()) throw 90;
//optymalny wybór klucza sesji (A lub B)
else if(valDateA[prov]>=Data() && valDateB[prov]<Data())
{
decKey=DecAESKey(keyA[prov], decKey);
}
else if(valDateA[prov]<Data() && valDateB[prov]>=Data())
{
decKey=DecAESKey(keyB[prov], decKey);
}
else if(valDateA[prov]<valDateB[prov])
{
decKey=DecAESKey(keyA[prov], decKey);
}
else
{
decKey=DecAESKey(keyB[prov], decKey);
}

out=AESCTR(tmp,decKey,adres); //szyfrowanie
break;
}
case 1:     //szyfrowanie z kluczem podstawowym F
{
out=AESCTR(tmp,keyF[prov]^SS[3],adres);   //szyfrowanie
break;
}
}

tmp=in.SubString(1,32);       //wydzielenie nagłówka
out=tmp+out;      //końcowa postać wiadomości UDM
return out;       //zwrócenie gotowej wiadomości
}
```

Na wejściu powyższej metody znajdują się: jawna postać wiadomości (*in*); adresat (*adres*); numer realizacji systemu dostępnej w danym module i

wykorzystywanej dla tworzenia aktualnej wiadomości (*prov*); numer wykorzystywanej i dostępnej grupy (*group*); typ klucza szyfrującego (*key*) oraz typ wiadomości UDM (*typ*).

Występuje w tym miejscu podobny problem jak w przypadku uwierzytelniania. Chodzi tu mianowicie o optymalny wybór klucza szyfrującego, gdy ma nim być klucz sesji. Rozwiązanie jest tu analogiczne do tego, przedstawionego w przypadku uwierzytelniania.

W dalszej części niniejszego paragrafu przedstawione zostaną procedury odbiorcze wiadomości SMM oraz UDM. Zważywszy jednak na fakt, że analiza otrzymanych danych jest dużo bardziej złożona niż w przypadku tworzenia i nadawania wiadomości, to również ilość wynikowego kodu źródłowego jest spora. Nie zostaną tu zatem przedstawione całościowe i szczegółowe metody realizujące poszczególne działania, a jedynie schematy blokowe autorskich algorytmów, wykorzystywanych w trakcie analizy odbieranych danych.

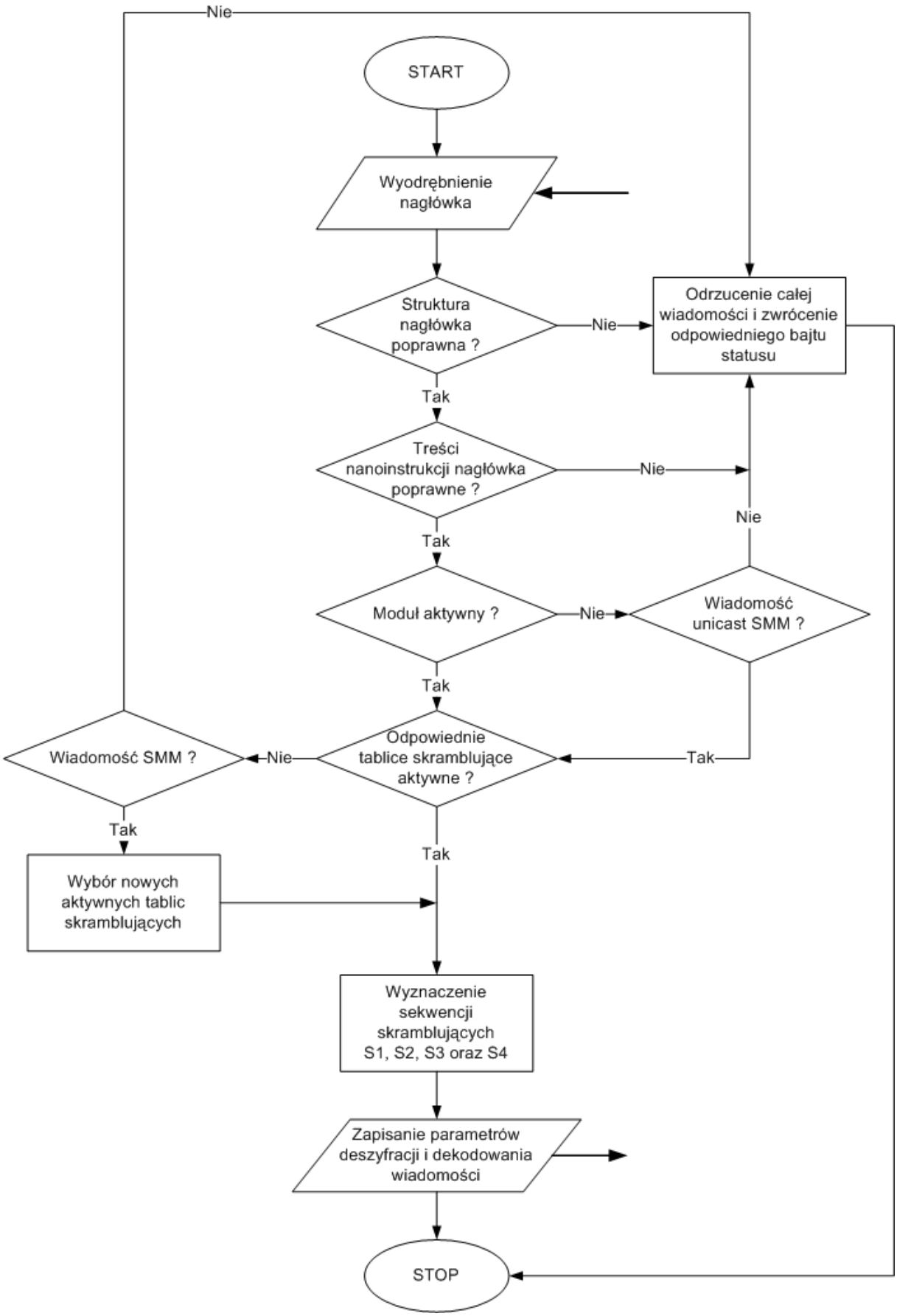
W ogólności, procedurę odbioru wiadomości podzielić można na dwa podstawowe etapy:

- Analiza nagłówka wiadomości i uzyskanie na jej podstawie danych do dalszego przetwarzania wiadomości;
- Deszyfracja, kontrola integralności, uwierzytelnienie nadawcy oraz pełne dekodowanie wiadomości z wykorzystaniem informacji uzyskanych z nagłówka oraz zawartych w samym module kryptograficznym.
    - W przypadku wiadomości SMM wykonywane są w tym etapie wszystkie instrukcje otrzymane od centrum bezpieczeństwa.
    - W przypadku z kolei wiadomości UDM wyodrębniane i udostępniane są dane użytkownika.

Warto w tym miejscu dodać, że w trakcie realizacji obu tych etapów analizowana jest również poprawność strukturalna wszystkich elementów wiadomości.

W razie jakichkolwiek nieprawidłowości w trakcie przeprowadzania analizy odebranej wiadomości, zwracany jest odpowiedni bajt statusu (kod błędu) – patrz poprzedni rozdział niniejszej rozprawy doktorskiej.

Pierwszym etapem jest jak już wspomniano analiza nagłówka odebranej wiadomości. Sposób realizacji tego procesu przedstawiono na rysunku 4.2.



Rys. 4.2. Algorytm analizy nagłówka odebranej wiadomości.

Algorytm ten przedstawiono w sposób ogólny i uproszczony. W pierwszej kolejności następuje wyodrębnienie z otrzymanych danych pierwszych 128 bitów (jawna część wiadomości - nagłówek). Następnie kontroli podlega sama struktura nagłówka i wszelkie jej nieprawidłowości powodują odrzucenie całej wiadomości.

Przykładowo pierwszy bajt musi mieć wartość *0x86* (System ID), a kolejny to identyfikator jednej z realizacji kryptosystemu obsługiwanej przez odbiorczy moduł kryptograficzny. Kompletna struktura nagłówka przedstawiona została w poprzednim rozdziale. Sprawdzeniu podlega również sama treść nanoinstrukcji zawartych w nagłówku.

Istotnym elementem jest tu również fakt aktywności modułu dla danej realizacji systemu (*providera*). W przypadku wiadomości unicast SMM element ten nie jest istotny, ponieważ łatwo wyobrazić sobie sytuację, w której to odbierana właśnie wiadomość z centrum bezpieczeństwa zawiera instrukcje aktywujące moduł. W przypadku UDM lub multicast SMM jednak nieaktywny *provider* oznacza automatycznie brak możliwości odbioru tych typów wiadomości, a więc ich odrzucenie.

Ponadto w przypadku samego tylko przekroczenia terminu końca aktywacji odbiór przez moduł (lub próba nadania przez niego) jakiejkolwiek wiadomości oznacza automatyczną dezaktywację danej realizacji kryptosystemu w tymże module. Proces ten jest realizowany automatycznie podczas analizy dowolnego nagłówka (na rysunku 4.2 nie zaprezentowano tego przypadku) lub w momencie próby nadania wiadomości UDM. Pełna definicja procesu dezaktywacji przedstawiona została w dalszej części ninejszego rozdziału.

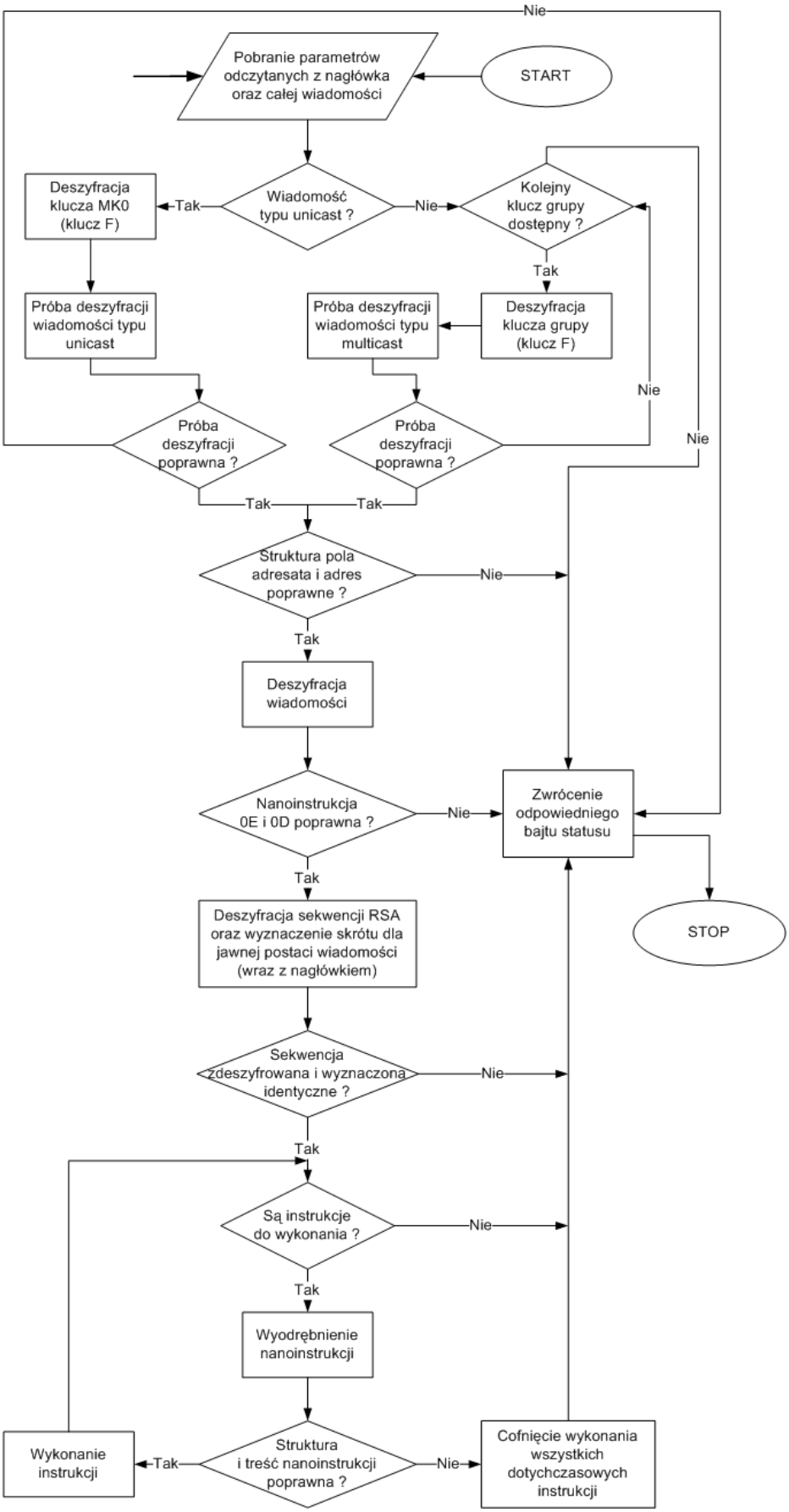
Dla wiadomości, przesyłanych pomiędzy terminalami, również niezgodność numerów aktualnie aktywnych w module tablic skramblujących z tymi użytymi do wygenerowania danej wiadomości UDM (nano *0x14* w nagłówku) oznacza automatyczne jej odrzucenie.

W przypadku jednak transmisji odebranej od centrum bezpieczeństwa numery aktualnie aktywnych tabklic skramblujących są tymczasowo aktualizowane na nowe – te zawarte w nagłówku (nano *0x14*). Jeśli jednak analizowana w tym momencie wiadomość SMM zostanie odrzucona (na którymkolwiek etapie), to przywracane zostają poprzednie numery tablic. Po jej pozytywnym zwertyfikowaniu z kolei

aktualizacja taka staje się obowiązującą w daanym module (do czasu ewentualnej kolejnej zmiany).

Jeśli okażę się, że procedura analizy nagłówka przebiegnie poprawnie, to wyznaczane są sekwencje skramblujące (od S1 do S4 – patrz rozdział trzeci) i zapisane zostają parametry deszyfracji i dekodowania wiadomości (są to właściwie wszystkie informacje zawarte w nagłówku i ustalone na jego podstawie – patrz rozdział poprzedni). W tej sytuacji moduł przechodzi do drugiego etapu analizy odebranej wiadomości.

Dalsze działanie jest już w dużej mierze zależne od informacji zawartych w samym nagłówku, a główne znaczenie ma typ wiadomości, która jest właśnie przetwarzana. Jeśli okaże się bowiem, że odebrana została wiadomość SMM, to procedura postępowania jest zgodna z tą przedstawioną na rysunku 4.3.



Rys. 4.3. Algorytm deszyfracji i dekodowania wiadomości SMM.

W pierwszym etapie przetwarzania wiadomości SMM pobierane są wszystkie parametry ustalone na podstawie analizy nagłówka oraz kompletna wiadomość. W zależności od tego czy transmisja jest typu unicast czy multicast, w kolejnym kroku wybierany jest do deszyfracji (na podstawie informacji z nagłówka) klucz główny (MK0) lub jeden z kluczy grupy (na tym etapie nie wiadomo której konkretnie grupy) dla realizacji systemu określonej w nagłówku. Dla klucza MK0 lub grupy, zrealizowana musi być najpierw jego deszyfracja, którą wykonuje się z wykorzystaniem schematu pośredniego jednoetapowego (patrz rozdział poprzedni), a kluczem deszyfrującym jest tu klucz podstawowy F. Warto tu również przypomnieć, że do deszyfracji kluczy, stosowanych w realizacji mechanizmu poufności stosuje się sekwencję skramblującą S4.

W dalszej kolejności moduł przechodzi do próby deszyfracji wiadomości. Należy pamiętać, że analizie poddawany jest na tym etapie tylko pierwszy tajny blok wiadomości (128 bitów po jawnym nagłówku). W celu sprawdzenia poprawności deszyfrowania kontrolowane jest pierwsze tajne nano (*0x2F*) wraz z jego treścią. W przypadku deszyfracji z użyciem klucza grupy proces może być powtórzony kilkukrotnie, ponieważ moduł może wykorzystywać kilka grup (do ośmiu) w danej realizacji systemu. Jeśli próby deszyfracji dla wszystkich dostępnych kluczy okazują się nieskuteczne, to wiadomość należałoby odrzucić.

W praktyce zaimplementowano jednak pewien dodatkowy mechanizm, który nie jest przedstawiony na rysunku 4.3, ponieważ zaciemniłby on ogólną ideę omawianego algorytmu. Technika ta stosowana jest przy odbiorze każdego typu wiadomości (SMM jak i UDM). Należy w tym miejscu bowiem przypomnieć, iż do realizacji funkcji poufności wykorzystywany jest zmodyfikowany algorytm AES-CTR, w którym stosuje się tak zwany znacznik czasu (minutowy przedział). W przypadku niedokładnej synchronizacji zegarów (a taką się zakłada) pomiędzy poszczególnymi terminalami lub ze względu na działanie protokołu ARQ[14] (patrz rozdział drugi) mogłaby bowiem wystąpić sytuacja, gdy terminal nadawczy wysyła wiadomość z danym znacznikiem, a u odbiorcy na przykład aktualny znacznik jest już inny

[14] Błędy w transmisji spowodować mogą wielokrotne próby nadania tej samej już zaszyfrowanej, a więc z konkretnym znacznikiem czasu, wiadomości. Kryptosystem z założenia umiejscowiony jest jednak wyżej w modelu OSI niż protokół ARQ i dlatego stempel czasu nie ulega aktualizacji w trakcie kolejnych retransmisji. Działanie protokołu ARQ wymaga więc czasu i między innymi dlatego stosuje się opisany mechanizm oraz minutową ważność znacznika czasu.

(kolejny). W tym momencie poprawny odbiór wiadomości nie byłby możliwy nawet w sytuacji, gdy adresat i inne parametry byłyby poprawne. Z tego też powodu, w przypadku niepowodzenia próby deszyfracji wiadomości dla wszystkich potencjalnych kluczy, proces ten powtarza się najpierw dla znacznika poprzedniego, a następnie dla przyszłego w celu wyeliminowania błędów synchronizacji zegarów i opóźnień, wynikających z ewentualnego stosowania protokołu ARQ.

Jeśli jednak pomimo zrealizowania powyższych działań wiadomość zostanie odrzucona, to przyczyny takiej sytuacji mogą być następujące:

- Wiadomość została zaadresowana do innej (nieobsługiwanej) grupy lub innego modułu;
- Posiadany klucz grupy (w przypadku transmisji punkt-wielopunkt) jest nieważny (został wcześniej zmieniony w innych terminalach poprzez np. wiadomości SMM typu unicast);
- Wiadomość utraciła ważność (każda wiadomość jest ważna przez minutę od momentu, gdy u potencjalnego odbiorcy utracił ważność aktualny znacznik czasu nadawcy – próba deszyfracji zrealizowana zostanie bowiem również dla znacznika poprzedniego, ale tylko tego, który zdezaktualizował się jako ostatni);
- Wiadomość została zmodyfikowana lub posiada błędy.

W tym miejscu należy zauważyć pewien bardzo istotny aspekt. Wydawałoby się bowiem, że w przypadku transmisji multicastowej wielokrotne próby deszyfracji będą wymagały dużej ilości czasu. W praktyce jednak nie ma takiego zagrożenia, ponieważ należy pamiętać, że dla weryfikacji poprawności deszyfracji wystarczy poddać analizie tylko jedną sekwencję 128-bitową (nano *0x2F* wraz z treścią znajdują się w pierwszym tajnym bloku wiadomości). Deszyfracji z wykorzystaniem algorytmu AES-CTR podlega na typ etapie zatem tylko jeden blok (jest to tylko próba deszyfracji).

Po pozytywnej weryfikacji nana *0x2F* kontroli podlega kolejne pole – adresat (znajduje się ono również w pierwszym tajnym bloku wiadomości). Sprawdzana jest jego struktura i treść tej nanoinstrukcji (adresatem w sposób oczywisty musi być terminal odbiorczy – unicast SMM – lub obsługiwana grupa – multicast SMM).

W dalszej kolejności, po poprawnej deszyfracji już całej wiadomości, weryfikowane są z kolei nana *0x0D* oraz *0x0E*. W tym miejscu należy uwypuklić fakt,

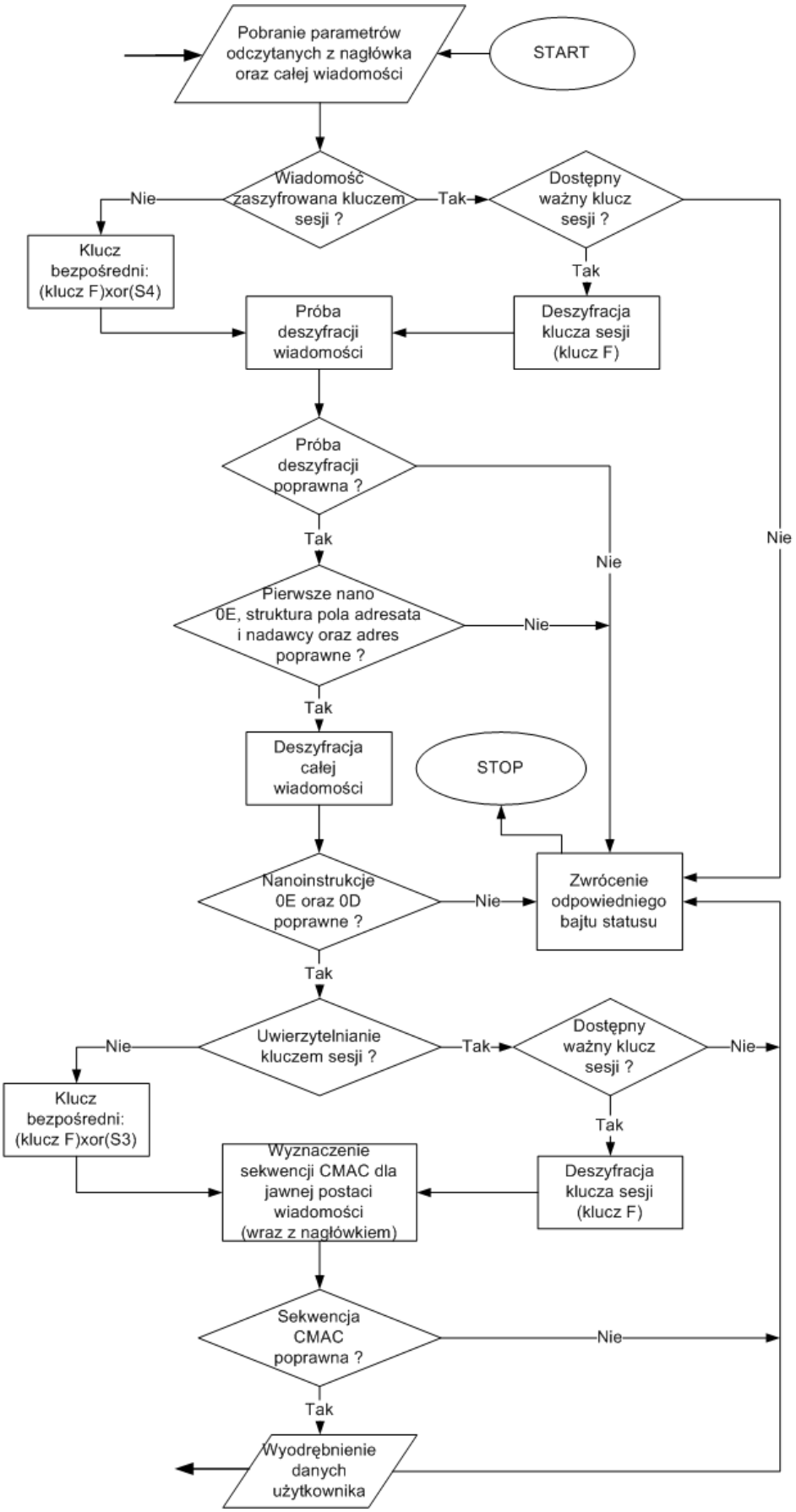
że jakiekolwiek, napotkane w trakcie analizy, błędy struktury czy treści elementów wiadomości powodują jej automatyczne odrzucenie i ewentualne cofnięcie zmian dokonanych nanem *0x14* z nagłówka.

Po wstępnej i pozytywnej kontroli poprawności wiadomości następuje etap jednoznacznej weryfikacji jej integralności oraz wiarygodności nadawcy (centrum bezpieczeństwa systemu *0x86*). W tym celu deszyfracji z wykorzystaniem klucza publicznego RSA poddany zostaje podpis cyfrowy, umieszczony na końcu wiadomości. Dodatkowo dla jawnych już danych (wraz z nagłówkiem oraz nanoinstrukcją *0x0D*) wyznaczona zostaje sekwencja skrótu SHA-256. Oba uzyskane w ten sposób ciągi muszą być identyczne. W przeciwnym razie wiadomość z oczywistych względów zostaje odrzucona.

Końcowym etapem analizy wiadomości SMM jest wyodrębnianie, kontrola oraz wykonywanie poszczególnych nanoinstrukcji otrzymanych od CB. Szczegółowe mechanizmy wykonywania instrukcji nie będą tu prezentowane, ponieważ jest to zagadnienie czysto implementacyjne, a ważny jest efekt końcowy – dane w module kryptograficznym zostają zaktualizowane. Warto tylko wspomnieć, że spis dostępnych rozkazów umieszczono w rozdziale poprzednim. Istotny jest tu natomiast fakt, iż jakikolwiek błąd w strukturze lub treści choćby jednej instrukcji powoduje nie tylko odrzucenie całej wiadomości, ale również cofnięcie wszystkich zmian dokonanych przez inne nana analizowanej wiadomości (łącznie z nanem *0x14* z nagłówka).

Efektem końcowym poprawnie zdekodowanej wiadomości SMM jest zwrócenie odpowiedniego statusu; w tym wypadku będzie to bajt *0x10*. Jeśli z kolei analiza odebranych danych na którymkolwiek etapie nie powiodła się, to również zwracany jest odpowiedni status, ale w tym wypadku, odzwierciedlający powód odrzucenia wiadomości (listę statusów kryptosystemu 0x86 przedstawiono w rozdziale trzecim).

Jeśli na podstawie informacji z nagłówka okaże się, że odebrano wiadomość UDM, to algorytm działania jest odmienny. Co więcej procedura postępowania jest inna w zależności od typu wiadomości, ale można tu wyróżnić dwa podstawowe sposoby działania. Pierwszy z nich został schematycznie przedstawiony na rysunku 4.4 i jest wykorzystywany, gdy wiadomość UDM jest typu broadcast lub unicast poza grupę.



4.4. Algorytm deszyfrowania i dekodowania wiadomości UDM typu broadcast lub unicast poza grupę.

Analogicznie jak w przypadku wiadomości SMM, tu również w pierwszej kolejności pobrane zostają informacje, odczytane z nagłówka oraz sama wiadomość.

Dla prezentowanego przypadku deszyfracja odbywać się może z wykorzystaniem klucza podstawowego F lub któregoś z dwóch kluczy sesji – w zależności od informacji uzyskanych z nagłówka. W pierwszej sytuacji można bezpośrednio przejść do próby deszyfracji wiadomości, a kluczem deszyfrującym staje się sekwencja powstała z wyznaczenia sumy modulo 2 klucza F oraz ustalonej wcześniej sekwencji skramblującej S4 (schemat bezpośredni uzyskiwania klucza – patrz rozdział poprzedni). W drugim przypadku natomiast ważny klucz sesji podlega najpierw deszyfracji z wykorzystaniem klucza F oraz ciągu S4, a następnie w ten sposób uzyskana sekwencja stosowana jest w dalszej analizie.

Na rysunku 4.4 dla jego uproszczenia nie umieszczono przypadku, w którym ważne okazują się oba klucze sesji. Jeśli taka sytuacja ma miejsce, to w momencie błędnej deszyfracji pierwszym (starszym, ale ważnym) kluczem, deszyfruje się drugi, a następnie podejmowana jest kolejna próba analizy wiadomości. W dalszej kolejności, jeśli w przypadku deszyfracji wiadomości kluczem F lub dostępnymi ważnymi kluczami sesji proces ten nie powiedzie się (nie zostanie znalezione poprawne nano *0x2F*), to jest on ponawiany dla poprzedniego znacznika czasu i w razie konieczności również dla przyszłego. Powody, dla których deszyfracja wiadomości UDM mimo wszystko może się nie powieść, przedstawione zostaną w trakcie omawiania kolejnego algorytmu.

Po poprawnej próbie deszyfracji kontroli podlega struktura i treść pola adresata (w wiadomości typu broadcast ono nie występuje) oraz struktura pola nadawcy (jego treść zostaje zapisana), a następnie pierwsze nano *0x0E*. W dalszej kolejności następuje już całkowita deszyfracja wiadomości i kontrolowane są nana *0x0D* oraz *0x0E* (kolejne, występujące po danych użytecznych). Jeśli na tym etapie występują jakieś błędy to wiadomość jest odrzucana.

W tym miejscu konieczna jest kontrola integralności i wiarygodności wiadomości. Do tego celu służy oczywiście autorska modyfikacja algorytmu CMAC. Dla jego realizacji wymagany jest klucz, który podobnie jak przy deszyfracji może być zdeszyfrowanym kluczem sesji lub sekwencją uzyskaną poprzez schemat bezpośredni z kluczem F. Przy uzyskiwaniu klucza uwierzytelniającego korzysta się z

tych samych algorytmów jak przy wyznaczaniu klucza deszyfrującego, ale stosuje się tu sekwencję S3 zamiast S4.

Informacja o tym, który klucz będzie wykorzystywany została wcześniej odczytana z nagłówka wiadomości. Jeśli dla algorytmu CMAC, użyty ma być klucz sesji, to stosuje się tu właściwie identyczną procedurę jak w przypadku deszyfracji z wykorzystaniem właśnie tego klucza. Wybiera się bowiem ważny klucz sesji, a gdy aktualne są oba, to najpierw stosuje się ten starszy, a w razie potrzeby nowszy. Wyznaczenie sekwencji CMAC może więc być zrealizowane dwukrotnie, jeśli dla pierwszego aktywnego klucza sesji sekwencja ta okaże się błędna, a dostępny będzie kolejny ważny klucz sesji, to proces można powtórzyć.

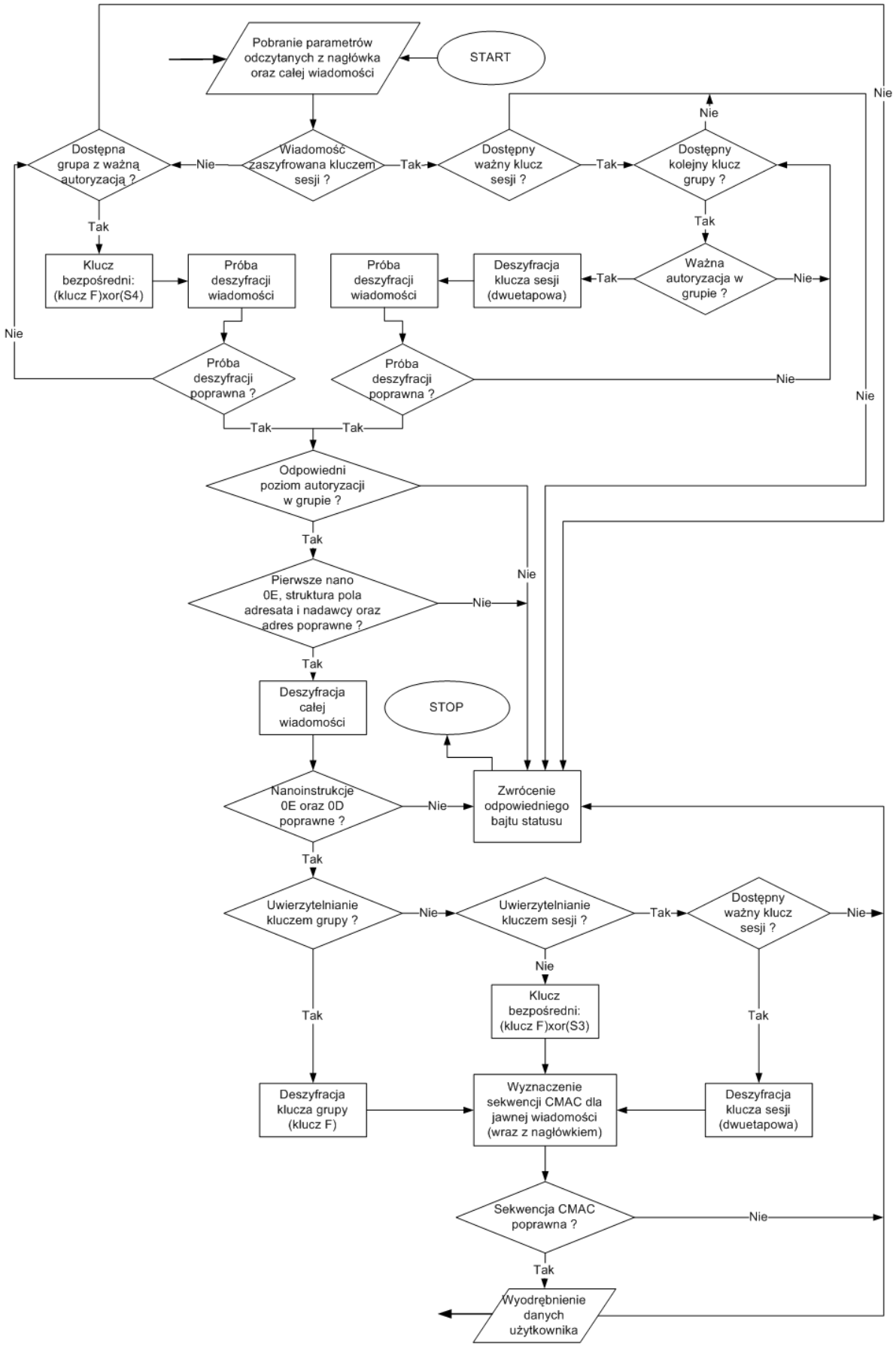
Jeśli końcowa sekwencja CMAC wyznaczona dla całej jawnej postaci wiadomości wraz z nagłówkiem oraz nanoinstrukcją *0x0D* będzie identyczna z tą umieszczoną w wiadomości, to wiadomość taka jest finalnie uznana przez moduł za poprawną i w dalszej kolejności wyodrębnieniu ulegają dane użytkownika. Nie podlegają one już żadnej kontroli strukturalnej (struktura jest dowolna, a właściwie nie istnieje), choć oczywiście ich integralność, wiarygodność i poufność została potwierdzona.

W końcowym etapie zwracany jest bajt statusu, który dla poprawnie zdekodowanej wiadomości UDM ma postać *0x11*. W przypadku odrzucenia wiadomości na którymś etapie zwracany jest oczywiście inny status (status błędu).

Algorytm deszyfrowania i dekodowania wiadomości UDM typu multicast oraz unicast w grupie różni się dość znacznie w stosunku do poprzedniego i przedstawiono go na rysunku 4.5.

W tym miejscu należy zwrócić uwagę na fakt, iż poniższa procedura jest podstawowym i zalecanym sposobem komunikowania się terminali. Jest ona bowiem bezpieczniejsza niż algorytm prezentowany na rysunku 4.4. Ze względu na fakt możliwości wykorzystywania kluczy grup (transmisja odbywa się zawsze wewnątrz konkretnej grupy użytkowników) możliwe staje się bardziej wiarygodne uwierzytelnianie modułów kryptograficznych, wykorzystanie kontroli poziomu ich autoryzacji w danej grupie oraz stosowanie prostszych schematów adresacji.

W przypadku algorytmu z rysunku 4.5, przedstawione zostaną tylko zmiany w jego realizacji w stosunku do poprzednio prezentowanego. Pomiędzy obiema procedurami występują trzy zasadnicze różnice. Pierwszą jest tu kontrola poziomu autoryzacji modułu w danej grupie.



4.5. Algorytm deszyfrowania i dekodowania wiadomości UDM typu multicast lub unicast w grupie.

W przypadku wiadomości typu unicast lub zwykłej multicast proces ten sprowadza się tylko do sprawdzenia daty wygaśnięcia autoryzacji i realizuje się go jeszcze przed próbą deszyfracji. Jeśli terminal nie jest uprawniony do transmisji (i odbioru) w danej grupie, to moduł automatycznie odrzuci taką wiadomość. Gdy transmisja z kolei odbywa się z wykorzystaniem kanałów wirtualnych, kontroli podlega poza datą również maska uprawnień (poziom autoryzacji) dla danej grupy. Proces ten realizowany jest po pozytywnej próbie deszyfracji, ponieważ nano *0x1F* znajduje się w pierwszym tajnym bloku wiadomości. Terminal nie uzyska danych użytecznych, jeśli moduł kryptograficzny stwierdzi brak możliwości odbioru (i również nadawania) w danym kanale wirtualnym.

W tym miejscu należy przypomnieć, że próba deszyfracji wiadomości UDM może się oczywiście nie powieść. Przyczyny takiej sytuacji mogą być następujące:

- Wiadomość została zaadresowana do innej (nieobsługiwanej) grupy, innego użytkownika grupy lub innego modułu;
- Posiadany klucz grupy (w przypadku transmisji punkt-wielopunkt) jest nieważny (został wcześniej zmieniony w innych terminalach poprzez np. wiadomości SMM typu unicast);
- Posiadany ważny klucz sesji mógł być również wcześniej zmieniony przez CB;
- Wiadomość utraciła ważność (każda wiadomość jest ważna przez minutę od momentu, gdy u potencjalnego odbiorcy utracił ważność aktualny znacznik czasu – próba deszyfracji zrealizowana zostanie bowiem również dla znacznika poprzedniego, ale tylko tego, który zdezaktualizował się jako ostatni);
- Wiadomość została zmodyfikowana lub posiada błędy.

Warto tu nadmienić, że transmisja w kanale wirtualnym jest najbardziej pożądana w systemie, ponieważ oferuje najwyższy poziom bezpieczeństwa oraz umożliwia dużą elastyczność komunikowania się i prosty mechanizm hierarchizacji terminali należących do danej grupy. Dodatkowo kanały wirtualne są wygodnym sposobem adresowania wiadomości. Wystarczy bowiem wybrać grupę (jej adres) użytkowników, do której ma się dostęp i z którą chce się skontaktować, a następnie wybrać kanał, w którym odbywać się będzie transmisja. Takie podejście ma dwie zasadnicze zalety. Pozwala bowiem wydzielić niezależne transmisje (o różnym poziomie autoryzacji) w danej grupie i dodatkowo umożliwia wykorzystanie intuicyjnego rozumienia przez

użytkowników sieci radiowych pojęcia *kanał*. Dla przykładu: aby skomunikować się z danymi terminalami w grupie wystarczy wybrać kanał nr np. 16 (tu jednak wirtualny!).

Kolejna różnica w stosunku do poprzedniego algorytmu odbioru wiadomości UDM występuje właściwie wcześniej w procedurze, ale przedstawiona zostanie w tym miejscu, ponieważ wiąże się w pewien sposób z trzecim elementem, różniącym oba rozwiązania. Mowa tu o mechanizmie deszyfracji wiadomości i o ile sam algorytm deszyfracji jest tu identyczny (zmodyfikowany AES-CTR), to sposób uzyskiwania klucza w jego postaci jawnej jest w tym przypadku inny.

Jeśli wiadomość zaszyfrowana jest kluczem podstawowym F, to postępowanie jest identyczne jak w poprzednim przypadku. Stosowany jest bowiem schemat bezpośredni uzyskiwania klucza deszyfrującego – kluczem tym staje się ciąg, utworzony jako suma modulo 2 klucza F oraz sekwencji skramblującej S4.

W przypadku jednak wykorzystywania klucza sesji (przypadek zalecany), stosuje się schemat pośredni dwuetapowy, gdzie kluczem deszyfrującym jest klucz F, natomiast wspomagającym – klucz danej grupy. Takie podejście pozwala na wygenerowanie klucza deszyfrującego, który jest poprawny i znany tylko dla danej grupy (wykorzystanie klucza grupy) w danej realizacji systemu (wykorzystanie klucza F) i samym systemie *0x86* (użycie sekwencji skramblującej S4 wyznaczonej na podstawie odpowiednich wektorów z systemowych tablic skramblujących A oraz B) i oczywiście jest również ważnym kluczem sesji (wykorzystanie ważnego klucza sesji A lub B).

Trzeci element, odróżniający algorytm deszyfracji i dekodowania wiadomości UDM przesyłanych w obrębie grupy od poprzedniego, to sposób uzyskiwania klucza uwierzytelniającego. Ponownie może być tu wykorzystany schemat bezpośredni (klucz F i sekwencja S3) i jest to przypadek najprostszy oraz najmniej bezpieczny. W celu wyznaczenia sekwencji CMAC użyć można również klucza sesji. Wykorzystuje się tu jednak schemat pośredni dwuetapowy (klucz F + klucz grupy), więc klucz deszyfrujący i uwierzytelniający będą w pewnym sensie do siebie „podobne” (nie identyczne, bo stosuje się tu sekwencję S3 a nie S4). Dodatkowo schemat dwuetapowy jest dość złożony obliczeniowo. Zalecanym zatem sposobem uzyskania klucza uwierzytelniającego i wykorzystywanego do kontroli integralności wiadomości jest schemat pośredni jednoetapowy, w którym to klucz grupy deszyfrowany jest z

wykorzystaniem klucza podstawowego F oraz sekwencji S3. W ten sposób nadawca wiadomości może zostać uwiarygodniony jako członek danej grupy.

Po poprawnym procesie analizy wiadomości UDM wyodrębnione zostają oczywiście dane użyteczne i zwrócony zostaje status *0x11*. W tym miejscu warto przypomnieć, że niezależnie od typu przetwarzanej wiadomości (SMM/UDM) moduł kryptograficzny zawsze zwraca bajt statusu. Nawet jeśli wiadomość zostanie na którymś etapie odrzucona, to na wyjście podany zostanie kod błędu (patrz poprzedni rozdział).

W niniejszym paragrafie przedstawiono sposób implementacji zaproponowanego w rozdziale poprzednim kryptosystemu dla transmisji danych w łączu krótkofalowym. Zaprezentowano tu metody realizacji zarówno najistotniejszych autorskich modyfikacji znanych algorytmów kryptograficznych jak i całej zaproponowanej struktury systemu bezpiecznej transmisji danych w łączu HF.

W kolejnym paragrafie po krótkim wstępie, dotyczącym interfejsu zaimplementowanej aplikacji, przedstawione zostaną badania oraz testy funkcjonalne samego kryptosystemu, pozwalające na ocenę oferowanego poziomu bezpieczeństwa transmisji. Ocena ta będzie tematem kolejnego rozdziału niniejszej rozprawy doktorskiej.

4.2. Badania poziomu bezpieczeństwa zaproponowanego kryptosystemu

Do celów badania bezpieczeństwa zaproponowanego rozwiązania zrealizowana została aplikacja, której główne mechanizmy i algorytmy omówiono w poprzednim paragrafie. W tym miejscu przedstawiony zostanie pokrótce jej interfejs.

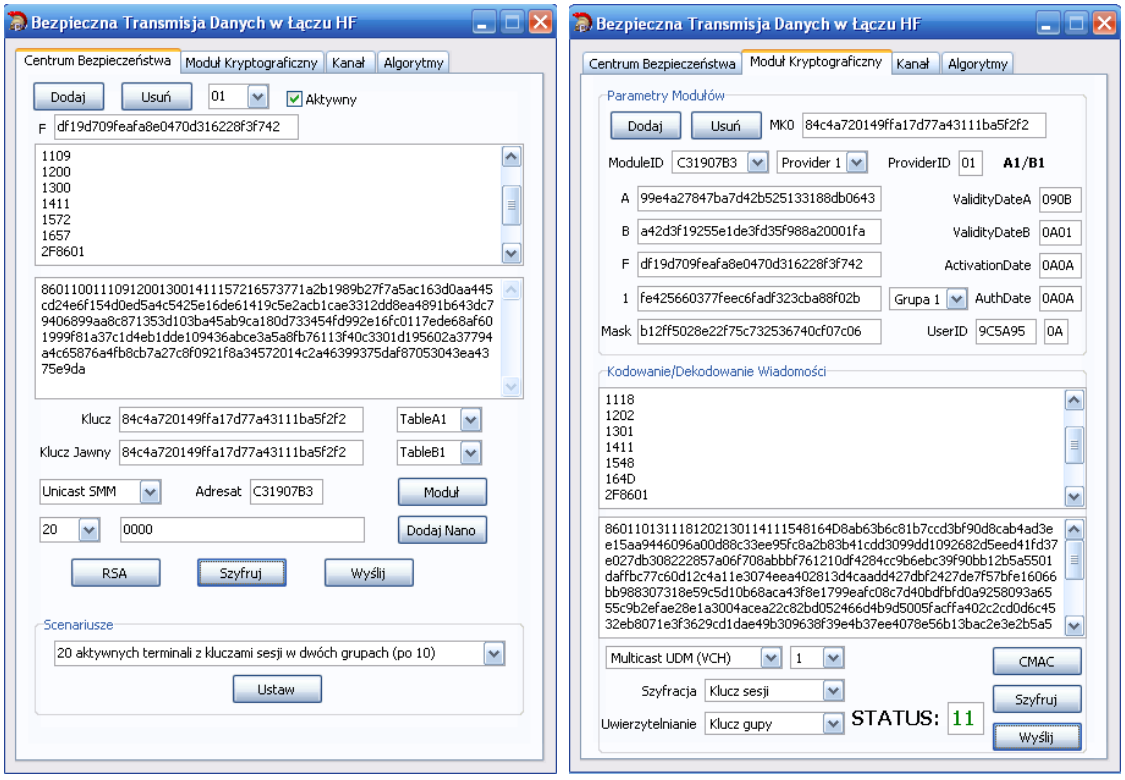
4.2.1. Interfejs użytkownika narzędzia symulacyjnego

W niniejszej pracy nie zostanie zaprezentowana szczegółowa procedura obsługi zaimplementowanej aplikacji, a jedynie ogólna struktura interfejsu użytkownika oraz możliwości jakie daje omawiane narzędzie. Wybrano takie podejście ze względu na fakt, że prezentowany w tej pracy symulator jest środowiskiem testowym, zaproponowanej koncepcji systemu bezpiecznej transmisji danych w łączu krótkofalowym. Ma on służyć celom badawczym, a w szczególności w tym

konkretnym przypadku pozwoli pokazać, że zaprojektowany kryptosystem można wykorzystać do realizacji bezpiecznej transmisji danych.

Należy tu dodatkowo zaznaczyć, że narzędzie to w tej postaci nie jest przeznaczone do współpracy z istniejącymi modemami HF i nie będzie wykorzystywane przez konkretnych użytkowników, a w więc w związku z tym nie ma tu konieczności prezentowania swoistej instrukcji obsługi. Niemniej jednak poniżej przedstawiono w sposób poglądowy, zrealizowane na potrzeby niniejszej rozprawy doktorskiej, narzędzie symulacyjne.

Na rysunku 4.6a zaprezentowano okno główne aplikacji, którego celem jest przede wszystkim definiowanie centrów bezpieczeństwa oraz tworzenie i wysyłanie wiadomości zarządzających typu SMM.



a) b)

Rys. 4.6. Interfejs użytkownika narzędzia symulacyjnego.

Rysunek 4.6b przedstawia z kolei okno narzędzia symulacyjnego, pozwalające na definiowanie nowych modułów kryptograficznych, obserwowanie wszystkich ich parametrów, odczytywanie odebranych przez nie wiadomości oraz tworzenie z ich użyciem konkretnych wiadomości UDM, zawierających dane użyteczne. Dane te dla prostoty obsługi symulatora są po prostu znakami ASCII wpisywanymi w oknie aplikacji. W tej zakładce również odczytać można aktualny status konkretnego

elementu, a w pliku dziennika aplikacji dostępne są posortowane bajty stanu dla wszystkich w danym momencie istniejących modułów.

Aplikacja posiada również możliwość przechwytywania przesyłanych wiadomości i ich modyfikowania oraz testowania funkcjonowania zaimplementowanych modyfikacji algorytmów kryptograficznych. Istnieje również funkcja generowania i inicjalizowania nowych tablic skramblujących (A i B) oraz kluczy RSA czy AES.

Istotny jest również fakt, że symulator pozwala tworzyć właściwie dowolne, ale oczywiście poprawne strukturalnie wiadomości i dodatkowo analizować je na wszystkich etapach ich powstawania. Poniżej przedstawiono przykładową wiadomość SMM typu unicast oraz sposób jej syntezy.

Postać jawna (wydzielone nana)[15]:

```
86    -> identyfikator systemu
04    -> identyfikator realizacji systemu (provider'a)
1001  -> typ wiadomości (unicast SMM)
110D  -> długość wiadomości bez nagłówka (liczba sekwencji 128-bitowych)
1200  -> typ klucza wykorzystanego do szyfrowania (MK0)
1300  -> uwierzytelnianie i kontrola integralności (sekwencja RSA)
1411  -> numery wykorzystywanych tablic skramblujących (A1/B1)
15EE  -> numer wykorzystywanego wektora z tablicy A (0xEE)
161D  -> numer wykorzystywanego wektora z tablicy B (0x1D)
2F8604      -> pierwsze tajne nano
4ADEAFFA08  -> adresat (identyfikator modułu)
200A0A      -> nadanie daty aktywności modułu dla danej realizacji systemu
4112345678  -> nadanie identyfikatora użytkownika w obrębie grupy nr 1
210A0A      -> nadanie daty ważności autoryzacji w grupie nr 1
90101122334455667788    -> nadanie pierwszej części maski w grupie nr 1
90111122334455667788    -> nadanie drugiej części maski w grupie nr 1
91101122334455667788    -> nadanie pierwszej części klucza grupy nr 1
91111122334455667788    -> nadanie drugiej części klucza grupy nr 1
02    -> skasowanie uprawnień w grupie nr 2
220000      -> wyzerowanie daty ważności autoryzacji w grupie nr 2
4200000000  -> wyzerowanie identyfikatora użytkownika w obrębie grupy nr 2
```

Postać jawna podpisana[16]:

```
86041001110D12001300141115EE161D2F86044ADEAFFA08200A0A4112345678210A0A90101
1223344556677889011112233445566778891101122334455667788911111223344556677888
022200004200000000E0C0C0C0C0C0C0C0C0C0C0D0e43f99694cb83f4ca1a087cfea961844
9420212e71215527c7abb6805bc62f2f5c997782a6636253d3beaca95c1b70220ae6c5d3ed9
af8c0cc3fa0cc2a001112fd1ef482031b2b629ebd403da62117a76edeaffc507822b0695bec
de92bcd375633b1c760cee939ac1c427f021746a7c9dcdf8826c7ca28e7f09d90be2cf00d
```

[15] Oznaczenia: kolor niebieski – identyfikator systemu i jego realizacji oraz typ nanoinstrukcji; kolor zielony – treść nanoinstrukcji

[16] Oznaczenia: kolor zielony – nagłówek; kolor niebieski – inne nanoinstrukcje; kolor żółty – sekwencja RSA

Postać zaszyfrowana (końcowa)[17]:

```
86041001110D12001300141115EE161D7e2155f4369c0218d237b6df7d4a3f21b297f0e21dc
cc6f1de723716203b30a8084736a1ba4fa7a10300f6cfa35cd79df193dcc37b9d54d95a9016
381afd519bb750e7f9ddd4ca56ad8852fe2b19385dcb0292a87bc041e26314ba453cb147492
9da4442727760ff9c173a74316922dd6e51bf2e58f8a14e7171844ce48400db54a228f46b8f
31cf0b52fbcd752ce134cebcb45177a6aaeadebad44fc70164951648506a8d957feed32c68c
e38f95a3d487b5bbd31e79f8358d1dfa8e58e5c9c92d126d7e6ab33e1087b9bf10d02cf74
```

Z kolei synteza, przeprowadzona dla przykładowej wiadomości UDM, przedstawiona została poniżej. W tym konkretnym przypadku rozważana jest transmisja multicastowa w kanale wirtualnym.

Postać jawna (wydzielone nana)[18]:

```
86    -> identyfikator systemu
01    -> identyfikator realizacji systemu (provider'a)
1013  -> typ wiadomości (multicast UDM w kanale wirtualnym)
1103  -> długość wiadomości bez nagłówka (liczba sekwencji 128-bitowych)
1202  -> typ klucza wykorzystanego do szyfrowania (klucz sesji)
1301  -> uwierzytelnianie i kontrola integralności (CMAC z kluczem grupy)
1411  -> numery wykorzystywanych tablic skramblujących (A1/B1)
15DE  -> numer wykorzystywanego wektora z tablicy A (0xDE)
16A1  -> numer wykorzystywanego wektora z tablicy B (0xA1)
2F8601      -> pierwsze tajne nano
3A3E569E    -> adresat (identyfikator grupy)
1B10  -> nadawca (identyfikator w grupie 3E569E)
1F04  -> numer wykorzystywanego kanału wirtualnego
```

Postać jawna z sekwencją CMAC[19]:

```
8601101311031202130114111 5DE16A12F86013A3E569E1B101F040E416C61206D61206B6F7
4612E0E0C0C0C0C0C0C0D2a4f70e49eb60c1585de2c472bdd232b
```

Postać zaszyfrowana (końcowa)[20]:

```
8601101311031202130114111 5DE16A111a6c46d59b77ab42ae473c9ad7eb9b31cce3e76148
0250f44932642978bb6596dd3821b91a28a9c35e163472b48de7b
```

W kolejnym paragrafie przedstawiony zostanie problem optymalnego doboru długości wiadomości w zależności od długości przeplotu stosowanego podczas transmisji danych w łączu krótkofalowym.

[17] Oznaczenia: kolor zielony – część jawna; kolor czerwony – część zaszyfrowana

[18] Oznaczenia: kolor niebieski – identyfikator systemu i jego realizacji oraz typ nanoinstrukcji; kolor zielony – treść nanoinstrukcji

[19] Oznaczenia: kolor zielony – nagłówek; kolor niebieski – inne nanoinstrukcje; kolor pomarańczowy – dane użyteczne; kolor żółty – sekwencja CMAC

[20] Oznaczenia: kolor zielony – część jawna; kolor czerwony – część zaszyfrowana

4.2.2. Optymalna długość wiadomości

W założeniach początkowych stwierdzono (patrz rozdział poprzedni), że zaproponowany kryptosystem powinien ograniczać pasmo użyteczne w niewielkim tylko stopniu. Autor przyjął, że w przypadku komunikacji pomiędzy terminalami nadmiarowość, wynikająca z zaproponowanego protokołu komunikacyjnego, powinna pozwolić na osiągnięcie 10-procentowego poziomu zmniejszenia przepływności użytecznej przy jednoczesnym ograniczeniu długości wiadomości do jak najkrótszej długości przeplotu, zdefiniowanego w najpopularniejszym standardzie modemów krótkofalowych (patrz [52] oraz rozdział drugi).

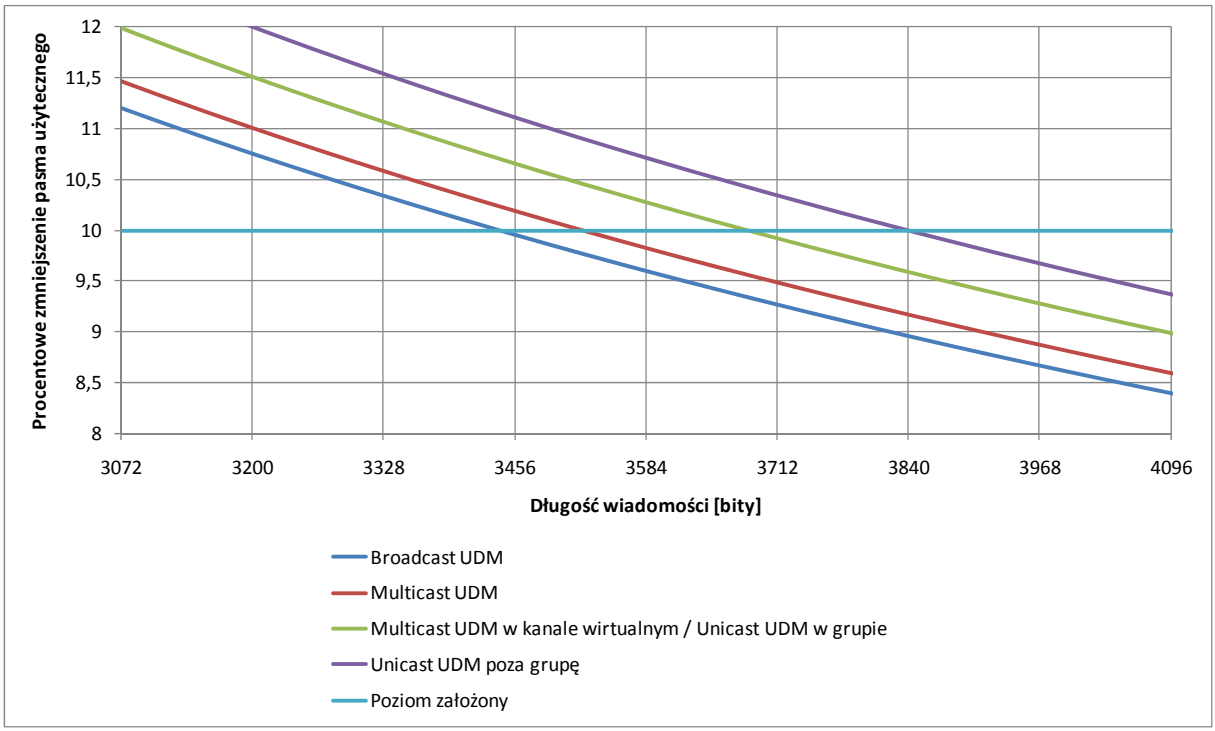
Takie podejście wynika z faktu, że modemy HF pracować mogą z różnymi głębokościami przeplotu, zależnymi od warunków panujących w kanale krótkofalowym. Autor założył w tym miejscu, że długość pojedynczej wiadomości w zaproponowanym kryptosystemie nie może być dłuższa niż aktualna głębokość przeplotu. Co więcej te wielkości powinny być równe w taki sposób, aby wiadomość mieściła się dokładnie w jednym przeplatanym bloku. Takie rozwiązanie pozwoli bowiem zmaksymalizować użyteczność kodowania kanałowego standardowych modemów krótkofalowych w powiązaniu z zaproponowanym kryptosystemem. Niekorygowalny błąd w jednym bloku przeplotu oznaczać będzie odrzucenie jednej tylko wiadomości i dodatkowo kolejne bloki (a zatem wiadomości) będą nadal użyteczne.

Krótkie przeploty mogą mieć jednak długość nawet tylko jednego (384 bity dla QPSK) lub trzech bloków danych. Uzyskanie 10-procentowego progu dla tych przypadków byłoby trudne do zrealizowania i pozbawiłoby zaprojektowane rozwiązanie jednej z jego kluczowych zalet – oferowany wysoki poziom bezpieczeństwa poprzez realizację wszystkich istotnych funkcji bezpieczeństwa.

Autor założył zatem, że kryptosystem wykorzystywany z tak krótkimi przeplotami nie będzie w pełni optymalny choć nadal funkcjonalny i efektywny. W takim przypadku bowiem długość wiadomości powinna być wielokrotnością długości przeplotu. W tym momencie błąd w jednym bloku przeplotu wpływa nadal na jedną tylko wiadomość, lecz niestety kolejne bloki, zawierające dane z tej samej wiadomości, stają się bezużyteczne (cała wiadomość zostanie i tak odrzucona przez moduł). Jest to niestety koszt, który trzeba ponieść, aby móc stosować zaproponowane rozwiązanie dla tak krótkich przeplotów. Alternatywnie można

również stosować krótsze wiadomości i godzić się na większy nadmiar protokolarny. Wykorzystując jednak dłuższe przeploty (dziewięć bloków i więcej), zaproponowane rozwiązanie może być stosowane w pełni optymalnie.

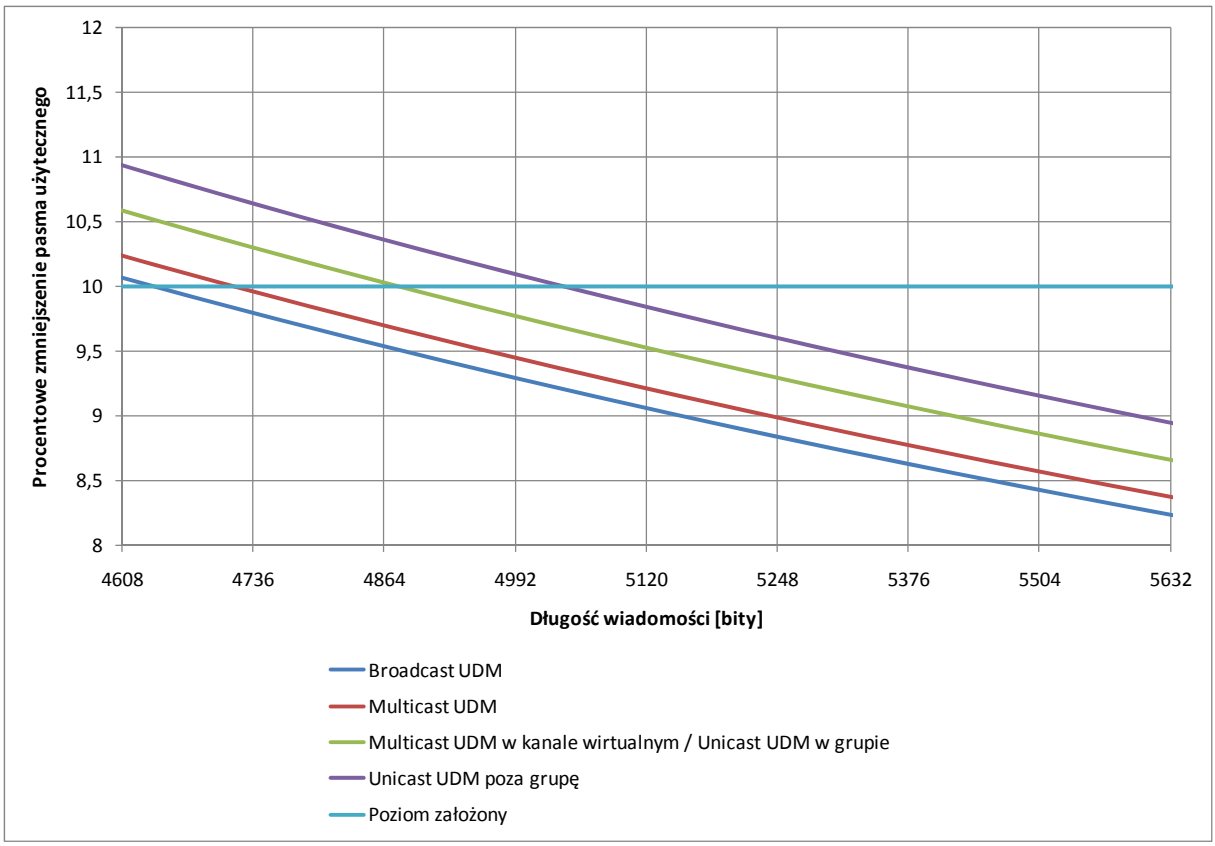
Na rysunku 4.7 przedstawiono wykres zależności procentowego zmniejszenia pasma użytecznego od długości wiadomości przy minimalnym poziomie dopełnienia (brak nanoinstrukcji *0x0C*) dla różnych typów wiadomości UDM.



Rys. 4.7. Procentowe ograniczenie pasma użytecznego w wiadomościach UDM przy minimalnym dopełnieniu.

Zgodnie ze specyfikacją modemów krótkofalowych [52] dla przeplotu o długości dziewięciu bloków danych w przypadku modulacji QPSK 10-procentowy próg można uzyskać tylko dla wiadomości typu broadcast. Dla wyższych wartościowości modulacji jednak już wszystkie typy wiadomości mogą być optymalnie transmitowane. Dodatkowo dla modulacji 64QAM wystarczy nawet przeplot o długości trzech bloków, by dla wiadomości rozsiewczych uzyskać założony próg. Powyższe rozważania przeprowadzono przy warunku, że w bloku przeplotu ma znajdować się dokładnie jedna (cała) wiadomość.

Na rysunku 4.8 przedstawiono dodatkowo wykres zależności procentowego zmniejszenia pasma użytecznego od długości wiadomości przy maksymalnym poziomie dopełnienia nanoinstrukcjami *0x0C* dla różnych typów wiadomości UDM.



Rys. 4.8. Procentowe ograniczenie pasma użytecznego w wiadomościach UDM przy maksymalnym dopełnieniu.

W tym przypadku, przy identycznych jak poprzednio założeniach, próg docelowy uzyskany będzie dla przeplotu o długości co najmniej dziewięciu bloków, począwszy od modulacji 8PSK (a właściwie 16QAM – patrz niżej) i niezależnie od typu wiadomości.

Warto tu wspomnieć, że wiadomości SMM nie są rozważane pod kątem ograniczenia pasma użytecznego, ponieważ nie są w nich przesyłane dane użytkowników. Istotny jest jednak fakt, że wiadomości SMM będą miały co najmniej 160 bajtów długości (sekwencja RSA, nagłówek i inne nana) i z tego powodu optymalna ich transmisja może być uzyskiwana dla przeplotu o długości co najmniej trzech bloków danych, począwszy od modulacji 8PSK (a właściwie 16QAM – patrz niżej).

W tym miejscu należy dodać, że dodatkowy problem stanowią tu modulacje 8PSK oraz 32QAM dla długości przeplotu 1, 3 lub 9 bloków danych. Dla tych bowiem przypadków blok danych wejściowych nie jest wielokrotnością 128 bitów, a jedynie 64. W takich sytuacjach w sposób oczywisty długość wiadomości powinna być parzystą wielokrotnością wielkości bloku danych wejściowych.

Ponadto dla długości przeplotu 36 (dla 32 i 64QAM) oraz 72 (począwszy od 8PSK) bloków nie ma również możliwości doboru w pełni optymalnej długości wiadomości. Blok danych wejściowych jest bowiem zbyt duży – większy od maksymalnej długości pojedynczej wiadomości. W takich sytuacjach w bloku danych wejściowych umieścić należy dwie lub trzy wiadomości i najlepiej w taki sposób, by dokładnie wypełnić dany blok.

W tabeli 2.8 w rozdziale drugim kolorem zielonym zaznaczono najbardziej optymalne wielkości bloków danych wejściowych dla kryptosystemu *0x86*. Wyboru tego dokonano na podstawie informacji przedstawionych w niniejszym paragrafie.

W kolejnym punkcie przedstawiona zostanie metodologia badań, a w kolejnym już konkretne scenariusze symulacyjne. Pozwolą one na zbadanie poprawności strukturalnej kryptosystemu, jego odporności na różnego rodzaju ataki i niepożądaną ingerencję z zewnątrz oraz wrażliwość na naruszenie mechanizmu dostępności. W ten sposób dokonana zostanie pośrednio ocena poziomu oferowanego bezpieczeństwa całości zaproponowanego rozwiązania.

4.2.3. Metodologia badań

Na całościowy poziom bezpieczeństwa oferowanego przez dany kryptosystem wpływ mają przede wszystkim trzy zasadnicze składniki. Pierwszym z nich są zastosowane algorytmy kryptograficzne. Jest to bardzo istotny aspekt, ponieważ nawet w przypadku najlepiej zaprojektowanego rozwiązania złamanie przez kryptoanalityków algorytmów wykorzystywanych w danym systemie oznaczać będzie najpewniej całkowitą jego bezużyteczność. Z tego też powodu w początkowych fazach budowaniu projektu krytposystemu wybór konkretnych rozwiązań analitycznych może być czynnikiem bardzo istotnym dla całości późniejszego rozwiązania.

W niniejszej pracy aspekt ten został rozwiązany poprzez wykorzystanie takich algorytmów, które na dzień dzisiejszy uważa się za w pełni bezpieczne i nie ma podstaw do tego, by móc sądzić, że sytuacja ta mogłaby zmienić się w najbliższym czasie. Nie są bowiem dostępne żadne wyniki najnowszych badań, które pozwoliłyby sądzić, że rozwiązania takie jak AES, RSA (dla kluczy o długości co najmniej 1024 bitów) czy SHA-256 są podatne na jakiekolwiek ataki kryptoanalityczne. Autor zakłada zatem w niniejszej rozprawie doktorskiej, że powyższe trzy algorytmy w swych pełnych formach są całkowicie bezpieczne. Praca ta bowiem nie ma na celu

badania poziomu bezpieczeństwa podstawowych rozwiązań kryptograficznych, a skupia się ona na analizie aspektu bezpieczeństwa w nieco innym zakresie. W ogólności niniejsze rozważania poddają analizie pozostałe dwa elementy składające się na końcową ocenę poziomu bezpieczeństwa konkretnego kryptosystemu.

Jednym z tych składników jest struktura danego rozwiązania. Chodzi tu zatem o poziom bezpieczeństwa oferowany przez kryptosystem jako całość. Struktura systemu, a więc jego architektura, mechanizmy zarządzania i protokół komunikacyjny. Są to bowiem elementy zupełnie nowe – rozwiązania autorskie – i dlatego mogą być one potencjalnym źródłem słabości czy wad, uniemożliwiających bezpieczną transmisję danych pomiędzy modemami krótkofalowymi.

Analiza tego właśnie aspektu będzie tematem rozważań dalszej części niniejszego rozdziału, w którym przedstawione zostaną pewne scenariusze symulacyjne, pozwalające na określenie poziomu bezpieczeństwa, zaproponowanego rozwiązania pod kątem poprawności jego całościowej struktury.

Trzecim elementem oceny poziomu bezpieczeństwa jest tak zwany „czynnik ludzki”, czyli wszystkie te elementy, które w tej chwili nie są możliwe do przewidzenia, a poprzez na przykład naruszenie mechanizmów dostępności, pozwolić mogą na utrudnienie lub uniemożliwienie realizacji przez system pozostałych funkcji bezpieczeństwa. Element ten rozważany będzie na końcu niniejszego oraz w ostatnim rozdziale tej pracy, w którym to przedstawiona zostanie kompleksowa ocena zaproponowanego rozwiązania.

4.2.4. Scenariusze symulacyjne

W celu dokonania oceny poziomu bezpieczeństwa prezentowanego kryptosystemu w pierwszej kolejności poddać należy weryfikacji poprawność i funkcjonalność samej jego struktury. Autor zaproponował więc przeprowadzenie szeregu badań symulacyjnych, pozwalających potwierdzić wysokie bezpieczeństwo i dużą efektywność zaproponowanego rozwiązania.

W celu zrealizowania wspomnianych analiz zaproponowano grupę scenariuszy symulacyjnych, pozwalających na zbadanie i przetestowanie poszczególnych elementów i mechanizmów kryptosystemu. Zbiór ten podzielono na kilka ogólnych części, zawierających logicznie powiązane ze sobą sytuacje. Badania symulacyjne realizowane będą zatem w zakresie obejmującym:

- Ogólne testy funkcjonalne, prezentujące poprawność działania kryptosystemu w warunkach idealnych;
- Próby ingerencji w treść przesyłanych wiadomości;
- Próby ingerencji w transmisję z wykorzystaniem technik powtarzania oraz filtrowania wiadomości;
- Mechanizmy zaradcze systemu w przypadku kradzieży modułu kryptograficznego lub nieautoryzowanego dostępu do kluczy bądź innych danych systemowych.

W tym miejscu warto ponownie zwrócić uwagę na fakt, że w trakcie badań symulacyjnych nie przewiduje się analiz związanych z klasycznymi atakami kryptoanalitycznymi, mającymi na celu złamanie danego algorytmu lub znalezienie jakiegoś klucza szyfrującego. Praca ta nie ma bowiem na celu tego typu działań. Do realizacji bowiem zaproponowanego rozwiązania wybrano algorytmy kryptograficzne, uznawane na dzień dzisiejszy za jedne z najbezpieczniejszych. Analizie i ocenie podlegać będą tu zatem nie algorytmy, ale powstały z ich wykorzystaniem kryptosystem – jako całość.

W dalszej części niniejszego paragrafu przedstawione zostaną zatem kolejne scenariusze symulacyjne i wyniki badań, uzyskane na ich podstawie wraz z wnioskami, płynącymi z tychże analiz.

W pierwszej kolejności zatem zaprezentowane zostaną ogólne testy funkcjonalne, obrazujące poprawność działania kryptosystemu w warunkach idealnych, które rozumie się tu jako brak jakiejkolwiek ingerencji (przez osoby nieuprawnione) w transmisję oraz w same urządzenia nadawczo-odbiorcze. Warunki idealne oznaczają również, że nie doszło do zdarzeń typu kradzież modułu czy ujawnienie kluczy lub innych danych systemowych, a cały kryptosystem jest wykorzystywany zgodnie z jego przeznaczeniem oraz funkcja bezpieczeństwa dostępność jest zapewniona.

Poniżej przedstawiono kilka standardowych sytuacji, mogących wystąpić w zaprojektowanym kryptosystemie.

Aktywacja pojedynczej realizacji systemu w przykładowym module kryptograficznym

Procedura taka może być w skrócie nazywana aktywacją modułu, jednakże należy w tym miejscu pamiętać, iż proces taki jest realizowany niezależnie dla każdego z maksymalnie czterech, obsługiwanych przez dany moduł, *provider'ów* systemu *0x86.* Może występować zatem sytuacja, w której jedna realizacja będzie

aktywna, a inna już nie. W dalszej części niniejszej pracy mowa będzie najczęściej o pewnych scenariuszach i sytuacjach w obrębie jednego, konkretnego *provider'a* i dlatego, wykorzystywane będzie określenie aktywacja modułu – w domyśle w obrębie tego właśnie konkretnego *provider'a*.

W ogólności proces aktywacji ma umożliwić działanie danego modułu w omawianym kryptosystemie, a tak właściwie w jednej z jego realizacji. Należy tu przypomnieć, że każdy moduł ma odgórnie zdefiniowaną i niemodyfikowalną listę provider'ów (maksymalnie cztery), z którymi może współpracować. W celu podstawowej aktywacji wystarczyłoby zatem jedynie wysłanie przez centrum bezpieczeństwa instrukcji ustawiającej datę końca aktywacji (*ActivationDate*), ponieważ po poprawnym wykonaniu takiego rozkazu moduł byłby zdolny do komunikowania się z wykorzystaniem wbudowanego i stałego klucza F.

Takie podejście spowoduje jednak, że dany moduł będzie mieć bardzo ograniczoną funkcjonalność (transmisje z wykorzystaniem tylko klucza F). Z tego powodu wprowadza się tu pojęcie pełnej aktywacji, która ma na celu przypisanie danego modułu do choćby jednej grupy (przesłanie klucza grupy oraz identyfikatora *User ID*) wraz ze zdefiniowaniem jego poziomu autoryzacji w tejże grupie (maska uprawnień oraz data końca autoryzacji w grupie).

Przypisanie do grupy wiąże się z przesłaniem instrukcji zmiany identyfikatora użytkownika (choćby jedno nano z zakresu od *0x41* do *0x48*) dla jednej z maksymalnie ośmiu obsługiwanych przez moduł grup w danej realizacji systemu. Dodatkowo moduł powinien otrzymać klucz danej grupy, swą maskę uprawnień w grupie oraz datę wygaśnięcia autoryzacji w tejże grupie.

W tym miejscu warto nadmienić klika spraw. Po pierwsze, wymienione tu instrukcje przesyłane powinny być w wiadomości typu unicast SMM, ponieważ dotyczą konkretnego terminala. Po drugie, klucze sesji mogą zostać również przesłane w tej wiadomości, lecz ich brak nie oznacza braku aktywacji. Dodatkowo warto wspomnieć, że klucze sesji wraz z ich datami ważności powinny być raczej przesyłane w wiadomościach typu multicast SMM.

Poniżej przedstawiono przykładowe nanoinstrukcje aktywujące moduł *0xB52CF1D0* w realizacji systemu *0x01* w grupie *0x123456*.

