POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

Imię i nazwisko autora rozprawy:	mgr inż. Mariusz Dzwonkowski
Dyscyplina naukowa:	Telekomunikacja

ROZPRAWA DOKTORSKA

Tytuł rozprawy w języku polskim:	Kwaternionowy system kryptograficzny dla zabezpieczania danych multimedialnych
Tytuł rozprawy w języku angielskim:	A quaternion cryptosystem for securing multimedia data

Promotor

podpis

dr hab. inż. Roman Rykaczewski

STRESZCZENIE

Problemem badawczym, którego dotyczy rozprawa jest kryptograficzne zabezpieczanie danych multimedialnych, głównie obrazów i sygnałów video, przed działaniami niepowołanych osób i organizacji, chcących uzyskać do nich dostęp i mieć możliwość ich nielegalnego wykorzystania oraz modyfikacji. W rozprawie wykazano, że wykorzystanie właściwości rachunku kwaternionowego umożliwia zbudowanie kwaternionowego systemu kryptograficznego dla bezpiecznej i wydajnej transmisji multimediów. Celem pracy było zaprojektowanie symetrycznego algorytmu kryptograficznego opartego na rotacjach kwaternionowych, cechującego się zarówno dużą szybkością przy szyfrowaniu i deszyfracji jak i dużą odpornością na ataki kryptoanalityczne. Algorytm powinien też obsługiwać dane dowolnego typu i rozmiaru. W części wstępnej rozprawy, przedstawiono podstawowe pojęcia związane z kryptografią, omówiono podstawy rachunku kwaternionowego, koncepcję szyfrowania kwaternionowego oraz tło naukowe proponowanych rozwiązań. W części głównej rozprawy przedstawiono opisy i analizę algorytmów modularnego szyfrowania kwaternionowego dla trzech trybów szyfrowania: ECB, CBC i CTR; algorytmu QFC wykorzystującego zmodyfikowaną sieć Feistela (w wersji podstawowej i rozszerzonej) oraz algorytmu kwaternionowego dla fingerprintingu (wykorzystującego kontrolowane zniekształcenia generowane przy deszyfracji). Przedstawiono też algorytm obustronnego generatora kluczy dla systemów symetrycznych oraz algorytm generacji kluczy wykorzystujący kwaternionowe zbiory Julia. Wszystkie zaproponowane rozwiązania zostały dokładnie przebadane pod względem wydajnościowym i pod względem odporności na podstawowe ataki kryptoanalityczne. W podsumowaniu rozprawy omówiono otrzymane wyniki oraz przedstawiono kierunki kontynuacji badań.

Gdańsk, rok 2017

ABSTRACT

The research problem dealt with in this dissertation concerns the security of multimedia contents (i.e. images and video signals) by using cryptographic means in order to prevent their illegal modification and use by a third party. It is demonstrated that the application of quaternion calculus makes it possible to create a quaternion cryptosystem for secure and efficient transmission of multimedia data. The aim of this study was to design a symmetric cryptosystem based on quaternion rotations with efficient encryption and decryption processes as well as with robustness to cryptographic attacks. The encryption algorithm was meant to handle data of any size and type. The introductory part presents basic information on cryptography, quaternion calculus, quaternion encryption as well as the literature of the research field. The main part of this dissertation describes and analyzes a modular quaternion encryption for three encryption modes: ECB, CBC and CTR; a QFC algorithm (with two implementations: fast and secure) based on the modified Feistel network; and a quaternion algorithm for fingerprinting (which features controlled artifacts during the decryption process). A both-sided key generation scheme is also presented as well as a key generation algorithm based on quaternion Julia sets. All of the presented methods are closely scrutinized and tested for their efficiency and robustness to cryptographic attacks. The summary presents a discussion of the results and further research directions.

SPIS TREŚCI

1. WSTĘP

Bezpieczeństwo przesyłania informacji przez niezabezpieczone kanały staje się problemem skupiającym coraz większą uwagę. Ponieważ liczba metod pozwalających na przechwycenie transmitowanych danych wzrasta bardzo dynamicznie, zatem ciągle poszukuje się skutecznych sposobów, aby utrudnić, bądź uniemożliwić przeprowadzenie potencjalnych ataków ze strony osób trzecich. Dodatkowo, widoczne jest coraz większe zapotrzebowanie na systemy przetwarzania informacji wizualnej. Dotyczy to zarówno pojedynczych obrazów statycznych, jak i sekwencji obrazów video. Przetwarzanie tego typu danych staje się coraz bardziej popularne także ze względu na rosnące możliwości techniczne i dostępną moc obliczeniową dzisiejszych komputerów. Również niezwykle silny rozwój komunikacji bezprzewodowej, szczególnie podatnej na działania osób trzecich, powoduje, że poszukuje się nowych metod, które skutecznie zabezpieczą treści przesyłanych danych przed niepowołanym dostępem.

Próbą rozwiązania tego problemu jest system zarządzania prawami cyfrowymi DRM (ang. *Digital Rights Management*) (DRM 2003). DRM to system zabezpieczeń oparty na mechanizmach kryptograficznych lub metodach ukrywania treści, takich jak watermarking i fingerprinting. Zadaniem systemu DRM jest uniemożliwienie używania danych w formacie elektronicznym w sposób sprzeczny z wolą ich wydawcy.

Sposób zabezpieczania przesyłanych informacji zwykle zależny jest od rodzaju danych i ich przeznaczenia. Dla zabezpieczania danych o dużej ważności (np. dane wagi państwowej) priorytetem będzie zastosowanie mechanizmów kryptograficznych oferujących wysoki poziom bezpieczeństwa kosztem mniejszej wydajności przesyłania. W przypadku zabezpieczania danych mniejszej wagi jak np. dane multimedialne, nacisk kładziony jest głównie na szybkość działania stosowanego rozwiązania. Sytuacja wygląda analogicznie w przypadku szyfrowania danych medycznych, których wysoka jakość wizualna okupiona jest dużą objętością pliku. Aby zapewnić szybki dostęp do danych wymagane jest zastosowanie wydajnych mechanizmów kryptograficznych, tak by opóźnienie, wprowadzone przez algorytmy szyfrowania/deszyfrowania, mogło być utrzymywane na akceptowalnym poziomie.

W rozprawie przedstawiono rozwiązania systemów kryptograficznych dla wydajnego zabezpieczania danych dużego rozmiaru, takich jak dane multimedialne i dane medyczne. Zaproponowane rozwiązania oferują zarówno dużą szybkość przetwarzania jak i wysoki poziom bezpieczeństwa, pozwalając na szyfrowanie i deszyfrację danych różnego typu. Z uwagi na przeznaczenie zaproponowanego systemu kryptograficznego, przedstawione w pracy testy, przeprowadzone zostały głównie dla obrazów.

W rozprawie, po rozdziale 2. wprowadzającym w tematykę pracy, przedstawiono cel, tezę i zakres pracy (rozdział 3.), a także dokładnie opisano strukturę i zawartość kolejnych rozdziałów. Rozdział poświęcony przeglądowi literatury obszaru badań (rozdział 7.), został przedstawiony po rozdziałach wprowadzających z kryptografii, podstawowych informacji z zakresu kryptoanalizy i rachunku kwaternionowego (rozdziały 4., 5. i 6.).

2. WPROWADZENIE

Kwestia bezpieczeństwa sieciowego to nie tylko zapewnienie poufności przesyłanych przez sieć danych. Pojęcie bezpieczeństwa sieciowego jest dziedziną znacznie szerszą i należy rozważać ją opierając się na czterech podstawowych pojęciach: poufności danych, integralności danych, uwierzytelnienia i niezaprzeczalności (Schneier 1996, Stallings 2006).

Poufność oznacza niedostępność treści zawartej w danych dla wszystkich podmiotów poza podmiotami uprawnionymi. Dla zapewnienia poufności informacji stosowane są powszechnie systemy kryptograficzne. Procedury uwierzytelniania, ograniczania uprawnień dostępu czy ograniczanie fizycznego dostępu do systemu komputerowego są środkami pośrednimi prowadzącymi do osiągnięcia tego celu. Pomimo stosowania środków zapewniających poufność, istnieje niebezpieczeństwo przypadkowego lub celowego jej naruszenia. W związku z tym system ochrony powinien nie tylko zapewniać poufność, lecz także gwarantować możliwość wykrycia prób i przypadków naruszenia danych.

Integralność danych oznacza, że dane nie zostaną w żaden nieuprawniony sposób zmienione, a tym samym ich stan pozostanie zgodny z wymaganym i oczekiwanym stanem właściwym. Integralność danych może być naruszona przez nieuprawnionego użytkownika jak i błędy i zaniedbania popełniane przez użytkownika upoważnionego. Dodatkowo błędy w oprogramowaniu, zakłócenia transmisji czy działania wirusów również niekorzystnie wpływają na utrzymanie integralności danych. Nieupoważniona modyfikacja nie musi wiązać się z naruszeniem poufności danych. Podobnie jak poufność, integralność musi być zapewniona podczas przetwarzania, przechowywania i przesyłania informacji. Integralność danych weryfikuje się, stosując głównie algorytmy kryptograficzne zwane funkcjami skrótu. W sposób pośredni można przyczynić się do osiągnięcia tego celu poprzez stosowanie protokołów uwierzytelniania, ograniczanie uprawnień dostępu, ograniczanie fizycznego dostępu do systemu komputerowego, stosowanie metod zwiększających niezawodność sprzętu i tolerancję na błędy.

Uwierzytelnienie to proces mający na celu potwierdzeniu zadeklarowanej tożsamości podmiotu biorącego udział w procesie komunikacji. Strona weryfikująca tożsamość podmiotu musi uzyskać ustalony poziom pewności, że dany podmiot jest w rzeczywistości tym, za który się podaje. Do uwierzytelniania podmiotów służą protokoły kryptograficzne. Protokół jest algorytmem wykonywanym przez co najmniej dwie strony. Kryptografia oferuje protokoły uwierzytelniania tworzone za pomocą szyfrów symetrycznych, asymetrycznych i funkcji skrótu. Wskutek ich wykonania jedna strona nabywa pewność co do tożsamości strony drugiej.

Niezaprzeczalność to brak możliwości wyparcia się swego uczestnictwa w całościowej lub częściowej wymianie danych przez jeden z podmiotów uczestniczących w tej wymianie. Mechanizmami kryptograficznymi używanymi do weryfikacji niezaprzeczalności są szyfry asymetryczne służące do wykonywania podpisów cyfrowych i funkcje skrótu z kluczem kryptograficznym (czyli kody uwierzytelniające wiadomość). Mówiąc o niezaprzeczalności mamy na uwadze niezaprzeczalność pochodzenia danych od ich nadawcy, dostarczenia danych do ich odbiorcy, ustalenia danych do transportu w sieci i wykonania transportu w sieci.

3. CEL, TEZA I ZAKRES PRACY

Do podstawowych funkcji systemów informatycznych zaliczamy przesyłanie i przechowywanie danych. W obu przypadkach dane muszą być chronione przed różnorodnymi naruszeniami bezpieczeństwa. Ponieważ obecnie do czynienia mamy z ogromną liczbą zagrożeń istniejących w Internecie, zatem coraz bardziej widoczny jest wzrost znaczenia problematyki ochrony danych.

Do najważniejszych metod ochrony transmitowanych danych należą metody kryptograficzne. Dane wymagające ochrony są szyfrowane na czas transmisji. Za pomocą metod kryptograficznych można także wykryć przypadki fałszowania adresu i tożsamości nadawcy oraz nieuprawnionego modyfikowania treści komunikatów.

Jednym z podstawowych problemów stosowania kryptografii w sieciach jest znaczna (rosnąca w sposób dynamiczny) ilość transmitowanych danych. Procesy szyfrowania i deszyfracji dużej ilości danych w istotny sposób wpływają na całkowity czas transmisji. Konieczne staje się więc stosowanie wydajnych algorytmów kryptograficznych, szczególnie w przypadku transmisji danych o treści multimedialnej i medycznej. Dostęp dla tego typu danych nie powinien być ograniczany przez duże opóźnienie wynikające z implementacji dodatkowych warstw bezpieczeństwa. Zastosowany algorytm kryptograficzny dla zabezpieczania danych multimedialnych i danych medycznych musi więc cechować się przede wszystkim dużą wydajnością.

Przykładowo, w pracy Nagase i in. (2004) pokazano możliwość wykorzystania szczególnych właściwości kwaternionów do szybkiego przeprowadzania obrotów (rotacji) wektorów danych, interpretowanych jako nośniki informacji. Zaproponowany tam algorytm szyfrujący cechuje wysoka szybkość działania (z uwagi na operacje kwaternionowe), jednakże dla zaproponowanej implementacji nie uzyskano wysokiego poziomu bezpieczeństwa. Taka możliwość wykorzystania kwaternionów stała się podstawą sformułowania następującej tezy pracy: **"możliwe jest zbudowanie kwaternionowego systemu kryptograficznego dla bezpiecznej i wydajnej transmisji danych multimedialnych i medycznych"**.

Celem pracy jest:

- zbadanie właściwości ciała kwaternionów pod kątem kryptograficznej implementacji rotacji kwaternionowej,
- zaprojektowanie symetrycznego algorytmu kryptograficznego opartego na rotacjach kwaternionowych, cechującego się zarówno dużą szybkością przy szyfrowaniu i deszyfracji jak i dużą odpornością na ataki kryptoanalityczne,
- przeprowadzenie analizy porównawczej opracowanego algorytmu z popularnym algorytmem symetrycznym AES,
- dostosowanie i zbadanie algorytmu pod kątem szyfrowania różnego rodzaju danych, m.in. danych tekstowych, 8-bitowych obrazów kolorowych i 16-bitowych obrazów medycznych DICOM.

Praca składa się z 13 rozdziałów. Rozdziały od 1. do 7. stanowią część wstępną rozprawy, która zawiera opis tematyki pracy, przedstawienie problemu badawczego, oraz analizę źródeł literaturowych. Rozdziały od 8. do 12. stanowią główną część rozprawy, w której przedstawiono algorytmy szyfrowania kwaternionowego wraz z ich praktycznym zastosowaniem. Rozdział 13. stanowi zakończenie i podsumowanie

rozprawy. Poniżej znajduje się opis zagadnień omawianych w poszczególnych rozdziałach.

W rozdziale 4. scharakteryzowano podstawowe zagadnienia dotyczące kryptografii i kryptoanalizy. Omówiono koncepcję kryptografii symetrycznej i asymetrycznej. Wyjaśniono pojęcia funkcji skrótu i podpisu cyfrowego oraz przedstawiono podstawowe ataki kryptoanalityczne.

W rozdziale 5. zostały przedstawione podstawy matematyczne rachunku kwaternionowego. Przedstawiono w nim pojęcie ciała kwaternionów, a także macierzową i algebraiczną reprezentację kwaternionów, których znajomość jest wymagana do pełnego zrozumienia dalszych, zawartych w tej pracy, zagadnień. Przedstawiono także ogólne właściwości kwaternionów i charakterystyczne dla tego ciała, operacje: dodawanie, mnożenie, sprzężenie, odwrotność.

W rozdziale 6. zawarta została koncepcja szyfrowania kwaternionowego, opierającego się na pojęciu kwaternionowej rotacji przestrzennej wektora danych względem kwaternionu-klucza. Opisywana w tej pracy metoda szyfrowania może zostać zrealizowana na dwa sposoby. Pierwszy sposób polega na obliczeniu macierzy rotacji i przeprowadzeniu szyfrowania jako zwykłego mnożenia macierzowego (metoda macierzowa), drugi natomiast realizuje rotację kwaternionową zgodnie z rachunkiem kwaternionów (metoda kwaternionowa). W rozdziale tym omówiono także postać macierzy rotacji, wykorzystywanej m.in. do generacji przestrzeni kluczy szyfrujących. Rozdział kończy propozycja sposobu optymalizacji metody szyfrowania za pomocą jednorazowej rotacji większych porcji danych.

Rozdział 7. został poświęcony przeglądowi literatury związanej z szyfrowaniem symetrycznym i asymetrycznym, wykorzystujących rotacje kwaternionowe.

W rozdziale 8. przedstawiono koncepcję zaproponowanego kwaternionowego systemu kryptograficznego w dwóch wersjach: szybkiej i bezpiecznej. Omówiono także możliwe do zastosowania tryby szyfrowania symetrycznego (ECB, CBC, CTR) przy wykorzystaniu większej liczby bloków dla danych wejściowych.

Rozdział 9. został poświęcony propozycji generatora kluczy dla symetrycznego systemu kryptograficznego, generującego klucze symetryczne jednocześnie dla obu komunikujących się stron. Umożliwia to komunikację bez udziału strony trzeciej, odpowiedzialnej za dostarczenie klucza tajnego. Dodatkowo, w tym rozdziale przedstawiono modyfikację algorytmu generacji kluczy poprzez zastosowanie kwaternionowych zbiorów Julia.

W rozdziale 10. omówiono możliwe zastosowania zaproponowanego algorytmu szyfrującego. Zastosowania dotyczą m.in. możliwości umieszczania fingerprintów w odszyfrowanych obrazach oraz wydajne szyfrowanie 16-bitowych obrazów medycznych DICOM.

W rozdziale 11. zamieszczone zostały wyniki badań i testów przeprowadzonych przy użyciu oprogramowania MATLAB. Testy przeprowadzono na przykładowych obrazach (kolorowych oraz w odcieniach szarości). Zbadano m.in. wpływ zastosowanego trybu szyfrowania na właściwości uzyskiwanych szyfrogramów, przeprowadzono szereg testów losowości wykorzystując pakiet Diehard (Marsaglia 1995) i darmowy program Cryptool (CrypTool 2015). Zbadano też efekt lawinowy dla zaproponowanego algorytmu. W tym celu porównano szyfrogramy uwzględniając zmianę 1 bitu w kluczu, zmianę 1 bitu w danych do zaszyfrowania oraz

przeprowadzono deszyfrację po uwzględnieniu zmiany 1 bitu w kluczu. Dodatkowo, przeprowadzono analizę porównawczą zaproponowanego algorytmu ze standardowym algorytmem symetrycznym AES.

Rozdział 12. poświęcony jest zagadnieniom kryptoanalizy. W rozdziale przeanalizowano atak brutalny na przestrzeń kluczy dla obydwu wersji algorytmu, postaci kluczy inicjalizujących, generujących słabe klucze rundowe oraz atak ze znanym tekstem jawnym dla algorytmu w wersji bezpiecznej.

W rozdziale 13. przedstawiono podsumowanie całej rozprawy. Wyszczególniono m.in. oryginalny dorobek autora, ocenę osiągniętych rezultatów oraz perspektywy dalszych badań.

4. PODSTAWOWE POJĘCIA KRYPTOLOGII

Celem rozdziału jest bardzo zwięzłe przedstawienie najważniejszych pojęć z zakresu kryptografii i kryptoanalizy, niezbędnych dla zrozumienia przedstawianych w rozprawie rozwiązań.

Kryptografia jest dziedziną skupiającą się na projektowaniu i konstruowaniu algorytmów kryptograficznych. Do jej przedmiotu zainteresowań należą algorytmy umożliwiające przekształcanie wiadomości jawnej w jej postać tajną, czyli szyfrogram oraz algorytmy działające odwrotnie. Algorytmy zazwyczaj uzależnione są od wielkości zwanej kluczem kryptograficznym.

System kryptograficzny jest terminem ogólnym obejmującym zbiór podstawowych urządzeń kryptograficznych implementujących algorytmy kryptograficzne w celu zapewnienia niezbędnych usług w zakresie bezpieczeństwa informacji. Ze względu na rodzaj klucza kryptograficznego, systemy kryptograficzne dzielimy na symetryczne (nazywane też systemami z kluczem tajnym) oraz asymetryczne (z kluczem publicznym).

4.1. Kryptografia symetryczna

Kryptografia symetryczna zakłada wykorzystanie pojedynczego klucza kryptograficznego zarówno przy szyfrowaniu danych po stronie nadawczej, jak i przy odszyfrowaniu otrzymywanych danych po stronie odbiorczej. Istnieją algorytmy symetryczne, w których klucz wykorzystywany po stronie odbiorczej nie jest identyczny z kluczem używanym po stronie nadawczej, lecz jest jedynie z nim powiązany – może być na przykład jego odwrotnością. Takie klucze są elementem tajnym, znanym tylko komunikującym się ze sobą użytkownikom. Stąd też bardzo często używa się określenia szyfrowanie z kluczem tajnym.

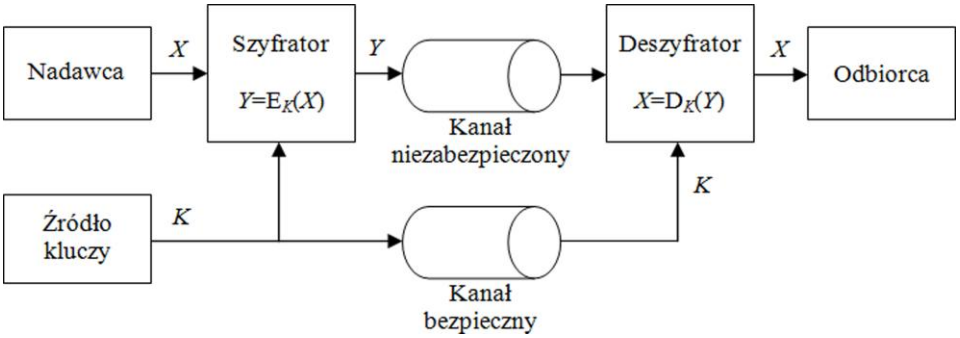
Kryptograficzny system symetryczny definiowany jest przez parę funkcji: szyfrującej (4.1) i deszyfrującej (4.2):

$$Y = \mathrm{E}_K(X) \tag{4.1}$$

$$X = \mathrm{D}_K(Y) \tag{4.2}$$

gdzie E_K to funkcja szyfrująca wiadomość X przy użyciu klucza tajnego K, natomiast D_K to funkcja deszyfrująca zastosowana do zaszyfrowanej wiadomości Y przy użyciu klucza tajnego K.

Aby przeprowadzić szyfrowanie danych w systemie symetrycznym, klucze tajne muszą zostać dostarczone obu stronom. Najczęściej jest to realizowane przez niezależną stronę trzecią, gdzie transmisja kluczy prowadzona jest przy użyciu bezpiecznego kanału za pośrednictwem szyfrowania asymetrycznego, przed rozpoczęciem właściwej transmisji (Rys. 1).



Rys. 1. Schemat symetrycznego systemu kryptograficznego.
Źródło: Opracowanie własne.

Algorytmy symetryczne cechuje duża szybkość przetwarzania (w porównaniu do algorytmów asymetrycznych) dzięki czemu dobrze nadają się do wykonywania przekształceń kryptograficznych na dużych strumieniach danych.

Istnieją dwa rodzaje algorytmów symetrycznych: szyfry blokowe (ang. *block ciphers*) oraz szyfry strumieniowe (ang. *stream ciphers*).

Zasada działania szyfrów blokowych polega na podziale wiadomości szyfrowanej na bloki o ustalonej długości i wykonaniu operacji kryptograficznych na każdym bloku z osobna. Jeśli ostatni fragment wiadomości jest zbyt krótki, to jest on uzupełniany (ang. *padded*) do zadanej długości bloku. Do podstawowych operacji przeprowadzanych na blokach zaliczamy operacje podstawienia (ang. *substitution*), czyli zastąpienie jednej grupy znaków inną, oraz operacje permutacji (ang. *transposition*), czyli operacji przestawiania grup znaków wewnątrz bloku. Jak udowodnił w swojej publikacji Claude Shannon (Shannon 1949), powyższe operacje stosowane osobno nie gwarantują wystarczającego poziomu bezpieczeństwa, ale wykonywane naprzemiennie (iloczynowe systemy kryptograficzne) mogą zapewnić wysokie bezpieczeństwo. Bardzo często operacje podstawienia i permutacji grupuje się w pary tworząc tzw. rundę. W projektowaniu współczesnych szyfrów blokowych stosuje się wiele rund, gdzie każda runda szyfrowana jest innym podkluczem, wyznaczonym na podstawie klucza głównego. Powszechnie znaną implementacją tego typu szyfrów jest sieć Feistela (Menezes i in. 1996). Dodatkowo, implementacja większości szyfrów blokowych bardzo często związana jest z zastosowaniem dodatkowych trybów szyfrowania. Więcej informacji na temat trybów szyfrowania zawarto w rozdziale 8.1.

Przykładem najpopularniejszych szyfrów blokowych są algorytmy szyfrowania symetrycznego: Data Encryption Standard (DES), Triple DES (3DES), Advanced Encryption Standard (AES), IDEA, Blowfish, RC2 i inne.

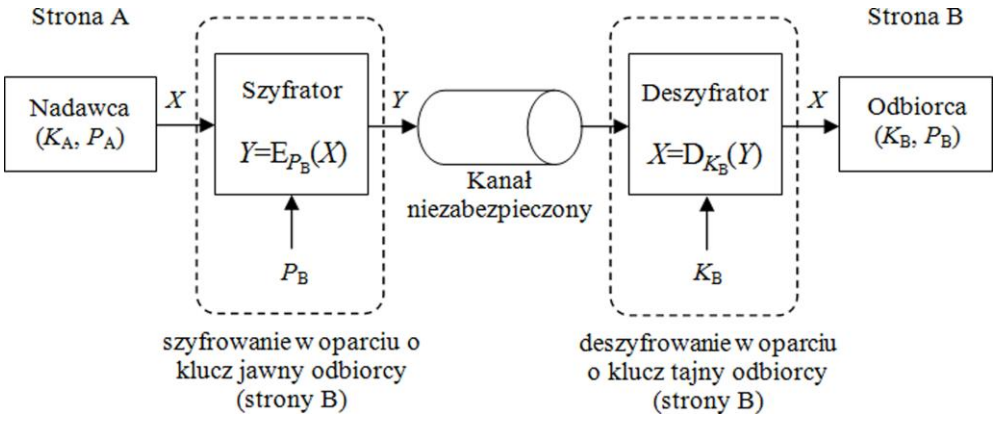
W przypadku szyfrów strumieniowych, ich struktura jest w ogólności podobna do szyfrów blokowych. Zasadniczą różnicą jest długość bloku danych, która w tym przypadku równa jest jednemu znakowi. Strumień znaków tekstu jawnego jest łączony z odpowiadającym strumieniem znaków klucza (ang. *keystream*) tworząc zaszyfrowany strumień danych. Ponieważ we współczesnej praktyce za znak uznaje się pojedynczy bit, operacją łączącą obydwa strumienie (tekstu jawnego i klucza) jest zwykle funkcja XOR wykonywana bit po bicie.

Możliwe jest utworzenie szyfru strumieniowego, o którym wiadomo na pewno (Shannon 1949), że jest niemożliwy do złamania – jest to tzw. szyfr jednokrotny (ang. *one-time pad*). Wykorzystuje on operację XOR wiążącą kolejno znaki strumienia tekstu jawnego z odpowiadającymi znakami strumienia klucza. Dla takiego szyfru strumień klucza musi zostać wygenerowany w sposób całkowicie losowy o długości co najmniej równej długości strumienia tekstu jawnego i być użyty wyłącznie jeden raz. Z tego też powodu implementacja szyfru jednokrotnego w praktyce jest dość kłopotliwa. Stosuje się go głównie do zabezpieczania bardzo ważnych informacji.

4.2. Kryptografia asymetryczna

Odmienną grupą algorytmów szyfrujących, wykorzystywaną w kryptografii są algorytmy szyfrowania asymetrycznego. W przeciwieństwie do szyfrowania symetrycznego, algorytm asymetryczny zakłada użycie dwóch różnych kluczy do przeprowadzenia szyfrowania i do odszyfrowania danych. Mechanizmy mające zastosowania w kryptografii asymetrycznej wykorzystują funkcje jednokierunkowe, tzn. takie, które można łatwo przeprowadzić w jedną stronę, a bardzo trudno w drugą. Kryptografia asymetryczna jest znacznie wolniejsza w porównaniu z algorytmami symetrycznymi, dlatego też najczęściej wykorzystuje się ją do bezpiecznego przesłania kluczy tajnych w kryptografii symetrycznej (Goldwasser i Bellare 2008, Schneier 1996).

W asymetrycznym systemie kryptograficznym każda z komunikujących się stron posiada zestaw dwóch kluczy. Jeden udostępniany jest publicznie i nosi nazwę klucza jawnego (publicznego). Drugi klucz pozostaje tajny i znany jest tylko danemu użytkownikowi. W zależności od sposobu użycia kluczy, można uzyskać różne zastosowania kryptografii asymetrycznej. Przykładowo, na zrealizowanie szyfrowania danych u nadawcy zakładamy użycie klucza publicznego odbiorcy, a deszyfrację tych danych u odbiorcy realizujemy wykorzystując jego klucz tajny. Zostało to przedstawione na Rys. 2. Dodatkowo, w przypadku odwrotnego użycia kluczy i obliczenia skrótu wiadomości, mamy możliwość uzyskania podpisu cyfrowego (Goldwasser i Bellare 2008, Schneier 1996).



Rys. 2. Schemat asymetrycznego systemu kryptograficznego.
Źródło: Opracowanie własne.

Zarówno strona A jak i strona B posiada swój własny zestaw kluczy: jawny P i tajny K. Proces wyglądałby analogicznie, gdyby strona B chciała wysłać wiadomość do strony A. Strona B użyłaby do zaszyfrowania danych klucza publicznego P_A, natomiast strona A zdeszyfrowałaby wiadomość swoim kluczem tajnym K_A.

Najpowszechniej wykorzystywanym w praktyce algorytmem asymetrycznym jest algorytm RSA (Johnson i Kaliski 2013), który wziął swoją nazwę od pierwszych liter nazwisk twórców: Ronalda Rivesta, Adiego Shamira oraz Leonarda Adlemana. Bezpieczeństwo algorytmu RSA jest oparte o trudny problem faktoryzacji liczby złożonej na czynniki pierwsze.

Innym przykładem algorytmów asymetrycznych są algorytmy, których zabezpieczenie wynika z trudności obliczenia logarytmów dyskretnych w ciałach skończonych (Schneier 1996), np. algorytm Diffiego-Hellmana (opracowany przez Witfielda Diffiego oraz Martina Hellmana) oraz algorytm ElGamala (opracowany przez Taher Elgamal).