```
86
01
1001  -> unicast SMM
110C
1200  -> szyfrowanie kluczem MK0
1300  -> uwierzytelnianie RSA
1411
1521
1605
2F8601
4AB52CF1D0  -> adresat
200A0A      -> ustawienie daty końca aktywacji (do końca października 2010)
210A0A      -> ustawienie daty końca autoryzacji w grupie nr 1 (jw.)
4112345601  -> nadanie identyfikatora użytkownika
9010F000000000000000    -> możliwość transmisji w kanałach od 1 do 4
90110000000000000000
91101122334455667788    -> pierwsza część klucza grupy nr 1
91111122334455667788    -> druga część klucza grupy nr 1
```

Z powyższych instrukcji na wyjaśnienie zasługuje nano *0x4112345601*, które oznacza:

- Przypisanie grupie pierwszej w module identyfikatora użytkownika (*User ID*) postaci *0x12345601;*
- Grupa pierwsza w module ma odtąd identyfikator (*Group ID*) postaci *0x123456*, co oznacza w praktyce, że dany moduł przypisano do grupy o identyfikatorze *0x123456;*
- Identyfikator modułu w grupie (*InGroup ID*) pierwszej o identyfikatorze *0x123456* jest postaci *0x01*, co oznacza, że w przypadku transmisji w grupie *0x123456* moduł jest jednoznacznie identyfikowany przez jednobajtową sekwencję *0x01.*

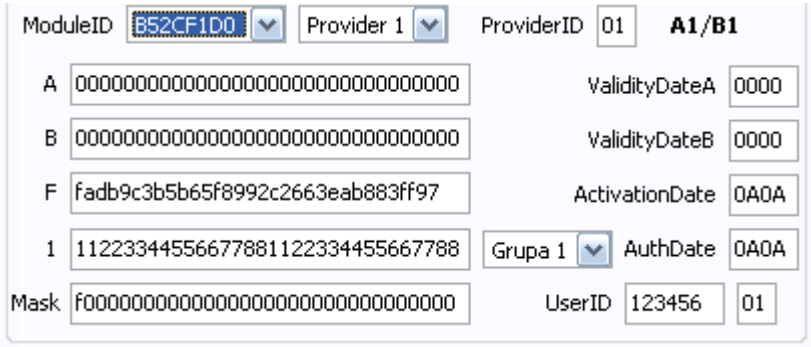
W tym miejscu warto dodać, że moduł może być przypisany podczas aktywacji od razu do kilku grup. W przyszłości konfiguracja taka może zostać oczywiście całkowicie zmodyfikowana.

Po przesłaniu wiadomości w pliku dziennika aplikacji można znaleźć z kolei:

	ModuleID	Status
1	B52CF1D0	10
2	948C0B3D	50
3	D925F217	50
4	9920F900	50
5	BD65077A	50
6	CCB601A6	50
7	BBB10EF0	50
8	F8D7F331	50
9	C9FDF475	50
10	9E510AC2	50

Na podstawie powyższych danych jasno widać, że wiadomość odebrana została w tym przypadku przez 10 różnych modułów, ale tylko adresat zdekodował ją w całości i wykonał zawarte w niej instrukcje. Status *0x50* pozostałych modułów oznacza błąd deszyfracji, spowodowany niepoprawnie wyznaczoną sekwencją licznika w algorytmie AES-CTR. Dzięki temu wiadomość adresowana do innego modułu może zostać odrzucona już podczas próby deszyfracji jej pierwszego bloku (128 bitów).

Efekt przesłanych i wykonanych instrukcji w module *0xB52CF1D0* przedstawiono na rysunku 4.9.



4.9. Moduł *0xB52CF1D0* po aktywacji *provider'a 0x01.*

Od momentu aktywacji dany moduł może już poprawnie funkcjonować w systemie. Dostępne stają się wszystkie typy wiadomości UDM, choć oczywiście bezpieczeństwo transmisji nie jest pełne ze względu na brak kluczy sesji w module. Jednakże uwierzytelnianie oraz kontrola integralności wiadomości mogą już być w pełni realizowane, ale oczywiście tylko w obrębie grupy *0x123456*, dla której to dostępny jest w tym przypadku klucz grupy.

Aktualizacja kluczy sesji w całej grupie modułów

Aktualizacja kluczy sesji jest procedurą, która z racji zachowania najwyższego możliwego poziomu poufności w systemie (a przez to wysokiego bezpieczeństwa transmisji), powinna być wykonywana znacznie częściej niż na przykład modyfikacja kluczy grup czy innych identyfikatorów w modułach.

Stosunkowo częsta aktualizacja kluczy sesji jest wymagana ze względu na ryzyko utraty poufności, wynikające z ciągłego stosowania tych samych kluczy szyfrujących. Dodatkowo występuje niebezpieczeństwo, że istnieć mogą moduły, posiadające aktualne klucze sesji, a w rzeczywistości powinny były one odebrać na przykład instrukcje, kasujące je lub nawet dezaktywujące daną realizację systemu w module.

Z tego też powodu wprowadzono pojęcie ważności kluczy sesji. Takie podejście pozwala na automatyczną dezaktualizację kluczy po upłynięciu pewnego i zdefiniowanego okresu czasu. Każdy klucz sesji jest zatem jednoznacznie powiązany z datą końca jego ważności.

Klucze jednak nie mogą być również aktualizowane zbyt często, ponieważ wprowadziłoby to bowiem duże zamieszanie w systemie, a w skrajnym przypadku spowodowałoby brak możliwości poufnej transmisji. Taka sytuacja mogłaby wystąpić, ponieważ moduły posiadałyby klucze sesji, które szybko tracą ważność, a ich aktualizacja w tak krótkim czasie nie byłaby możliwa. Tylko część modułów odebrałaby bowiem aktualne dane, a w innych klucze utraciłyby ważność. W takim przypadku możliwa byłaby tylko transmisja z użyciem klucza podstawowego sesji – mało bezpieczna.

Ze względu na powyższe fakty, autor zaproponował wykorzystanie dwóch kluczy sesji: aktualnego i przyszłego. Należy tu dodać, że klucz przyszły jest oczywiście również aktualny, ale wykorzystany do szyfrowania będzie tylko, gdy poprzedni utraci swą ważność. Przy deszyfracji z kolei, jeśli oba klucze sesji są aktualne, wykorzystać należy najpierw ten starszy, a dopiero w przypadku niepowodzenia deszyfracji ten nowszy. Dla uwierzytelniania i kontroli integralności z użyciem kluczy sesji (nie zaleca się) należy zastosować to samo podejście. Takie rozwiązanie pozwala na automatyczną zmianę przez moduł nieważnego już klucza sesji na aktualny bez konieczności odbioru instrukcji aktualizacyjnych. Każda aktualizacja z kolei zawierać powinna klucz aktualny oraz przyszły, co pozwoli zapewnić elastyczność funkcjonowania zaproponowanego rozwiązania.

Problemem pozostaje tylko ustalenie długości czasu ważności poszczególnych kluczy sesji. Autor proponuje tu tylko, aby klucz przyszły był ważny co najmniej tak długo jak jego poprzednik, w celu zmaksymalizowania szansy na odbiór instrukcji aktualizujących. Jeśli na przykład jeden klucz ważny był przez miesiąc, to po jego dezaktualizacji klucz przyszły powinien być poprawny przynajmniej jeszcze przez miesiąc. Dzięki temu moduł kryptograficzny będzie miał cały miesiąc na odebranie następnej aktualizacji kluczy sesji. Okres miesiąca jest tu jednak tylko przykładem, a jego długość powinna być dobrana w zależności od aktualnych potrzeb i na podstawie długotrwałych testów zaproponowanego rozwiązania w warunkach rzeczywistych, co wykracza znacznie poza ramy tej pracy. Biorąc jednak pod uwagę

fakt korzystania w zaproponowany rozwiązaniu z tablic skramblujących, hierarchizacji kluczy, licznika AES-CTR, wykorzystującego aktualną datę oraz na dzień dzisiejszy duży poziom bezpieczeństwa oferowany przez algorytm AES, można założyć, że nawet stosunkowo długo stosowany klucz, będzie bezpieczny.

Proces aktualizacji kluczy może niestety wymagać jednak przesyłania dużej liczby wiadomości co pewien okres czasu, co byłoby uciążliwe zarówno dla centrum bezpieczeństwa jak i dla samych modułów oraz terminali odbiorczych. Ze względu na ten fakt klucze sesji przesyłane powinny być w wiadomościach typu multicast SMM. Przykład przedstawiono poniżej.

```
86
01
1002  -> multicast SMM
110C
1201  -> szyfrowanie kluczem grupy
1300  -> uwierzytelnianie RSA
1411
1506
1601
2F8601
3AF2FB00    -> adresat (grupa 0x077B36)
91A01122334455667788    -> pierwsza część klucza sesji A
91A11122334455667788    -> druga część klucza sesji A
91B01122334455667788    -> pierwsza część klucza sesji B
91B11122334455667788    -> druga część klucza sesji B
2A090B      -> data ważności klucza A (do końca grudnia 2009 roku)
2B0A01      -> data ważności klucza B (do końca stycznia 2010 roku)
```

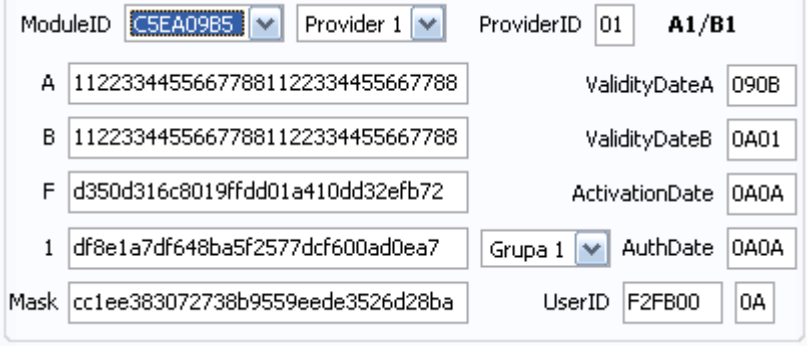
Powyższa wiadomość przesłana została do grupy o identyfikatorze *0xF2FB00*. Po przesłaniu wiadomości w pliku dziennika aplikacji można znaleźć:

```
      ModuleID    Status
1     DEA40F59    10
2     EC7AF504    10
3     9A74041A    10
4     88410FF5    10
5     8E280989    10
6     E76EF9F3    10
7     A4810874    10
8     D585052D    10
9     EBFB032C    10
10    C5EA09B5    10

11    D65202E4    50
12    A4810AF7    50
13    BEB90898    50
14    8233F9D7    50
15    96AB0903    50
16    C1A7FF86    50
17    84550E5C    50
18    AA04FDF9    50
19    9088F0F4    50
20    93DFFE93    50
```

Na podstawie powyższych informacji łatwo stwierdzić, że klucze zostały zaktualizowane w dziesięciu z dwudziestu modułów, do których dana wiadomość dotarła. Stało się tak dlatego, ponieważ moduły od 11-tego do 20-tego należą do grupy o identyfikatorze *0x455C66*, więc wiadomość ta nie była adresowana do nich i dlatego wystąpiła niepoprawna próba deszyfracji (status *0x50*). Widać tu jednak, że wysłanie jednej tylko wiadomości SMM pozwoliło na zaktualizowanie większej liczby modułów.

Na rysunku 4.10 przedstawiono przykład modułu, w którym zrealizowano poprawną aktualizację kluczy sesji oraz dat ich ważności.



4.10. Moduł *0xC5EA09B5* po aktualizacji kluczy sesji A oraz B.

W tym miejscu warto dodać, iż w pełni aktywny moduł (a tak właściwie *provider* w module) wraz z aktualnymi kluczami sesji może nie tylko wysyłać i odbierać (oczywiście adresowane do niego i dla jego poziomu autoryzacji) wszystkie typy wiadomości UDM, ale również korzystać z zaproponowanego kryptosystemu z maksymalnym oferowanym przez niego poziomem bezpieczeństwa.

Oczywistym jest tu fakt, że aktualizacja kluczy sesji dla różnych realizacji systemu musi być przeprowadzona niezależnie.

Podsumowując scenariusz aktualizacji kluczy można stwierdzić, iż wiadomości typu multicast SMM są bardzo elastyczne i pozwalają efektywnie aktualizować klucze w całej grupie. Najczęściej występować będą jednak sytuacje, w których to CB będzie chciało zaktualizować klucze i ewentualnie utrzymać autoryzację w danej grupie dla tylko części modułów. Taka procedura przedstawiona zostanie w jednym z kolejnych scenariuszy, a w następnym zaprezentowany zostanie przykładowy sposób dezaktywacji danej realizacji systemu w module.

Dezaktywacja pojedynczej realizacji w przykładowym module kryptograficznym

W ogólności proces dezaktywacji jest odwrotny w stosunku do procesu aktywacji. Dokładniej rzecz ujmując, dezaktywacja ma na celu doprowadzenie jednej konkretnej obsługiwanej przez moduł realizacji systemu do stanu sprzed aktywacji. Dezaktywacja dodatkowo najczęściej połączona jest ze skasowaniem kluczy sesji – moduł (realizacja) nieaktywny i tak nie będzie mógł ich wykorzystać.

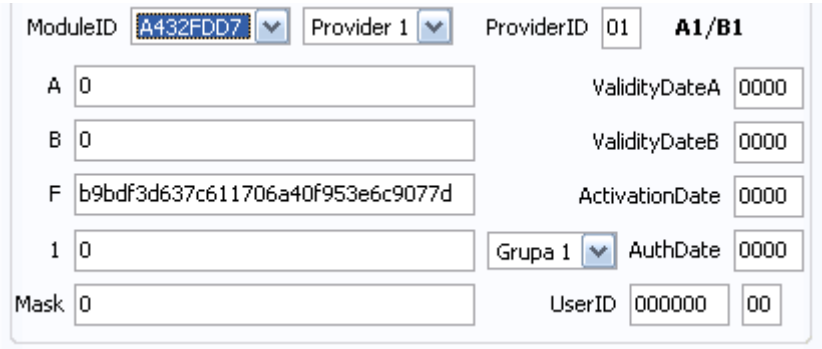
Podczas procesu dezaktywacji najczęściej kasuje się uprawnienia jak i inne parametry we wszystkich grupach, do których moduł został przydzielony. Można sobie jednak wyobrazić sytuacje, w której to moduł usuwany jest z pewnej tylko grupy i na przykład dodawany do innej (lub nie). Operacji związanych z przydziałem do konkretnych grup, które może przeprowadzić CB w stosunku do konkretnego modułu, jest sporo. W tym przypadku jednak przedstawiony zostanie proces całkowitej dezaktywacji modułu *0xA432FDD7* w obrębie realizacji systemu *0x01* z użyciem wiadomości unicast SMM i przy założeniu, że moduł ten posiada uprawnienia tylko w grupie nr 1. Nanoinstrukcje realizujące to zadanie przedstawiono poniżej.

```
86
01
1001  -> unicast SMM
1109
1200  -> szyfrowanie kluczem MK0
1300  -> uwierzytelnianie RSA
1411
1573
1635
2F8601
4AA432FDD7 -> adresat
01    -> anulowanie uprawnień w grupie nr 1
0A    -> skasowanie klucza A oraz wyzerowanie daty jego ważności
0B    -> skasowanie klucza B oraz wyzerowanie daty jego ważności
200000 -> wyzerowanie daty końca aktywacji modułu
```

Spośród powyższych nanoinstrukcji na uwagę zasługuje nano *0x01*, które powoduje następujące zdarzenia:

- Skasowanie klucza grupy nr 1;
- Wyzerowanie maski uprawnień grupy nr 1;
- Wyzerowanie daty końca uprawnień w grupie nr 1;
- Wyzerowanie identyfikatora *User ID* w grupie nr 1, a co za tym idzie wyzerowanie identyfikatora grupy (*Group ID 1*) oraz w grupie (*InGroup ID 1*).

Na rysunku 4.11 przedstawiono parametry modułu *0xA432FDD7* po dezaktywacji.



4.11. Moduł *0xA432FDD7* po całkowitej dezaktywacji.

Istotny jest w tym momencie fakt, że po całkowitej dezaktywacji konkretnej realizacji w module mogą oczywiście pozostawać w pełni aktywne inne *provider'y*. W przypadku jednak nieaktywnej realizacji systemu dostępny jest jedynie jej podstawowy klucz sesji F. Pozostają również klucz główny (MK0) oraz identyfikator modułu (*Module ID*), będące parametrami nie tylko nieusuwalnymi i niemodyfikowalnymi, ale oczywiście również wspólnymi dla wszystkich realizacji w danym module.

W tym miejscu warto przypomnieć, że w module znajdują się również identyfikatory (maksymalnie cztery) potencjalnie obsługiwanych realizacji systemu (*Provider ID 1 – 4*), tablice skramblujące A i B oraz publiczny klucz RSA systemu *0x86* – te parametry w sposób oczywisty również nie podlegają zmianom i nie ma możliwości ich usunięcia.

Sposób anulowania uprawnień lub całkowitej dezaktywacji przedstawiony w tym scenariuszu jest jednak mało elastyczny, ponieważ wymaga przesyłania wiadomości typu punkt-punkt. W dalszej kolejności omówiony zostanie dużo bardziej funkcjonalny oraz efektywny sposób jednoczesnej aktualizacji kluczy w danej grupie wraz z anulowaniem w niej uprawnień dla pewnej podgrupy modułów z możliwością całkowitej ich dezaktywacji w konkretnej i obsługiwanej realizacji systemu.

Aktualizacja kluczy sesji w całej grupie z jednoczesną dezaktywacją lub anulowaniem uprawnień podgrupy modułów

Elastyczność i efektywność zaproponowanego kryptosystemu ujawnia się między innymi poprzez możliwość zarządzania parametrami poszczególnych modułów kryptograficznych w sposób grupowy z wykorzystaniem wiadomości multicastowych przesyłanych do pewnych zbiorów terminali.

Takie podejście jest bardzo użyteczne w sytuacji, gdy centrum bezpieczeństwa chce przesłać na przykład klucze sesji do całej grupy. W warunkach rzeczywistych jednak okazać się może, że CB poza aktualizacją chce usunąć pewną grupę modułów z grupy bądź nawet dokonać ich dezaktywacji. Taka możliwość została przewidziana w zaproponowanym rozwiązaniu i zaprezentowano ją poniżej w postaci przykładowych nanoinstrukcji.