Kryptografia krzywych eliptycznych ECC (ang. *Elliptic Curve Cryptography*) również realizuje koncepcje kryptografii asymetrycznej, wykorzystując jako podstawową technikę matematyczną właściwości krzywych eliptycznych (Miller 1985, Koblitz 1987, Gawinecki i in. 2014).

4.3. Funkcja skrótu

Algorytmami kryptograficznymi, które są stosowane w celu zapewnienia weryfikacji integralności przesyłanych danych, to m.in. funkcje skrótu (ang. *hash function*). Funkcja skrótu przekształca dowolnie długi ciąg danych binarnych w ciąg bitów o ustalonej, stosunkowo niewielkiej długości – tzw. skrót. Chcąc mieć pewność, że dane nie zostały zmienione podczas transmisji, komunikujące się strony muszą najpierw uzgodnić rodzaj funkcji skrótu, a następnie do każdej wysyłanej wiadomości dołączać obliczony skrót. Odbiorca, po otrzymaniu danych, oblicza po swojej stronie skrót wiadomości i porównuje go z nadesłanym skrótem. Jeżeli obydwa skróty są takie same, to odbiorca ma pewność, że otrzymał dane niezmienione.

Funkcja skrótu jest algorytmem kryptograficznym, do działania którego nie jest wymagany klucz kryptograficzny. Niemniej, w prosty sposób można rozbudować ten mechanizm, wprowadzając do niego klucz tajny. Funkcję skrótu z kluczem nazywa się kodem uwierzytelniającym wiadomość MAC (ang. *Message Authentication Code*). MAC służy zarówno do badania integralności danych, jak i ich uwierzytelniania – tajny klucz jest wyłącznie w posiadaniu dwóch określonych stron wymieniających dane.

4.4. Podpis cyfrowy

Inną możliwość uwierzytelnienia danych daje podpis cyfrowy. Do wykonania podpisu cyfrowego wykorzystujemy algorytmy asymetryczne. Są one stosowane w nieco inny sposób, niż w sytuacji zapewniania poufności przesyłanych danych. Chcąc otrzymać podpis cyfrowy, należy po stronie nadawczej obliczyć skrót wiadomości i zaszyfrować go kluczem tajnym nadawcy. Podpis weryfikuje się korzystając z tego samego algorytmu asymetrycznego, lecz z kluczem publicznym. Klucz publiczny,

wykorzystywany przez odbiorcę danych, jest jawny, zatem każdy może zweryfikować podpis. Zastosowanie podpisu cyfrowego umożliwia odbiorcy uwierzytelnienie źródła wiadomości oraz weryfikację integralności danych. Dodatkowo nadawca nie ma możliwości wyparcia się wysłania wiadomości do odbiorcy.

4.5. Kryptoanaliza

Kryptoanaliza jest sztuką łamania zabezpieczeń systemów kryptograficznych, poprzez analizę przechwyconych, zaszyfrowanych danych. Kryptoanaliza to synonim szeroko rozumianego ataku na algorytmy kryptograficzne. Celem kryptoanalityka jest odszyfrowanie przechwyconych danych bez znajomości kluczy szyfrujących lub też odgadnięcie wszystkich kluczy użytych podczas szyfrowania. Dlatego też, każdy algorytm kryptograficzny, określany mianem bezpiecznego, powinien cechować się wysokim stopniem złożoności oraz dużą odpornością na kryptoanalizę.

Podstawowym atakiem kryptoanalitycznym jest tzw. atak brutalny (ang. *brute-force attack*). Polega on na sprawdzeniu wszystkich możliwych kluczy do przechwyconego szyfrogramu. Przy bardzo licznej przestrzeni kluczy i dużej złożoności obliczeniowej algorytmu, atak ten staje się mało opłacalny. Z tego też powodu, obok ataku brutalnego, zwykle analizuje się inne ataki. Do najpopularniejszych ataków zaliczamy:

- atak z szyfrogramem (ang. *ciphertext-only attack*)
- atak ze znanym tekstem jawnym (ang. *known-plaintext attack*)
- atak z wybranym szyfrogramem (ang. *chosen-ciphertext attack*)
- atak z wybranym tekstem jawnym (ang. *chosen-plaintext attack*)

W przypadku ataku z szyfrogramem, kryptoanalityk ma dostęp jedynie do pewnej liczby, przechwyconych szyfrogramów. Atak skupia się przede wszystkim na znalezieniu zależności między szyfrogramami w celu ich odszyfrowania. Odgadnięcie kluczy szyfrujących jest celem drugorzędnym z uwagi na ograniczenia ataku.

O ataku ze znanym tekstem jawnym mówi się wtedy, kiedy kryptoanalityk dysponuje odpowiadającymi sobie parami tekst jawny i szyfrogram. Kryptoanalityk nie ma wpływu na treść przechwyconego tekstu jawnego. Analizując zdobyte pary tekstów jawnych i ich szyfrogramów, kryptoanalityk stara się odgadnąć, wykorzystywane w systemie, klucze szyfrujące.

Przy ataku z wybranym szyfrogramem, kryptoanalityk ma wpływ na treść szyfrogramu i ma dostęp do wyniku jego deszyfracji. Atak polega na podawaniu do systemu osobiście spreparowanych szyfrogramów i analizie efektu deszyfracji. W większości przypadków efekt po odszyfrowaniu nie będzie przedstawiać żadnego sensu. Kryptoanalityk skupia się jedynie na analizie porównawczej otrzymywanych w ten sposób danych odszyfrowanych, by w konsekwencji odgadnąć używane przez system klucze szyfrujące.

Atak z wybranym tekstem jawnym (w odróżnieniu od ataku z wybranym szyfrogramem) skupia się na podawaniu do systemu, osobiście spreparowanych, tekstów jawnych i analizowaniu otrzymywanych dla nich szyfrogramów. Celem ataku jest dedukcja wykorzystywanych w systemie kluczy szyfrujących.

Dodatkowo, dla algorytmów symetrycznych, oprócz wyżej wymienionych form ataku, stosuje się także inne metody. Jedną z nich jest kryptoanaliza różnicowa (będąca przykładem ataku z wybranym tekstem jawnym), dla której porównuje się pary szyfrogramów, powstałych w wyniku zaszyfrowania par tekstów jawnych o ustalonych różnicach (Biham i Shamir 1990). Innym podejściem jest kryptoanaliza liniowa, polegająca na opisie działania urządzenia szyfrującego poprzez aproksymację liniową (Matsui 1993).

W dziedzinie kryptoanalizy wyróżnić można także grupę ataków algebraicznych, których celem jest znalezienie i rozwiązanie układu równań wielomianowych (charakterystycznych dla danego algorytmu szyfrującego) nad ciałem skończonym (Kipnis i Shamir 1999).

Kryptoanaliza to bardzo rozległa, złożona i dynamicznie rozwijająca się dziedzina nauki. Ponieważ proponowane w rozprawie rozwiązania kryptograficzne dotyczą bezpieczeństwa multimediów, szczegółowy opis dostępnych ataków krypto-analitycznych wykracza poza zakres niniejszego dokumentu, ponieważ treści multimedialne nie wymagają na ogół bardzo silnego zabezpieczania.

5. PODSTAWY RACHUNKU KWATERNIONÓW

Rozdział ten zawiera podstawy matematyczne rachunku kwaternionów. Pojęcie szyfrowania kwaternionowego wymaga zaznajomienia się z właściwościami ciała kwaternionów oraz z operacjami charakterystycznymi dla tej algebry.

5.1. Ciało kwaternionów

Kwaterniony to struktura algebraiczna, będąca rozszerzeniem ciała liczb zespolonych. Twórcą kwaternionów był irlandzki matematyk William Rowan Hamilton (Hamilton 1844), który wprowadził je związku ze swoimi pracami nad mechaniką obrotów ciał w przestrzeni trójwymiarowej. Hamilton w swoich pracach doszedł do wniosku, że właściwe rozszerzenie liczb zespolonych, tak aby możliwe było opisanie obrotów w przestrzeni trójwymiarowej będzie wymagało wprowadzenia jeszcze jednego, dodatkowego wymiaru. Ostatecznie, rozszerzenie liczb zespolonych zostało zdefiniowane jako wektor czterowymiarowy, składający się z części rzeczywistej i trzech różnych komponentów urojonych: *i*, *j*, *k*. Zapis kwaternionu, będącego sumą części rzeczywistej i części urojonych, przedstawia wzór:

$$q = w + xi + yj + zk \tag{5.1}$$

gdzie *w*, *x*, *y*, *z* są rzeczywistymi współczynnikami kwaternionu *q*. Warto zaznaczyć, że Hamilton wprowadzając kwaterniony musiał zaproponować działania addytywne i multiplikatywne, tak aby zbiór liczb kwaternionów wraz z tymi działaniami tworzył strukturę algebraiczną. Tak jak ma to miejsce w przypadku liczb zespolonych, działanie dodawania polega na sumowaniu odpowiadających sobie komponentów. Inaczej jest jednak w przypadku działania multiplikatywnego. Obydwa te działania zostaną szerzej omówione w rozdziałach 5.3 i 5.4. Hamilton ustalił właściwości jednostek urojonych pozwalające na mnożenie kwaternionów:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{5.2}$$

Związki (5.2) opisują w sposób jednoznaczny i kompletny działanie multiplikatywne w ciele kwaternionów. Należy jednak zauważyć, że związki (5.2) są działaniami nieprzemiennymi (Zhang 1997, Eberly 2010); np. $ij \neq ji$. Struktura algebraiczna, w której działanie multiplikatywne jest nieprzemienne i są spełnione pozostałe warunki dla istnienia ciała nosi nazwę ciała nieprzemiennego. Ciało kwaternionów jest więc ciałem nieprzemiennym.

Oprócz postaci algebraicznej (5.1), kwaterniony są również często przedstawiane w tzw. postaci dwuczłonowej, tzn. w postaci składającej się z części skalarnej i części wektorowej (5.3). W dalszych rozdziałach kwaterniony zapisane będą najczęściej w tej postaci.

$$q = (w, \vec{v}) = (w, [x\ y\ z]^{\mathrm{T}}) \tag{5.3}$$

Należy podkreślić, że postaci (5.3) dla kwaternionu q, używa się jedynie dla czytelniejszej prezentacji wartości współczynników tego kwaternionu, a interpretować należy ją jako wektor:

$$q = [w\ x\ y\ z]^{\mathrm{T}} \tag{5.4}$$

gdzie $[...]^{\mathrm{T}}$ oznacza transpozycję.

Reprezentacja dwuczłonowa kwaternionu (5.3) okaże się szczególnie wygodna przy zapisie mnożenia dwóch kwaternionów, a także przy analizie rotacji kwaternionowej w rozdziale 6.

5.2. Właściwości kwaternionów

Dla ciała kwaternionów, podobnie jak ma to miejsce w przypadku ciała liczb zespolonych, można zdefiniować sprzężenie, normę oraz wartość odwrotną kwaternionu. Ich definicje są niezbędne dla dalszej analizy. Poniższe wzory definiują odpowiednio: sprzężenie, normę i odwrotność kwaternionu:

$$q^{*} = w - xi - yj - zk \tag{5.5}$$

$$\|q\| = \sqrt{w^2 + x^2 + y^2 + z^2} \tag{5.6}$$

$$q^{-1} = \frac{q^{*}}{\|q\|^2} = \frac{w - xi - yj - zk}{w^2 + x^2 + y^2 + z^2} \tag{5.7}$$

Dodatkowo, zdefiniować należy także pojęcie kwaternionu jednostkowego, dla którego zachodzi:

$$\|q\| = 1 \quad \Rightarrow \quad q^{-1} = q^{*} \tag{5.8}$$

Oznacza to, że kwaternion jednostkowy, to kwaternion, którego norma równa się jedności i z wzoru (5.7) wynika, że jego wartość odwrotna jest równa wartości sprzężenia tego kwaternionu. Właściwość ta będzie szczególnie przydatna podczas analizy szyfrowania kwaternionowego przedstawionego w rozdziale 6.

5.3. Dodawanie kwaternionów

W ciele kwaternionów, dodawanie i odejmowanie wykonywane jest względem odpowiadających sobie komponentów – analogicznie jak w przypadku liczb zespolonych. Operacja sprowadza się do sumowania odpowiadających sobie elementów części skalarnej i części wektorowej (Eberly 2010):

$$q_1 + q_2 = w_1 + x_1 i + y_1 j + z_1 k + w_2 + x_2 i + y_2 j + z_2 k = \\ = (w_1 + w_2) + (x_1 + x_2) i + (y_1 + y_2) j + (z_1 + z_2) k \tag{5.9}$$

Dodatkowo, dla dodawania kwaternionowego spełniona jest zasada łączności:

$$(q_1 + q_2) + q_3 = q_1 + (q_2 + q_3) \tag{5.10}$$

5.4. Mnożenie kwaternionów

Omawianie mnożenia kwaternionów należy rozpocząć od przedstawienia związków między jednostkami urojonymi (Nalty 2008), które wynikają z zależności (5.2):

$$ij = k \qquad ji = -k \tag{5.11}$$

$$jk = i \qquad kj = -i \tag{5.12}$$

$$ki = j \qquad ik = -j \tag{5.13}$$

Z powyższych związków między jednostkami urojonymi wynika właściwość nieprzemienności mnożenia kwaternionów:

$$q_1 \cdot q_2 \neq q_2 \cdot q_1 \tag{5.14}$$

gdzie symbol • w tej pracy, oznacza operację mnożenia kwaternionowego. Przy mnożeniu kwaternionu przez skalar, zasada przemienności jest zachowana. Kolejną cechą operacji mnożenia kwaternionów, podobnie jak w przypadku operacji dodawania, jest zachowanie zasady łączności występujących w iloczynie czynników:

$$(q_1 \cdot q_2) \cdot q_3 = q_1 \cdot (q_2 \cdot q_3) \tag{5.15}$$

Iloczyn dwóch kwaternionów daje wynik, który również jest kwaternionem, a więc składa się z części skalarnej i części wektorowej (Eberly 2010):

$$q_1 \cdot q_2 = (w_1, \vec{v}_1) \cdot (w_2, \vec{v}_2) = (w_1 w_2 - \vec{v}_1 \circ \vec{v}_2,\ w_1 \vec{v}_2 + w_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2) \tag{5.16}$$

gdzie symbol ◦ oznacza operację mnożenia skalarnego dla dwóch wektorów, natomiast symbol × oznacza operacje mnożenia wektorowego dla dwóch wektorów. Rozpisana postać mnożenia skalarnego dla dwóch wektorów $\vec{v}_1$ i $\vec{v}_2$ jest następująca:

$$\vec{v}_1 \circ \vec{v}_2 = [x_1\ y_1\ z_1]^{\mathrm{T}} \circ [x_2\ y_2\ z_2]^{\mathrm{T}} = x_1 x_2 + y_1 y_2 + z_1 z_2 \tag{5.17}$$

Uwzględniając wzory (5.16) i (5.17), ostateczna postać części skalarnej iloczynu dwóch kwaternionów jest następująca:

$$w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 \tag{5.18}$$

Podobnie, dla mnożenia wektorowego traktując części urojone *i*, *j*, *k* jako odpowiednie wersory przestrzeni trójwymiarowej otrzymamy wynik:

$$\vec{v}_1 \times \vec{v}_2 = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix} \begin{matrix} (y_1 z_2 - y_2 z_1)i + \\ = (x_2 z_1 - x_1 z_2)\, j + \\ (x_1 y_2 - x_2 y_1)k \end{matrix} \tag{5.19}$$

Ostatecznie, część wektorowa iloczynu dwóch kwaternionów, przedstawionego we wzorze (5.16), ma postać algebraiczną:

$$\begin{matrix} (w_1 x_2 + w_2 x_1 + y_1 z_2 - y_2 z_1)i + \\ (w_1 y_2 + w_2 y_1 + x_2 z_1 - x_1 z_2)\, j + \\ (w_1 z_2 + w_2 z_1 + x_1 y_2 - x_2 y_1)k \end{matrix} \tag{5.20}$$

Uwzględniając rozpisaną powyżej postać algebraiczną części skalarnej (5.18) i części wektorowej (5.20) iloczynu dwóch kwaternionów, można wzór (5.16) przedstawić następująco:

$$q_1 \cdot q_2 = (w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2,\ \begin{bmatrix} w_1 x_2 + w_2 x_1 + y_1 z_2 - y_2 z_1 \\ w_1 y_2 + w_2 y_1 + x_2 z_1 - x_1 z_2 \\ w_1 z_2 + w_2 z_1 + x_1 y_2 - x_2 y_1 \end{bmatrix}) \tag{5.21}$$

Przedstawione powyżej mnożenie dwóch kwaternionów będzie wykorzystane w rozdziale 6, przy opisie operacji rotacji kwaternionowej.

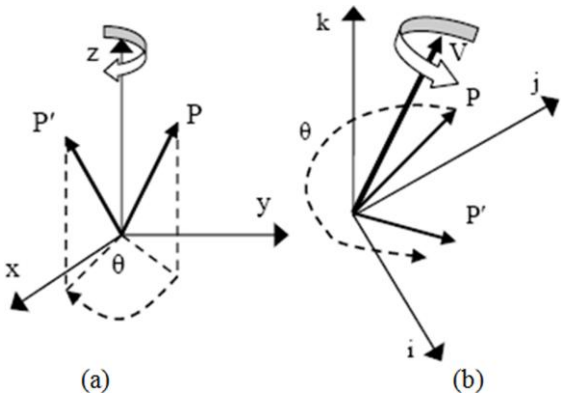
6. SZYFROWANIE KWATERNIONOWE

Rozdział 6 poświęcony jest opisaniu podstawowej formy szyfrowania kwaternionowego, wykorzystywanego w omawianym, w tej pracy, systemie kryptograficznym. Jak pokazano w pracach Nagase i in. (2004a, 2004b, 2005), szyfrowanie kwaternionowe posiada wiele zalet i stanowi rozwiązanie wzmacniające bezpieczeństwo transmisji danych w sieciach bezprzewodowych. W rozdziale omówiono dwie metody realizacji szyfrowania kwaternionowego uwzględniając wady i zalety obydwu rozwiązań.

6.1. Rotacja kwaternionowa

Istnieje wiele sposobów parametryzacji przestrzeni trójwymiarowej dla opisu obrotów obiektów. Jednym z ważniejszych i bardziej popularnych sposobów na osiągniecie tego celu są kwaterniony (Eberly 2010). Do obrotu obiektów można też wykorzystać metodę rotacji Eulera (Rys. 3a), jednakże kwaterniony, z uwagi na swoje właściwości, obecnie cieszą się znacznie większą popularnością przy opisie i realizacji obrotów obiektów w przestrzeni trójwymiarowej (Katz 1996, Kuipers 1999, Kunze i Schaeben 2004, Eberly 2010).

Przedstawienie strumienia danych w postaci strumienia wektorów, umożliwia wykorzystanie właściwości obrotu kwaternionowego w systemach kryptograficznych dla szyfrowania danych.



Rys. 3. Rotacja przy wykorzystaniu reprezentacji Eulera (a) i kwaternionu (b)
Źródło: Opracowanie własne na podstawie artykułu Nagase i in. (2005).

Rotacja kwaternionowa (Rys. 3b), w przeciwieństwie do rotacji Eulera (Rys. 3a), wykonywana jest względem wektora **V**, a nie względem poszczególnych osi przestrzeni trójwymiarowej. Zatem w przypadku rotacji kwaternionowej, aby mówić o obrocie należy dysponować dwoma kwaternionami.

Rozpatrzmy dwa kwaterniony: kwaternion $q=[w\ x\ y\ z]^T$ oraz kwaternion $P=[0\ a\ b\ c]^T$, gdzie wektor reprezentujący cześć wektorową $[a\ b\ c]^T$ kwaternionu P o zerowej części skalarnej, przechowywać będzie informację o danych, które poddane zostaną obrotowi. Jeżeli pomnożylibyśmy kwaternion P przez kwaternion q, to otrzymalibyśmy

kwaternion o niezerowej części skalarnej, co bezpośrednio wynika z zasady mnożenia dwóch kwaternionów przedstawionej w rozdziale 5.4. we wzorach (5.16) i (5.21). Oznacza to, że w takiej sytuacji wektor danych, zapisany w postaci kwaternionowej, przekształcilibyśmy w czysty kwaternion (po pominięciu części rzeczywistej kwaternionu), który w żaden sposób nie jest przestrzennym odzwierciedleniem obróconego wektora. Aby koncepcja kwaternionowego obrotu wektora $[a\ b\ c]^T$ została spełniona, to po obrocie czystego kwaternionu powinniśmy otrzymać także czysty kwaternion.

Jeżeli kwaternion P pomnożymy z jeden strony przez kwaternion q, natomiast z drugiej strony przez jego odwrotność q^{-1}, to otrzymamy wynik w postaci kwaternionu, którego część rzeczywista będzie równa zero (Eberly 2010):

$$\mathrm{Re}[q \cdot P \cdot q^{-1}] = 0 \tag{6.1}$$

W ten sposób mamy możliwość kwaternionowego odwzorowania wektora danych $[a\ b\ c]^T$ w inny wektor. Na podstawie powyższych rozważań otrzymaliśmy, że pożądany wzór na rotację kwaternionową ma postać:

$$P_{\mathrm{rot}} = q \cdot P \cdot q^{-1} \tag{6.2}$$

Wzór na rotację kwaternionową (6.2) ma zasadnicze znaczenie dla koncepcji szyfrowania kwaternionowego. Realizacja szyfrowania kwaternionowego sprowadza się do dwóch tożsamych rozwiązań. Pierwsze podejście zakłada przekształcenie wzoru (6.2) do postaci macierzowej i realizacji mnożenia macierzowego przy użyciu macierzy rotacji (Nagase i in. 2004a, 2004b, 2005). Drugie rozwiązanie to bezpośrednia realizacja wzoru (6.2) zgodnie z zasadami mnożenia kwaternionowego. Obydwie metody posiadają swoje wady i zalety.

Dodatkowo, na podstawie algorytmu zaproponowanego przez Nagase i in. (2004a, 2004b, 2005), macierz rotacji umożliwia także wyznaczenie dowolnie dużej przestrzeni kluczy dla obydwu metod szyfrowania kwaternionowego.

6.2. Macierz rotacji

By wektor danych mógł być poddany obrotowi, należy wykorzystać narzędzie, które w odpowiedni sposób przeprowadzi rotację kwaternionową. Istotną rolę odgrywa odpowiednia interpretacja właściwości kwaternionów oraz realizacja ich mnożenia. Stworzenie narzędzia pozwalającego na przeprowadzenie szyfrowania wektora danych poprzez jego obrót kwaternionowy jest wykonalne, jednakże wiąże się z dodatkowymi problemami wdrożeniowymi.

Alternatywnie można przedstawić rotację kwaternionową w postaci macierzowej. Rozwiązanie to nie wymaga wdrażania dodatkowych narzędzi obsługujących rachunek kwaternionowy i jednocześnie pozwala uzyskać identyczny efekt, jak w przypadku rotacji kwaternionowej. Dlatego też, by móc zrealizować szyfrowanie kwaternionowe metodą macierzową, należy najpierw zadbać o wyznaczenie macierzy rotacji.

Macierz rotacji można w łatwy sposób obliczyć, wykorzystując postać algebraiczną

rotacji kwaternionowej (Zhang 1997, Eberly 2010). Realizując mnożenie kwaternionowe ze wzoru (6.2) na rotację wektora danych oraz przyjmując dla uproszczenia, że obrót wektora danych wykonywany jest względem jednostkowego kwaternionu q, otrzymamy następujący kwaternion P_{rot}:

$$P_{\text{rot}} = q \,.\, P \,.\, q^{-1} = \\ = (0, \begin{bmatrix} (w^2 + x^2 - y^2 - z^2)a + (2xy - 2wz)b + (2xz + 2wy)c \\ (2wz + 2xy)a + (w^2 - x^2 + y^2 - z^2)b + (2yz - 2wx)c \\ (2xz - 2wy)a + (2yz + 2wx)b + (w^2 - x^2 - y^2 + z^2)c \end{bmatrix}) \tag{6.3}$$

Z wzoru (6.3) możemy wyznaczyć macierz rotacji. Należy zwrócić uwagę, że we wzorze (6.3), prezentującym obrócony i zapisany w postaci kwaternionu wektor danych, znajdują się elementy a, b i c, które są rzeczywistymi współczynnikami części wektorowej kwaternionu P i stanowią wektor danych. Stąd wzór (6.3) może być przekształcony do postaci macierzowej (Zhang 1997, Eberly 2010):

$$\mathbf{P}_{\text{rot}} = \mathbf{\Gamma}(q)\mathbf{P}$$

$$\mathbf{P}_{\text{rot}} = \begin{bmatrix} w^2 + x^2 - y^2 - z^2 & 2xy - 2wz & 2xz + 2wy \\ 2wz + 2xy & w^2 - x^2 + y^2 - z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & w^2 - x^2 - y^2 + z^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{6.4}$$

gdzie $\mathbf{\Gamma}(q)$ oznacza macierz rotacji, $\mathbf{P}$ oznacza wektor danych $[a\ b\ c]^{\text{T}}$, natomiast $\mathbf{P}_{\text{rot}}$ oznacza zaszyfrowany (obrócony przestrzennie) wektor danych. Macierz rotacji umożliwia przeprowadzenie szyfrowania kwaternionowego poprzez zastosowanie mnożenia macierzowego.

Algorytm wykorzystujący macierz rotacji do wygenerowania dowolnie dużej przestrzeni kluczy został przedstawiony w rozdziale 6.5.

6.3. Szyfrowanie kwaternionowe metodą macierzową

Przedstawiona wcześniej postać macierzy rotacji we wzorze (6.4) pokazuje, że do zaszyfrowania wektora danych wystarczy nam realizacja mnożenia macierzowego. Ponieważ macierz rotacji jest wymiaru 3×3, zatem można tak rozszerzyć wektor danych, aby zoptymalizować mnożenie macierzowe. Zamiast wektora danych $\mathbf{P}$ o wielkości 3×1 wprowadźmy macierz danych $\mathbf{B}$ o wielkości 3×3. W taki sposób, nie tylko zachowana zostanie koncepcja szyfrowania kwaternionowego, ale również lepiej wykorzystamy mnożenie macierzowe, jednorazowo szyfrując większą porcję danych.

$$\mathbf{P} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \rightarrow \quad \mathbf{B} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \tag{6.5}$$

Dla tak rozszerzonej macierzy danych **B**, proces szyfrowania (6.6) i deszyfrowania (6.7) kwaternionowego zapisuje się następująco (Nagase i in. 2004a, 2004b, 2005):

$$\mathbf{B}_{\text{rot}} = \mathbf{\Gamma}(q)\mathbf{B} \tag{6.6}$$

$$\mathbf{B} = \mathbf{\Gamma}(q)^{-1}\mathbf{B}_{\text{rot}} \tag{6.7}$$

gdzie $\mathbf{B}_{\text{rot}}$ oznacza zaszyfrowaną macierz danych rozmiaru 3×3. Postać macierzy rotacji $\mathbf{\Gamma}(q)$ jest taka jak przedstawiona we wzorze (6.4).

Szyfrując dane metodą macierzową jesteśmy ograniczeni wielkością macierzy danych – maksymalnie 3×3. Uwarunkowane jest to postacią macierzy rotacji, która jest zawsze rozmiaru 3×3. Przykładowo, szyfrując wartości pikseli trójkanałowego obrazu kolorowego RGB otrzymamy:

$$[\mathbf{B}_{\text{rot}}]_{3x3} = [\mathbf{\Gamma}(q)]_{3x3} \cdot \begin{bmatrix} [\quad red \quad]_{1x3} \\ [\quad green \quad]_{1x3} \\ [\quad blue \quad]_{1x3} \end{bmatrix}_{3x3} \tag{6.8}$$

Podsumowując, metoda macierzowa jest łatwa do zaimplementowania, jednakże pozostaje mniej wydajna niż metoda wykorzystująca mnożenie kwaternionowe dla realizacji szyfrowania.

6.4. Szyfrowanie kwaternionowe metodą mnożenia kwaternionowego

Alternatywną metodą realizacji szyfrowania kwaternionowego jest metoda bezpośrednio realizująca wzór na rotację kwaternionową (6.2). W tym przypadku jednak należy liczyć się z wdrożeniem odpowiedniego narzędzia, które umożliwi przeprowadzanie operacji mnożenia kwaternionowego. Narzędzie wykorzystywane w tej pracy zastosowano dla środowiska *Matlab* (MathWorks 1994-2016).

Podobnie jak w przypadku metody macierzowej (opisanej w rozdziale 6.3.), tutaj również dokonać można optymalizacji szyfrowania. Ponieważ metoda mnożenia kwaternionowego bazuje bezpośrednio na wzorze (6.2), zatem zamiast szyfrować pojedynczy wektor danych $[a\ b\ c]^{\text{T}}$, umieszczony w części wektorowej kwaternionu P, zastosujemy szyfrowanie macierzy danych wymiaru 3×3 przechowywanej w części wektorowej rozszerzonego kwaternionu B:

$$P = (0, \begin{bmatrix} a \\ b \\ c \end{bmatrix}) \quad \rightarrow \quad B = (0, \begin{bmatrix} [a_1 & a_2 & a_3] \\ [b_1 & b_2 & b_3] \\ [c_1 & c_2 & c_3] \end{bmatrix}) \tag{6.9}$$

Dla tak rozszerzonego kwaternionu danych B, proces szyfrowania (6.10) i deszyfrowania (6.11) kwaternionowego zapisuje się następująco:

$$B_{\text{rot}} = q \cdot B \cdot q^{-1} \tag{6.10}$$

$$B = q^{-1} \cdot B_{\text{rot}} \cdot q \tag{6.11}$$

gdzie B_{rot} oznacza kwaternion danych po rotacji (zaszyfrowany), którego część wektorowa jest rozmiaru 3×3. Kwaternion q, jest kwaternionem-kluczem dookoła którego przeprowadzany jest obrót kwaternionu danych B.