```
86
01
1002  -> multicast SMM
110C
1201  -> szyfrowanie kluczem MK0
1300  -> uwierzytelnianie RSA
1411
152D
162A
2F8601
3AA2EDB5    -> adresat (grupa 0xA2EDB5)
91A01122334455667788    -> pierwsza część klucza sesji A
91A11122334455667788    -> druga część klucza sesji A
91B01122334455667788    -> pierwsza część klucza sesji B
91B11122334455667788    -> druga część klucza sesji B
2A090B      -> data ważności klucza A (do końca grudnia 2009 roku)
2B0A01      -> data ważności klucza B (do końca stycznia 2010 roku)
30010203    -> usunięcie trzech modułów z grupy 0xA2EDB5
31060708    -> dezaktywacja trzech modułów
```

Nanoinstrukcje *0x30* oraz *0x31* pozwalają na jednoczesne odwołanie się do trzech modułów poprzez ich identyfikatory *InGroup ID* wpisane w treść tychże instrukcji. Dostępne są również nana postaci *0x50* i *0x51* oraz *0xF0* i *0xF1*, które pozwalają na skomunikowanie się jednocześnie z odpowiednio pięcioma bądź piętnastoma modułami. Takie rozwiązanie zastosowano dla zminimalizowania nadmiarowości w przesyłanych wiadomościach. Oczywistym jest fakt, że w danej wiadomości wystąpić może wiele wspomnianych tego typu instrukcji, co pozwala elastycznie dobrać ich typy oraz liczbę w celu zarządzania w dużym stopniu dowolną liczbą modułów z danej grupy. W przypadku jednak braku możliwości odpowiedniego dobrania instrukcji dla konkretnej liczby modułów, w treści nanoinstrukcji wpisać można bajty zerowe lub powtórzyć któreś z identyfikatorów *InGroup ID*.

Zaprezentowany tu mechanizm pozwala zminimalizować ilość wysyłanych wiadomości typu SMM poprzez ułatwienie dezaktywacji modułów z wykorzystaniem transmisji typu punkt-wielopunkt.

Do tego miejsca przedstawiono podstawowe scenariusze zarządzania modułami kryptograficznymi systemu *0x86* w warunkach idealnych (bez ingerencji osób

niepowołanych), co oczywiście nie oznacza, że są to wszystkie dostępne procedury. Na ich podstawie można jednak w łatwy sposób opracować wiele modyfikacji i rozszerzeń zaprezentowanych tu rozwiązań z wykorzystaniem dostępnych instrukcji systemowych.

Warto tu jeszcze nadmienić, iż różnorodne mechanizmy reagowania CB na sytuacje prób lub choćby ryzyka naruszenia funkcji bezpieczeństwa systemu, przedstawione zostaną w dalszej części omawianego rozdziału.

W kolejnych scenariuszach omówione zostaną z kolei standardowe schematy komunikowania się modułów (a w ten sposób pośrednio terminali użytkowników) pomiędzy sobą przy wykorzystaniu omawianego kryptosystemu.

Transmisja rozsiewcza

Zaproponowane rozwiązanie systemu bezpiecznej transmisji danych dla klasycznych modemów krótkofalowych pozwala na komunikacje pomiędzy nimi na wiele różnych sposobów. Jednym z nich jest właśnie transmisja rozsiewcza, która pozwala na odbiór wiadomości UDM przez wszystkie terminale, znajdujące się akurat w zasięgu nadajnika, gdy spełnione są następujące założenia:

- Moduły kryptograficzne terminali obsługują tą samą realizację systemu *0x86* i są w niej aktywne (przynajmniej w sposób podstawowy);
- Moduły wykorzystują te same numery tablic skramblujących A oraz B;
- W przypadku wykorzystywania schematów pośrednich uzyskiwania kluczy dla realizacji poufności konieczny jest również ważny i identyczny klucz sesji we wszystkich modułach.

Omawiany tu sposób transmisji jest możliwy do wykorzystania w systemie, lecz nie zapewnia on najwyższego poziomu bezpieczeństwa szczególnie w przypadku stosowania tylko klucza F do realizacji zarówno szyfrowania jak i uwierzytelniania oraz kontroli integralności. Dodatkowo terminale, komunikując się w ten sposób, tracą możliwość adresowania wiadomości oraz korzystania z udogodnień podziału na grupy tj.:

- Transmisja w obrębie własnej grupy;
- Kontrola poziomu autoryzacji nadawcy i odbiorcy (data jej wygaśnięcia oraz kanały wirtualne);
- Uwierzytelnianie nadawcy na poziomie grupy, a nie tylko całego systemu;
- Bezpieczniejsze szyfrowanie i kontrola integralności wiadomości.

Wiadomości rozsiewcze mają jednak również pewne zalety i są one następujące:

- Najmniejsza nadmiarowość protokolarna, wynikająca ze stosowania zaproponowanego kryptosystemu;
- Szybki dostęp do praktycznie wszystkich terminali w zasięgu;
- Możliwość transmisji nawet przy niepełnej aktywacji (bez przydziału do grup) i bez kluczy sesji (wystarczy klucz podstawowy sesji).

Przykładowa struktura nanoinstrukcji dla wiadomości broadcast UDM została zaprezentowana poniżej:

```
86
01
1011  -> broadcast UDM
1103
1202  -> szyfrowanie kluczem sesji
130F  -> uwierzytelnianie CMAC (klucz podstawowy F)
1411
15F9
1611
2F8601
4BED0C0AB6  -> nadawca wiadomości (module ID 0xED0C0AB6)
```

Warto w tym miejscu przypomnieć, że po powyższych instrukcjach występuje nano *0x0E*, a następnie dane użyteczne, które również kończą się nanem *0x0E*. W dalszej kolejności występują (jedna lub kilka – o ile dopełnienie jest w ogóle wymagane) nanoinstrukcje *0x0C*, a na końcu umieszczone zostaje nano *0x0D* z sekwencją uwierzytelniającą i pozwalającą na kontrolę integralności wiadomości – w tym przypadku jest to 128-bitowa sekwencja, wyznaczona z użyciem zmodyfikowanego algorytmu CMAC.

Transmisja w obrębie grupy użytkowników

Przesyłanie danych w grupie jest zalecanym sposobem komunikowania się terminali z wykorzystaniem zaproponowanego rozwiązania bezpiecznej transmisji danych (przesłanki przemawiające za tą tezą przedstawiono w poprzednim scenariuszu).

W kryptosystemie dostępne są trzy typy wiadomości UDM, które przesyłane są w obrębie grupy:

- Unicast;
- Multicast;
- Multicast w kanale wirtualnym.

W tabeli 4.1 przedstawiono podstawowe konstrukcje tych właśnie rodzajów wiadomości UDM.

Tab. 4.1. Przykładowe struktury nanoinstrukcji w wiadomościach UDM przesyłanych w obrębie grupy.

Unicast	Multicast	Multicast w kanale wirtualnym
86	86	86
01	01	01
1014 -> unicast	1012 -> multicast	1013 -> w kanale wirt.
1103	110A	1118
1202 -> klucz sesji	1202 -> klucz sesji	1202 -> klucz sesji
1301 -> klucz grupy	1301 -> klucz grupy	1301 -> klucz grupy
1411	1411	1411
1508	15BF	15C2
164F	1603	16F3
2F8601	2F8601	2F8601
3A13A52C -> Group ID	3A13A52C -> Group ID	3A13A52C -> Group ID
1A04 -> adresat	1B01 -> nadawca	1B01 -> nadawca
1B01 -> nadawca		1F01 -> nr kanału

W tym scenariuszu zakłada się, że istnieją dwie grupy po dziesięciu użytkowników i wykorzystywana jest tylko jedna realizacja systemu *0x86* o identyfikatorze *0x01*.

Po przesłaniu wiadomości multicastowej w obrębie jednej z grup (*0x13A52C*), w pliku dziennika znajdują się informacje przedstawione poniżej.

```
      ModuleID    Status
1     8776FA91    11    -> nadawca wiadomości multicastowej
2     D5AAF705    11
3     B65705CF    11
4     D0870895    11
5     A56F0577    11
6     E2A200DA    11
7     888BF12C    11
8     FB59F342    11
9     CCB2FED6    11
10    BB2D09AE    11

11    907807DB    50
12    D1B6F654    50
13    AB0100CB    50
14    F2CBFE6A    50
15    EB6F0005    50
16    E712FDD1    50
17    BB35F4DF    50
18    F6B8F141    50
19    EE77F8F0    50
20    F4DD04A8    50
```

Na podstawie powyższych danych łatwo stwierdzić, iż wszystkie moduły z grupy *0x13A52C* poprawnie odebrały przesłane dane (status *0x11*). W pozostałych przypadkach uzyskano status 50, który oznacza błędną deszyfrację, spowodowaną

niepoprawnym ustaleniem wartości licznika w algorytmie AES-CTR – użycie złego adresu *Group ID*.

W przypadku transmisji z użyciem z kolei kanałów wirtualnych przykładowy dziennik ma postać przedstawioną poniżej.

```
      ModuleID    Status
1     8776FA91    11    -> nadawca wiadomości w kanale wirtualnym
7     888BF12C    11

2     D5AAF705    41
3     B65705CF    41
4     D0870895    41
5     A56F0577    41
6     E2A200DA    41
8     FB59F342    41
9     CCB2FED6    41
10    BB2D09AE    41
11    907807DB    50
12    D1B6F654    50
13    AB0100CB    50
14    F2CBFE6A    50
15    EB6F0005    50
16    E712FDD1    50
17    BB35F4DF    50
18    F6B8F141    50
19    EE77F8F0    50
20    F4DD04A8    50
```

W przedstawionej tu sytuacji transmisja odbywała się w kanale numer 28. Tylko jeden z modułów (*0x888BF12C*), będących w grupie nadawcy, mógł wykorzystać wspomniany kanał i dlatego tylko on poprawnie odebrał dane. W pozostałych przypadkach zaobserwować można statusy błędne:

- *0x50* – oznacza, że moduł nie znajdował się w grupie nadawcy;
- *0x41* – odbiorca znajdował się w grupie nadawcy, ale nie posiadał odpowiedniego poziomu autoryzacji, a w tym konkretnym przypadku nie posiadał uprawnień do kanału numer 28.

W przypadku wiadomości unicastowych przesyłanych w obrębie grupy sytuacja podobna jest do transmisji w kanałach wirtualnych z dwoma zastrzeżeniami:

- Transmisja w kanale wirtualnym jest ciągle multicastowa i może być poprawnie odebrana przez wiele modułów, jeśli tylko obsługują one dany numer kanału;
- Transmisja unicastowa w grupie w przypadku błędnej adresacji zostaje odrzucona już na etapie próby deszyfracji wiadomości. W przypadku natomiast kanałów wirtualnych, niewystarczający poziom autoryzacji spowoduje odrzucenie wiadomości dopiero po pozytywnej próbie jej

deszyfracji – adresat jest bowiem poprawny nawet, gdy kanał wirtualny jest nieobsługiwany.

Ze względu na występowanie aż trzech typów wiadomości UDM przesyłanych w grupie, ten typ transmisji jest bardzo elastyczny i daje możliwość tworzenia wielu scenariuszy komunikacji pomiędzy terminalami.

W warunkach rzeczywistych jednak może okazać się, że istnieje konieczność skomunikowania dwóch terminali, które nie należą do wspólnej grupy. W takiej sytuacji istnieją zasadniczo trzy rozwiązania:

- Transmisja rozsiewcza – niezbyt dobre rozwiązanie zwarzywszy na fakt, iż inne terminale, obsługujące daną realizacje systemu, mogą odczytać wiadomość, a dodatkowo brak pola adresata powoduje trudności we wzajemnym komunikowaniu się;
- Stworzenie przez CB nowej grupy i umieszczenie w niej terminali, które chcą się komunikować w trybie punkt-punkt – zadanie złożone proceduralnie, ponieważ konieczna jest poinformowanie *providera* danej realizacji systemu – podejście to wymaga zatem czasu. Może być jednak użyteczne, gdy taki sposób transmisji wymagany będzie przez dłuższy okres czasu. Dodatkowo rozwiązanie to jest oczywiście możliwe do zrealizowania w systemie, ponieważ każdy moduł dla każdej realizacji systemu może zostać przypisany do maksymalnie ośmiu grup;
- Transmisja unicastowa poza grupę – nie tak bezpieczna jak w grupie, ale możliwa do zrealizowania w każdej chwili pomiędzy dowolnymi terminalami obsługującymi daną realizację systemu. Należy jednak pamiętać, że w takiej sytuacji nadawca musi znać pełen adres odbiorcy (pełen = *Module ID*).

Ostatnia wymieniona powyżej sytuacja omówiona zostanie krótko w następnym scenariuszu symulacyjnym.

Transmisja typu punkt-punkt poza grupą

Ten typ komunikacji podobnie jak w przypadku transmisji rozsiewczej nie oferuje przywilejów związanych z przydziałem modułów do grup, ale jednocześnie pozwala na właściwie dowolne konfiguracje typu punkt-punkt w obrębie danej realizacji systemu *0x86*.

W tym miejscu warto przypomnieć, iż w omawianym rozwiązaniu nie ma możliwości komunikowania się terminali pomiędzy konkretnymi realizacjami

kryptosystemu. Moduły kryptograficzne mogą jednak obsługiwać do czterech takich niezależnych realizacji, a więc stają się w ten sposób bardziej funkcjonalnymi i elastycznymi elementami w całym kryptosystemie *0x86*.

Ze względu na korzystanie w prezentowanym typie komunikacji z pełnej adresacji poprzez identyfikatory modułów, rozwiązanie to cechuje się niestety największą nadmiarowością protokolarną. W rzeczywistości jednak przyrost tej nadmiarowości nie jest duży i w stosunku do transmisji multicastowej w grupie wynosi (dla jednej wiadomości) cztery bajty, a pięć bajtów w przypadku trybu rozsiewczego.

Poniżej przedstawiono przykładową strukturę nanoinstrukcji wykorzystywaną w przypadku transmisji typu punkt-punkt poza grupą.