Szyfrując dane metodą mnożenia kwaternionowego, nie jesteśmy w żaden sposób ograniczeni wielkością części wektorowej kwaternionu danych B. Przykładowo chcąc zaszyfrować wartości pikseli trójkanałowego obrazu kolorowego RGB możemy część wektorową kwaternionu danych B rozszerzyć następująco:

$$B_{\text{rot}} = q \cdot (0, \begin{bmatrix} [& red &]_{3x3} \\ [& green &]_{3x3} \\ [& blue &]_{3x3} \end{bmatrix}) \cdot q^{-1} \tag{6.12}$$

Szyfrowanie metodą mnożenia kwaternionowego jest bardziej wydajne niż szyfrowanie metodą macierzową. Operacje kwaternionowe wykonywane są szybciej niż analogicznie zapisane operacje macierzowe (Goldman 2009, 2011). Dodatkowo, dla metody mnożenia kwaternionowego, istnieje możliwość zapisania części wektorowej kwaternionu danych B o dowolnych rozmiarach. W przypadku metody macierzowej jesteśmy ograniczeni zawsze do macierzy danych **B**, o rozmiarze maksymalnie 3×3.

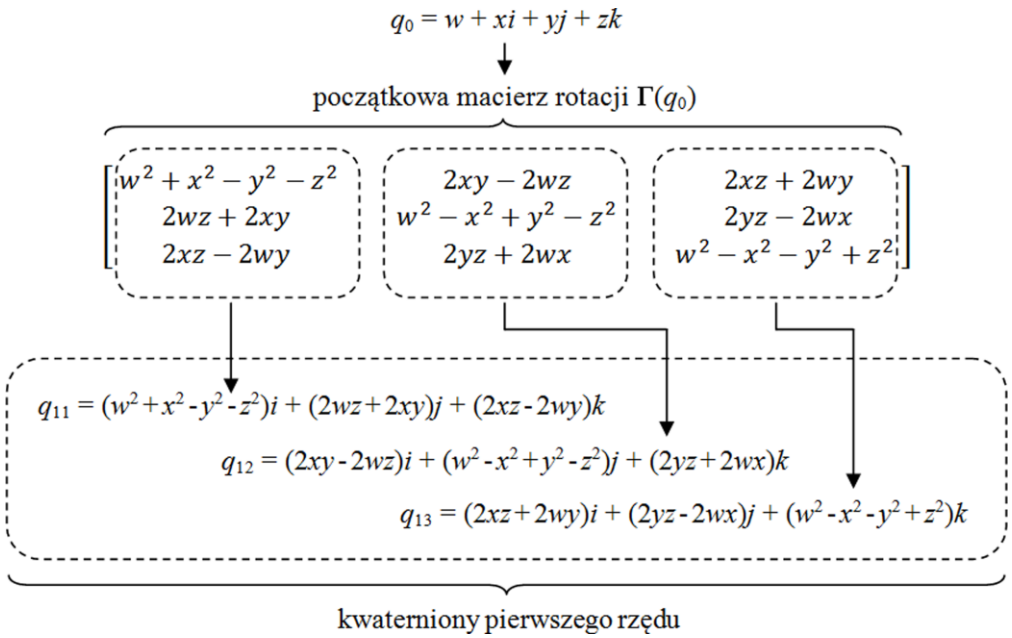
Podsumowując, metoda mnożenia kwaternionowego umożliwia bardzo wydajne szyfrowanie dużych porcji danych i jest to metoda szyfrowania, która została wybrana na potrzeby zaprojektowania omawianego w tej pracy kwaternionowego systemu kryptograficznego.

6.5. Algorytm generacji kluczy szyfrujących

Kwaternion q i macierz rotacji $\mathbf{\Gamma}(q)$ z niego wyznaczona, wykorzystywane są jako klucze dla szyfrowania kwaternionowego, w zależności od przyjętej metody szyfrowania. W przypadku metody macierzowej kluczem jest macierz rotacji $\mathbf{\Gamma}(q)$, natomiast dla metody mnożenia kwaternionowego kluczem jest kwaternion q.

Przedstawiona w rozdziale 6.2. macierz rotacji określana jest mianem macierzy początkowej $\mathbf{\Gamma}(q_0)$, wyznaczonej na podstawie kwaternionu początkowego q_0. Na podstawie znajomości $\mathbf{\Gamma}(q_0)$ można utworzyć kolejne kwaterniony q_i nazywane kwaternionami wyższych rzędów, a z nich odpowiednie macierze rotacji $\mathbf{\Gamma}(q_i)$ wyższych rzędów.

Wykazano (Nagase i in. 2004a), że do przeprowadzenia obrotu ciągu danych dużo korzystniej jest użyć nie jednej, lecz kilku różnych rotacji kwaternionowych. Algorytm, który pozwala na pozyskiwanie kolejnych kwaternionów q_i (i w konsekwencji odpowiednich macierzy rotacji $\mathbf{\Gamma}(q_i)$) z macierzy utworzonej wcześniej został przedstawiony na Rys. 4.



Rys. 4. Uzyskanie kwaternionów pierwszego rzędu z początkowej macierzy rotacji.
Źródło: Opracowanie własne.

Wyznaczoną, początkową macierz rotacji określa się mianem macierzy rzędu zerowego. Aby wyznaczyć kolejne macierze rotacji wyższych rzędów, należy zgrupować elementy macierzy rzędu niższego w taki sposób, aby trzy elementy każdej kolumny tej macierzy stanowiły współczynniki dla utworzenia nowego kwaternionu. Dla uproszczenia, przyjmuje się zerową część skalarną kwaternionów. Z tak uzyskanych kwaternionów wyznacza się kolejne macierze rotacji postępując analogicznie jak dla rzędu zerowego, aż do uzyskania wymaganej przestrzeni kluczy.

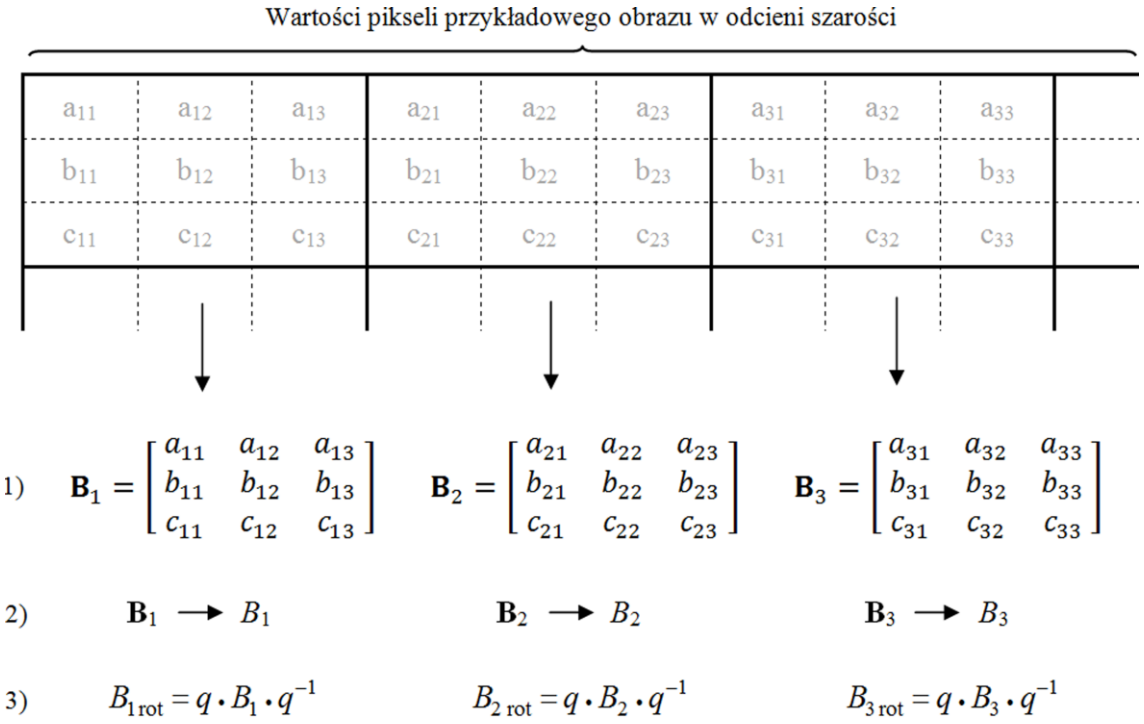
Jeżeli przez n oznaczymy wielkość zastosowanego rzędu, to uzyskana liczba kwaternionów q lub liczba macierzy rotacji jest równa 3^n.

Przykładowo, dla rzędu drugiego uzyskamy 9 kwaternionów kluczy (q_{21}, q_{22}, q_{23}, q_{24}, q_{25}, q_{26}, q_{27}, q_{28}, q_{29}), a dla rzędu trzeciego 27 kwaternionów kluczy (q_{31}, q_{32}, q_{33}, …, q_{327}). Jednakże, im wyższy rząd wybierzemy, tym więcej czasu będzie potrzebne na uzyskanie końcowych kwaternionów kluczy, wykorzystywanych do szyfrowania danych.

6.6. Szyfrowanie przykładowego obrazu

Szyfrowanie kwaternionowe przykładowego obrazu w odcieniach szarości, wykorzystujące metodę mnożenia kwaternionowego, zostało przedstawione na Rys. 5. Przykładowy obraz, którego piksele przyjmują wartości od 0 do 255, jest dzielony na odpowiednią liczbę mniejszych fragmentów. Każdy fragment zapisywany jest w postaci macierzy **B**. W przykładzie widocznym na Rys. 5. (dla uproszczenia) są to macierze rozmiaru 3×3. Ponieważ szyfrowanie ma być przeprowadzone metodą mnożenia kwaternionowego, zatem wartości przechowywane w macierzach **B** należy zapisać w części wektorowej odpowiednich kwaternionów *B*. Następnie, każdy kwaternion *B*

należy zaszyfrować wykorzystując wzór na rotację kwaternionową (6.10).



Rys. 5. Szyfrowanie kwaternionowe przykładowego obrazu w odcieniach szarości przy wykorzystaniu metody mnożenia kwaternionowego.
Źródło: Opracowanie własne.

Wykorzystując algorytm generacji kluczy szyfrujących podany w rozdziale 6.5., możemy zapewnić różnorodność kwaternionów kluczy q przy szyfrowaniu każdego kolejnego kwaternionu danych B.

7. PRZEGLĄD LITERATURY NA TEMAT SZYFROWANIA KWATERNIONOWEGO

Kwaterniony, z uwagi na swoje unikatowe właściwości dla przeprowadzania rotacji w przestrzeni trójwymiarowej i prosty zapis macierzowy, zyskały dużą popularność opisując mechanikę obrotu w przestrzeni 3-wymiarowej dla różnych dziedzin nauki. Współcześnie wykorzystuje się je nie tylko w matematyce, automatyce i robotyce czy grafice komputerowej (Kuipers 1999, Marins i in. 2001), ale także mogą być wykorzystane w kryptografii.

Szyfrowanie kwaternionowe po raz pierwszy zaproponowane zostało przez Tomoyuki Nagase w pracach (Nagase i in. 2004a, 2004b, 2005). Przedstawiony tam symetryczny, kwaternionowy algorytm szyfrujący QES (ang. *Quaternion Encryption Scheme*) został zaimplementowany w trybie elektronicznej książki kodowej ECB (ang. *Electronic Code Book*). W pracach tych pokazano m.in. realizację QES dla metody macierzowej oraz algorytm generacji nieskończonej przestrzeni kluczy na podstawie początkowej macierzy rotacji. Zaproponowana metoda szyfrowania wykorzystuje macierze danych rozmiaru 3×3, co w przypadku szyfrowania dużych obrazów znacząco wydłuża czas przetwarzania. Zaproponowana przez autorów implementacja trybu ECB, gdzie kolejne szyfrowane bloki danych nie są ze sobą powiązane, zapewnia zwiększoną wydajność szyfrowania i deszyfracji niż w przypadku innych trybów. Jednakże, tryb ECB wpływa niekorzystnie na poziom proponowanego bezpieczeństwa. Algorytm QES jest szczególnym przypadkiem szyfru Hilla (Stallings 2006), który jest podatny na atak ze znanym tekstem jawnym (Gupta i in. 2007), stąd też QES nie może być uznany za rozwiązanie bezpieczne.

Rozwiązanie naprawiające podatność QES na atak ze znanym tekstem jawnym zostało zaproponowane przez Evgueni Doukhnitch (Doukhnitch i in. 2013). Przedstawiona tam modyfikacja dla QES, określana mianem M-QES, skupia się na zmniejszeniu wielkości macierzy danych do rozmiaru wektora danych 1×3 i zmianie procedury generacji kluczy, tak by kolejne wygenerowane kwaterniony klucze wyższych rzędów posiadały wszystkie 4 współczynniki (część skalarna kwaternionów kluczy wyższych rzędów jest różna od 0). M-QES zakłada wielokrotne mnożenie macierzowe pojedynczego wektora danych 1×3 przez różne macierze rotacji (wyznaczane na podstawie zmodyfikowanego algorytmu generacji kluczy).

Dodatkowo, w pracy (Doukhnitch i in. 2013) zaproponowana została implementacja sprzętowa zmodyfikowanego QES, określana mianem HW-QES, wykonana na podstawie pracy (Doukhnitch i Ozen 2011, Hsiao i Delosme 1994). Przedstawiona tam implementacja pozwala na realizację M-QES bez wykorzystania operacji mnożenia, wykonywane są jedynie operacje dodawania i przesuwania.

Obydwa rozwiązania M-QES i HW-QES, wykazują odporność na atak ze znanym tekstem jawnym, jednakże pozostają mniej wydajne niż zwykły algorytm QES implementowany w trybie ECB, z wykorzystaniem metody macierzowej. HW-QES jest w przybliżeniu 2 razy wydajniejszy niż M-QES i około 10% mniej wydajny niż zwykły algorytm QES. Należy pamiętać, że przy szyfrowaniu dużych pakietów danych np. trójkanałowych obrazów kolorowych, metoda macierzowa wykorzystywana zarówno w QES, M-QES czy HW-QES nie jest optymalna - pozwalając na jednorazowe

szyfrowanie 9 próbek danych w przypadku QES i 3 próbek danych w przypadku M-QES i HW-QES.

Inne podejście dla implementacji algorytmu QES zostało przedstawione w pracach (Dzwonkowski i Rykaczewski 2012, 2013, 2014). Proponowane tam, zmodyfikowane algorytmy QES zostały zaimplementowane z wykorzystaniem metody mnożenia kwaternionowego. W pracy (Dzwonkowski i Rykaczewski 2012) zastosowano modyfikację QES zaimplementowaną w trybie wiązania bloków CBC (ang. *Cipher Block Chaining*). W pracy (Dzwonkowski i Rykaczewski 2013) wykorzystano tryb licznikowy CTR (ang. *Counter*) natomiast w pracy (Dzwonkowski i Rykaczewski 2014) przedstawiono algorytm oparty na sieci Feistela z wykorzystaniem arytmetyki modularnej. Zastosowanie metody mnożenia kwaternionowego umożliwia osiągnięcie dużo bardziej wydajnego systemu kryptograficznego niż w przypadku zastosowania metody macierzowej. Operacje mnożenia kwaternionowego wykonywane są szybciej niż analogicznie zapisane operacje macierzowe (Goldman 2009, 2011). Dodatkowo, możliwość jednorazowego szyfrowania dużych porcji danych (współczynniki wektorowe kwaternionu danych mogą mieć dowolną wielkość) sprawia, że algorytm QES, implementowany dla metody mnożenia kwaternionowego, jest idealnym narzędziem do szybkiego zabezpieczania danych multimedialnych. Z tego też powodu, rozszerzona wersja algorytmu zaproponowanego w (Dzwonkowski i Rykaczewski 2014) jest głównym tematem niniejszej rozprawy.

Oprócz wspomnianego wcześniej algorytmu QES, istnieją także inne metody pozwalające na zaszyfrowanie obrazów z wykorzystaniem kwaternionów. Jedną z nich jest metoda wykorzystująca podwójne, losowo-fazowe szyfrowanie oparte na dyskretnej kwaternionowej transformacie Fouriera DQFT (ang. *Discrete Quaternion Fourier Transform*) zaproponowana przez Xiaolei Wang w pracy (Wang i in. 2011). Reprezentacja kwaternionowa została tu użyta w celu jednoczesnej transformacji wszystkich trzech kanałów obrazu kolorowego. Wadą proponowanego rozwiązania jest relatywnie mała przestrzeń kluczy szyfrujących oraz fakt, że odszyfrowany obraz różni się w pewnym stopniu od oryginału. Dodatkowo, proponowana metoda nie została sprawdzona pod kątem wydajnościowym, zatem trudno porównywać ją z algorytmem QES.

Kolejna metoda, skupiająca się na szyfrowaniu obrazów kolorowych, zaproponowana przez Zhuhong Shao w pracy (Shao i in. 2013), wykorzystuje kwaternionową transformację żyratorową QGT (ang. *Quaternion Gyrator Transform*). Proponowana metoda, podobnie jak w przypadku metody DQFT, wykorzystuje małą przestrzeń kluczy oraz nie pozwala na odzyskanie obrazu identycznego z oryginałem po deszyfracji. Wydajność metody QGT również nie została zbadana.

Szyfrowanie kwaternionowe zostało także zaproponowane w pracy (Narayan i Ibrahim 2013), jednakże proponowana tam metoda jest podobna do algorytmu M-QES. Omawiany algorytm realizuje metodę macierzową dla QES, przy czym generacja kwaternionów kluczy (z których wyznaczane są macierze rotacji) jest przeprowadzana z wykorzystaniem ciągów Fareya. Macierz danych szyfrowana jest kilkukrotnie z udziałem różnych macierzy rotacji.

Algebra kwaternionowa znalazła również zastosowanie w szyfrowaniu asymetrycznym, poprzez modyfikację probabilistycznego systemu kryptograficznego NTRU. Główną zaletą asymetrycznego algorytmu NTRU, w zestawieniu z popularnym

algorytmem asymetrycznym RSA, jest duża szybkość przetwarzania i odporność na ataki kwantowe (Hermans i in. 2011). Wadą jest możliwość uzyskania błędnie odszyfrowanych danych. Prawdopodobieństwo błędnego dekryptażu oszacowano na ok. 2^{-80} (Hoffstein i in. 2008). Zmodyfikowaną wersją NTRU jest QTRU (Malekian i in. 2009), gdzie zastosowano operacje kwaternionowe. Zaproponowana modyfikacja dziedziczy właściwości probabilistyczne rozwiązania NTRU i jest od niego ok. 4 razy wolniejsza (Malekian i in. 2009). QTRU wykazuje odporność na zaproponowany w pracy (Coppersmith i Shamir 1997) atak kratowy (ang. *lattice attack*). QTRU może z powodzeniem zastąpić algorytm RSA wykorzystywany do dystrybucji kluczy dla algorytmów symetrycznych.

Ponieważ w niniejszej pracy skupiono się na wydajnym zabezpieczaniu danych multimedialnych i medycznych, kwaternionowa modyfikacja algorytmów asymetrycznych (dużo wolniejszych od algorytmów symetrycznych (Schneier 1996)), nie jest rozwiązaniem optymalnym i algorytmy te nie były przedmiotem zainteresowania w rozprawie. Zaproponowany w kolejnym rozdziale kwaternionowy system kryptograficzny oparty jest na kryptografii symetrycznej.

8. KWATERNIONOWY SYSTEM KRYPTOGRAFICZNY

Kwaternionowy system kryptograficzny przedstawiony w tej pracy oparty jest na metodzie mnożenia kwaternionowego (rozdział 6.4.). Wykorzystywana metoda szyfrowania kwaternionowego jest nowa i oprócz publikacji autora niniejszej rozprawy (rozdział 7.), nie została jeszcze zastosowana w dziedzinie kryptografii.

W przeciwieństwie do metody macierzowej przedstawionej w pracach (Nagase i in. 2004a, 2004b, 2005, Doukhnitch i in. 2013), metoda mnożenia kwaternionowego nie jest ograniczona rozmiarem danych wejściowych. Możliwe jest szyfrowanie dowolnie dużych bloków danych (np. obrazów wejściowych pełnej wielkości, bez ich uprzedniego podziału na mniejsze fragmenty) lub też zastosowanie podziału danych wraz z implementacją dodatkowych trybów szyfrowania. Ponieważ operacje kwaternionowe wykonywane są szybciej niż analogicznie zapisane operacje macierzowe (Goldman 2009, 2011), implementacja trybów szyfrowania (takich jak CBC czy CTR) nie wpływa w znaczący sposób na spadek wydajności systemu. Dlatego też, system kryptograficzny zbudowany z wykorzystaniem metody mnożenia kwaternionowego jest atrakcyjnym rozwiązaniem dla wydajnego zabezpieczania danych multimedialnych i medycznych.

8.1. Tryby szyfrowania

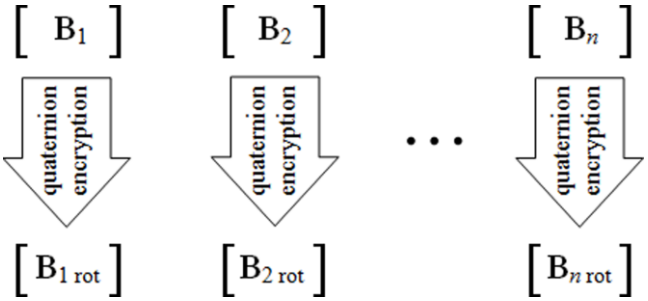
Przy stosowaniu symetrycznych szyfrów blokowych należy liczyć się z niebezpieczeństwem wykorzystania tych samych bitów klucza tajnego do zaszyfrowania takich samych fragmentów tekstu jawnego. Zastosowanie pojedynczego algorytmu deterministycznego dla pewnej liczby identycznych danych wejściowych daje w efekcie zawsze takie same dane wyjściowe.

Jest to bardzo niekorzystna sytuacja dla użytkowników szyfru. Ewentualny napastnik, nawet przy braku znajomości tekstu jawnego, jest w stanie uzyskać wiele informacji znając rozkład powtarzających się fragmentów zaszyfrowanej wiadomości.

Istnieją metody mieszające i uzależniające fragmenty tekstu jawnego z fragmentami tworzonego szyfrogramu, w taki sposób, aby uniemożliwić powstawanie takich samych bloków wyjściowych. Efekt ten osiągany jest dzięki zastosowaniu trybów działania szyfrów blokowych.

8.1.1. Tryb elektronicznej książki kodowej ECB

Tryb ECB to najprostszy sposób szyfrowania. Każdy z bloków wiadomości jest szyfrowany i deszyfrowany niezależnie przy użyciu tego samego klucza. Szyfrowanie kwaternionowe, zaimplementowane w tym trybie, cechuje niski poziom bezpieczeństwa lecz wysoka wydajność, z uwagi na możliwość zrównoleglenia operacji wykonywanych na kolejnych blokach.



Rys. 6. Ogólny schemat szyfrowania kwaternionowego w trybie elektronicznej książki kodowej ECB.
Źródło: Opracowanie własne.

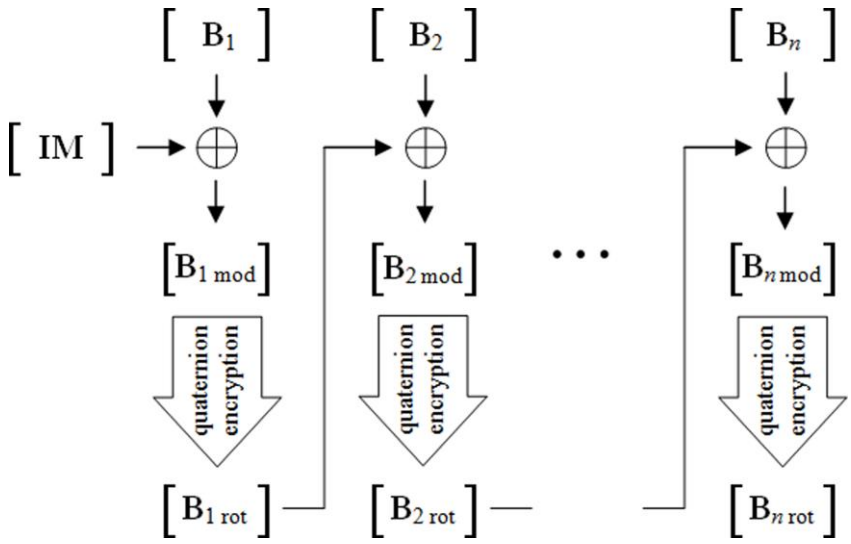
Rys. 6. przedstawia ogólny schemat szyfrowania kwaternionowego w trybie ECB, gdzie $\mathbf{B}_n$ oznacza kolejne bloki danych (np. wartości pikseli obrazu wejściowego). Bloki danych zapisywane są w części wektorowej odpowiadających im kwaternionów B_n, które biorą udział w szyfrowaniu kwaternionowym metodą mnożenia kwaternionowego. W konsekwencji otrzymujemy obrócone (zaszyfrowane) kwaterniony $B_{n\,\mathrm{rot}}$, których części wektorowe, dla uproszczenia, można również zapisać w postaci bloków danych $\mathbf{B}_{n\,\mathrm{rot}}$. Omawiany tryb szyfrowania implementowany jest z wykorzystaniem arytmetyki modularnej (rozdział 8.2.), co pozwala na uzyskania wartości całkowitoliczbowych dla bloków danych $\mathbf{B}_{n\,\mathrm{rot}}$.

8.1.2. Tryb wiązania bloków CBC

W trybie wiązania bloków CBC, tekst jawny jest podzielony na n bloków danych. Szyfrowanie polega na dodawaniu XOR każdego kolejnego bloku tekstu jawnego do poprzednio otrzymanego bloku szyfrogramu. Wynikowy blok (po operacji XOR) jest następnie szyfrowany wybranym algorytmem. W trybie CBC, każdy kolejny blok szyfrogramu jest zależny od bloku poprzedniego. Dla pierwszego bloku tekstu jawnego należy zastosować wektor bądź macierz inicjalizującą o długości równej długości pojedynczego bloku tekstu jawnego.

Szyfrowanie w trybie CBC cechuje wyższy poziom bezpieczeństwa niż szyfrowanie w trybie ECB. Niestety, występująca zależność bloków uniemożliwia zastosowanie zrównoleglenia operacji podczas szyfrowania, co czyni tryb CBC mniej wydajnym. Mimo tej wady, jest to bardzo popularny sposób przeprowadzania szyfrowania i często wykorzystywany we współczesnych algorytmach szyfrujących.

Rys. 7. ilustruje klasyczny schemat szyfrowania kwaternionowego w trybie CBC. Bloki danych $\mathbf{B}_n$ są poelementowo sumowane modulo 2 z poprzednio uzyskanym blokiem szyfrogramu $\mathbf{B}_{n\text{-}1\,\mathrm{rot}}$ (dla pierwszego bloku wykorzystywana jest macierz inicjalizująca **IM**). W wyniku tej operacji otrzymujemy wartości całkowitoliczbowe, zapisane w bloku $\mathbf{B}_{n\,\mathrm{mod}}$. Następnie, wartości z bloku $\mathbf{B}_{n\,\mathrm{mod}}$ zapisywane są w postaci kwaternionu $B_{n\,\mathrm{mod}}$ i poddawane rotacji kwaternionowej. W wyniku otrzymywany jest zaszyfrowany (obrócony) kwaternion $B_{n\,\mathrm{rot}}$, którego część wektorowa stanowi blok szyfrogramu $\mathbf{B}_{n\,\mathrm{rot}}$.



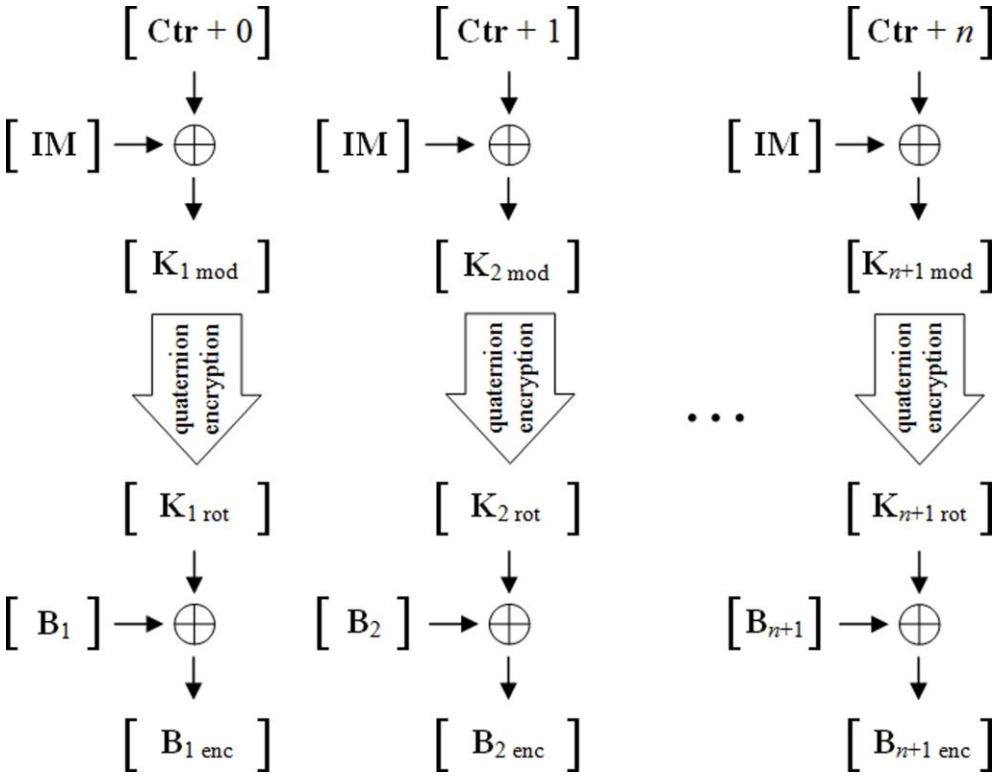
Rys. 7. Ogólny schemat szyfrowania kwaternionowego w trybie wiązania bloków CBC.
Źródło: Opracowanie własne.