```
86
01
1015  -> unicast poza grupą
1103
120F  -> szyfrowanie kluczem F
130F  -> sekwencja CMAC z kluczem F
1411
155E
169E
2F8601
4AAEAB0116  -> adresat (Module ID)
4BAEAB0116  -> nadawca (Module ID)
```

Na tym etapie zakończono omawianie podstawowych procedur stosowanych w zaproponowanym krytposystemie. Można sobie oczywiście wyobrazić wiele innych i bardziej złożonych scenariuszy transmisji czy zarządzania całym systemem. Wykorzystać do tego celu można by choćby inne realizacje systemu czy większą ilość grup czy modułów. W tym miejscu chodziło jednak o zaprezentowanie podstawowej funkcjonalności i efektywności kryptosystemu *0x86*, a nazbyt skomplikowane sytuacje symulacyjne przesłoniłyby ten fakt.

W kolejnych dwóch scenariuszach zostanie zaprezentowany wpływ nieautoryzowanych modyfikacji przesyłanych wiadomości na poprawność i bezpieczeństwo funkcjonowania zaproponowanego rozwiązania.

Próby nieautoryzowanej modyfikacji treści nagłówka wiadomości

W scenariuszu tym zaprezentowane zostaną sposoby reakcji modułów kryptograficznych na różnego rodzaju zmiany dokonywane w treści jawnej części przesyłanych wiadomości.

Zostaną przeprowadzone dwa typy zmian:

- Modyfikacje zmieniające zaproponowaną strukturę nagłówka lub wprowadzające w polach nagłówka wartości niezgodne z zaproponowaną specyfikacją systemu *0x86;*
- Modyfikacje spełniające wymogi kryptosystemu *0x86*.

Opis wszystkich zaproponowanych sytuacji przedstawiono w tabeli 4.2, w której to opisano również zaobserwowane reakcje modułów. Badania te przeprowadzono przy założeniu istnienia dwóch grup aktywnych terminali z ważnymi kluczami sesji w jednej realizacji systemu. Modyfikowano nagłówki wiadomości zarówno SMM jak i UDM.

Tab. 4.2. Wpływ modyfikacji nagłówka wiadomości na bezpieczeństwo kryptosystemu *0x86*.

Opis modyfikacji	**Reakcja systemu**
Modyfikacja pierwszego bajtu nagłówka, który powinien mieć zawsze stałą wartość równą *0x86*	Odrzucenie wiadomości przez wszystkie moduły. Status: *0x20* (błędny kryptosystem).
Zmiana typu lub kolejności naoinstrukcji w nagłówku	Odrzucenie wiadomości przez wszystkie moduły. Status: *0x62* (błędny typ nano w nagłówku).
Zmiana treści którejś z nanoinstrukcji na niepoprawną w systemie *0x86*	Odrzucenie wiadomości przez wszystkie moduły. Status: *0x63* (błędna treść nano w nagłówku).
Modyfikacja drugiego bajtu nagłówka, który określa typ realizacji systemu (identyfikator providera), na nieobsługiwany przez moduły	Odrzucenie wiadomości przez wszystkie moduły. Status: *0x30* (nieobsługiwany provider).
Zmiana typu wiadomości (na poprawny w systemie)	Odrzucenie wiadomości przez wszystkich adresatów*. Statusy mogą być różne, ale zawsze błędne. Możliwy jest na przykład: - status *0x50* (błędna deszyfracja); - status *0x60* (błędne nano); - status *0x63* (błędna treść nano w nagłówku).
Zmiana długości wiadomości	Odrzucenie wiadomości przez wszystkich adresatów*. Statusy mogą być różne, ale zawsze błędne. Możliwy jest na przykład: - status *0x70* (błąd kontroli integralności); - status *0x60* (błędne nano).
Zmiana sposobu szyfrowania (na poprawny w systemie)	Odrzucenie wiadomości przez wszystkie moduły. Status: *0x50* (błąd deszyfracji).
Zmiana sposobu uwierzytelniania/kontroli integralności (na poprawny w systemie)	Odrzucenie wiadomości przez wszystkich adresatów*. Status: *0x70* (błąd kontroli integralności).
Zmiana numerów wektorów skramblujących (na poprawne w systemie)	Odrzucenie wiadomości przez wszystkie moduły. Status: *0x50* (błąd deszyfracji).
Zmina numerów tablic skramblujących A/B (na poprawne w systemie)	Odrzucenie wiadomości przez wszystkie moduły. Statusy mogą być różne, ale zawsze błędne. Możliwy jest na przykład: - status *0x50* (błąd deszyfracji); - status *0x91* lub *0x92* (nieaktywna tablica skramblująca odpowiednio A lub B).
*) Wszystkie moduły niebędące adresatami odrzucają wiadomość ze statusem *0x50*.	

Nagłówek jako jedyna jawna część wiadomości jest potencjalnie najbardziej narażony na różnego rodzaju ataki czy próby jego modyfikacji. W projekcie systemu przewidziano jednak tego typu sytuacje i większość z nich jest wykrywana już w trakcie kontroli samego nagłówka bądź próby deszyfracji pierwszej sekwencji 128-bitowej, występującej zaraz za nim. Są to jednak tylko mechanizmy pomocnicze i dodatkowe, które pozwalają na odrzucenie wiadomości już na wczesnych etapach jej analizy. Dopiero jednak kontrola integralności całej wiadomości z wykorzystaniem zmodyfikowanego algorytmu CMAC czy podpisu cyfrowego daje ostateczną i pewną odpowiedź.

Uogólniając można zatem stwierdzić, że mechanizmy pomocnicze pozwalają szybko wykryć większość prostych modyfikacji nagłówka i w ten sposób natychmiast odrzucić całą wiadomość, ale to algorytm kontroli integralności decyduje o ostatecznym i pozytywnym zatwierdzeniu danej wiadomości.

W kolejnym scenariuszu przedstawione zostaną z kolei próby modyfikacji tajnej części wiadomości i ich wpływ na poprawność funkcjonowania całego systemu. Pod uwagę wzięty zostanie zatem ponownie i przede wszystkim mechanizm kontroli integralności wiadomości jako całość.

Próby nieautoryzowanej modyfikacji treści tajnej części wiadomości

W scenariuszu tym analizie poddane zostaną zmiany treści wiadomości w różnych jej obszarach i to zarówno w komunikacji pomiędzy terminalami jak i pomiędzy centrum bezpieczeństwa a terminalami. Opis i skutki wprowadzanych modyfikacji przedstawiono w tabeli 4.3. Analizę tą przeprowadzono przy założeniu istnienia dwóch grup aktywnych terminali z ważnymi kluczami sesji w jednej realizacji systemu *0x86*.

Przedstawione tu sytuacje jasno potwierdzają, że zaproponowane rozwiązanie jest bezpieczne i posiada dwupoziomowy mechanizm kontroli integralności przesyłanych wiadomości. W pierwszej kolejności kontroli podlega na bieżąco struktura i treść analizowanych elementów. Jeśli na którymś etapie tej wstępnej analizy zostanie stwierdzona jakaś niezgodność, to cała wiadomość jest odrzucana. Takie postępowanie pozwala szybko eliminować dużą część niepoprawnych wiadomości bez konieczności analizy sekwencji RSA czy CMAC.

Tab. 4.3. Wpływ modyfikacji elementów tajnej części wiadomości na bezpieczeństwo kryptosystemu.

Opis modyfikacji	**Reakcja systemu**
Modyfikacja w obrębie pierwszego tajnego nano 2F	Odrzucenie wiadomości przez wszystkie moduły. Status: *0x50* (błąd deszyfracji).
Modyfikacja w obrębie pola adresata	Odrzucenie wiadomości przez wszystkich adresatów*. Statusy mogą być różne, ale zawsze błędne. Możliwy jest na przykład: - status *0x40* (błędny adresat) – zmodyfikowano treść pola adresata; - status *0x60* (błędne nano) – zmodyfikowano typ nano adresata.
Modyfikacja w obrębie innych nanoinstrukcji	Odrzucenie wiadomości przez wszystkich adresatów*. Statusy mogą być różne, ale zawsze błędne. Możliwy jest na przykład: - status *0x61* (błędna treść nano); - status *0x60* (błędne nano); - status *0x70* (błąd kontroli integralności).
Modyfikacja w obrębie danych użytkowych	Odrzucenie wiadomości przez wszystkich adresatów*. Status: *0x70* (błąd kontroli integralności).
Modyfikacja w obrębie sekwencji uwierzytelniającej	Odrzucenie wiadomości przez wszystkich adresatów*. Status: *0x70* (błąd kontroli integralności).
*) Wszystkie moduły niebędące adresatami odrzucają wiadomość ze statusem *0x50*.	

Kolejne dwa scenariusze mają na celu ocenę możliwości ingerencji w transmisję z wykorzystaniem technik powtarzania i filtrowania wiadomości zarówno SMM jak i UDM. Badania te pozwolą między innymi odpowiedzieć na pytanie o możliwość manipulacji mechanizmami zarządzania w zaproponowanym kryptosystemie.

Próby ingerencji w transmisję z wykorzystaniem techniki powtarzania wiadomości

Ponowne przesyłanie wcześniej zapisanych wiadomości może być skutecznym atakiem na kryptosystemy niezabezpieczone przed tego typu działaniami. Atakujący, obserwując zachowania odbiorców pewnych wiadomości, może w przyszłości wymusić te same rezultaty, przesyłając taką samą informację po raz kolejny.

W zaproponowanym rozwiązaniu jednak działania takie są w dużej mierze utrudnione bądź nawet niemożliwe do zrealizowania. Jest to spowodowane przynajmniej kilkoma elementami zaimplementowanymi w zaproponowanym rozwiązaniu, jak choćby:

- Ograniczona ważność kluczy sesji – wiadomość po pewnym czasie (jeśli do szyfrowania wykorzystywano klucze sesji) przestanie być poprawna, ponieważ wygaśnie ważność klucza użytego do jej zakodowania. Zważywszy jednak na fakt, że klucz sesji może być aktualny przez na przykład cały miesiąc i

dodatkowo nie musiał on być wcale użyty (klucz podstawowy), to zabezpieczenie to w praktyce będzie mało skuteczne, lecz mimo wszystko należy o nim pamiętać.

- Tajne pole adresata i nadawcy wiadomości – ten mechanizm utrudnia już w dużym stopniu stosowanie ataku powtarzania wiadomości. Atakujący bowiem nie jest w stanie określić adresata i nadawcy wiadomości, ponieważ są to informacje zaszyfrowane. Oczywistym jest jednak fakt, że dokładna i długotrwała obserwacja zachowań terminali będących w zasięgu atakującego pozwoli w pewnych sytuacjach określić nadawcę bądź choćby odbiorcę wiadomości. Z tego też powodu zabezpieczenie to nie jest w pełni skuteczne.
- Znacznik czasu stosowany w algorytmie AES-CTR – jest to główny mechanizm, zabezpieczający system przed atakami, wykorzystującymi metodę powtarzania wiadomości. Autorska modyfikacja algorytmu AES-CTR zakłada bowiem wykorzystanie sygnatury czasowej przy wyznaczaniu wartości licznika dla kolejnych szyfrowanych sekwencji 128-bitowych. Dana sygnatura jest ważna przez minutę. Przy niepowodzeniu jednak próby deszyfracji proces ten jest powtarzany również dla sygnatury poprzedniej oraz następnej w celu wyeliminowania problemu niedokładnej synchronizacji zegarów pomiędzy terminalami oraz ewentualnych opóźnień, wynikających z możliwych retransmisji wiadomości, realizowanych przez protokół ARQ. Przy takim podejściu dana sygnatura może być ważna teoretycznie nawet przez 3 minuty, ale w rzeczywistości jest to najczęściej krótszy odcinek czasu. Oznacza to że wiadomość wysłana po upływie tego czasu zostanie automatyczne odrzucona przez wszystkie terminale już na etapie próby jej deszyfracji (status *0x50*).

Rozwiązania tu przedstawione zapewniają zatem w bardzo dużym stopniu ochronę przed atakami powtarzania wiadomości. Mechanizm ten jest niezależny zarówno od typu wiadomości jak i od samej jej struktury, a więc jest on wykorzystywany przy komunikowaniu się centrum bezpieczeństwa z terminalami jak i podczas przesyłania danych użytkowników.

Przy takim podejściu konkretny moduł jest wrażliwy na atak powtórzenia wiadomości tylko w bardzo krótkim czasie po jej otrzymaniu. W praktycznych sytuacjach jednak tego typu ataki stosuje z dużo większymi opóźnieniami, a kilka

minut nie stanowi rzeczywistego zagrożenia, ponieważ atakujący musiałby w bardzo krótkim czasie zapewnić warunki, w których ewentualne powtórzenie wiadomości przyniosłoby mu jakiekolwiek wymierne korzyści.

Próby ingerencji w zarządzanie systemem z wykorzystaniem techniki filtrowania wiadomości SMM

Metoda ataku na kryptosystem, polegająca na filtrowaniu odbieranych wiadomości z wykorzystaniem jawnych informacji zawartych w ich nagłówkach, jest często stosowanym i stosunkowo prostym rozwiązaniem. Może ono umożliwić na przykład uniknięcia dezaktywacji modułu kryptograficznego, stosowanego w terminalu użytkownika.

W zaproponowanym rozwiązaniu jednak ryzyko, wynikające ze stosowania wspomnianej tu techniki filtrowania, jest niewielkie. Należy pamiętać, że w omawianym systemie mogą być przesyłane dwa typy wiadomości SMM – typu punkt-punkt lub punkt-wielopunkt. Z tego powodu jedynym możliwym sposobem filtrowania jest blokowanie jednego (lub obu) z tych typów wiadomości.

W praktyce blokowanie wszystkich wiadomości SMM pozwoliłoby co prawda na uniknięcie dezaktywacji, lecz należy tu pamiętać, iż zarówno ważność kluczy sesji, autoryzacja w danej grupie jak i aktywacja danej realizacji kryptosystemu są ograniczone czasowo. Blokowanie zatem wszystkich typów wiadomości zarządzających byłoby skuteczne tylko do pewnego momentu.

Dla przykładu po wygaśnięciu ważności kluczy sesji szyfrowanie mogłoby być realizowana tylko w mało bezpieczny sposób – z użyciem klucza podstawowego sesji. Po wygaśnięciu autoryzacji w danej grupie uwierzytelnianie mogłoby być realizowane również jedynie z użyciem tego klucza, a sama transmisja odbywać musiałaby się poza obrębem wspomnianej grupy. Dodatkowo po upłynięciu daty końca aktywacji danej realizacji, wykonana zostałaby automatyczna dezaktywacja, uniemożliwiając w ten sposób jakąkolwiek transmisję z wykorzystaniem zaproponowanego rozwiązania.

Warto w tym miejscu dodać, iż pomimo braku możliwości natychmiastowej dezaktywacji modułu, gdy stosowana jest technika filtracji wszystkich wiadomości SMM, CB ma ciągle możliwość zarządzania pozostałymi modułami, a to daje kilka dodatkowych opcji, które mogą w pewien sposób powstrzymać atakującego lub ograniczyć możliwości jego działania. Trzy najbardziej standardowe rozwiązania to:

1. Zmiana stosowanych w systemie kluczy sesji – szybkie rozwiązanie i możliwe do zrealizowania z użyciem wiadomości multicastowych. Jest to modyfikacja poza harmonogramem aktualizacji i wymaga zmiany ważnych i stosowanych w danej realizacji systemu kluczy sesji, co przez pewien okres czasu może spowodować pewne trudności w komunikacji pomiędzy terminalami.
2. Zmiana klucza grupy i/lub identyfikatora grupy – aktualizacja ta powinna dotyczyć wszystkich grup, do których należy moduł, który powinien wykonać dezaktywację. Oczywiście rozwiązanie to jest skuteczne również w przypadku konieczności samego tylko anulowania uprawnień danego modułu w konkretnych grupach. Wszystkie moduły należące do wspomnianych grup powinny otrzymać dane aktualizacyjne (nowy klucz i/lub identyfikator grupy). Niestety rozwiązanie to nie jest tak szybkie jak poprzednie, ponieważ wymaga wysyłania wiadomości unicastowych do poszczególnych modułów.
3. Zmiana aktualnych tablic skramblujących (przynajmniej jednej) – szybka i stosunkowo prosta modyfikacja (wykorzystanie zarówno wiadomości unicastowych jak i multicastowych SMM). Jest to skuteczna obrona przed atakiem filtrowania wiadomości. Należy tylko pamiętać że dostępna jest ograniczona liczba takich zmian (cztery tablice A oraz cztery tablice B dostępne w systemie).

W praktyce pierwsze dwa rozwiązania mogą być zastosowane równorzędnie, co uniemożliwi atakującemu jakąkolwiek transmisję w grupach i dodatkowo posiadane przez niego klucze sesji staną się bezużyteczne. Pozostanie mu tylko możliwość korzystania z mało bezpiecznego klucza F. Próba zatem odbioru jakiejkolwiek transmisji w grupach lub z użyciem kluczy sesji zakończy się niepowodzeniem już na etapie próby deszyfracji (status *0x50*). Natomiast po odpowiednim czasie nastąpi kompletna i automatyczna dezaktywacja danej realizacji systemu i w tym momencie odbierane wiadomości UDM będą całkowicie pomijane, a ich nadawanie stanie się niemożliwe.

Wadą niestety wspomnianych tu dwóch metod jest stosunkowo spora ingerencja w konfigurację poszczególnych modułów i w rzeczywistych warunkach CB często będzie musiało wybrać stosowanie jednej z tych technik (lub obu) bądź oczekiwanie na automatyczną dezaktywację kluczy i uprawnień w module atakującego. To drugie

podejście może być oczywiście powiązane z ciągłymi próbami dezaktywacji poprzez na przykład wiadomości multicastowe, przenoszące aktualizacje kluczy sesji.

Metoda trzecia jest z kolei bardzo skuteczna i po jej zastosowaniu terminal filtrujący wiadomości nie ma żadnej możliwości odbioru wiadomości UDM, a ich nadawanie będzie nieskuteczne, bo wykorzystane zostaną nieaktywne tablice skramblujące. Jak już jednak wspomniano rozwiązanie to ma poważne ograniczenie – można je zastosować tylko ograniczoną liczbę razy (maksymalnie sześć dla każdej realizacji systemu *0x86*).

Docelowo zatem CB musi zdecydować się na któreś rozwiązanie lub zastosować podejście łączące zaprezentowane tu techniki. Sytuacja jest jednak zgoła odmienna, jeśli centrum bezpieczeństwa posiada informacje (lub zakłada) o tym, że atakujący odfiltrowuje tylko jeden typ wiadomości SMM.

Możliwe jest bowiem blokowanie przez atakującego tylko wiadomości multicastowych. Wiązałoby się to jednak z brakiem możliwości uzyskania przez niego aktualnych kluczy sesji, co z pewnością nie byłoby pożądane z jego perspektywy.

Z kolei odfiltrowywanie wiadomości unicastowych oznaczałoby teoretycznie zablokowanie możliwości bezpośredniego zażądania modułem, co mogłoby przynieść korzyści atakującemu. Możliwe stałoby się w ten sposób (przynajmniej teoretycznie) uzyskiwanie kluczy sesji (z wiadomości SMM punkt-wielopunkt), a dezaktywacja przesyłana w wiadomościach typu punkt-punkt nie zostałaby odebrana.

Taka sytuacja jednak nie jest niebezpieczna, ponieważ należy pamiętać, że w zaproponowanym systemie przewidziano dezaktywację pojedynczych modułów również z wykorzystaniem wiadomości multicastowych poprzez nana: *0x31*, *0x51* lub *0xF1*. Możliwe jest również samo anulowanie uprawnień w danej grupie z użyciem nanoinstrukcji: *0x30*, *0x50*, *0xF0*.

Podsumowując zatem niniejszy scenariusz, można stwierdzić, iż technika filtrowania wiadomości zarządzających w zaproponowanym systemie może być w pewnym ograniczonym stopniu skuteczna. Przewidziano tu jednak mechanizmy, które mogą wyeliminować ten problem lub zminimalizować jego wpływ na ogólny poziom bezpieczeństwa, oferowany przez zaproponowane rozwiązanie.

W kolejnych scenariuszach przedstawione zostaną pewne mechanizmy zaradcze systemu, stosowane w przypadku kradzieży modułu kryptograficznego lub nieautoryzowanego dostępu do kluczy bądź innych danych systemowych.

Działania CB w przypadku kradzieży/zagubienia aktywnego modułu kryptograficznego

Każdy moduł kryptograficzny niezależnie od metody jego realizacji (wbudowany w modem lub niezależny) powinien być zabezpieczony przed wydobyciem z niego jakichkolwiek kluczy czy innych danych systemowych. Na potrzeby tego scenariusza zakłada się zatem, że elementy te są bezpieczne, a jedynie sam moduł może zostać wykorzystany przez nieautoryzowane osoby. Dla uproszczenia przyjmuje się tu również, że w module aktywna jest tylko jedna realizacja systemu.

Procedura postępowania centrum bezpieczeństwa powinna być zrealizowana w następującej kolejności:

1. Natychmiastowe wysłanie wiadomości unicastowej SMM z instrukcjami dezaktywującymi dany moduł. Działanie to powinno być powtarzane co pewien czas (coraz rzadziej) do momentu automatycznego wygaśnięcia aktywacji w module.
2. Zawieranie instrukcji dezaktywującej dany moduł w wysyłanych wiadomościach multicastowych SMM, przenoszących aktualne klucze sesji dla grup, do których należał skradziony moduł kryptograficzny. Działanie to powinno być realizowane do momentu automatycznego wygaśnięcia aktywacji w module lub do czasu zmiany parametrów danej grupy (patrz punkt trzeci i poprzedni scenariusz).
3. Jeżeli w ocenie CB istnieje ryzyko ataku opartego o filtrowanie wiadomości zarządzających, centrum bezpieczeństwa może rozważyć zastosowanie mechanizmów przedstawionych w poprzednim scenariuszu.

Powyższa procedura wymaga niestety wielokrotnego powtarzania poszczególnych jej elementów, ponieważ CB nie posiada informacji zwrotnych i nie wie kiedy dane instrukcje zostaną wykonane. Zważywszy jednak na fakt, iż powtórzenia tu wspomniane nie muszą być częste, a procedury dezaktywujące mogą być na przykład dołączane do wiadomości aktualizujących klucze sesji, to proces dezaktywacji skradzionych modułów może zostać przeprowadzony bardzo sprawnie.

W kolejnym scenariuszu przedstawione zostaną procedury zaradcze CB, stosowane w przypadku naruszenia mechanizmu dostępności i wycieku kluczy lub innych danych systemowych.

Działania CB w przypadku wycieku kluczy i/lub innych danych systemowych

Prezentowany w niniejszej pracy kryptosystem został tak zaprojektowany, by posiadał pewne mechanizmy, zabezpieczające go w sytuacjach naruszenia funkcji dostępności i wycieku pewnych kluczy i innych danych systemowych, zawartych w poszczególnych modułach kryptograficznych. Nie rozważa się tu bowiem i jednocześnie uznaje za mało prawdopodobne (choć bardzo niebezpieczne), sytuacji, w której wyciek informacji pojawiłby się ze strony CB. Zdarzenie takie bowiem mogłoby spowodować możliwość wycieku najistotniejszego i decydującego o bezpieczeństwie parametru systemowego jakim jest prywatny klucz RSA. Taka sytuacja oznaczałaby niestety możliwość podszywania się pod centrum bezpieczeństwa, co automatycznie oznaczałoby bezużyteczność systemu *0x86*.

W prezentowanym scenariuszu zatem rozważa się tylko możliwość wycieku elementów zawartych w modułach kryptograficznych. W tabeli 4.4 przedstawiono przykładowe sytuacje ujawnienia pojedynczej informacji z konkretnego modułu wraz z opisem ewentualnych działań podejmowanych przez CB w takiej sytuacji.

Tab. 4.4. Procedury zaradcze CB przy wycieku pojedynczego parametru systemowego.

Opis sytuacji	**Procedura zaradcza**	**Ryzyko naruszenia bezpieczeństwa Przed/Po**
Wyciek któregokolwiek identyfikatora (modułu, grupy, itp.)	Brak (elementy jawne)	Brak/Brak
Wyciek publicznego klucza RSA systemu *0x86*	Brak (element jawny)	
Wyciek aktualnego klucza sesji	Brak lub ewentualna aktualizacja klucza sesji (Wyciek samego tylko klucza sesji nie zmniejsza poziomu bezpieczeństwa.)	
Wyciek klucza głównego MK0	Brak lub ewentualnie procedura jak przy kradzieży/zagubieniu modułu (Jawny klucz MK0 bez znajomości aktualnych tablic skramblujących jest właściwie bezużyteczny. Jest to jednak parametr niemodyfikowalny i zarazem główny klucz modułu, co stanowi pewne ryzyko samo w sobie.)	Niskie/Brak
Wyciek klucza grupy	Brak lub ewentualna zmiana klucza grupy (Samo ujawnienie klucza danej grupy nie stanowi zagrożenia bez znajomości innych parametrów systemowych.)	Brak/Brak
Wyciek klucza F	Brak (Klucza F nie można zaktualizować, ale bez znajomości aktualnych tablic skramblujących jego znajomość jest bezużyteczna.)	
Wyciek aktualnej tablicy skramblującej A lub B	Brak (Wyciek pojedynczej aktywnej tablicy skramblujących nie stanowi zagrożenia.)	

Dodatkowo w tabeli tej określono subiektywne ryzyko (brak, niskie, średnie lub wysokie) naruszenia bezpieczeństwa transmisji przed i po zastosowaniu procedur zaradczych przez centrum bezpieczeństwa. Dla uproszczenia w rozważaniach założono pojedynczą realizację systemu.

Jak można łatwo zauważyć, ujawnienie pojedynczego parametru systemowego zawartego w module kryptograficznym nie niesie ze sobą właściwie naruszenia poziomu bezpieczeństwa transmisji. Oczywistym jest jednak fakt, że jednocześnie wzrasta w takiej sytuacji prawdopodobieństwo wystąpienia takiej sytuacji w przyszłości.

W dalszej kolejności zatem analizie poddane zostaną podobne sytuacje jak w tabeli 4.4, ale tym razem przy założeniu, że wcześniej ujawniono klucz F danej realizacji systemu. W takim przypadku nie ma oczywiście możliwości jego aktualizacji i należy rozważyć jak taka sytuacja wpłynie na bezpieczeństwo transmisji w momencie ujawnienia kolejnych elementów systemowych. Rozważania te zaprezentowano w tabeli 4.5.

Tab. 4.5. Procedury zaradcze CB przy wycieku klucza F oraz innego parametru systemowego.

Opis sytuacji	**Procedura zaradcza**	**Ryzyko naruszenia bezpieczeństwa Przed/Po**
Wyciek aktualnego klucza sesji	Brak lub ewentualna aktualizacja klucza sesji (Wyciek klucza sesji i klucza F nie zmniejsza poziomu bezpieczeństwa, jeśli tablice skramblujące są tajne.)	Brak/Brak
Wyciek klucza głównego MK0	Brak lub ewentualnie procedura jak przy kradzieży/zagubieniu modułu (Jawny klucz MK0 bez znajomości aktualnych tablic skramblujących jest bezużyteczny. Jest to jednak parametr niemodyfikowalny i zarazem główny klucz modułu, co stanowi pewne ryzyko samo w sobie.)	Niskie/Brak
Wyciek klucza grupy	Brak lub ewentualna zmiana klucza grupy (Wyciek klucza danej grupy oraz klucza F nie stanowi zagrożenia bez znajomości tablic skramblujących.)	Brak/Brak
Wyciek aktualnej tablicy skramblującej A	Brak lub ewentualna zmiana aktualnej tablicy skramblującej A (Wyciek aktualnej tablicy skramblującej A oraz klucza F nie stanowi zagrożenia bez znajomości tablicy B)	
Wyciek aktualnej tablicy skramblującej B	Brak lub ewentualna zmiana aktualnej tablicy skramblującej B (Znajomość tablicy skramblującej B oraz klucza F nie stanowi zagrożenia bez znajomości tablicy A)	
Wyciek obu aktualnych tablic skramblujących	Zmiana aktualnych tablic skramblujących (Znajomość tablic skramblujących oraz klucza F umożliwia tworzenie poprawnych wiadomości UDM (w oparciu o klucz podstawowy) oraz ich deszyfrację o ile nie korzystano przy ich szyfrowaniu z kluczy sesji czy grupy.)	Niskie/Brak

Na podstawie analizy sytuacji przedstawionych w tabeli 4.5 można stwierdzić, że CB posiada mechanizmy zaradcze, pozwalające przeciwdziałać wyciekowi zarówno klucza F jak i innego parametru systemowego zawartego w module.

Warto tu również zauważyć, że szczególnie niebezpiecznym zjawiskiem jest równoczesny wyciek klucza F oraz aktualnych tablic skramblujących, ponieważ w takiej sytuacji CB powinno zmienić aktualnie stosowane tablice (przynajmniej jedną), a liczba takich zmian w danej realizacji jest ograniczona.

Warto zatem przeanalizować sytuację, w której to wyciekł klucz F oraz obie tablice skramblujące, a ich zmiana nie jest już możliwa w danej realizacji. W tabeli 4.6 przedstawiono więc sytuacje, w których to ujawniony zostanie jakiś klucz systemowy, a klucz F oraz tablice skramblujące są również jawne. Istotny jest tu fakt, że w takim przypadku, niezależnie od działań CB, atakujący będzie w stanie tworzyć poprawne wiadomości UDM (w obrębie danej realizacji kryptosystemu) z użyciem klucza podstawowego oraz deszyfrować takie, przy których generowaniu nie korzystano z kluczy sesji czy grupy. W takim wypadku istnieje więc zawsze pewne ryzyko naruszenia bezpieczeństwa transmisji. Będzie mieć ono jednak marginalne znaczenie, jeśli w praktycznej realizacji kryptosystemu wykorzystywane będą klucze sesji, a do uwierzytelniania i kontroli integralności klucze grup.

Na podstawie informacji zebranych w tabeli 4.6 można łatwo stwierdzić, iż nawet w sytuacji wycieku znacznej ilości danych systemowych centrum bezpieczeństwa ma zawsze możliwość reagowania na zaistniałą sytuację. Oczywistym jest jednak fakt, iż ujawnienie niemodyfikowalnych parametrów systemowych stanowi dla zaproponowanego rozwiązania pewne ryzyko. Kryptosystem jednak jako całość może nadal spełniać swe funkcje i być bezpiecznym o ile jego użytkownicy stosować będą (jeśli jest to możliwe) wiadomości UDM przesyłane w grupie, a do ich szyfrowania wykorzystywać będą klucze sesji, natomiast do uwierzytelniania klucze swoich grup.

W tym miejscu warto jeszcze wspomnieć, że sytuacje wycieku tajnych parametrów systemowych wydaja się nieść za sobą większe ryzyko niż na przykład kradzież modułu kryptograficznego. W tej drugiej sytuacji bowiem atakujący ograniczony jest do możliwości jakie daje mu skradziony element. Nie może on na przykład podszywać się pod inne moduły czy użytkowników grup. Ponadto CB może łatwiej dowiedzieć się o zaistniałej sytuacji – zgłoszenie kradzieży czy zagubienia. W

przypadku wycieku parametrów systemowych z kolei atakującego nie ogranicza sam moduł a jedynie ilość, posiadanych przez niego, informacji.

Tab. 4.6. Procedury zaradcze CB przy wycieku klucza F, tablic skramblujących oraz innego parametru systemowego.

Opis sytuacji	**Procedura zaradcza**	**Ryzyko naruszenia bezpieczeństwa Przed/Po**
Wyciek aktualnego klucza sesji	Aktualizacja klucza sesji (Wyciek klucza sesji, klucza F oraz aktualnych tablic skramblujących jest niebezpieczny, ponieważ pozwala na deszyfrację oraz tworzenie wszystkich wiadomości UDM przesyłanych poza obrębem grupy. Możliwa jest też deszyfracja wiadomości UDM przesyłanych w grupie, jeżeli do ich zakodowania stosowano klucz podstawowy. Generowanie wiadomości przesyłanych w grupie jest również możliwe, ale tylko przy użyciu klucza F.)	Średnie/Niskie
Wyciek klucza głównego MK0	Procedura jak przy kradzieży/zagubieniu modułu (Ujawniony klucz MK0 wraz z aktualnymi tablicami skramblującymi stanowi ogromne niebezpieczeństwo dla systemu, ponieważ atakujący może deszyfrować dane przesyłane w wiadomościach unicast SMM, kierowanych do danego terminala.)	Wysokie/Niskie
Wyciek klucza grupy	Zmiana klucza grupy (Wyciek klucza danej grupy, klucza F oraz tablic skramblujących stanowi ogromne niebezpieczeństwo dla systemu, ponieważ atakujący może deszyfrować dane przesyłane w wiadomościach multicast SMM, kierowanych do grupy danego terminala. Atakujący może również deszyfrować wiadomości UDM, jeśli do ich kodowania nie użyto kluczy sesji. Możliwe jest także generowanie wiadomości UDM z wykorzystaniem klucza podstawowego i z sekwencją CMAC opartą o klucz grupy.)	Wysokie/Niskie
Wyciek klucza grupy oraz aktualnego klucza sesji	Zmiana klucza grupy oraz klucza sesji (Ujawnienie tak dużej ilości danych systemowych jest bardzo niebezpieczne, ponieważ w tej sytuacji atakujący może deszyfrować oraz tworzyć wszystkie typy wiadomości UDM poza grupą oraz w obrębie grupy, dla której posiada klucz. Dodatkowo atakujący w tym przypadku może deszyfrować wiadomości multicast SMM, adresowane do danej grupy modułów.)	Wysokie/Niskie

Kontynuacja tych rozważań zostanie przeprowadzona w kolejnym rozdziale niniejszej pracy, w którym to zaproponowany kryptosystem poddany zostanie kompleksowej i w dużym stopniu obiektywnej ocenie z wykorzystaniem standardu CVSS. Przeanalizowane zostaną najważniejsze i najciekawsze sytuacje, przedstawione w niniejszym scenariuszu.

Podsumowując niniejszy rozdział, można stwierdzić, iż zaproponowany system bezpiecznej transmisji danych w łączu krótkofalowym może być skutecznie zarządzany nawet w sytuacjach ryzyka naruszenia bezpieczeństwa transmisji.

Dodatkowo techniki zaradcze w nim przewidziane mogą skutecznie eliminować lub chociaż minimalizować efekty naruszenia mechanizmu dostępności. Co więcej rozwiązanie to jest odporne na szereg różnorodnych prób ingerencji w transmisję i pozwala oprzeć się popularnym metodom ataków na systemy kryptograficzne.

Rozdział V

Ocena poziomu bezpieczeństwa zaproponowanego kryptosystemu

W poprzednich rozdziałach niniejszej pracy przedstawiono istotne zagadnienia kryptograficzne, związane ściśle z zaproponowanym rozwiązaniem systemu bezpiecznej transmisji danych w paśmie HF. Zaprezentowano również sam system; zarówno jego projekt wraz z teoretyczną specyfikacją jak i sposobem jego implementacji. W dalszej kolejności poddano go analizie pod kątem oceny poprawności jego funkcjonowania i możliwości przeciwdziałania różnego typu sytuacjom wyjątkowym.

W tej części pracy zostanie podjęta próba pełniejszej, uniwersalnej oraz porównywalnej i w pewnym stopniu obiektywnej oceny bezpieczeństwa zaproponowanego rozwiązania. Do tego celu konieczna jest jednak pewna metryka, umożliwiająca tego typu działanie. W literaturze nie ma wielu metod oceny bezpieczeństwa systemów [50, 76, 92], a ich uniwersalność i obiektywność stoi pod znakiem zapytania. Istnieje jednak rozwiązanie zatwierdzone przez NIST (*National Institute of Standards and Technology*), pozwalające na ocenę ryzyka jakie stanowią ogólnie rozumiane wrażliwości rozwiązań IT oraz określenie ich wpływu na te systemy. Standard CVSS (*Common Vulnerability Scoring System*) w wersji 2.10 [28, 47], bo o nim tu mowa, jest pewną podstawą, umożliwiającą ilościową ocenę i porównanie różnego typu systemów IT pod względem wpływu ich podatności na ogólnie rozumiany poziom bezpieczeństwa przez te rozwiązania oferowany.

Wrażliwość czy podatność jest tu rozumiana jako pewna wada lub słabość danego rozwiązania, która może prowadzić do naruszenia poufności, wiarygodności, integralności i/lub szeroko rozumianej dostępności systemu.