Deszyfracja otrzymanego szyfrogramu, polega na rotacji kwaternionowej wykonanej w przeciwnym kierunku oraz poelementowym sumowaniu modulo 2 tak otrzymanych danych z poprzednimi blokami szyfrogramu. Ponieważ odbiorca dysponuje wszystkimi blokami szyfrogramu w momencie otrzymania całej zaszyfrowanej wiadomości, proces deszyfracji może odbywać się z wykorzystaniem wielu wątków równocześnie.

8.1.3. Tryb licznikowy CTR

W trybie licznikowym CTR, w przeciwieństwie do trybu CBC, nie występuje zależność między kolejno szyfrowanymi blokami danych. Szyfrowaniu poddaje się kolejne wartości stale zwiększającego się licznika, zsumowane z dodatkową wartością tzw. nonce. Nonce pełni tu podobną rolę jak wektor (macierz) inicjujący w trybie CBC. Jest to jeden z najpopularniejszych trybów działania szyfrów blokowych. Zarówno szyfrowanie jak i deszyfrowanie może odbywać się z wykorzystaniem wielu wątków równocześnie.

Na Rys. 8. przedstawiono ogólny schemat szyfrowania kwaternionowego w trybie CTR. Blok licznika **Ctr** jest sekwencyjnie inkrementowany i sumowany modulo 256 zawsze z tym samym blokiem inicjalizującym **IM**. Operacja sumowania modulo 256 w tym przypadku traktowana jest jako funkcja jednokierunkowa. W konsekwencji otrzymywany jest blok klucza $\mathbf{K}_{n\,\mathrm{mod}}$ o wartościach całkowitoliczbowych z zakresu od 0 do 255. Blok ten zapisywany jest w części wektorowej kwaternionu $K_{n\,\mathrm{mod}}$ i poddawany rotacji kwaternionowej. Otrzymywany kwaternion $K_{n\,\mathrm{rot}}$ zapisywany jest w postaci bloku $\mathbf{K}_{n\,\mathrm{rot}}$ i poelementowo sumowany modulo 2 z blokiem danych $\mathbf{B}_n$. W wyniku tej operacji otrzymywany jest zaszyfrowany blok danych $\mathbf{B}_{n\,\mathrm{enc}}$.



Rys. 8. Ogólny schemat szyfrowania kwaternionowego w trybie licznikowym CTR.
Źródło: Opracowanie własne.

Dodatkowo, w trybie CTR uszkodzenie jednego bitu tekstu jawnego lub szyfrogramu powoduje uszkodzenie odpowiadającego mu jednego zaszyfrowanego lub zdeszyfrowanego bitu. Pozwala to na zastosowanie dodatkowych mechanizmów minimalizujących liczbę błędów w transmisji.

8.2. Arytmetyka modularna

Zastosowanie szyfrowania kwaternionowego (realizowanego zarówno metodą macierzową (6.6) jak i metodą mnożenia kwaternionowego (6.10)) wiąże się z uzyskaniem wartości zmiennoprzecinkowych w wynikowej macierzy danych bądź w części wektorowej wynikowego kwaternionu. Aby zachować wartości całkowitoliczbowe po szyfrowaniu kwaternionowym niezbędne jest zastosowanie arytmetyki modularnej przy wykorzystaniu liczb Lipschitza (kwaternionów o całkowitych współczynnikach) (Conway i Smith 2004).

Jak widać na Rys. 7. i Rys. 8. uzyskanie wartości całkowitoliczbowych zaraz po szyfrowaniu kwaternionowym bardzo ułatwia przeprowadzenie dalszych operacji XOR. W przypadku szyfrowania obrazów, można stosować zapis 8-bitowy dla otrzymywanych wartości. Z tego też powodu nie ma konieczności wykorzystywania zmiennoprzecinkowej reprezentacji binarnej (Kahan 1997) podczas sumowania modulo 2 oraz dodatkowych mechanizmów obsługujących pojawiające się wyjątki (Kahan 1997).

Implementacja arytmetyki modularnej wiąże się m.in. z wyznaczeniem modularnej odwrotności dowolnej liczby całkowitej, w tym także kwaternionu Lipschitza. Kwaternion odwrotny modularnie jest niezbędny z uwagi na zasadę szyfrowania metodą mnożenia kwaternionowego (6.10). Ponieważ omawiany algorytm ma być przystosowany głównie do zabezpieczania danych multimedialnych (m.in. trójkanałowe obrazy kolorowe RGB), zatem rozpatrywanym zakresem wartości całkowitoliczbowych jest przedział od 0 do 255. W tym przypadku, aby wyznaczyć odwrotność modularną dla zadanej przestrzeni liczb należy wykorzystać rozszerzony algorytm Euklidesa (Christensen 2005).

Algorytm obliczania odwrotności zakłada wybór odpowiedniego dzielnika (będącego liczbą pierwszą), tak aby największy wspólny dzielnik NWD, dla wybranego dzielnika i wszystkich liczb całkowitych z zadanego zakresu (w naszym przypadku są to wartości od 0 do 255), był równy 1 (Christensen 2005).

Z tego też powodu, dla danych multimedialnych dzielnikiem nie może być liczba 256, gdyż wartość ta nie jest liczbą pierwszą ($256 = 2^8$), co uniemożliwi obliczenie modularnej odwrotności w większości przypadków. Z tego powodu, w tej pracy, jako moduł wybrana została liczba pierwsza 257, co spowodowało uzyskanie wartości z zakresu od 0 do 256 zaraz po szyfrowaniu kwaternionowym.

Realizacja modularnego szyfrowania kwaternionowego metodą mnożenia kwaternionowego, sprowadza się do wyznaczenia modularnie odwrotnego kwaternionu q^{-1} (6.10). Aby tego dokonać, zgodnie z zapisem (5.7), należy najpierw wyznaczyć modularną odwrotność normy kwaternionu q podniesionej do kwadratu $||q||^2$, a następnie modularnie wymnożyć otrzymany wynik przez sprzężony kwaternion q^*. Po uzyskaniu modularnie odwrotnego kwaternionu q należy wykonać modularne mnożenie kwaternionów według wzoru (6.10). W konsekwencji, z uwagi na użyty dzielnik 257, otrzymany po szyfrowaniu kwaternion przechowywać będzie w swojej części wektorowej wartości z zakresu od 0 do 256.

Przyglądając się uważnie schematom szyfrowania (Rys.7. i Rys. 8.), można zauważyć, że uzyskanie nadmiarowej wartości 256 nie wpłynie negatywnie na proces deszyfracji. Podczas operacji poelementowego sumowania binarnego XOR, wartość 256 zapisywana jest również na 8 bitach i traktowana jako 0. Deszyfracja, we wszystkich przedstawionych trybach i z tak zaimplementowaną arytmetyką modularną, zawsze zwraca wartości identyczne z wartościami oryginalnymi. Odszyfrowany obraz będzie zawsze posiadać poprawne wartości pikseli z zakresu od 0 do 255.

8.3. Algorytm zmodyfikowanej sieci Feistela

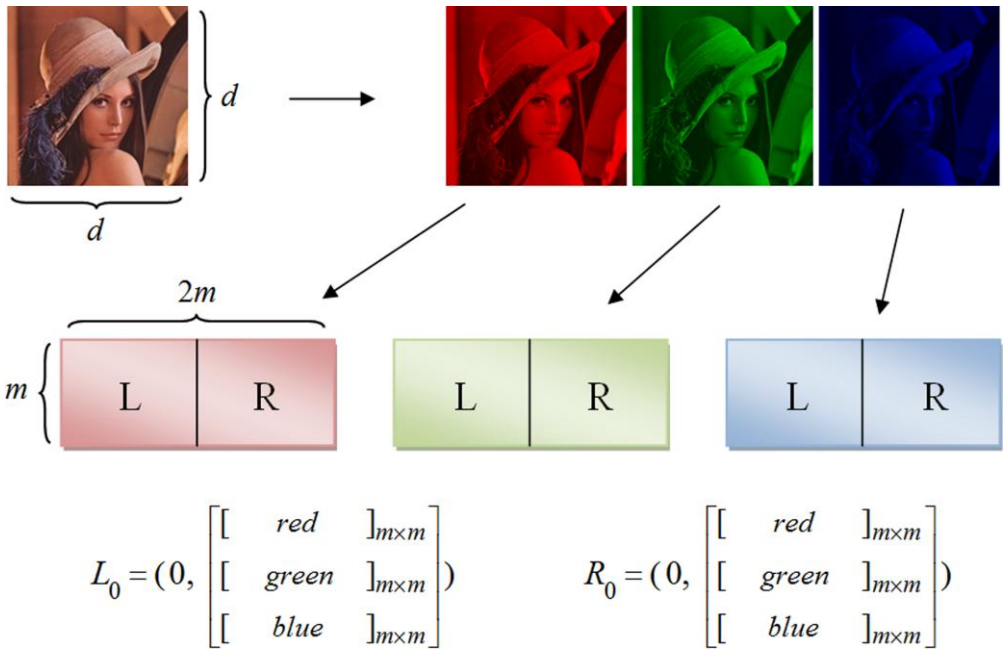
Opisany w tym rozdziale algorytm, który jest głównym osiągnięciem niniejszej rozprawy, może być wykorzystywany do szyfrowania zarówno trójkanałowych obrazów kolorowych RGB, obrazów w odcieni szarości jak również danych tekstowych. Podobny algorytm został opisany w pracy (Sastry i Kumar 2012), gdzie również wykorzystano sieć Feistela, ale proces szyfrowania i deszyfracji oparty był tylko na prostym mnożeniu macierzowym.

Rozważmy przykład przedstawiony na Rys. 9, który ilustruje sposób przygotowania danych wejściowych dla proponowanego algorytmu. Dysponujemy trójkanałowym

obrazem kolorowym RGB o rozmiarze $d \times d$ (dane wejściowe **B** mogą mieć dowolny rozmiar, niekoniecznie symetryczny). Chcąc zaszyfrować obraz należy najpierw wartości pikseli (0-255) z każdego kanału zapisać jako 3 macierze o tym samym rozmiarze $d \times d$ co obraz oryginalny. Następnie, macierze te przekształcamy w 3 nowe macierze o rozmiarze $m \times 2m$ każda. Wymiar m ustalany jest następująco:

$$m = \left\lceil \sqrt{\frac{d \cdot d}{2}} \right\rceil \tag{8.1}$$

Nadmiarowe elementy w macierzach o wymiarach $m \times 2m$ uzupełniane są losowymi liczbami całkowitymi z zakresu od 0 do 255. Otrzymane macierze ($m \times 2m$) są następnie dzielone na dwie macierze kwadratowe ($m \times m$) - odpowiednio na lewą **L** i prawą **R**. Uzyskane macierze kwadratowe zapisywane są jako współczynniki części wektorowej dwóch kwaternionów L_0 i R_0 (Rys. 9.).



Rys. 9. Zapisanie obrazu wejściowego w postaci dwóch kwaternionów L_0 i R_0.
Źródło: Opracowanie własne.

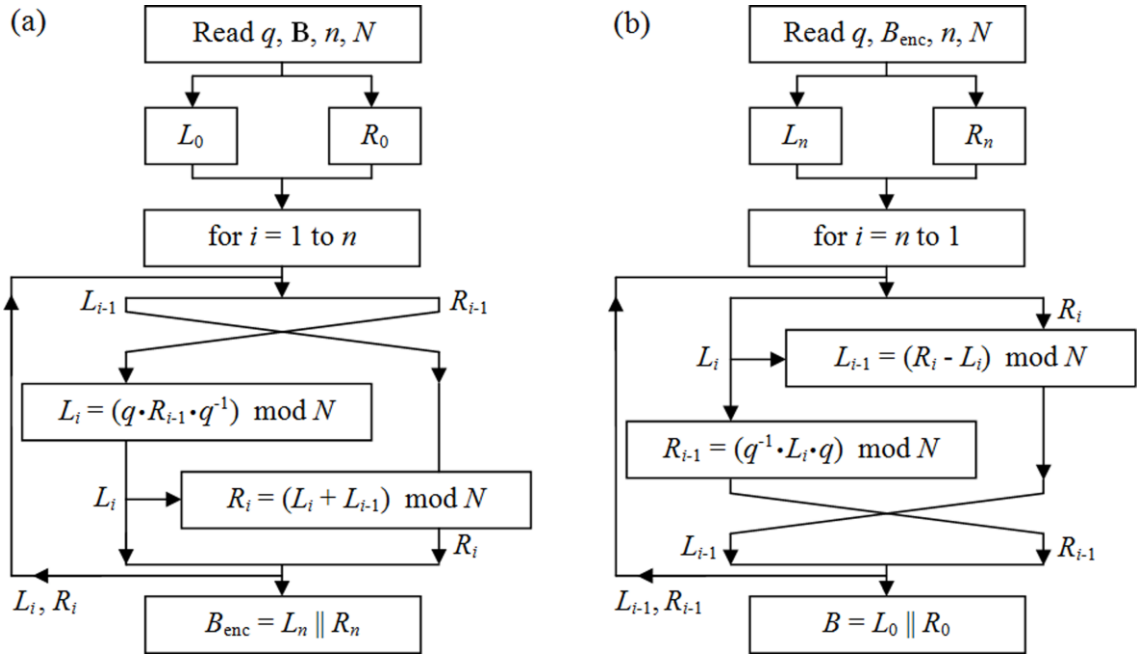
Wzory opisujące proces szyfrowania (8.2) i deszyfracji (8.3) dla omawianego algorytmu są następujące:

$$L_i = (q \cdot R_{i-1} \cdot q^{-1}) \mod N \qquad R_i = (L_{i-1} + L_i) \mod N \qquad \text{dla } i = 1, 2, \ldots n \tag{8.2}$$

$$R_{i-1} = (q^{-1} \cdot L_i \cdot q) \mod N \qquad L_{i-1} = (R_i - L_i) \mod N \qquad \text{dla } i = n, n\text{-}1, \ldots 1 \tag{8.3}$$

Wszystkie operacje przedstawione we wzorach (8.2) i (8.3) wykonywane są na kwaternionach z uwzględnieniem arytmetyki modularnej modulo $N = 257$ (rozdział

8.2.). Schemat procesu szyfrowania i deszyfracji dla omawianego algorytmu został przedstawiony na Rys. 10.



Rys. 10. Schemat szyfrowania (a) i deszyfracji (b) dla algorytmu kwaternionowego wykorzystującego zmodyfikowaną sieć Feistela.
Źródło: Opracowanie własne.

Występujący na Rys. 10. symbol || oznacza połączenie części wektorowych kwaternionów L_n i R_n. W konsekwencji otrzymujemy kwaternion B_{enc} o współczynnikach części wektorowej wielkości $m \times 2m$ każdy. Parametr n oznacza liczbę rund zastosowanych w algorytmie, przy czym każda runda szyfrowana jest z wykorzystaniem innego kwaternionu klucza q_i i = 1, 2, ..., n. Klucze dla każdej rundy uzyskiwane są na podstawie algorytmu generacji kluczy, przedstawionego w rozdziale 6.5.