W skład CVSS wchodzą trzy grupy metryk: podstawowa, czasowa oraz środowiskowa. Każda z nich pozwala wyznaczyć ocenę z zakresu od 0 do 10, której nieodłączną część stanowi wektor, zawierający pewną tekstową reprezentację elementów wykorzystanych do wyznaczenia konkretnej wartości liczbowej. Im jest ona wyższa tym większe ryzyko niesie ze sobą dana wrażliwość i tym większy niekorzystny wpływ ma ona na dany system.

Grupa podstawowa reprezentuje właściwą wielkość wpływu podatności, która jest względnie stała w danym systemie. Grupa czasowa z kolei odzwierciedla charakterystyki wrażliwości danego rozwiązania, które ulegają zmianie w czasie. Wreszcie grupa środowiskowa reprezentuje z kolei cechy podatności unikalne dla każdej grupy użytkowników w danym środowisku.

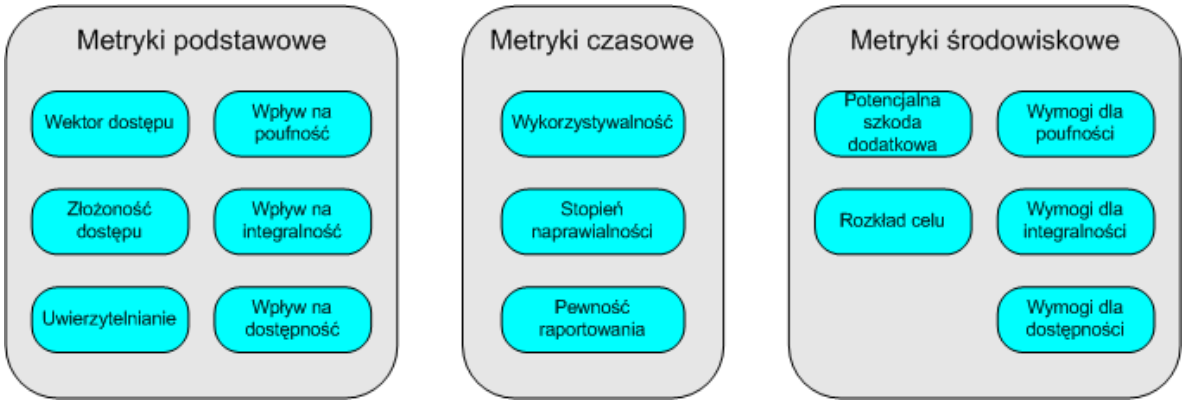
W ogólności standard CVSS umożliwia menadżerom, dostawcom, twórcom systemów IT jak i naukowcom na wykorzystanie tego uniwersalnego narzędzia do oceny i porównania wpływu różnego rodzaju podatności pod kątem ryzyka jakie niosą one ze sobą. Dzięki temu rozwiązaniu możliwe jest zatem pośrednio określenie poziomu bezpieczeństwa oferowanego przez konkretny system w przypadku, gdy obarczony jest on pewną wrażliwością czy słabością.

5.2. System oceny CVSS

Aktualnie identyfikacja podatności systemów jest trudna ze względu na wielość platform sprzętowych i programowych oraz konieczność priorytetyzacji elementów danego rozwiązania pod względem poziomu zagrożenia dla bezpieczeństwa całości systemu. Dodatkowo, jak już wspomniano, dostępne metody oceny bezpieczeństwa systemów nie są standaryzowane i ujednolicone. Rozwiązaniem jest zatem wykorzystanie CVSS, które zapewnia standaryzowaną ocenę podatności, priorytetyzuje ryzyko oraz jest rozwiązaniem otwartym.

5.2.1. Informacje podstawowe

CVSS składa się z trzech grup metrycznych: podstawowej (*Base*), czasowej (*Temporal*) oraz środowiskowej (*Environmental*). W skład każdej z nich wchodzi pewna grupa metryk (patrz rysunek 5.1).



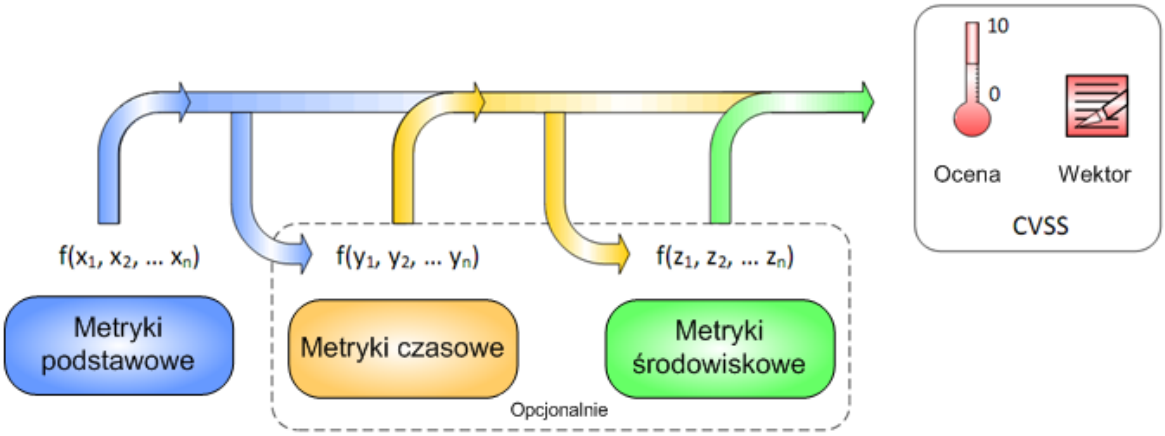
Rys. 5.1. Grupy metryczne w CVSS [47].

Poniżej przedstawiono definicje poszczególnych grup:

- Podstawowa – reprezentuje nieodłączne i fundamentalne charakterystyki podatności, które są stałe w czasie i nie zależą od środowiska;
- Czasowa – odzwierciedla charakterystyki wrażliwości, które są zmienne w czasie, ale niezależne od środowiska użytkownika danego systemu;
- Środowiskowe – reprezentuje charakterystyki podatności, które są istotne i unikalne dla konkretnego środowiska użytkownika.

Celem podstawowej grupy metryk CVSS jest definicja charakterystyk podatności i przekazanie użytkownikowi jasnej oraz intuicyjnej informacji na ich temat. W dalszej kolejności wykorzystana może zostać również informacja czasowa i środowiskowa, która dokładniej odzwierciedli ryzyko w konkretnym środowisku użytkownika.

Po przypisaniu metrykom podstawowym konkretnych wartości, na podstawie odpowiednich równań wyznaczana jest ocena w skali od 0 do 10 oraz tworzony jest pewien wektor (patrz rysunek 5.2). Sekwencja ta jest łańcuchem tekstowym zawierającym wartości przypisane do każdej z metryk. Służy ona przedstawieniu sposobu wyznaczenia punktacji dla konkretnej wrażliwości. Istotne jest zatem, aby wektor był zawsze prezentowany wraz z punktacją. Więcej szczegółów odnośnie tychże łańcuchów tekstowych znaleźć można w dalszej części niniejszego rozdziału.



Rys. 5.2. Schematyczny sposób wyznaczania metryk CVSS [47].

Opcjonalnie punktacja podstawowa może zostać wykorzystana dla pewnej inicjalizacji metryk czasowych i środowiskowych, których wyznaczenie dostarcza dodatkowych i kontekstowych informacji odnośnie wrażliwości w danym środowisku. Nie jest to jednak konieczne i czasem wystarcza tylko punktacja i wektor podstawowy. Po wyznaczeniu metryk czasowych i środowiskowych uzyskiwany jest również wynik z zakresu od 0 do 10 dla każdej z grup.

Oceny z wykorzystaniem systemu CVSS dla grupy podstawowej i czasowej dokonywać powinni sami twórcy i właściciele danego rozwiązania IT, ponieważ posiadają większą wiedzę na temat wrażliwości ich systemów niż sami użytkownicy. Oceny środowiskowej jednak powinni podjąć się już wykorzystujący daną aplikację, ponieważ są oni w stanie w pełni ocenić wpływ wrażliwości na bezpieczeństwo wewnątrz ich własnego środowiska.

5.2.2. Grupy metryczne

W tej części niniejszego rozdziału, szczegółowo przedstawione zostaną metryki systemu CVSS z uwzględnieniem ich podziału na grupy. W ten sposób dostępne stanie się narzędzie, pozwalające na ocenę poziomu bezpieczeństwa różnego rodzaju systemów IT, a w szczególności zaproponowanego rozwiązania bezpiecznej transmisji danych w paśmie HF.

5.2.2.1. Metryki podstawowe

Grupa metryk podstawowych uwzględnia charakterystyki podatności danego systemu, które są stałe w czasie i nie zależą od środowiska użytkownika. Metryki: wektor dostępu (*Access Vector*), złożoność dostępu (*Access Complexity*) oraz uwierzytelnianie (*Authentication*) określają czy w ogóle istnieją pewne słabości systemu, a jeśli tak, to czy wymagane są jakieś szczególne warunki dla ich wykorzystania.

Trzy kolejne metryki (metryki „wpływu”) opisują jaki bezpośredni wpływ na system miałoby wykorzystanie jego potencjalnych podatności. Oceniane są tu niezależnie efekty utraty poufności, integralności oraz dostępności. Nie zawsze bowiem słabość systemu wpływać musi na wszystkie te elementy.

W dalszej części tego paragrafu metryki podstawowe zostaną przedstawione w sposób bardziej szczegółowy.

Wektor dostępu jest metryką, która odzwierciedla w jaki sposób pewna wada systemu jest wykorzystywana. Możliwe wartości tejże miary przedstawiono w tabeli 5.1. W ogólności jednak można stwierdzić, że im bardziej „zdalny” może być atakujący system tym wyższa jest punktacja podatności, czyli sama wrażliwość danego rozwiązania jest większa – ma ona większy wpływ i stanowi większe ryzyko.

Tab. 5.1. Wartości metryki wektor dostępu (AV - *Access Vector*).

Wartość metryki	**Wartość liczbowa**	**Opis**
Lokalna (L - *Local*)	0,395	Podatność może być wykorzystana tylko lokalnie. Wymagany jest tu albo fizyczny dostęp do systemu albo konto lokalne w systemie.
Przyległa sieciowa (A – *Adjacent Network*)	0,646	Słabość może być wykorzystana z użyciem dostępu przyległego do sieci. Konieczny jest tu albo dostęp do domeny rozsiewczej albo kolizyjnej systemu. Przykładowe sposoby takiego dostępu: podsieci IP, Bluetooth, IEEE 802.11 lub lokalny segment Ethernetowy.
Sieciowa (N - *Network*)	1,0	Wrażliwość może być wykorzystana z użyciem dostępu sieciowego – system musi być podłączony do sieci. Nie ma konieczności dostępu lokalnego czy nawet poprzez sieć lokalną. Taka podatność jest często określana jako „wykorzystanie zdalne”.

Złożoność dostępu jest metryką określającą trudność wykorzystania wrażliwości systemu, gdy atakujący uzyska już do niego dostęp. Zwykle samo już uzyskanie dostępu do systemu pozwala na wykorzystanie jego słabości, ale niekiedy wymagane są pewne dodatkowe działania. Możliwe wartości wspomnianej metryki przedstawiono w tabeli 5.2. Warto tu tylko dodać, że im niższa jest złożoność tym wyższa jest punktacja podatności (większy wpływ wrażliwości na system).

Tab. 5.2. Wartości metryki złożoność dostępu (AC - *Access Complexity*).

Wartość metryki	**Wartość liczbowa**	**Opis**
Wysoka (H - *High*)	0,35	Istnieją specjalizowane warunki dostępu, na przykład: • W większości konfiguracji atakujący musi posiadać podwyższone uprawnienia lub zdobyć dostęp do jakiegoś dodatkowego systemu; • Działania atakującego mogą być łatwo wykryte. W systemie są przeprowadzane na przykład jakieś nietypowe i podejrzane działania; • Dana konfiguracja wrażliwości jest bardzo rzadka w praktyce. • Jeśli nawet istnieją warunki sprzyjające, to czasowe okno dostępu jest bardzo wąskie;
Średnia (M - *Medium*)	0,61	Warunki dostępu są w pewnym stopniu specjalizowane: • Atakujący jest ograniczony do grupy użytkowników systemu, na pewnym, ograniczonym poziomie autoryzacji; • Zebrane muszą zostać pewno dodatkowe informacje zanim atak może zostać przeprowadzony; • Dana konfiguracja nie jest domyślną i standardową (podatność istnieje na przykład tylko dla pewnych schematów uwierzytelniania, a już dla innych system jest odporny); • Działania atakującego nie mogą być już tak łatwo wykryte – istnieją sytuacje, w których nie uda się tego zrobić;
Niska (L - *Low*)	0,71	Specjalizowane warunki dostępu nie istnieją: • Atakowany system jest udostępniony szerokiemu gronu użytkowników anonimowych lub niewiarygodnych; • Dana konfiguracja systemu jest domyślna i standardowa; • Atak może zostać przeprowadzony manualnie i nie wymaga dużych umiejętności ani zbierania dodatkowych informacji; • Często występują warunki sprzyjające;

Kolejna metryka określa minimalną liczbę uwierzytelnień do systemu, których musi dokonać atakujący, aby wykorzystać wrażliwość danego rozwiązania. Miara ta nie bierze pod uwagę złożoności procesu uwierzytelniania, a jedynie sam fakt czy atakujący musi potwierdzić swą tożsamość w systemie zanim będzie mógł wykorzystać jego podatność. Możliwe wartości tejże metryki przedstawiono w tabeli 5.3. Im mniej żądań autoryzacji jest wymaganych tym wyższa jest punktacja dla niniejszej metryki.

Warto w tym miejscu dodać, że metryka uwierzytelniania jest inna niż wektor dostępu. Uwierzytelnianie bowiem jest rozważane dopiero w momencie, gdy uzyskano już dostęp do systemu. W przypadku na przykład lokalnie wykorzystywanej wrażliwości, uwierzytelnianie jest uznawane (jednokrotne lub wielokrotne – patrz tabela 5.3), jeśli jest realizowane oprócz samego logowania się do systemu. Przykładem lokalnie wykorzystywanej podatności, która wymaga uwierzytelnienia jest atak na bazę danych pracującą pod kontrolą systemu Unix. Jeśli bowiem konieczne jest uwierzytelnienie się jako prawidłowy użytkownik tejże bazy, to metryka niniejsza powinna być ustawiona jako uwierzytelnianie jednokrotne.

Tab. 5.3. Wartości punktacji metryki uwierzytelniania (Au - *Authentication*).

Wartość metryki	**Wartość liczbowa**	**Opis**
Wielokrotna (M - *Multiple*)	0,45	Wykorzystanie słabości systemu wymaga wielokrotnego uwierzytelniania przez atakującego. Nie jest tu istotny fakt czy za każdym razem używane są te same dane uwierzytelniające (ta sama tożsamość).
Jednokrotna (S - *Single*)	0,56	Wymagane jest jednokrotne uwierzytelnienie aby uzyskać dostęp i wykorzystać podatność systemu.
Brak (N - *None*)	0,704	Uwierzytelnienie nie jest konieczne, aby uzyskać dostęp i wykorzystać wrażliwość systemu.

Metryka ta powinna być zastosowana w oparciu o uwierzytelnienie jakiego wymaga atakujący przed rozpoczęciem samego ataku. Jeśli bowiem jakiś system jest wrażliwy na pewne niepożądane działania jeszcze przed uwierzytelnieniem, to metryka niniejsza powinna być ustawiona na wartość N (*None*). Z kolei jeśli podatność może być wykorzystana dopiero po uwiarygodnieniu, to metryka ta powinna być ustawiona na S (*Single*) lub M (*Multiple*) w zależności od liczby wymaganych uwierzytelnień.

Następna metryka mierzy wpływ wykorzystania słabości systemu na jego poufność, która jest tu rozumiana jako ograniczony dostęp do informacji tylko dla autoryzowanej grupy użytkowników. Poufność jest tu jednak rozumiana również jako

zapobieganie dostępowi oraz ujawnieniu tychże informacji użytkownikom nieautoryzowanym. Możliwe wartości tej metryki przedstawiono w tabeli 5.4. Warto tu dodać, że większy wpływ na poufność systemu oznacza wyższą punktację wrażliwości.

Tab. 5.4. Wartości punktacji metryki wpływu na poufność (C – *Confidentiality Impact*).

Wartość metryki	**Wartość liczbowa**	**Opis**
Brak (N - *None*)	0	Brak wpływu na poufność systemu.
Częściowa (P - *Partial*)	0,275	Występuje pewne ujawnienie informacji. Dostęp do niektórych informacji jest możliwy, ale atakujący nie posiada kontroli nad tym, co jest możliwe do uzyskania lub zakres utraty ograniczeń systemowych jest ściśle określony i częściowy.
Całkowita (C - *Complete*)	0,66	Występuje całkowite ujawnienie, powodujące wyjawienie wszystkich informacji. Atakujący ma możliwość odczytania wszelkich danych przechowywanych/przesyłanych w systemie.

Kolejna metryka określa wpływ wykorzystania podatności systemu na jego integralność, która jest tu rozumiana jako wiarygodność oraz gwarantowana prawdziwość informacji. Możliwe wartości tejże metryki przedstawiono w tabeli 5.5. W ogólności większy wpływ na integralność oznacza wyższą punktację podatności.

Tab. 5.5. Wartości punktacji metryki wpływu na integralność (I – *Integrity Impact*).

Wartość metryki	**Wartość liczbowa**	**Opis**
Brak (N - *None*)	0	Brak wpływu na integralność systemu.
Częściowa (P - *Partial*)	0,275	Możliwa jest modyfikacja pewnych tylko danych, ale atakujący nie posiada kontroli nad tym co może zostać zmodyfikowane lub zakres jego działań jest ściśle ograniczony.
Całkowita (C - *Complete*)	0,66	Występuje pełne naruszenie integralności systemu. Ma miejsce utrata bezpieczeństwa systemu, co w rezultacie naraża go jako całość. Wszystkie dane mogą zostać zmodyfikowane.

Następna metryka określa wpływ wykorzystania wrażliwości systemu na jego osiągalność, która jest tu rozumiana jako dostępność[21] do zasobów informacyjnych. Atakujący, wykorzystując na przykład pasmo lub inne zasoby, ogranicza w ten sposób osiągalność danego systemu. Możliwe wartości tej metryki przedstawiono w tabeli 5.6. Warto tu nadmienić, że większy wpływ na dostępność zwiększa ryzyko jakie niesie ze sobą dana wrażliwość systemu.

[21] Dostępność do zasobów jest odmiennym zagadnieniem w stosunku do dostępności, rozumianej jako funkcja bezpieczeństwa (mechanizm kryptograficzny).

Tab. 5.6. Wartości punktacji metryki wpływu na dostępność (A – *Availability Impact*).

Wartość metryki	Wartość liczbowa	Opis
Brak (N - *None*)	0	Brak wpływu na dostępność systemu.
Częściowa (P - *Partial*)	0,275	Ma miejsce redukcja wydajności lub przerwy w dostępie do zasobów. Nie występuje jednak całkowite od nich odcięcie.
Całkowita (C - *Complete*)	0,66	Występuje całkowite wstrzymanie dostępu do zasobów informacyjnych. Atakujący może uczynić zasoby całkowicie niedostępnymi - użytkownicy są od nich odcięci.

5.2.2.2. Metryki czasowe

Niebezpieczeństwo powodowane podatnością systemu może ulegać zmianom w czasie. Standard CVSS definiuje trzy czynniki uwzględniające to zjawisko: potwierdzenie szczegółów technicznych wrażliwości, poprawa statusu wrażliwości oraz dostępność rozwiązań i technik wykorzystania.

Ze względu na fakt, że metryki czasowe są opcjonalne w standardzie CVSS, każda z nich zawiera pewną niezdefiniowaną wartość, która nie ma wpływu na końcową punktację. Wybranie jej pozwala na pominięcie konkretnej metryki, gdy nie ma ona na przykład zastosowania w danym przypadku lub nie ma dostępu do odpowiednich informacji.

Jedną z metryk czasowych jest „wykorzystywalność", która określa aktualny stan dostępności rozwiązań i technik, pozwalających na wykorzystanie podatności danego systemu. Publiczna dostępność sposobu (algorytmu/kodu), który może być zastosowany dla wykorzystania słabości, powoduje wzrost liczby potencjalnych ataków na system, zwiększając w ten sposób wagę jego podatności.

W rzeczywistości początkowo istnienie algorytmu, umożliwiającego wykorzystanie słabości systemu, może być tylko teoretyczne. Następnie mogą na przykład powstać publikacja przedstawiające dowód istnienia takiego kodu lub prezentujące już gotowe rozwiązanie funkcjonalne albo też zawierające wymagane detale techniczne konieczne do wykorzystania wrażliwości danego systemu.

Dostępność sposobu wykorzystania słabości może ewoluować od dowodu jego istnienia po funkcjonalne rozwiązanie, mogące być skutecznym i łatwo dostępnym. Możliwe wartości prezentowanej metryki przedstawiono w tabeli 5.7. W ogólności im łatwiej wykorzystać podatność tym większy wpływ na bezpieczeństwo ma wrażliwość danego rozwiązania.

Tab. 5.7. Wartości punktacji metryki „wykorzystywalność" (E - *Exploitability*).

Wartość metryki	Wartość liczbowa	Opis
Niedowiedziona (U - *Unproven*)	0,85	Nie istnieje sposób umożliwiający wykorzystanie podatności lub jest on całkowicie teoretyczny.
Dowód istnienia (POC – Proof-Of-Concept)	0,9	Dostępny jest dowód istnienia sposobu wykorzystującego podatność lub demonstracja ataku, ale rozwiązanie to nie jest praktyczne dla większości istniejących systemów. Taki algorytm lub technika nie jest funkcjonalna we wszystkich sytuacjach i może wymagać znacznych modyfikacji przez atakującego o dużej wiedzy i umiejętnościach.
Funkcjonalna (F - *Functional*)	0,95	Istnieje funkcjonalny sposób wykorzystania wrażliwości. Rozwiązanie takie jest skuteczne w większości sytuacji, gdy tylko istnieje jakaś słabość systemu.
Wysoka (H - *High*)	1,0	Istnieje funkcjonalny i autonomiczny sposób wykorzystania podatności lub wykorzystanie nie jest konieczne a szczegóły są szeroko dostępne. Rozwiązanie takie działa w każdej sytuacji lub jest aktywnie dostarczane poprzez mobilnego i autonomicznego agenta (np. wirusa).
Niezdefiniowana (ND – *Not Defined*)	1,0	Przypisanie tej wartości do metryki nie zmieni końcowej punktacji – oznacza pominięcie niniejszej metryki.

Kolejną metryką czasową jest stopień naprawialności danej wrażliwości, który jest ważnym czynnikiem dla ustalania priorytetów podatności. Typowa słabość jakiegoś systemu występuje najczęściej przy jego pierwszej publikacji. Pewne obejścia i modyfikacje mogą oferować tymczasową poprawę, a czasem należy wykorzystać mechanizmy wbudowane w samo rozwiązanie lub oficjalne modyfikacje dostarczone przez właściciela lub twórcę danego systemu. Możliwe wartości prezentowanej metryki przedstawiono w tabeli 5.8. W ogólności im mniej oficjalna i permanentna jest poprawka tym wyższa punktacja wrażliwości.

Tab. 5.8. Wartości punktacji metryki stopień naprawialności (RL – *Remediation Level*).

Wartość metryki	Wartość liczbowa	Opis
Oficjalna Poprawka (OF – *Official Fix*)	0,87	Istnieje kompletne rozwiązanie udostępnione przez dostawcę systemu. Jest to oficjalna poprawka lub przewidziany gotowy mechanizm zaradczy w systemie.
Tymczasowa poprawka (TF – *Temporary Fix*)	0,9	Istnieje oficjalna, ale tymczasowa poprawka lub mechanizm zaradczy. Uwzględnia się tu również obejścia lub dodatkowe narzędzia udostępnione przez dostawcę.
Obejście (W - *Workaround*)	0,95	Dostępne jest nieoficjalne rozwiązanie. Jest to poprawka lub mechanizm, pozwalający na obejście lub złagodzenie danej wrażliwości systemu.
Niedostępna (U - *Unavailable*)	1,0	Nie ma dostępnego rozwiązania problemu lub mechanizm jest niemożliwy do zastosowania.
Niezdefiniowana (ND – *Not Defined*)	1,0	Przypisanie tej wartości do metryki nie zmieni końcowej punktacji – oznacza pominięcie niniejszej metryki.

Kolejną metryką czasową jest pewność raportowania, która określa stopień ufności w fakt istnienia słabości systemu oraz wiarygodności jej detali technicznych.

Często bowiem zdarza się, że publikowana jest tylko informacja o istnieniu wrażliwości danego rozwiązania, bez podawania jakichkolwiek szczegółów technicznych. Słabość systemu może być w dalszej kolejności potwierdzona przez autora/właściciela danej technologii. Ryzyko związane z daną podatnością jest tym większe im wyższa jest pewność istnienia takiej słabości.

Przedstawiana metryka określa również pośrednio minimalny poziom wiedzy technicznej, który musi posiadać ewentualny atakujący, aby móc wykorzystać daną wrażliwość systemu. Możliwe wartości prezentowanej tu miary zestawiono w tabeli 5.9. Warto tu jeszcze dodać, że im pewniejszy jest fakt istnienia wrażliwości tym wyższa jest punktacja podatności.

Tab. 5.9. Wartości punktacji metryki pewność raportowania (RC – *Report Confidence*).

Wartość metryki	**Wartość liczbowa**	**Opis**
Niepotwierdzona (UC - *Unconfirmed*)	0,9	Istnieje pojedyncze i niepotwierdzone źródło lub kilka sprzecznych informacji. Wiarygodność tych danych jest niewielka.
Niepotwierdzona oficjalnie (UR - Uncorroborated)	0,95	Dostępnych jest wiele nieoficjalnych i najczęściej niezależnych źródeł. Na tym etapie występować mogą pewne techniczne nieścisłości lub inne niejednoznaczności odnośnie szczegółów.
Potwierdzona (C - *Confirmed*)	1,0	Wrażliwość została potwierdzona przez autora bądź sprzedawcę danej technologii. Podatność może być również potwierdzona „zewnętrznie” przez publikację funkcjonalnego sposobu/kodu lub dowodu jego istnienia albo poprzez powszechne jego wykorzystywanie.
Niezdefiniowana (ND – *Not Defined*)	1,0	Przypisanie tej wartości do metryki nie zmieni końcowej punktacji – oznacza pominięcie niniejszej metryki.

5.2.2.3. Metryki środowiskowe

Zróżnicowanie środowiska może mieć ogromny wpływ na poziom ryzyka jaki powodować będzie dana podatność systemu. Metryki środowiskowe standardu CVSS opisują charakter wrażliwości w powiązaniu ze środowiskiem użytkowników IT. Ze względu na fakt, że miary te są opcjonalne, to każda z nich może mieć wartość niezdefiniowaną, która nie ma wpływu na końcową punktację. Opcja taka jest wykorzystywana, gdy dana metryka nie ma zastosowania lub w celu jej pominięcia.

Jedną z metryk środowiskowych jest potencjalna szkoda dodatkowa, która określa możliwość utraty, poprzez zniszczenie bądź kradzież zasobów materialnych lub innej własności (na przykład informacji). Miara ta może odnosić się bezpośrednio do strat ekonomicznych (zmniejszenie przychodu i/lub produkcji). Możliwe wartości tej metryki przedstawiono w tabeli 5.10. Oczywistym jest tu fakt, że im wyższa możliwość szkody tym wyższa jest punktacja wrażliwości.

Tab. 5.10. Wartości punktacji metryki potencjalna szkoda dodatkowa (CDP – *Collateral Damage Potential*).

Wartość metryki	Wartość liczbowa	Opis
Brak (N - *None*)	0	Nie ma ryzyka utraty zasobów materialnych czy zmniejszenia produkcyjności i/lub zysków.
Niska (L - *Low*)	0,1	Sukcesywne wykorzystywanie danej słabości systemu może prowadzić do niewielkich strat lub szkód. Może też wystąpić niewielkie zmniejszenie produkcyjności i/lub zysków.
Średnio-niska (LM – *Low-Medium*)	0,3	Sukcesywne wykorzystywanie danej słabości systemu może prowadzić do umiarkowanych strat lub szkód. Może też wystąpić umiarkowane zmniejszenie produkcyjności i/lub zysków.
Średnio-wysoka (MH – *Medium-High*)	0,4	Sukcesywne wykorzystywanie danej słabości systemu może prowadzić do znacznych strat lub szkód. Może też wystąpić znaczne zmniejszenie produkcyjności i/lub zysków.
Wysoka (H - *High*)	0,5	Sukcesywne wykorzystywanie danej słabości systemu może prowadzić do katastrofalnych strat lub szkód. Może też wystąpić katastrofalne zmniejszenie produkcyjności i/lub zysków.
Niezdefiniowana (ND – *Not Defined*)	0	Przypisanie tej wartości do metryki nie zmieni końcowej punktacji – oznacza pominięcie niniejszej metryki.

Precyzyjne znaczenie słów: niewielkie, umiarkowane, znaczące oraz katastrofalne musi zostać zdefiniowane niezależnie dla każdego systemu.

Kolejną metryka środowiskową jest rozkład celu, który określa procentową część systemu (realizacji/aplikacji systemu), na którą wpływ będzie mieć dana podatność. Możliwe wartości wspomnianej metryki przedstawiono w tabeli 5.11. Warto w tym miejscu dodać, że im większa część systemu będzie pod wpływem wrażliwości, tym wyższa będzie punktacja końcowa.

Tab. 5.11. Wartości punktacji metryki rozkład celu (TD – *Target Distribution*).

Wartość metryki	Wartość liczbowa	Opis
Brak (N - *None*)	0	System (lub jego realizacje) nie jest celem lub cele są wysoce specjalizowane i istnieją tylko w warunkach laboratoryjnych. Efektywnie 0 % środowiska podlega ryzyku.
Niska (L - *Low*)	0,25	Cele istnieją wewnątrz środowiska, ale na małą skalę. Od 1 do 25 % środowiska podlega ryzyku.
Średnia (M – *Medium*)	0,75	Cele istnieją wewnątrz środowiska, ale na średnią skalę. Od 26 do 75 % środowiska podlega ryzyku.
Wysoka (H – *High*)	1,0	Cele istnieją wewnątrz środowiska i to na dużą skalę. Od 76 do 100 % środowiska podlega ryzyku.
Niezdefiniowana (ND – *Not Defined*)	1,0	Przypisanie tej wartości do metryki nie zmieni końcowej punktacji – oznacza pominięcie niniejszej metryki.

Kolejne trzy metryki środowiskowe należą do pewnej grupy (wymogi bezpieczeństwa) i dlatego omówione zostaną łącznie. Umożliwiają one przystosowanie punktacji CVSS do danego rozwiązania w zależności od poziomu ważności dla systemu (jego właścicieli, użytkowników, itp.) elementów takich jak:

poufność (CR – *Confidentiality Requirement*), integralność (IR – *Integrity Requirement*) oraz dostępność (AR – *Availability Requirement*). Jeśli na przykład dla pewnej aplikacji najistotniejsza jest dostępność, to tejże metryce przypisana powinna być wartość wyższa niż innym składnikom wymogów bezpieczeństwa.

Całkowity wpływ oceny środowiskowej wyznaczany jest w oparciu o odpowiednie metryki podstawowe – metryki wpływu. Trzy przedstawione tu miary są niejako współczynnikami wagowymi dla podstawowych metryk wpływu. Należy jednak w tym miejscu dodać, że same metryki podstawowe nie ulegają oczywiście żadnym zmianom. Można ponownie posłużyć się przykładem. Metryka wpływu na poufność (C) będzie mieć zwiększoną ważność, gdy miara wymóg poufności (CR) przyjmie wartość Wysoka (H – *High*). Dodatkowo metryki z grupy wymogi bezpieczeństwa o wartości Niska (L – *Low*) zmniejszają wagę danej miary wpływu, a o wartości Średnia (M – *Medium*) są neutralne. Te zasady tyczą się oczywiście również integralności i dostępności.

Warto w tym miejscu dodać, że odpowiednie metryki wymogi bezpieczeństwa nie będą mieć znaczenia dla danych miar wpływu, gdy te ostatnie będą zdefiniowane jako Brak (N – *None*). Dodatkowo zmiana którychkolwiek wymogów z wartości „Średniej” na „Wysoką” nie będzie mieć znaczenia, gdy dana metryka wpływu osiągnie poziom „Całkowity” (C – *Complete*) – uogólniona i całkowita metryka wpływu (*Impact* – patrz kolejne paragrafy) będzie mieć już i tak wartość maksymalną.

Możliwe wartości metryki wymogi bezpieczeństwa (zarówno dla poufności, integralności oraz dostępności) przedstawiono w tabeli 5.12. Im wyższe są wymogi odnośnie każdego z trzech elementów, tym wyższa jest punktacja wrażliwości, a poziom „Średni” rozumiany jest tu jako domyślny.

Tab. 5.12. Wartości punktacji metryki wymogi bezpieczeństwa (CR, IR, AR).

Wartość metryki	**Wartość liczbowa**	**Opis**
Niska (L - *Low*)	0,5	Utrata poufności/integralności/dostępności będzie skutkować tylko ograniczonymi następstwami zarówno dla użytkowników, właścicieli jak i wszystkich powiązanych z danym systemem.
Średnia (M – *Medium*)	1,0	Utrata poufności/integralności/dostępności będzie skutkować poważnymi następstwami zarówno dla użytkowników, właścicieli jak i wszystkich powiązanych z danym systemem.
Wysoka (H – *High*)	1,51	Utrata poufności/integralności/dostępności będzie skutkować katastrofalnymi następstwami zarówno dla użytkowników, właścicieli jak i wszystkich powiązanych z danym systemem.
Niezdefiniowana (ND – *Not Defined*)	1,0	Przypisanie tej wartości do metryki nie zmieni końcowej punktacji – oznacza pominięcie niniejszej metryki.

5.2.3. Wektory podstawowe, czasowe oraz środowiskowe standardu CVSS

Wektory standardu CVSS buduje się w celu lepszej wizualizacji końcowej oceny danego rozwiązania. Każda grupa metryk to jeden wektor, więc docelowo otrzymuje się trzy takie elementy.

Konkretny wektor składa się ze skrótu nazwy danej metryki (patrz poprzedni paragraf), po której występuje dwukropek, a następnie skrótowa wartość metryki. W danym wektorze kolejne metryki oddziela się znakiem „/" (*slash*). Jeśli jakaś metryka czasowa bądź środowiskowa nie jest wykorzystywana, to przypisuje się jej wartość ND (*Not Defined*). Możliwe wartości wektorów podstawowych, czasowych i środowiskowych przedstawiono w tabeli 5.13.

Tab. 5.13. Wektory grup metrycznych standardu CVSS.

Grupa metryczna	Wektor
Podstawowa	AV:[L,A,N]/AC:[H,M,L]/Au:[M,S,N]/C:[N,P,C]/I:[N,P,C]/A:[N,P,C]
Czasowa	E:[U,POC,F,H,ND]/RL:[OF,TF,W,U,ND]/RC:[UC,UR,C,ND]
Środowiskowa	CDP:[N,L,LM,MH,H,ND]/TD:[N,L,M,H,ND]/CR:[L,M,H,ND]/ IR:[L,M,H,ND]/AR:[L,M,H,ND]

Można tu rozpatrzyć prosty przykład. Konkretna wrażliwość jakiegoś systemu uzyskała następujące wartości metryk podstawowych:

- Wektor dostępu (*Access Vector*): Niska (*Low*);
- Złożoność dostępu (*Access Complexity*): Średnia (*Medium*);
- Uwierzytelnianie (*Authentication*): Brak (*None*);
- Wpływ na poufność (*Confidentiality Impact*): Brak (*None*);
- Wpływ na integralność (*Integrity Impact*): Częściowa (*Partial*);
- Wpływ na poufność (*Availability Impact*): Całkowita (*Complete*).

Dla powyższego przykładu wektor podstawowy będzie mieć zatem postać: "AV:L/AC:M/Au:N/C:N/I:P/A:C".

5.2.4. Podstawowe zasady oceniania w standardzie CVSS

Poniżej przedstawione zostaną najważniejsze zasady oceniania systemów z wykorzystaniem standardu CVSS.

1. Ocena danej wrażliwości systemu nie powinna brać pod uwagę żadnych interakcji pomiędzy poszczególnymi podatnościami. Każda słabość systemu powinna być ewaluowana niezależnie.

2. W trakcie oceniania należy rozważyć bezpośredni wpływ na ewentualny cel ataku, nawet jeśli pośredni wpływ na przykład na użytkowników systemu jest większy.
3. Wiele systemów może być wykorzystywanych na różnym poziomie autoryzacji. Podczas oceny użyć należy poziomu najczęściej wykorzystywanego lub domyślnego, gdy ten popularny nie jest znany.
4. W trakcie oceny wpływu podatności, która może zostać wykorzystana na wiele sposobów, wybrać należy metodę najskuteczniejszą, a nie najczęściej stosowaną.
5. Jeśli wrażliwość może być wykorzystana zarówno lokalnie jak i zdalnie, to wybrać należy metodę zdalną (możliwie najbardziej zdalną – Przyległą sieciową lub Sieciową).
6. Wiele systemów posiada słabości lokalne, które jednak mogą być wykorzystane zdalnie. W takim przypadku metryka Wektor dostępu powinna przyjąć wartość: Sieciowa lub Przyległa sieciowa.
7. Jeśli wrażliwość istnieje w samym schemacie uwierzytelniania lub w systemie, w którym uwierzytelnianie nie jest realizowane, to metryka Au powinna być ustalona jako Brak, ponieważ aby wykorzystać podatność takiego rozwiązania nie trzeba uwiarygodniać tożsamości.
8. Jeśli dana podatność daje najwyższy poziom dostępu do systemu, to wszystkie podstawowe metryki wpływu powinny być zdefiniowane jako Całkowite. W przypadku dostępu na poziomie konkretnego użytkownika metryki te powinny być ustalone jako Częściowe.
9. Wrażliwości z częściową lub całkowitą utratą integralności mogą mieć również wpływ na dostępność. Przykład: jeśli można coś modyfikować, to prawdopodobnie można to również usuwać i w ten sposób uniemożliwić dostęp do tego elementu użytkownikom systemu.

5.2.5. Wyznaczanie punktacji końcowej

W niniejszym paragrafie przedstawione zostaną równania, pozwalające wyznaczyć końcowe wartości punktacji we wszystkich trzech grupach metrycznych: podstawowej, czasowej oraz środowiskowej.

5.2.5.1. Punktacja dla metryk podstawowych

Poniżej przedstawione zostaną równania, pozwalające wyznaczyć ocenę dla podstawowej grupy metrycznej. W celu wyznaczenia punktacji podstawowej (*BaseScore*) należy posłużyć się równaniem 5.1:

$$BaseScore = round\Big(\big((0.6 \cdot Impact) + (0.4 \cdot Exploitability) - 1.5\big) \cdot f(Impact)\Big), \quad (5.1)$$

gdzie *round()* oznacza zaokrąglenie do jednego miejsca po przecinku.

W równaniu 5.1 *Impact* definiuje się następująco:

$$Impact = 10.41 \cdot \big(1 - (1 - C) \cdot (1 - I) \cdot (1 - A)\big), \quad (5.2)$$

gdzie *C*, *I* oraz *A*, to skróty nazw odpowiednich metryk podstawowych (patrz paragraf 5.1.2.1).

Exploitability z równania 5.1 wyznacza się w oparciu o zależność:

$$Exploitability = 20 \cdot AV \cdot AC \cdot Au, \quad (5.3)$$

gdzie *AV*, *AC* oraz *Au* to również skróty nazw odpowiednich metryk podstawowych.

Parametr *f(Impact)* w równaniu 5.1 przyjmuje wartość równą zero, gdy *Impact* jest również zerowy, a w przeciwnym wypadku *f(Impact)* = 1,176.

Ostatecznie punktacja podstawowa może mieć wartość z przedziału od 0 do 10.

5.2.5.2. Punktacja dla metryk czasowych

Jeśli wykorzystuje się metryki czasowe, to do wyznaczenia oceny konieczna jest wartość punktacji podstawowej (*BaseScore*), która jest modyfikowana w celu uzyskania punktacji czasowej (*TemporalScore*):

$$TemporalScore = round(BaseScore \cdot E \cdot RL \cdot RC), \quad (5.4)$$

gdzie *round()* ma takie samo znaczenie jak w przypadku równania 5.1, a parametry *E*, *RL* oraz *RC* to skróty nazw odpowiednich metryk czasowych (patrz paragraf 5.1.2.2).

Warto w tym miejscu dodać, że punktacja czasowa mieści się w przedziale od 0 do 10 i nie jest nigdy większa od punktacji podstawowej, a mniejsza może być od niej maksymalnie o 33%.

5.2.5.3. Punktacja dla metryk środowiskowych

Jeśli metryki środowiskowe są wykorzystywane, to do wyznaczenia punktacji wykorzystuje się punktację czasową (*TemporalScore*), która jest w odpowiedni sposób modyfikowana w celu uzyskania oceny środowiskowej (*EnvironmentalScore*):

$$EnvironmentalScore = round\big((AdjustedTemporal + (10 - AdjustedTemporal) \cdot CDP) \cdot TD\big), \quad (5.5)$$

gdzie *CDP* oraz *TD* to odpowiednie skróty nazw metryk środowiskowych (patrz paragraf 5.1.2.3), natomiast skorygowana punktacja czasowa (*AdjustedTemporal*), wyznaczana jest na podstawie wzoru 5.4, w którym punktacja podstawowa (*BaseScore*), uzyskana jest zgodnie z zależnością 5.1, w której to z kolei parametr *Impact* zastępuje się jego skorygowaną wersją (*AdjustedImpact*):

$$AdjustedImpact = min\left(10; 10.41 \cdot \left(1 - (1 - C \cdot CR) \cdot (1 - I \cdot IR) \cdot (1 - A \cdot AR)\right)\right). \quad (5.6)$$

We wzorze 5.6 *C*, *I* oraz *A* to odpowiednie metryki podstawowe, a *CR*, *IR* oraz *AR* to odpowiednie skróty nazw metryk środowiskowych.

Warto tu tylko nadmienić, że punktacja środowiskowa może zmieniać się w przedziale od 0 do 10, ale wartość ta nigdy nie jest większa od punktacji czasowej.

Podsumowując, w paragrafie 5.1 niniejszej pracy przedstawiono pewien standard oceny bezpieczeństwa ogólnie rozumianych systemów IT.

W kolejnej części tego rozdziału zaprezentowana zostanie już autorska implementacja tego rozwiązania, mająca na celu kompleksową ocenę zaproponowanego kryptosystemu w oparciu właśnie o standard CVSS.

5.3. Nowy kryptosystem dla łącza HF w świetle standardu CVSS

W niniejszym paragrafie przedstawiona zostanie ocena modelu systemu bezpiecznej transmisji danych w łączu krótkofalowym z zastosowaniem standardu CVSS. Zadanie to zrealizowane zostanie na podstawie analiz przeprowadzonych w scenariuszach symulacyjnych z rozdziału poprzedniego. Wykorzystane zostaną tu najciekawsze i najważniejsze sytuacje, które mogą mieć bezpośredni wpływ na oferowany w kryptosystemie poziom bezpieczeństwa.

Analizę tu przeprowadzoną zrealizowano przy założeniu istnienia w systemie jednej jego realizacji. Fakt bowiem większej ich ilości sztucznie skomplikowałby zagadnienie i zaciemnił wyniki oraz płynące z nich wnioski. Dla potrzeb tego paragrafu warto rozumieć poszczególne realizacje kryptosystemu jako zupełnie niezależne elementy i stosować wyniki tu zaprezentowane oddzielnie dla każdego z nich.

W pierwszej kolejności analizie poddana zostanie sytuacja, w której to ryzyko utraty poufności, integralności oraz wiarygodności przesyłanych wiadomości jest stosunkowo niewielkie. Mowa tu o przypadku, w którym to ujawnieniu uległ klucz podstawowy danej realizacji systemu oraz obie aktualne tablice skramblujące

(A oraz B). Podatnością systemu, która może naruszać bezpieczeństwo transmisji, jest zatem właśnie wyciek wspomnianych parametrów. Sytuacja ta przedstawiona została w tabeli 5.14.

Tab. 5.14. Ocena wpływu ujawnienia klucza F i tablic skramblujących kryptosystemu *0x86* na oferowany poziom bezpieczeństwa transmisji.

Metryki podstawowe	**Wartość metryki**	**Metryki czasowe**	**Wartość metryki**	**Metryki środowiskowe**	**Wartość metryki**
AV	Lokalna	E	Funkcjonalna	CDP	Niska
AC	Średnia	RL	Oficjalna poprawka	TD	Niska
Au	Brak	RC	Potwierdzona	CR	Wysoka
C	Częściowa	**Punktacja**	**3,6**	IR	Wysoka
I	Częściowa			AR	Średnia
A	Częściowa			**Punktacja**	**1,3**
Punktacja	**4,4**				

Dla pełniejszej zgodności ze standardem oceny poniżej zaprezentowano również wektory CVSS dla trzech zdefiniowanych w poprzednim paragrafie metryk.

Wektor dla metryki podstawowej: AV:L/AC:M/Au:N/C:P/I:P/A:P.

Wektor dla metryki czasowej: E:F/RL:OF/RC:C.

Wektor dla metryki środowiskowej: CDP:L/TD:L/CR:H/IR:H/AR:M.

Jeśli chodzi o metrykę podstawową, to uzyskana tu ocena 4,4 (w skali do 10) wydaje się stosunkowo wysoka, ale należy tu jednak pamiętać, że kryptosystem po ujawnieniu wspomnianych danych jest narażony w pewnym stopniu zarówno na częściową utratę poufności jak i integralności. Stąd właśnie wynikają wartości metryk odpowiednio C oraz I. Należy jednak zauważyć, że sam fakt nieautoryzowanej możliwości transmisji w systemie naraża również jego dostępność. Nie chodzi tu nawet o „zalewanie" odbiorców dużą ilością danych, ale o sam fakt korzystania z zasobów systemu. Z tego też powodu metryka wpływu na dostępność (A) ma również wartość Częściowa.

Zakłada się dodatkowo w tym miejscu, że dostęp do zaproponowanego systemu (we wszystkich omawianych w tym paragrafie sytuacjach) jest lokalny i w taki sposób zdefiniowano metrykę AV. Podejście takie uzasadnić można faktem, że domyślnie nie zakłada się tu podłączania zaproponowanego rozwiązania do żadnej zewnętrznej czy nawet lokalnej sieci. Za każdym razem, gdy występuje potrzeba skorzystania z systemu, należy posiadać zatem fizyczny dostęp do modemu krótkofalowego, wyposażonego w moduł kryptograficzny (lub informacje z niego ujawnione).

Jeśli powyższe założenie nie będzie spełnione, to analizę można oczywiście przeprowadzić dla takiego przypadku. Autor chciał jednak poddać ocenie bezpieczeństwo transmisji w samym tylko systemie łączności krótkofalowej z pominięciem możliwości pośredniczenia jakiejś dodatkowej sieci.

Metryce złożoność dostępu w omawianym przypadku przypisano wartość Średnia, ponieważ z wykorzystaniem samego tylko klucza F oraz tablic skramblujących wiąże się ograniczenie atakującego do pewnych tylko schematów uwierzytelniania oraz transmisji. Szczegóły dotyczące zakresu dostępu do systemu w takim przypadku przedstawiono w rozdziale poprzednim.

Metryka Au w prezentowanej sytuacji ma wartość Brak, ponieważ podatność systemu, wynikająca z ujawnienia klucza F oraz tablic skramblujących, wiąże się również z naruszeniem mechanizmu uwierzytelniania. Możliwe jest bowiem podszywanie się przez atakującego pod innych użytkowników w przypadku transmisji pomiędzy terminalami, gdy do uwierzytelniania i szyfrowania wykorzystuje się klucz podstawowy sesji. Oznacza to tak naprawdę, że dla pewnych schematów transmisji atakujący nie będzie przesyłać wiarygodnych danych, a zostanie poprawnie uwierzytelniony przez odbiorcę.

Jeśli chodzi natomiast o metryki czasowe to dla omawianego przypadku występuje oczywiście funkcjonalna metoda wykorzystania zaistniałej wrażliwości systemu. Sytuacja taka ma miejsce, ponieważ dla znanych tablic skramblujących oraz klucza F można realizować pewne ograniczone schematy transmisji w zaproponowanym systemie. Dodatkowo zostały one przedstawione i zdefiniowane w niniejszej pracy, która jest jednocześnie potwierdzeniem ich istnienia. Z tego powodu metryka „wykorzystywalność” (E) przyjmuje tu wartość Funkcjonalna, natomiast pewność raportowania (RC) jest Potwierdzona przez autora systemu.

Ważnym jest tu jednak aspekt, iż w kryptosystemie przewidziano mechanizmy zaradcze, mające na celu przeciwdziałanie takiej sytuacji. Stąd też metryka stopień naprawialności (RL) przyjmuje tu wartość Oficjalna poprawka.

Podsumowując ocenę wyznaczoną na podstawie metryk czasowych (3,6), można stwierdzić, iż wynik uzyskany z użyciem metryki podstawowych został tu zredukowany ze względu na istniejącą oficjalną poprawkę, polegającą na zmianie przez centrum bezpieczeństwa aktualnych tablic skramblujących. Należy jednak pamiętać, że jeśli środki zaradcze nie zostaną wprowadzone to ryzyko

niebezpieczeństwa będzie ciągłe, ponieważ klucz F jest ważny zawsze, a automatyczna dezaktywacja nie ma tu zastosowania, bo atakujący nie posługuje się modułem kryptograficznym, a jedynie ujawnionym kluczem i tablicami skramblującymi. To samo tyczy się upłynięcia daty autoryzacji w grupie, ponieważ atakujący może używać zdobytych informacje, gdy tylko tak postanowi.

Metryki środowiskowe, odzwierciedlające najbardziej praktyczne zagrożenia, wynikające z danej wrażliwości systemu, pokazują jednak, iż w rzeczywistości prezentowana tu sytuacja niesie niewielkie tylko niebezpieczeństwo (ocena 1,3).

Taki stan rzeczy spowodowany jest faktem, że zarówno metryka potencjalna szkoda dodatkowa (CDP) jak i rozkład celu (TD) przyjmują tu wartość Niska. Jest to efekt wycieku klucza F oraz tablic skramblujących, który daje dostęp jedynie do najmniej bezpiecznych schematów transmisji w zaproponowanym kryptosystemie. Nie można bowiem wykorzystać tu ani kluczy sesji dla bezpiecznego szyfrowania, ani klucza grupy dla pełnej realizacji procesu kontroli integralności i uwierzytelniania. W ten sposób szkody wywołane przez atakującego będą stosunkowo nieduże, a zakres jego działania będzie ograniczony do pewnej tylko grupy mało bezpiecznych schematów transmisji. Pozwala to założyć, że niewielka tylko część użytkowników systemu będzie chciała z nich korzystać, jeśli do dyspozycji będą bezpieczniejsze rozwiązania.

Pozostałe metryki środowiskowe będą miały stałe wartości w tym jak i w każdym następnym analizowanym przypadku, ponieważ reprezentują one wymogi danego systemu w stosunku do mechanizmu poufności, integralności oraz dostępności. W omawianym kryptosystemie dostępność jest oczywiście bardzo istotna, ale z perspektywy jego zastosowań to poufność i integralność są elementami ważniejszymi. Z tego właśnie powodu metryki CR oraz IR przyjmować będą zawsze wartość Wysoka, natomiast metryka AR wartość Średnia.

Podsumowując niniejszy przypadek, można stwierdzić, że omawiana tu sytuacja niesie ze sobą pewne teoretyczne ryzyko naruszenia bezpieczeństwa (metryki podstawowe – ocena 4,4), z perspektywy jednak czasu maleje ono (metryki czasowe – ocena 3,6), ponieważ w systemie przewidziano tego typu sytuacje i istnieją wbudowane w niego mechanizmy zaradcze. Dodatkowo praktyczne niebezpieczeństwo w odniesieniu do środowiska użytkowników jest znikome (metryki

środowiskowe – ocena 1,3), ponieważ dotyczy ono tylko jego części i tylko w ograniczonym zakresie.

W tym miejscu należy jeszcze tylko podkreślić jeden aspekt, związany z możliwością błędnej interpretacji, zaprezentowanych tu wyników. Trzeba mieć bowiem na uwadze, że zamieszczone tu wyniki dotyczą ryzyka, wynikającego z sytuacji wycieku pewnych parametrów systemowych, gdy takie ujawnienie miało już miejsce. Nie można zatem rozumieć niniejszej oceny jako ewaluacji ryzyka samego faktu wystąpienia wycieku klucza F i tablic skramblujących, a jedynie jego wpływu na omawiany kryptosystem. Jest to bowiem jakby próba przewidzenia efektów potencjalnego zaistnienia takiego zagrożenia.

Kolejna sytuacja omawiana w niniejszym paragrafie jest podobna do poprzedniej, a różni się od niej jedynie faktem ujawnienia dodatkowo (poza kluczem podstawowym i tablicami) również aktualnego klucza sesji. Przypadek ten przedstawiono w tabeli 5.15.

Tab. 5.15. Ocena wpływu ujawnienia klucza F, tablic skramblujących oraz aktualnego klucza sesji kryptosystemu *0x86* na oferowany poziom bezpieczeństwa transmisji.

Metryki podstawowe	**Wartość metryki**	**Metryki czasowe**	**Wartość metryki**	**Metryki środowiskowe**	**Wartość metryki**
AV	Lokalna	E	Funkcjonalna	CDP	Średnio-Niska
AC	Średnia	RL	Oficjalna poprawka	TD	Niska
Au	Brak	RC	Potwierdzona	CR	Wysoka
C	Częściowa	**Punktacja**	**3,6**	IR	Wysoka
I	Częściowa			AR	Średnia
A	Częściowa			**Punktacja**	**1,5**
Punktacja	**4,4**				

Ponownie dla pełnej zgodności ze standardem CVSS poniżej przedstawiono wektory, odpowiadające konkretnym grupom metryk.

Wektor dla metryki podstawowej: AV:L/AC:M/Au:N/C:P/I:P/A:P.

Wektor dla metryki czasowej: E:F/RL:OF/RC:C.

Wektor dla metryki środowiskowej: CDP:LM/TD:L/CR:H/IR:H/AR:M.

W przypadku zaistnienia sytuacji wycieku wspomnianych elementów teoretycznie wzrasta zagrożenie naruszenia bezpieczeństwa (patrz rozdział poprzedni) w kryptosystemie *0x86*. Jeśli jednak spojrzeć na oceny, to zarówno dla przypadku grupy podstawowej jak i czasowej uzyskano te same wyniki, co więcej konkretne

metryki przyjmują identyczne wartości. Taka sytuacja ma miejsce, ponieważ w standardzie CVSS nie przewiduje się szerokiego zakresu stopniowania wartości w przypadku na przykład metryk wpływu (C, I oraz A). Można jedynie zdecydować, czy dana sytuacja ogranicza poufność, integralność i dostępność w systemie w sposób częściowy, całkowity bądź nie robi tego wcale. W omawianej sytuacji występuje oczywiście ponownie sytuacja częściowego wpływu na te elementy, ponieważ bez kluczy wszystkich grup, występujących w danej realizacji systemu nie ma możliwości całkowitej ingerencji w transmisję. Nawet jednak w takim przypadku wpływ nie byłby całkowity, ponieważ atakujący nie posiadałby kluczy głównych modułów oraz prywatnego klucza RSA krytposystemu *0x86*. Ingerencja zatem w transmisję wiadomości SMM typu punkt-punkt nie byłaby możliwa.

Podobna sytuacja tyczy się metryki złożoność dostępu (AC). Dostęp jest bowiem nadal w pewien sposób specjalizowany i dotyczy tylko pewnego możliwego zakresu działań atakującego i ograniczonego zbioru schematów transmisji.

Identyczne jak w poprzedniej sytuacji wartości metryk czasowych są jakby oczywiste i wynikają ponownie z faktu, iż w niniejszej pracy przedstawiono i potwierdzono funkcjonalny sposób wykorzystania takiej potencjalnej wrażliwości systemu. Jednocześnie jednak zaprezentowano oficjalny i wbudowany w system mechanizm (aktualizacja kluczy sesji i ewentualnie zmiana tablic skramblujących), pozwalający na przeciwdziałanie takiemu zdarzeniu.

W przypadku metryk środowiskowych jednak sytuacja uległa niewielkiej zmianie (ocena 1,5), a mianowicie dla prezentowanego przypadku potencjalna szkoda dodatkowa (CDP) jest nieco wyższa. Atakujący, posiadając bowiem aktualny klucz sesji, ma dostęp do większej liczby schematów transmisyjnych i w ten sposób zagrożenie dla systemu jest wyższe. Docelowo jednak rzeczywiste niebezpieczeństwo jest ciągle niewielkie, bo bez dostępu do jakiegokolwiek klucza grupy i powiązania go z jej konkretnym identyfikatorem, atakujący posiada jedynie ograniczone możliwości ingerencji w transmisje. Dodatkowo aktualny klucz sesji po pewnym czasie straci po prostu ważność, a uzyskanie nowego nie będzie możliwe (chyba że znowu zostanie ujawniony). Metryka rozkładu celu (TD) natomiast nie ulega zwiększeniu, ponieważ w systemie zakłada się głównie komunikację w obrębie grup z użyciem klucza grupy, co wyklucza możliwość ataku w tym zakresie.

Podsumowując można stwierdzić, że niebezpieczeństwo, wynikające z ewentualnego wystąpienia omawianej sytuacji, jest ograniczone czasowo, bo nawet bez reakcji CB aktualny klucz sesji utraci swą ważność po pewnym czasie. Ryzyko jednak może być mimo wszystko znaczne, ponieważ atakującego nie ograniczają kanały wirtualne, data końca aktywacji i autoryzacji. Ponadto nie ma pewności, że CB będzie posiadać informacje o wycieku parametrów systemowych.

Kolejna sytuacja jest jakby rozwinięciem obu poprzednich, ponieważ zakłada się tu że poza ujawnieniem klucza F, tablic skramblujących oraz aktualnego klucza sesji wyciekowi ulega również klucz konkretnej grupy (wraz z jej identyfikatorem) użytkowników. Sytuację tą przedstawiono w tabeli 5.16.

Tab. 5.16. Ocena wpływu ujawnienia klucza F, tablic skramblujących, aktualnego klucza sesji oraz pojedynczego i aktywnego klucza grupy kryptosystemu *0x86* na oferowany poziom bezpieczeństwa transmisji.

Metryki podstawowe	**Wartość metryki**	**Metryki czasowe**	**Wartość metryki**	**Metryki środowiskowe**	**Wartość metryki**
AV	Lokalna	E	Funkcjonalna	CDP	Średnio-Wysoka
AC	Średnia	RL	Oficjalna poprawka	TD	Średnia
Au	Brak	RC	Potwierdzona	CR	Wysoka
C	Częściowa	**Punktacja**	**3,6**	IR	Wysoka
I	Częściowa			AR	Średnia
A	Częściowa			**Punktacja**	**5**
Punktacja	**4,4**				

Poniżej przedstawiono również wektory CVSS dla omawianej wrażliwości.

Wektor dla metryki podstawowej: AV:L/AC:M/Au:N/C:P/I:P/A:P.

Wektor dla metryki czasowej: E:F/RL:OF/RC:C.

Wektor dla metryki środowiskowej: CDP:MH/TD:M/CR:H/IR:H/AR:M.

W przypadku tym po raz kolejny metryki podstawowe oraz czasowe uzyskują takie same oceny. Spowodowane jest to ponownie faktem ciągle częściowego naruszenia poufności, integralności oraz dostępności systemu. Należy bowiem pamiętać, że ujawniono klucz jednej tylko grupy użytkowników. Metryka złożoność dostępu przyjmuje również wartość Średnią, ponieważ atakujący jest ograniczony do jednej tylko grupy użytkowników danej realizacji kryptosystemu *0x86*, a więc warunki dostępu są ciągle specjalizowane.

Jeśli chodzi z kolei o metryki czasowe, to przypadek ten jest analogiczny do poprzednich i nie będzie w tym miejscu ponownie rozważany.

Sytuacja jest jednak inna jeśli chodzi o metryki środowiskowe (ocena 5), które oceniają praktyczne zagrożenie, wynikające z danej wrażliwości systemu, w środowisku użytkowników. Potencjalna szkoda dodatkowa jest już dużo większa (Średnio-Wysoka) niż poprzednio, ponieważ atakujący posiada możliwość transmisji i odbioru każdej wiadomości w konkretnej grupie użytkowników, a więc może korzystać w niej ze wszystkich przywilejów oferowanych przez system. Dodatkowo jednak może realizować podszywanie się pod dowolnego użytkownika grupy i korzystać z dowolnych kanałów wirtualnych, uwiarygodniając te działania kluczem grupy – bardzo niebezpieczna sytuacja. Wcześniej miał do dyspozycji tylko klucz sesji i podstawowy, a więc uwierzytelnianie realizowane było jedynie na poziomie całego systemu.

Atakujący zyskuje w tym momencie również możliwość deszyfrowania i dekodowania multicastowych wiadomości SMM, kierowanych do grupy, której klucz posiada. Ta sytuacja wiąże się zatem z dodatkowym niebezpieczeństwem, ponieważ atakujący ma dostęp do aktualizacji kolejnych kluczy sesji, wysyłanych przez CB.

Metryka rozkładu celu (TD) ma również wyższą wartość, ponieważ atakujący uzyskuje wspomniane udogodnienia, co pozwala mu na większy zakres nieautoryzowanych działań w systemie i dostęp do większej liczby schematów transmisji.

Podsumowując tą hipotetyczną sytuację, która mogłaby mieć miejsce w systemie, gdyby ujawniono wspomniane wcześniej parametry systemowe, można stwierdzić, że w przypadku braku reakcji ze strony CB konsekwencje mogą być bardzo niebezpieczne. Omawiany tu przypadek niesie ze sobą bowiem wysokie i rzeczywiste zagrożenie naruszenia bezpieczeństwa transmisji w systemie, które może być ciągłe ze względu na możliwość odbioru aktualnych kluczy sesji przez atakującego. Centrum bezpieczeństwa powinno zatem (o ile uzyska informacje o wystąpieniu takiej sytuacji) jak najszybciej zaktualizować ujawniony klucz grupy i sesji oraz najlepiej również, jeśli jest to możliwe, zmienić wykorzystywane tablice skramblujące.

Kolejna sytuacja zagrożenia bezpieczeństwa transmisji jest już odmienna od poprzednio prezentowanych. Analizie poddany zostanie bowiem przypadek, w

którym to skradziony bądź zagubiony (CB musi zakładać w tym momencie, że dostał się on w niepowołane ręce) został moduł kryptograficzny kryptosystemu *0x86*. Taka sytuacja została oceniona w tabeli 5.17.

Tab. 5.17. Ocena wpływu kradzieży modułu kryptograficznego kryptosystemu *0x86* na oferowany poziom bezpieczeństwa transmisji.

Metryki podstawowe	**Wartość metryki**	**Metryki czasowe**	**Wartość metryki**	**Metryki środowiskowe**	**Wartość metryki**
AV	Lokalna	E	Funkcjonalna	CDP	Średnio-Wysoka
AC	Średnia	RL	Oficjalna poprawka	TD	Średnia
Au	Wielokrotna	RC	Potwierdzona	CR	Wysoka
C	Częściowa	**Punktacja**	**3,1**	IR	Wysoka
I	Częściowa			AR	Średnia
A	Częściowa			**Punktacja**	**4,8**
Punktacja	**3,8**				

Ponownie poniżej przedstawiono odpowiednie wektory CVSS.

Wektor dla metryki podstawowej: AV:L/AC:M/Au:M/C:P/I:P/A:P.

Wektor dla metryki czasowej: E:F/RL:OF/RC:C.

Wektor dla metryki środowiskowej: CDP:MH/TD:M/CR:H/IR:H/AR:M.

Jeśli chodzi o metryki podstawowe, to główna różnica, w stosunku do poprzednio omawianych sytuacji, widoczna jest w przypadku uwierzytelniania (Au). Wartość Wielokrotna oznacza bowiem, że atakujący chcąc wykorzystać skradziony moduł musi w celu przeprowadzenia każdej transmisji w systemie uwiarygodnić się w nim z wykorzystaniem identyfikatorów w nim zawartych. Ne ma bowiem dostępu do kluczy i innych parametrów (choćby tablic), a może jedynie stosować moduł w sposób nieautoryzowany, ale zgodny z jego przeznaczeniem w systemie. Każde zatem nadanie wiadomości wiąże się z jej uwiarygodnieniem poprzez dane zawarte w skradzionym module. Jest to zatem uwierzytelnianie wielokrotne i stąd bierze się wybrana wartość wspomnianej metryki.

Metryka AC ma tak jak w poprzednich sytuacjach wartość Średnia, ponieważ sam moduł ogranicza w pewien sposób zakres dostępu atakującego do systemu, a więc dostęp jest specjalizowany. Ponadto za wybraną wartością metryki złożoność dostępu przemawia fakt, że CB nie zawsze będzie posiadać informacje o kradzieży modułu.

Metrykom C, I oraz A przypisano wartość Częściowa, ponieważ pojedynczy moduł nie pozwala na dostęp do wszystkich informacji przesyłanych w systemie ani na tworzenie dowolnych wiadomości. Można sobie bowiem wyobrazić transmisję w grupach, do których skradziony moduł nie ma dostępu. Dodatkowo żaden moduł nie pozwala na tworzenie wiadomości SMM a tym bardziej na ich uwiarygodnienie (potrzeba prywatnego klucza RSA systemu *0x86*). Sam moduł nie umożliwi również całkowitego ograniczenia dostępu do systemu dla innych użytkowników.

Ze względu zatem na konieczność wielokrotnego uwierzytelniania atakującego w systemie wartość punktacji metryk podstawowych jest niższa niż w poprzednich przypadkach i wynosi 3,8.

Metryki czasowe uzyskały te same wartości co w poprzednich sytuacjach, ponieważ skradziony moduł sam w sobie jest funkcjonalnym sposobem wykorzystania podatności, a sposób ten potwierdza niniejsza praca. W przypadku wystąpienia omawianej sytuacji w rzeczywistości, przewidziano jednak mechanizm zaradczy (Oficjalna poprawka), który przedstawiono w rozdziale poprzednim niniejszej pracy.

Wartość punktacji metryk czasowych jest jednak niższa niż dla pierwszych trzech sytuacji i wynosi 3,1. Jest to spowodowana faktem, że do wyznaczenia tej wartości wykorzystuje się punktację, uzyskaną na podstawie metryk podstawowych, a ta jest niższa niż w poprzednich przypadkach. Okazuje się ponownie, że zagrożenie naruszenia bezpieczeństwa transmisji z perspektywy czasu jest niższe niż wskazywałaby na to grupa metryk podstawowych. Jest to oczywiście spowodowane możliwością reagowania CB na zaistniałą sytuację.

Metryki środowiskowe uzyskały te same wartości jak w poprzednim przypadku. Aktywny moduł kryptograficzny, który może mieć na przykład dostęp do wielu grup użytkowników i posiadać w nich wysokie poziomy autoryzacji, może w dużym stopniu zaszkodzić użytkownikom i dostawcom danej realizacji systemu. Z tego też powodu metryka CDP uzyskała wartość Średnio-wysoka. Potencjalne uprawnienia w wielu grupach użytkowników nie oznaczają jednak z reguły dostępu do całej czy choćby znacznej większości realizacji systemu, dlatego też metryka TD ma wartość Średnia.

Końcowa zatem ocena (4,8) wpływu omawianej wrażliwości systemu na środowisko użytkowników jest nieco niższa niż poprzednio. Podsumowując można stwierdzić, że różnica ta wynika z faktu, że dostęp do parametrów systemowych

danego modułu (nawet nie wszystkich) może przynieść więcej korzyści atakującemu niż sam moduł zawierający wszystkie te parametry. Atakujący bowiem, posiadając skradziony moduł, nie może podszywać się pod inne terminale. Obowiązują go ponadto ograniczenia, wynikające z daty końca aktywacji i autoryzacji (jak również jej poziomu – kanały wirtualne) w grupie oraz fakt, że jest podatny na zarządzanie ze strony CB. Oczywiście należy tu wspomnieć, że atakujący może skorzystać z techniki filtrowania wiadomości zarządzających, ale ten aspekt przedstawiony zostanie w jednej z kolejnych sytuacji omawianych w tym paragrafie.

Dodatkowo warto tu nadmienić, że ważność kluczy sesji również z czasem upłynie, a CB na pewno nie wyśle aktualizacji kluczy dla skradzionego terminala (o ile oczywiście wie, że jest on skradziony). W przypadku poprzedniej sytuacji atakujący, znając klucz grupy (zakładając, że CB go nie zmieniło), mógł wydobyć aktualne klucze sesji z multicastowych wiadomości SMM kierowanych na grupę, której klucz posiadał. W sytuacji skradzionego modułu z kolei CB umieści zapewne w wiadomościach aktualizujących klucze sesji odpowiednie instrukcje, dezaktywujące skradziony moduł. Odebranie zatem aktualizacji przez taki moduł nie będzie możliwe.

W kolejnej sytuacji przedstawiono wpływ ataku opartego o powtarzanie wiadomości przesyłanych w systemie na całościowy poziom bezpieczeństwa realizowanych w nim transmisji. Przypadek ten zaprezentowano w tabeli 5.18.

Tab. 5.18. Ocena wpływu ataku powtarzania wiadomości na oferowany poziom bezpieczeństwa transmisji.

Metryki podstawowe	**Wartość metryki**	**Metryki czasowe**	**Wartość metryki**	**Metryki środowiskowe**	**Wartość metryki**
AV	Lokalna	E	Dowód istnienia	CDP	Brak
AC	Wysoka	RL	Niedostępna	TD	Wysoka
Au	Brak	RC	Potwierdzona	CR	Wysoka
C	Brak	**Punktacja**	**2,3**	IR	Wysoka
I	Częściowa			AR	Średnia
A	Częściowa			**Punktacja**	**3,1**
Punktacja	**2,6**				

Poniżej przedstawiono odpowiednie wektory CVSS.

Wektor dla metryki podstawowej: AV:L/AC:H/Au:N/C:N/I:P/A:P.

Wektor dla metryki czasowej: E:POC/RL:U/RC:C.

Wektor dla metryki środowiskowej: CDP:N/TD:H/CR:H/IR:H/AR:M.

Sytuacja ta jest odmienna od poprzednich, ponieważ nie zakłada się tu, że atakujący posiada jakiekolwiek klucze, tablice skramblujące czy moduły kryptograficzne. Jedyną jego wiedzą jest znajomość architektury systemu i protokołów komunikacyjnych w nim wykorzystywanych. Ponadto musi on zebrać dodatkowe informacje, obserwując zachowania terminali w czasie transmisji. Będą mu one potrzebne w późniejszych próbach ataku. Jego celem jest bowiem zebranie wiadomości przesyłanych w systemie i wywołujących konkretne zachowania ich odbiorców. Późniejsze ponowne ich przesłanie może być przynajmniej teoretycznie użyteczne dla atakującego, który za ich pośrednictwem będzie chciał wywołać pewne działania odbiorców korzystne z jego perspektywy w danej chwili. Szczegóły tego procesu omówiono w rozdziale poprzednim.

Jeśli chodzi o metryki podstawowe, to złożoność dostępu jest tu bardzo wysoka, ponieważ istnieje bardzo wąskie okno (mniej niż trzy minuty – ze względu na wykorzystywany znacznik czasu w liczniku algorytmu AES-CTR), w którym to zachowana wiadomość może być ponownie przesłana. W praktyce zatem bardzo rzadko działanie takie okaże się użyteczne. Dodatkowo ponowne wysyłanie tych samych wiadomości będzie podejrzane i stosunkowo łatwe do wykrycia. Metryce AC przypisano zatem wartość Wysoka.

Uwiarygodnienie w systemie nie jest wymagane, ponieważ atakujący przesyła tylko ponownie wiadomości różnych użytkowników. Mechanizm uwierzytelniania jest tu zatem w pewien sposób naruszony i wykorzystywany w tym ataku. Metryka Au ma zatem wartość Brak.

Sytuacja tu prezentowana ma również pewien wpływ na integralność systemu. Teoretycznie bowiem nawet powtórzona wiadomość może dokonać pewnych zmian w module (czy modułach), choć często atakujący nie ma kontroli nad tym co zostanie zmienione. Mógłby on na przykład przesłać ponownie pewną wiadomość zarządzającą, która zmodyfikuje jakieś parametry modułu kryptograficznego.

Można sobie wyobrazić sytuacje, w której to CB wysyła pewne instrukcje kasujące, a już następna wiadomość zawiera nana aktualizujące dane modułu. Jest to oczywiście czysto hipotetyczna sytuacja, ponieważ CB nie musi realizować tego w ten sposób. Co więcej atakujący musiałby się najpierw dowiedzieć, że takie instrukcje zostały wysłane. Teoretycznie jednak mógłby on przechować wiadomość kasującą i przesłać ją ponownie zaraz po odebraniu przez moduł wiadomości aktualizującej.

Należy tu tylko pamiętać, że na zrealizowanie tych wszystkich działań musiałoby wystarczyć kilka minut, co jest mało realne, ponieważ trudno sobie wyobrazić, że atakujący w tak krótkim czasie dowie się jakie zadania miały wysłane wiadomości od CB oraz do kogo były adresowane. Co więcej trudno się spodziewać tak nierozsądnych działań ze strony centrum bezpieczeństwa. W praktyce bowiem kasowanie pewnie w ogóle nie byłoby realizowane skoro te same parametry miałyby być zaraz zaktualizowane, a więc wystarczyłaby jedna tylko wiadomość zarządzająca. A jeśli nawet CB skasowałoby najpierw pewne dane, to sytuacja modułu nie zmieniłaby się na tyle szybko (kilka minut), aby musiały być one znowu modyfikowane. Ze względu jednak na teoretyczną możliwość realizacji takiego ataku metryka I ma wartość Częściowa.

Ponowne przesyłanie wiadomości wiąże w sposób oczywisty z wykorzystywaniem zasobów systemu, dlatego też atak tu omawiany ma pewien wpływ na dostępność systemu. Wartość metryki A ma zatem wartość Częściowa.

Omawiany tu typ ataku nie ma wpływu na poufność przesyłanych wiadomości. Działanie to nie umożliwia bowiem deszyfracji jakichkolwiek wiadomości przesyłanych z wykorzystaniem kryptosystemu *0x86*. Z tego też powodu wartość metryki C to Brak.

Wartość oceny uzyskana na podstawie metryk podstawowych jest dużo niższa od wszystkich omawianych do tej pory przypadków i wynosi 2,6. Oznacza to, że system jest stosunkowo niewrażliwy na tego typu działania.

Metryki czasowe z kolei jeszcze obniżają tą punktację do poziomu 2,3. Jest to spowodowane faktem, że istnieje tylko teoretyczny dowód istnienia możliwości wykorzystania prezentowanego ataku. W praktyce jednak jego skuteczność jest wątpliwa ze względu na bardzo ograniczony czas jego ewentualnej realizacji. Metryka „wykorzystywalność” ma zatem wartość Dowód istnienia.

Wrażliwość ta jest jednak potwierdzona przez autora i dlatego metryka pewność raportowania ma wartość Potwierdzona. Dodatkowo nie ma dostępnego mechanizmu przeciwdziałania takiemu atakowi, który byłby wbudowany w specyfikację systemu, dlatego też metryka stopień naprawialności przyjmuje wartość Niedostępna.

W tym miejscu konieczne jest pewne wyjaśnienie. W zaproponowanym rozwiązaniu jest oczywiście wbudowany mechanizm chroniący przed atakami powtarzania wiadomości. Mowa tu o sposobie wyznaczania licznika w

zmodyfikowanym algorytmie AES-CTR, gdzie bierze się pod uwagę znacznik czasu. Ważny jest tu jednak fakt, że mechanizm ten chroni przed atakami, w których to wiadomość zostaje powtórzona po dłuższym czasie (przynajmniej po kilku minutach). W prezentowanym tu jednak przypadku analizie poddano sytuacje, w których teoretyczny atak następuje bardzo szybko po nadaniu oryginalnej wersji wiadomości, a przed takim działaniem system nie jest chroniony. W sposób oczywisty w wątpliwość poddać można praktyczną skuteczność takiego podejścia, ale ono właśnie podlega w tym przykładzie ocenie.

Jeśli chodzi o metryki środowiskowe, to autor stwierdza, że potencjalna szkoda jest znikoma lub żadna, ponieważ trudno sądzić że działania tu omawiane mogą w jakiś praktyczny sposób zaszkodzić transmisji z wykorzystaniem kryptosystemu *0x86*. Metryka CDP przyjmuje zatem wartość Brak.

Jeśli chodzi z kolei o metrykę rozkładu celu, to sytuacja jest taka, że teoretycznie każdy terminal w zasięgu atakującego jest narażony na próbę ataku. Z tego powodu metryka TD przyjmuje wartość Wysoka.

Końcowa ocena środowiskowa jest stosunkowo niska (chociaż wyższa niż podstawowa i czasowa) i wynosi 3,1. Można zatem stwierdzić, że stosowanie takiego typu ataku na system niesie ze sobą niewielkie tylko niebezpieczeństwo praktyczne. Wartość ostateczna nie jest mniejsza, choć wydaje się że jest to tylko czysto hipotetyczny przypadek, ze względu na to, że na działania tego typu narażone są właściwie wszystkie terminale i teoretycznie jest to niezależne od schematu transmisji przez nie wykorzystywanego.

Następna analiza ma na celu ocenę wpływu filtrowania wiadomości zarządzających na ogólny poziom bezpieczeństwa transmisji oferowany przez kryptosystem *0x86*. Sytuacja ta przedstawiona została w tabeli 5.19.

Prezentowana w tym miejscu sytuacja cechuje się wysoką złożonością dostępu, ponieważ atakujący musi posiadać już pewien poziom autoryzacji w systemie, aby jego działania były skuteczne. Filtrowanie wiadomości zarządzających może w pewnym stopniu uchronić go przed utratą, przysługujących mu wcześniej przywilejów. Dodatkowo jego działania mogą być stosunkowo łatwo wykryte, ponieważ pomimo prób na przykład jego dezaktywacji przez CB będzie on nadal korzystać z danej realizacji kryptosystemu *0x86*. Ze względu na powyższe fakty metryka AC przyjmuje tu wartość Wysoka.

Tab. 5.19. Ocena wpływu ataku filtrowania wiadomości zarządzających na oferowany poziom bezpieczeństwa transmisji.

Metryki podstawowe	**Wartość metryki**	**Metryki czasowe**	**Wartość metryki**	**Metryki środowiskowe**	**Wartość metryki**
AV	Lokalna	E	Funkcjonalna	CDP	Średnio-Niska
AC	Wysoka	RL	Oficjalna poprawka	TD	Średnia
Au	Wielokrotna	RC	Potwierdzona	CR	Wysoka
C	Częściowa	**Punktacja**	**2,8**	IR	Wysoka
I	Częściowa			AR	Średnia
A	Częściowa			**Punktacja**	**4,1**
Punktacja	**3,4**				

Odpowiednie wektory CVSS dla tego przypadku przedstawiono poniżej.

Wektor dla metryki podstawowej: AV:L/AC:H/Au:M/C:P/I:P/A:P.

Wektor dla metryki czasowej: E:F/RL:OF/RC:C.

Wektor dla metryki środowiskowej: CDP:LM/TD:M/CR:H/IR:H/AR:M.

Działania atakującego będą wymagały dodatkowo wielokrotnego uwierzytelniania się u odbiorców jego wiadomości, ponieważ korzystać on będzie z konkretnego modułu (podobnie jak w sytuacji kradzieży). Nie będzie on zatem mógł podszywać się pod inne moduły, a więc metryka Au przyjmuje tu wartość Wielokrotna.

Wszystkie działania atakującego będą miały częściowy wpływ na zarówno poufność, integralność jak i dostępność, ponieważ dany moduł jest zawsze ograniczony do pewnego tylko zakresu działania w systemie. Nigdy nie posiada on na przykład pełnego poziomu autoryzacji we wszystkich grupach w danej realizacji. Dodatkowo żaden moduł nie ma oczywiście możliwości deszyfrowania wiadomości SMM typu punkt-punkt przesyłanych do innych modułów kryptograficznych. Nie ma również możliwości uwiarygodniania wiadomości SMM – nie może ich nawet tworzyć. Warto w tym miejscu przypomnieć, że atakujący niezależnie właściwie od typu jego działań korzysta z pewnych zasobów systemu *0x86*. Na podstawie powyższych stwierdzeń metrykom C, I oraz A przypisano w tym przypadku wartość Częściowa.

Przy użyciu metryk podstawowych uzyskano ocenę ryzyka naruszenia bezpieczeństwa transmisji o wartości 3,4. Jest ona wyższa niż w przypadku ataku powtarzania wiadomości, ale również niższa w stosunku do wszystkich omawianych do tego momentu sytuacji ujawnienia informacji systemowych czy kradzieży modułu kryptograficznego. Atak filtrowania jest skuteczniejszy niż ten prezentowany poprzednio, ale mimo to ma on ciągle ograniczone możliwości zastosowania.

Metryki czasowe przyjmują w prezentowanej sytuacji wartości identyczne jak przy ujawnieniu parametrów systemowych. Istnieje bowiem funkcjonalny (choć ograniczony) sposób wykorzystania podatności systemu jakim jest filtrowanie wiadomości zarządzających. Dodatkowo niniejsza praca jest potwierdzeniem jego istnienia. Co jednak ważne w zaproponowanym kryptosystemie przewidziano pewne mechanizmy przeciwdziałania takiemu typowi ataku.

Wartość oceny uzyskana z wykorzystaniem metryk czasowych ma wartość 2,8 i jest niższa w stosunku do ewaluacji podstawowej. Powodem jest oczywiście fakt istnienia oficjalnej poprawki, a więc mechanizmu, który może zastosować CB, w celu minimalizacji wpływu omawianego typu ataku. Szczegóły tego rozwiązania znaleźć można w rozdziale poprzednim niniejszej pracy.

Jeśli chodzi o metryki środowiskowe, to potencjalna szkoda dodatkowa jest stosunkowo niewielka – metryka CDP ma tu wartość Średnio-Niska. Jest to spowodowane faktem czasowej tylko stosowalności ataku filtrowania. Dodatkowo ze względu na ograniczone możliwości atakującego, który nie ma dostępu do informacji systemowych zawartych w module, potencjalna szkoda jaką może on wyrządzić w systemie jest stosunkowo niewielka. Ponadto atak może być dość łatwo wykryty, co odróżnia ten przypadek od sytuacji zagubienia lub kradzieży modułu, o której CB może w ogóle nie wiedzieć.

Metryka rozkładu celu przyjmuje tu z kolei wartość Średnia, ponieważ w tym aspekcie sytuacja jest podobna jak w przypadku kradzieży modułu. Atakujący może bowiem przez pewien czas mieć dostęp do stosunkowo sporego zakresu schematów transmisji w systemie oraz do nawet kilku grup użytkowników w danej jego realizacji.

Końcowo ocena zagrożenia w środowisku użytkowników ma tu zatem wartość 4,1 i jest ona stosunkowo wysoka. Należy jednak pamiętać, iż jest ona ciągle mniejsza niż w przypadku kradzieży modułu. Dodatkowo atakujący stosując takie działania nie uzyskuje nigdy parametrów systemowych zawartych w module. Ponadto nawet w przypadku braku reakcji ze strony CB omawiana tu wrażliwość z czasem sama utraci na znaczeniu, a po automatycznej dezaktywacji modułu zostanie wyeliminowana.

Ostatnim omawianym w tym paragrafie przypadkiem jest czysto hipotetyczna sytuacja dostępu atakującego do danych zawartych w CB oraz bazie danych użytkowników systemu. Celem tej analizy jest głównie porównanie ryzyka naruszenia bezpieczeństwa transmisji w stosunku do wcześniej prezentowanych przypadków.

Zakłada się tu, że atakujący uzyskał już wspomniane dane, a więc nie podlega tu zatem ocenie samo ryzyko dostępu do nich, lecz niebezpieczeństwo dla systemu, wynikające z faktu ich posiadania przez osobę do tego nieupoważnioną. Analiza tej sytuacji przedstawiona została w tabeli 5.20.

Tab. 5.20. Ocena wpływu ujawnienia danych centrum bezpieczeństwa oraz bazy danych użytkowników na oferowany poziom bezpieczeństwa transmisji.

Metryki podstawowe	**Wartość metryki**	**Metryki czasowe**	**Wartość metryki**	**Metryki środowiskowe**	**Wartość metryki**
AV	Lokalna	E	Wysoka	CDP	Wysoka
AC	Niska	RL	Niedostępna	TD	Wysoka
Au	Brak	RC	Potwierdzona	CR	Wysoka
C	Całkowita	**Punktacja**	**7,2**	IR	Wysoka
I	Całkowita			AR	Średnia
A	Całkowita			**Punktacja**	**8,6**
Punktacja	**7,2**				

Podobnie jak w poprzednich przypadkach poniżej przedstawiono odpowiednie wektory CVSS dla konkretnych grup metrycznych.

Wektor dla metryki podstawowej: AV:L/AC:L/Au:N/C:C/I:C/A:C.

Wektor dla metryki czasowej: E:H/RL:U/RC:C.

Wektor dla metryki środowiskowej: CDP:H/TD:H/CR:H/IR:H/AR:M.

Prezentowana sytuacja cechuje się niską złożonością dostępu, ponieważ atakujący, posiadając wszystkie informacje systemowe, może w dowolnym momencie przesyłać wszelkiego rodzaju wiadomości systemowe. Często występują zatem warunki sprzyjające do przeprowadzenia nieautoryzowanych działań. Atakujący dodatkowo nie jest ograniczony do pewnych tylko schematów transmisji czy grup użytkowników. Nie obowiązują go żadne poziomy autoryzacji czy obostrzenia czasowe. Ze względu na powyższe fakty metryka AC przyjmuje wartość Niska.

Atakujący wykorzystuje również wrażliwość mechanizmu uwierzytelniania. Może on bowiem podszywać się pod dowolnych użytkowników systemu, a nawet pod samo centrum bezpieczeństwa. Z tego powodu metryka Au przyjmuje tu wartość Brak.

Sytuacja tu omawiana ma ponadto całkowity wpływ na poufność i integralność danych oraz parametrów systemowych, ponieważ atakujący ma dostęp do wszystkich schematów transmisji i danych użytkowników. Może zatem transmitować dowolne wiadomości jako dowolny użytkownik danej realizacji systemu lub nawet

jako centrum bezpieczeństwa (posiada prywatny klucz RSA kryptosystemu *0x86*). Dodatkowo atakujący może również całkowicie zablokować dostęp do systemu dla innych użytkowników poprzez choćby dezaktywację ich modułów kryptograficznych. Taki stan rzeczy powoduje, że metryki C, I oraz A przyjmują wartości Całkowita.

Wartość oceny uzyskana z wykorzystaniem metryk podstawowych jest zatem bardzo wysoka i wynosi 7,2. Oznacza to, że taka hipotetyczna sytuacja oznaczałaby praktycznie krytyczne naruszenie bezpieczeństwa transmisji w kryptosystemie *0x86*.

Metryki czasowe dodatkowo nie zmniejszają punktacji podstawowej, ponieważ „wykorzystywanlość" prezentowanej tu hipotetycznej podatności jest bardzo wysoka. Dodatkowo nie są dostępne żadne mechanizmy zaradcze, które umożliwiłyby choćby częściową poprawę takiej sytuacji. Co więcej w momencie wystąpienia wspomnianego wycieku informacji systemowych wrażliwość w ten sposób powstała jest oczywista i została potwierdzona przez autora kryptosystemu *0x86* w niniejszej pracy.

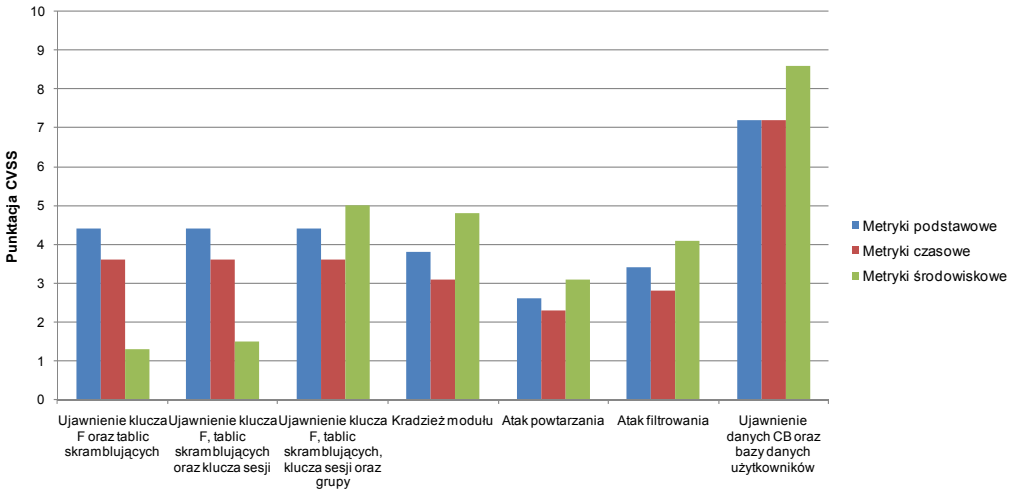
Metryki środowiskowe również potwierdzają wielkie niebezpieczeństwo, wynikające z ewentualnego zaistnienia omawianej sytuacji. Potencjalna szkoda dodatkowa jest bowiem ogromna ze względu na nieograniczone w tym momencie w żaden sposób działania atakującego. Wartość metryki CDP została ustalona zatem jako Wysoka. Ponadto ze względu na bardzo szeroki możliwy zakres ataku metryka TD uzyskuje również wartość Wysoka.

Końcowa punktacja środowiskowa, oceniająca praktyczne zagrożenie powstałe w wyniku zaistnienia prezentowanego zdarzenia, jest bardzo wysoka i wynosi 8,6. Wartość ta oznacza, że sytuacja taka miałaby krytyczne znaczenie dla systemu, a wrażliwość w ten sposób powstała spowodowałaby całkowite naruszenie bezpieczeństwa transmisji w środowisku użytkowników.

Należy w tym miejscu jednak pamiętać, iż ryzyko wystąpienia takiej sytuacji w praktyce jest mało prawdopodobne, a analiza ta ma pokazać jedynie jak ważna w prezentowanym (jak i w każdym innym) kryptosystemie jest kontrola dostępu do informacji. Czynnikiem krytycznym dla systemów tego typu jest bowiem najczęściej człowiek i nawet najlepiej zaprojektowany system bezpiecznej transmisji będzie bezużyteczny w momencie, gdy ten element zawiedzie.

Jako podsumowanie niniejszego paragrafu na rysunku 5.3. przedstawiono zestawienie wszystkich sytuacji tu prezentowanych i analizowanych.

Na podstawie rysunku 5.3 łatwo stwierdzić, iż praktycznym zagrożeniem dla systemu są głównie sytuacje, w których to ujawnieniu uległa znaczna liczba informacji systemowych. Jeśli jednak taki przypadek miałby miejsce, a centrum bezpieczeństwa uzyskałoby informacje o jego zaistnieniu, to dla większości sytuacji tego typu dostępne są mechanizmy zaradcze wbudowane w zaproponowany kryptosystem. Rozwiązania te pozwalają w dużym stopniu eliminować te zagrożenia i ponownie zapewnić pełne bezpieczeństwo transmisji.



Rys. 5.3. Ocena wpływu różnorodnych i niepożądanych zdarzeń na oferowany poziom bezpieczeństwa transmisji w kryptosystemie *0x86*.

Inną niebezpieczną sytuacją jest kradzież modułu kryptograficznego. W tym momencie CB jednak także posiada możliwości reagowania i minimalizowania wpływu takiego nieautoryzowanego dostępu do systemu. Może zatem również zapewnić w tej sytuacji ochronę poufności, integralności oraz wiarygodności przesyłanych z wykorzystaniem zaproponowanego kryptosystemu informacji.

Przeanalizowane z kolei próby ataku na kryptosystem z użyciem technik powtarzania i filtrowania wiadomości pokazują, że w większości przypadków metody te albo nie mają praktycznego zastosowania, bądź są stosunkowo łatwe do wykrycia. Dodatkowo wbudowane mechanizmy przeciwdziałania takim atakom będą w większości wypadków skuteczne i pozwolą na minimalizowanie ich wpływu.

Krytyczną jednak dla zachowania wysokiego poziomu bezpieczeństwa transmisji jest kontrola dostępu do informacji. Parametry systemowe muszą bowiem podlegać ścisłej ochronie przed nieautoryzowanym do nich dostępem. Czynnik ludzki jest tu zatem najistotniejszy i nawet najlepsze mechanizmy wiarygodności, poufności i kontroli integralności okazać się mogą bezużyteczne, gdy zawiedzie człowiek.

Podsumowanie

Celem niniejszej rozprawy doktorskiej było opracowanie nowego kryptosystemu dla transmisji danych w łączu krótkofalowym. Zadanie to zrealizowano poprzez dobór i modyfikację odpowiednich algorytmów kryptograficznych oraz mechanizmów zarządzania systemem jak i parametrami użytkowników. Założono w tym miejscu, że wybrane rozwiązania muszą być uznawane obecnie za w pełni bezpieczne, a przeprowadzone ich autorskie rozszerzenia nie mogą zmieniać tego stanu rzeczy. Między innymi z tego powodu, dla realizacji zaproponowanego w tej pracy kryptosystemu, zdecydowano się na cztery algorytmy: AES-128 (w trybie CTR), CMAC-128 (w oparciu o AES-128), SHA-256 oraz RSA-1024. Modyfikacjom poddano głównie rozwiązanie AES-CTR oraz CMAC. Dokonano ich jednak w ramach dopuszczonych przez dany standard bądź rozbudowano istniejący algorytm bez ingerencji w podstawową jego strukturę. Wybór powyższych rozwiązań oraz ich modyfikacji miał na celu głównie: minimalne ograniczenie pasma użytecznego, poprawę bezpieczeństwa, oferowanego przez kryptosystem, umożliwienie elastycznego i efektywnego nim zarządzania oraz przystosowanie go do specyficznego charakteru warstwy fizycznej istniejących modemów krótkofalowych.

Kolejnym etapem pracy był dobór pozostałych elementów kryptosystemu. Zaproponowano zatem zupełnie nowy protokół komunikacji, w którym zdefiniowano pewne typy i struktury wiadomości, przesyłane w systemie oraz pełną listę rozkazów w nim wykorzystywanych. Określono ponadto strukturę samego kryptosystemu i scharakteryzowano dwa podstawowe jego elementy: centrum bezpieczeństwa oraz moduł kryptograficzny. Wybrano również odpowiedni algorytm generacji kluczy szyfrujących, sposób ich hierarchizacji, dystrybucji, deszyfracji oraz wykorzystania w zaproponowanym rozwiązaniu. Na potrzeby kryptosystemu zdefiniowano ponadto parametry takie jak choćby: różnego typu identyfikatory, tablice skramblujace, maski uprawnień, znaczniki czasu, sekwencje skramblujące czy kanały wirtualne. Ostatecznie określono autorskie metody i algorytmy syntezy oraz analizy przesyłanych wiadomości oraz pewne ogólne zasady funkcjonowania kryptosystemu.

W pracy scharakteryzowano również sposób realizacji zaproponowanego rozwiązania w klasycznych modemach HF. Implementacja taka jest zgodna z odpowiednimi wymogami zawartymi w standardzie tych urządzeń i nie wymaga ona

ich modyfikacji, a jedynie wykorzystania wirtualnego modułu kryptograficznego. Element ten może zostać zrealizowany w postaci aplikacji uruchamianej na komputerze PC, podłączonym do modemu.

Na podstawie projektu kryptosystemu zaprezentowanego w niniejszej rozprawie doktorskiej, zrealizowano środowisko symulacyjne, pozwalające na analizę różnego typu rzeczywistych scenariuszy, mogących mieć miejsce w praktycznej implementacji zaproponowanego rozwiązania. Przedstawiono tu zatem podstawowe sytuacje zarządzania parametrami użytkowników. Dla przykładu zaprezentowano procedury aktywacji i dezaktywacji konkretnych realizacji systemu w modułach kryptograficznych. Omówiono również metody aktualizacji różnego typu kluczy czy innych elementów systemowych. W tym miejscu skupiono się jednak przede wszystkim na zbadaniu reakcji krytposystemu na wszelkiego rodzaju próby ingerencji i nieautoryzowanego do niego dostępu. Przeanalizowano między innymi ataki, polegające na modyfikowaniu treści przesyłanych wiadomości oraz opierające się o techniki powtarzania i filtrowania wiadomości. Zaprezentowano również mechanizmy zaradcze systemu w sytuacji kradzieży modułu kryptograficznego oraz wycieku różnego typu kluczy i parametrów systemowych, spowodowane naruszeniem mechanizmu dostępności, a więc często po prostu błędem człowieka lub błędnym albo niewystarczającym zabezpieczeniem przechowywanych informacji systemowych.

Po przeprowadzeniu powyższych badań można stwierdzić, że zaproponowane rozwiązanie jest rzeczywiście w bardzo dużym stopniu odporne na nieautoryzowane próby dostępu. W niektórych sytuacjach ryzyko naruszenia oferowanego poziomu bezpieczeństwa w ogóle nie występuje, a w innych system posiada pewne wbudowane rozwiązania pozwalające minimalizować a często nawet eliminować ewentualnie powstałe zagrożenia.

Ostatnim etapem realizacji niniejszej rozprawy doktorskie była próba obiektywnej oceny zaproponowanego kryptosystemu. Ewaluacja ta miała na celu określenie rzeczywistego wpływu ewentualnego wystąpienia pewnych słabości systemu na ogólnie rozumiany, oferowany przez niego poziom bezpieczeństwa transmisji. Analizie poddano tu sytuacje prób nieautoryzowanego dostępu, o których mowa była wcześniej. Do tego celu wykorzystano standard CVSS, który pozwala na określenie stopnia wpływu pewnych wrażliwości czy wad na ogólnie rozumiane systemy IT. Za

pomocą tego rozwiązania dokonano pośredniej oceny (zarówno z perspektywy czasu jak i środowiska użytkowników) ryzyka jakie niosą ze sobą niepożądane sytuacje, mogące wystąpić w warunkach rzeczywistych.

Głównie ze względu na bardzo ograniczony charakter ewentualnego nieautoryzowanego dostępu oraz z powodu istnienia w systemie wbudowanych mechanizmów zaradczych, uzyskane oceny odzwierciedlają wysoki ogólny poziom bezpieczeństwa zaproponowanego kryptosystemu. Jedynym właściwie źródłem ryzyka są tu sytuacje spowodowane naruszeniem mechanizmu dostępności, gdy wyciekowi uległa duża ilość informacji systemowych. Przypadkiem krytycznym i najniebezpieczniejszym jest ujawnienie prywatnego klucza RSA kryptosystemu oraz danych przechowywanych w bazie użytkowników. Taka sytuacja uznawana jest jednak za dużo mniej prawdopodobną niż na przykład nieautoryzowany dostęp do danych, zawartych w danym module kryptograficznym.

Końcowa ocena oferowanego poziomu bezpieczeństwa zaproponowanego kryptosystemu jest wysoka i wynika głównie z wzięcia pod uwagę faktu istnienia wielu zabezpieczeń przed potencjalnymi atakami jak i możliwości efektywnego oraz elastycznego nim zarządzania nawet w przypadkach ryzyka wystąpienia nieautoryzowanego dostępu.

Zaprezentowane w niniejszej rozprawie doktorskiej analizy i badania potwierdzają postawioną tezę, która dotyczyła wykazania, że w paśmie krótkofalowym można zapewnić bezpieczeństwo transmisji danych poprzez zastosowanie mechanizmów poufności i kontroli integralności informacji oraz uwierzytelniania i ograniczania dostępu do urządzeń i sieci. Ponadto, poprzez odpowiednie zarządzanie bezpieczeństwem, możliwe stało się zintegrowanie wszystkich tych funkcji, co umożliwiło opracowanie całościowej koncepcji systemu bezpieczeństwa dla łącza HF.

Wszystkie przeprowadzone w tej pracy analizy potwierdzają wysoki poziom bezpieczeństwa oferowany przez zaproponowane rozwiązanie, dużą elastyczność nim zarządzania jak i możliwość stosunkowo prostego wykorzystania go w istniejących modemach krótkofalowych.

Jedynym właściwie czynnikiem mogącym nieść ryzyko jest tak zwany czynnik ludzki i wynikające z tego faktu ewentualne naruszenie mechanizmu kontroli dostępności, powodujące ujawnienie tajnych parametrów systemowych. Nawet

jednak w takiej sytuacji dla większości przypadków wycieku informacji istnieją w systemie odpowiednie mechanizmy zaradcze, które umożliwiają ograniczenie wpływu tego typu zagrożeń.

Niniejsza rozprawa doktorska jest ważnym etapem na drodze do realizacji w przyszłości modelu centrum bezpieczeństwa oraz w pełni funkcjonalnego modułu kryptograficznego możliwego do wykorzystania w dzisiejszych modemach krótkofalowych, zrealizowanych w technologii radia programowalnego i nie tylko. Rozważania i analizy, przeprowadzone w tej pracy, potwierdzają bowiem, że rozwiązania tu zaprezentowane mogą być skutecznie wykorzystane dla realizacji bezpiecznej transmisji danych w łączu HF.

Bibliografia

[1] 3GPP Standards, 35 Series, *Security algorithms*.

[2] Bard J., Kovarik Jr. V. J., *Software Defined Radio, The Software Communications Architecture*, John Wiley & Sons, 2007.

[3] Bausspies F., *An approach to secure networks*, Proceedings of SECURICOM '90, str. 159-164, 1990.

[4] Biham E., Shamir A., Differential *cryptanalysis of DES-like cryptosystems*, Journal of Cryptology, nr 4, str. 3-72, 1991.

[5] Biham E., Shamir A., Differential *cryptanalysis of the data encryption standard*, Springer-Verlag, 1993.

[6] Biham E., Shamir A., Differential *cryptanalysis of the full 16-round DES*, Lecture Notes in Computer Science, nr 740, str. 487-496, Springer, Berlin – Heidelberg – New York, 1992.

[7] **Bronk K.**, *Koncepcja systemu bezpiecznej transmisji danych w paśmie krótkofalowym*, Przegląd Telekomunikacyjny, nr 6, 2009, str. 443.

[8] **Bronk K.**, *Koncepcja systemu bezpiecznej transmisji w paśmie krótkofalowym – Etap I*, Instytut Łączności, Gdańsk 2008.

[9] **Bronk K.**, *Koncepcja systemu bezpiecznej transmisji w paśmie krótkofalowym – Etap II*, Instytut Łączności, Gdańsk 2009.

[10] **Bronk K.**, Lipka A., *Model modemu krótkofalowego definiowanego programowo dla potrzeb radiokomunikacji morskiej*. Przegląd Telekomunikacyjny i Wiadomości Telekomunikacyjne nr 4, s. 233-236, 2008.

[11] **Bronk K.**, Stefański J., *Software Defined HF Data Modem*, Zeszyty Naukowe Akademii Marynarki Wojennej w Gdynia, 2007, str. 53-59.

[12] **Bronk K.**, Stefański J., *Zdefiniowany programowo modem krótkofalowy dla potrzeb radiokomunikacji morskiej*, Zeszyty Naukowe Wydziału Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej; Radiokomunikacja, Radiofonia i Telewizja, nr 1, 2007, str. 113-116.

[13] **Bronk K.**, *The Concept of the Secure HF Data Transmission*, Polish Journal of Environmental Studies, Vol. 18, No. 4B, HARD, Olsztyn 2009, str. 7-11.

[14] Chi-Feng L., Yan-Shun K., Hsia-Ling C., Chung-Huang Y., *Fast implementation of AES cryptographic algorithms in smart cards*, Security Technology, 2003.

[15] Churchhouse R. F., *The Achilles heel of the ENIGMA cipher machine, and some of its consequences*, IMA Bulletin, 1993.

[16] Coppersmith D., *The Data Encryption Standard (DES) and its strength against attacks*, IBM Journal of Research and Development, nr 38(3), str. 243-250, 1994.

[17] Cramer R., Damagard I., *New generation of secure and practical RSA-based signatures*, Lecture Notes in Computer Science, nr 1109, str. 487-496, Springer, Berlin – Heidelberg – New York, 1996.

[18] Daemen J., Rijmen V., *The design of Rijndael*, Springer, Berlin – Heidelberg – New York, 2002.

[19] Daniluk A., *Kompendium programisty C++ Builder Borland Developer Studio 2006*, Helion, 2006.

[20] Denning D., Sacco G., *Timestamps in key distribution protocols*, Communications of the ACM, nr 24(8), str. 533-536, 1981.

[21] Deshpande A.M., Deshpande M.S., Kayatanavar D.N., *FPGA implementation of AES encryption and decryption*, INCACEC 2009.

[22] Diffie W., Van Oorschot P., Wiener M., *Authentication and authenticated key exchange*, Designs, Codes and Cryptography, str. 107-125, 1992.

[23] Federal Information Processing Standards (FIPS), PUB 140-2, *Security Requirements for Cryptographic Modules*, 2002.

[24] Federal Information Processing Standards (FIPS), PUB 180-2, *Secure Hash Standard,* 2002.

[25] Federal Information Processing Standards (FIPS), PUB 186-2, *Digital Signature Standard (DSS),* 2000.

[26] Federal Information Processing Standards (FIPS), PUB 197, *Advanced Encryption Standard,* 2001.

[27] Federal Information Processing Standards (FIPS), PUB 198, *The Keyed-Hash Message Authentication Code (HMAC)*, 2002.

[28] Forum of Incident Response and Security Teams (FIRST), Common Vulnerability Scoring System (CVSS-SIG), Dostępny pod adresem: http://www.first.org/cvss/.

[29] Gollmann D., *Computer Security*, Wiley, New York, 1999.

[30] Gollmann D., *What do we mean by entity authentication*, IEEE Symposium on Research in Security and Privacy, str. 46-54, IEEE, 1996.

[31] Gręboszl J., *Pasja C++ - Szablony, pojemniki, i obsługa sytuacji wyjątkowych w języku C++*, Edition 2000, Kraków.

[32] Gręboszl J., *Symfonia C++ standard – Programowanie w języku C++ orientowane obiektowo*, Edition 2000, Kraków 2006.

[33] Haber S., Stornetta W. S., *How to timestamp a digital document*, Journal of Cryptology, nr 3, str. 99-111, 1991.

[34] Hartson H. R., *Database security – system architectures*, Information Systems, nr 6, str. 1-22, 1981.

[35] Henning R. R., Walker S. A., *Computer architecture and database security*, Proceedings of the 9th National Computer Conference, str. 216-230, National Bureau of Standards/National Computer Security Center, Gaithersburg, 1986.

[36] IEEE Standard for Information technology, Telecommunications and information exchange between systems, Local and metropolitan area networks, Specific requirements, *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2007.

[37] Kahn D., *Łamacze kodów, Historia Kryptologii*, WNT, 2004.

[38] Katulski R., Marczak A., Stefański J., *Programowalny system radiokomunikacyjny do zastosowań wojskowych*. XII Konf. Nauk. "Automatyzacja dowodzenia", Gdynia, 2004.

[39] Katulski R., *Materiały pomocnicze do przedmiotu Systemy Radiokomunikacyjne*.

[40] Kaukonen K., Thayer R., *A Stream Cipher Encryption Algorithm "Arcfour" (RC4),* Internet-Draft, 1999.

[41] Koblitz N., Algebraic Aspects of Cryptography, Springer, Berlin – Heidelberg – New York, 1999.

[42] Koblitz N., *Elliptic curve cryptosystems*, Mathematics of Computation, nr 48(177), str. 203-209, 1987.

[43] Kolakowski J., Cichocki J., *UMTS System telefonii komórkowej trzeciej generacji*, WKŁ, Warszawa 2003.

[44] Mao W., *Modern Cryptography – Theory and Practice*, Prentice Hall, lipiec 2003.

[45] Matsui M., *Linear cryptanalysis method for the DES cipher*, Lecture Notes in Computer Science, nr 765, str. 386-397, Springer, Berlin – Heidelberg – New York, 1994.

[46] MediaCrypt, *International Data Encryption Algorithm (IDEA).*

[47] Mell P., Scarfone K., Romanosky S., *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*, NIST, 2007.

[48] Menezes A., *Elliptic Curve Public Key Cryptosystems*, Kluwer, Dordrecht, 1993.

[49] Menezes A., Van Oorschot P., Vanstone S., *Handbook of Applied Cryptography*, CRC, Boca Raton, FL, 1997.

[50] Microsoft Corporation, Microsoft Security Response Center Security Bulletin, Severity Rating System, 2002. Dostępny pod adresem: http://www.microsoft.com/technet/security/bulletin/rating.mspx/.

[51] MIL-STD-188-110A, *Interoperability and Performance Standards for Data Modems*, Military Standard, U.S. Army Information Systems Engineering Command, 1991.

[52] MIL-STD-188-110B, *Interoperability and Performance Standards for Data Modems*, Departament of Defense Interface Standard, July 2004.

[53] Mitola J., *Software Radio Architecture*. John Wiley & Sons, 2000.

[54] National Institute of Standards and Technology (NIST), *Data Encryption Standard (DES),* 1999.

[55] National Institute of Standards and Technology (NIST), *Recommendation for Block Cipher Modes of Operation*, 2001.

[56] National Institute of Standards and Technology (NIST), *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*, 2005.

[57] National Institute of Standards and Technology (NIST), *Recommendation for Key Management – Part 1: General (Revised)*, 2007.

[58] Neibauer A. R., *Języki C i C++*. Komputerowa Oficyna Wydawnicza „HELP", 2004.

[59] Pfitzmann B., *Digital Signatures Schemes*, Lecture Notes in Computer Science, nr 1100, Springer, Berlin – Heidelberg – New York, 1996.

[60] Pieprzyk J., Hardjono T., Seberry J., *Teoria bezpieczeństwa systemów komputerowych*, HELION.

[61] PKCS - RSA Laboratories, *RSA Cryptography Standard,* 2002.

[62] Preneel B., Goyaerts R., Vandewalle J., Hash functions based on block ciphers: a synthetic approach, Lecture Notes in Computer Science, nr 773, str. 368-378, 1994.

[63] Preneel B., Goyaerts R., Vandewalle J., *Information authentication: hash functions and digital signatures*, Lecture Notes in Computer Science, nr 741, str. 87-131, 1993.

[64] Recommendation ITU-R F.1487, *Testing of HF Modems with Bandwidths of up to about 12 kHz Using Ionospheric Channel Simulators*, 2000.

[65] RF-5710A 9600/12800 BPS HF/LF MODEM, Harris RF Communications, USA, http://www.rfcomm.harris.com/products/tactical-networking-data/#2.

[66] RF-5710A-MD002 2400/4800 BPS HF MODEM, Harris RF Communications, USA, http://www.rfcomm.harris.com/products/tactical-networking-data/#2.

[67] Rhee M. Y., *Cryptography and Secure Communications*, McGraw-Hill, 1994.

[68] Rivest R. L., *The MD4 message digest algorithm*, Lecture Notes in Computer Science, nr 537, str. 303-311, Springer, Berlin – Heidelberg – New York, 1991.

[69] Rivest R. L., *The MD5 message-digest algorithm*, Internet Request for Comments, RFC1321, 1992.

[70] Rivest R., Network Working Group, *The MD5 Message-Digest Algorhitm,* 1992.

[71] Rizk M. R. M., Morsy M., *Optimized Area and Optimized Speed Hardware Implementations of AES on FPGA*, Design and Test Workshop, 2007.

[72] RM6 HF Modem & ALE Controller, RapidM Pty (Ltd), Płd. Afryka, 2004, http://www.rapidm.com.

[73] Rueppel R. A., *Analysis and Design of Stream Ciphers*, Springer-Verlag, 1986.

[74] Rueppel R. A., Van Oorschot P. C., *Modern Key agreement techniques*, Computer Communications, 1994.

[75] Salomaa A., *Public-Key Cryptography*, Springer, Berlin – Heidelberg – New York, 1996.

[76] SANS Institute, SANS Critical Vulnerability Analysis Archive, 2007. Dostępny pod adresem: http://www.sans.org/newsletters/cva/.

[77] Schneier B., *Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C*, John Wiley & Sons, 1996.

[78] Simmons G. J., *A survey of information authentication*, The Science of Information Integrity, str. 379-420, IEEE, 1992.

[79] SKANTI M1400A High Speed HF Data Modem, SKANTI A/S, Dania, http://www.themys-sa.com/document/17_SK_M_1400_A_LOSBL.PDF.

[80] Specification of the Bluetooth System, *Bluetooth specification version 2.1 + edr*, 2007.

[81] Stallings W., *Cryptography and Network Security – 4th edition*, Prentice Hall, listopad 2005.

[82] STANAG 5066, The adoption of a Profile for HF Data Communications, supporting Selective Repeat ARQ error control, HF E-Mail and IP-over-HF operations, NATO.

[83] STANAG 4539 (Edition 1), *Technical Standards for Non-hopping HF Communications Waveforms*, NATO 2000.

[84] Standards for Efficient Cryptography, *Elliptic Curve Cryptography*, 2000.

[85] Stefański J., **Bronk K.**, Lipka A., Radziwanowski M., *Szybka transmisja danych w paśmie krótkofalowym; Etap 3: Opracowanie platformy programowej modemu,* Instytut Łączności, Warszawa 2007.

[86] Stefański J., **Bronk K.**, Niski R., Radziwanowski M., *Szybka transmisja danych w paśmie krótkofalowym; Etap 2: Opracowanie uniwersalnej platformy sprzętowej modemu*, Instytut Łączności, Warszawa 2006.

[87] Stefański J., Gencza S., Niski R., Radziwanowski M., *Szybka transmisja danych w paśmie krótkofalowym; Etap 1: Opracowanie pakietu symulującego pracę toru nadawczo-odbiorczego modemu w krótkofalowym kanale radiowym*, Instytut Łączności, Warszawa 2005.

[88] Stefański J., *Trendy rozwojowe technologii radia programowalnego*. Zesz. Nauk. Wydz. ETI Polit. Gdań., Technologie Informacyjne nr 4, str. 264-270, 2004.

[89] Stinson D. R., *Kryptografia w teorii i praktyce*, WNT, Warszawa 2005.

[90] Sutton R. J., *Bezpieczeństwo telekomunikacji*, WKŁ, Warszawa 2004.

[91] Tsudik G., *Message authentication with one-way hash functions*, ACM SIGCOMM, Computer Communication Review, nr 22(5), str. 29-38, 1992.

[92] United States Computer Emergency Readiness Team (US-CERT), US-CERT Vulnerability Note Field Descriptions, 2006. Dostępny pod adresem: http://www.kb.cert.org/vuls/html/fieldhelp/.

[93] *Ustawa o ochronie informacji niejawnych*, Dz. U. Nr 11, poz. 95, styczeń 1999.

[94] Welschenbach M., *Kryptografia w C i C++*, MIKOM, Warszawa 2002.

[95] Wesołowski K.: *Podstawy cyfrowych systemów telekomunikacyjnych*. WKŁ, Warszawa, 2003.

[96] Winternitz R. S., *Producing a one-way hash function from DES*, Advances in Cryptology, str. 203-207, Plenum, New-York.

[97] Zheng Y., Matsumoto T., Imai H., *Structural properties of one-way hash functions*, Lecture Notes in Computer Science, nr 537, str. 285-302, Springer, Berlin – Heidelberg – New York, 1991.