Algorytm przystosowany jest do jednorazowego szyfrowania dowolnie dużych danych wejściowych. Nic jednak nie stoi na przeszkodzie, by dane wejściowe podzielić na mniejsze bloki, a całość szyfrować w trybie CBC lub CTR (rozdział 8.1.).

8.3.1. Obustronne mnożenie macierzowe

Dla zaproponowanego algorytmu istnieje dodatkowo możliwość zwiększenia bezpieczeństwa poprzez zastosowanie obustronnego mnożenia macierzowego przy wykorzystaniu arytmetyki modularnej. Mnożenie wykonywane jest dla wszystkich 3 współczynników części wektorowej kwaternionu R_{i-1} i realizowane jest następująco:

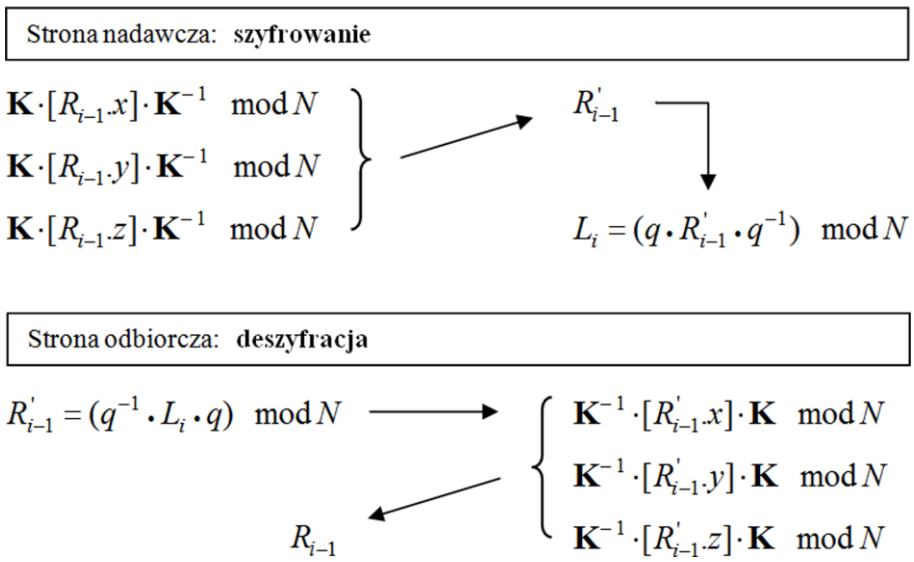
$$\mathbf{K} \cdot [R_{i-1}.x] \cdot \mathbf{K}^{-1} \mod N \tag{8.4}$$

$$\mathbf{K} \cdot [R_{i-1}.y] \cdot \mathbf{K}^{-1} \mod N \tag{8.5}$$

$$\mathbf{K} \cdot [R_{i-1}.z] \cdot \mathbf{K}^{-1} \mod N \tag{8.6}$$

gdzie $[R_{i-1}.x]$, $[R_{i-1}.y]$ i $[R_{i-1}.z]$ oznaczają odpowiednio współczynniki x, y i z kwaternionu R_{i-1} zapisane w postaci macierzy $m \times m$, natomiast **K** oznacza macierz klucza o rozmiarze $m \times m$. Modularne mnożenie macierzowe wykonywane jest dla $N = 257$.

Macierz klucza **K** jest wypełniona losowymi liczbami całkowitymi z zakresu od 0 do 256. Obustronne mnożenie macierzowe wykonywane jest zarówno po stronie nadawczej jak i odbiorczej. Po stronie odbiorczej występuje zmiana kolejności mnożenia macierzy klucza **K** przez współczynniki kwaternionu R_{i-1}, według schematu przedstawionego na Rys. 11.



Rys. 11. Obustronne mnożenie macierzowe dla współczynników części wektorowej kwaternionu R_{i-1} i macierzy klucza **K**, przedstawione dla strony nadawczej i odbiorczej. *Źródło: Opracowanie własne.*

Wyznaczenie macierzy odwrotnej o dowolnie dużym rozmiarze $m \times m$ jest realizowane przy wykorzystaniu metody Gaussa-Jordana oraz arytmetyki modularnej modulo $N = 257$. Warunkiem istnienia modularnej odwrotności macierzy jest, aby macierz **K**, utworzona z losowych wartości, nie była macierzą osobliwą - jej wyznacznik musi być różny od zera oraz względnie pierwszy do N (Christensen 2005).

Wprowadzenie obustronnego mnożenia macierzowego zwiększa poziom proponowanego przez algorytm bezpieczeństwa, poprzez uzależnienie każdego elementu w macierzy wynikowej od wszystkich elementów macierzy wejściowej. Wywoła to efekt lawinowy dla powstających szyfrogramów: jeżeli w tekście jawnym (obrazie wejściowym) zostanie zmieniona wartość pojedynczego bitu, to w szyfrogramie zmienione zostaną wartości połowy (w przybliżeniu) wszystkich bitów wyjściowych - więcej informacji na ten temat w rozdziale 11.

Dla lepszego zrozumienia uzyskanej zależności w macierzy wynikowej rozważymy następujący przykład. Dla uproszczenia wykorzystajmy macierz wejściową wielkości 3×3 (np. współczynnik x kwaternionu R_{i-1}). Macierz klucza $\mathbf{K}$ jest więc również rozmiaru 3×3. Zapis mnożenia macierzowego, zgodnie z (8.4), jest następujący (dla uproszczenia pomijamy arytmetykę modularną):

$$\begin{aligned}\mathbf{K}\cdot[R_{i-1}.x] &= \begin{bmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \\ k_7 & k_8 & k_9 \end{bmatrix}\cdot\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} = \\ &= \begin{bmatrix} x_1k_1 + x_4k_2 + x_7k_3 & x_2k_1 + x_5k_2 + x_8k_3 & x_3k_1 + x_6k_2 + x_9k_3 \\ x_1k_4 + x_4k_5 + x_7k_6 & x_2k_4 + x_5k_5 + x_8k_6 & x_3k_4 + x_6k_5 + x_9k_6 \\ x_1k_7 + x_4k_8 + x_7k_9 & x_2k_7 + x_5k_8 + x_8k_9 & x_3k_7 + x_6k_8 + x_9k_9 \end{bmatrix}\end{aligned} \tag{8.7}$$

$$(\mathbf{K}\cdot[R_{i-1}.x])\cdot\mathbf{K}^{-1} = \tag{8.8}$$

$$= \begin{bmatrix} x_1k_1 + x_4k_2 + x_7k_3 & x_2k_1 + x_5k_2 + x_8k_3 & x_3k_1 + x_6k_2 + x_9k_3 \\ x_1k_4 + x_4k_5 + x_7k_6 & x_2k_4 + x_5k_5 + x_8k_6 & x_3k_4 + x_6k_5 + x_9k_6 \\ x_1k_7 + x_4k_8 + x_7k_9 & x_2k_7 + x_5k_8 + x_8k_9 & x_3k_7 + x_6k_8 + x_9k_9 \end{bmatrix}\cdot\begin{bmatrix} k_1^{inv} & k_2^{inv} & k_3^{inv} \\ k_4^{inv} & k_5^{inv} & k_6^{inv} \\ k_7^{inv} & k_8^{inv} & k_9^{inv} \end{bmatrix} =$$

$$= \begin{bmatrix} (x_1k_1 + x_4k_2 + x_7k_3)k_1^{inv} + (x_2k_1 + x_5k_2 + x_8k_3)k_4^{inv} + (x_3k_1 + x_6k_2 + x_9k_3)k_7^{inv} & \dots \\ (x_1k_4 + x_4k_5 + x_7k_6)k_1^{inv} + (x_2k_4 + x_5k_5 + x_8k_6)k_4^{inv} + (x_3k_4 + x_6k_5 + x_9k_6)k_7^{inv} & \dots \\ (x_1k_7 + x_4k_8 + x_7k_9)k_1^{inv} + (x_2k_7 + x_5k_8 + x_8k_9)k_4^{inv} + (x_3k_7 + x_6k_8 + x_9k_9)k_7^{inv} & \dots \end{bmatrix}$$

gdzie k_i oznacza element macierzy klucza $\mathbf{K}$, x_i oznacza element macierzy wejściowej (element macierzy współczynnika x kwaternionu R_{i-1}) natomiast k_i^{inv} element odwróconej macierzy klucza $\mathbf{K}^{-1}$.

Zgodnie z zapisem (8.8) można stwierdzić, że macierz końcowa posiadać będzie elementy wyrażone przez wszystkie elementy macierzy wejściowej. Przykładowo, element (1,1) w macierzy wynikowej wyrażony jest przez wszystkie elementy macierzy wejściowej tj. x_1, x_2, ..., x_9.

Głównym celem wprowadzenia obustronnego mnożenia macierzowego jest uzyskanie zależności elementów macierzy wynikowej od elementów macierzy wejściowej dla uzyskania efektu lawinowego szyfrogramów (rozdział 11.) oraz uodpornienie algorytmu na atak ze znanym tekstem jawnym (rozdział 12.2.).

Ponieważ rozproszenie informacji w macierzy wynikowej jest uzyskiwane już po pierwszym, obustronnym mnożeniu macierzowym, zatem dla uzyskania najlepszej wydajności, wystarczy mnożenie to przeprowadzić tylko jeden raz, w jednej rundzie algorytmu. Można też, oczywiście zastosować mnożenie obustronne co kilka rund - zwiększając tym samym stopień złożoności algorytmu.

9. OBUSTRONNY GENERATOR KLUCZA

Istotnym problemem w przypadku systemów symetrycznych jest dostarczenie wspólnego klucza obu komunikującym się stronom, tak aby nie został on przechwycony przez osobę niepożądaną. Zagrożenie to zostałoby wyeliminowane, gdyby każda ze stron była odpowiedzialna za wygenerowanie własnego klucza. Wygodna zatem staje się koncepcja zrealizowania systemu symetrycznego bez procedury dostarczania obu stronom wspólnego klucza, opartego na generacji kluczy symetrycznych po obu komunikujących się stronach.

Zaproponowany w (Anand i in. 2009) model, wykorzystujący kwaternionowe zbiory Julia (ang. *Quaternion Julia Set*), zakłada ciągłą generację kluczy symetrycznych w czasie rzeczywistym, które służą do zabezpieczania przesyłanych przez kanał danych, w obrębie jednej sesji. Sposób ten, polegający na wymianie znaczników czasowych obu komunikujących się stron, może skutecznie utrudnić hakerowi dekryptaż przechwyconych danych.

W przypadku omawianego kwaternionowego systemu kryptograficznego, obustronny generator klucza można zrealizować na dwa sposoby. Pierwszy sposób skupia się na uzyskaniu dodatkowego poziomu bezpieczeństwa poprzez uniezależnienie kwaternionów kluczy wyższych rzędów od niższych. Modyfikacja algorytmu generacji kluczy (przedstawionego w rozdziale 6.5.) polega na uwzględnieniu dodatkowej rotacji kwaternionowej kwaternionów wyższych rzędów wokół kwaternionów wyznaczonych ze struktury fraktala.

Drugi sposób realizacji obustronnego generatora kluczy zakłada uzyskanie maksymalnej wydajności. Podejście to nie uwzględnia dodatkowej rotacji kwaternionowej, a skupia się jedynie na wyznaczeniu takich samych parametrów inicjalizujących dla algorytmu generacji kluczy po obu komunikujących się stronach.

Dla proponowanego rozwiązania można wykorzystać dowolne generatory ciągów pseudolosowych (np. eliptyczne generatory pseudolosowe (Kaczorowski 2014)) zapewniając synchronizację ich pracy dla obu komunikujących się stron. Możliwe rozwiązanie takiej synchronizacji zostało opisane w rozdziale 9.2.

9.1. Fraktale i zbiór Julia

Fraktal jest obiektem samopodobnym (jego części są podobne do całości) posiadającym subtelne detale nawet przy wielokrotnym powiększeniu. Ze względu na olbrzymią różnorodność przykładów matematycy unikają podawania ścisłej definicji i proponują określać fraktal jako zbiór o konkretnych właściwościach (Falconer 1997). Przykładowo, fraktal musi posiadać strukturę nietrywialną, trudną do opisania w języku tradycyjnej geometrii euklidesowej. Określany jest natomiast przez względnie prostą definicję rekurencyjną (Falconer 1997).

Zbiory Julia konstruuje się na płaszczyźnie zespolonej, wykorzystując funkcje iteracyjne wielomianu co najmniej stopnia drugiego. Ostatecznie zbiór Julia tworzą te punkty $p \in \mathbb{C}$, dla których ciąg opisany równaniem rekurencyjnym (Kudrewicz 1993):

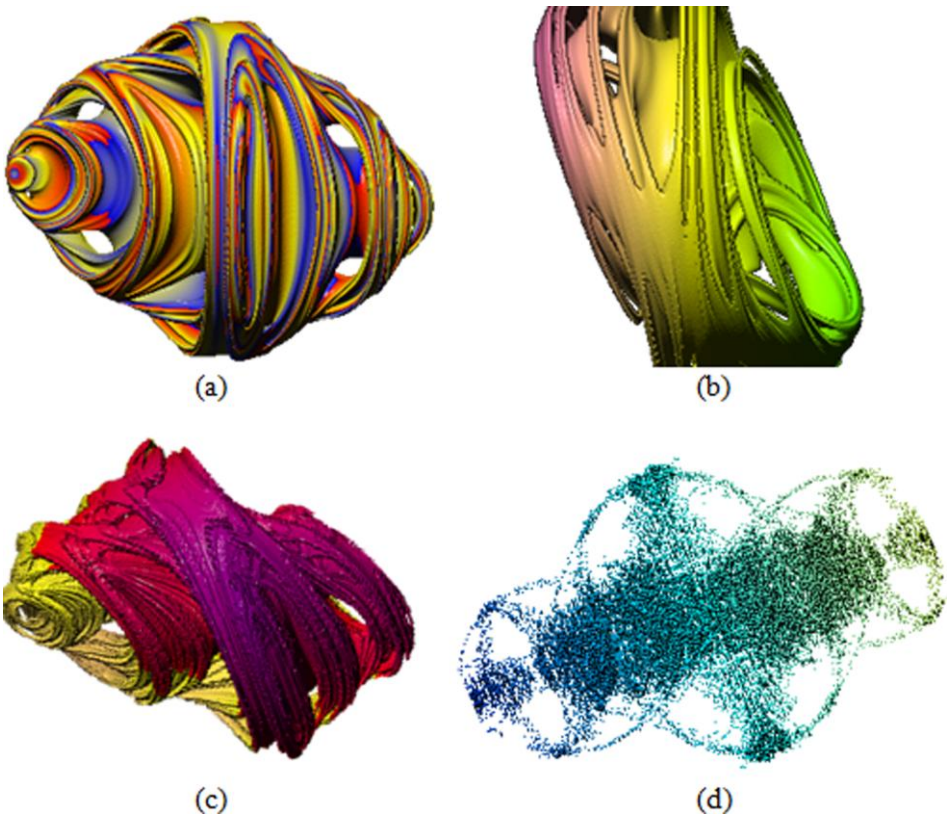
$$z_0 = p \tag{9.1}$$

$$z_{n+1} = z_n^2 + c \quad \text{dla } n = 0, 1, \ldots \tag{9.2}$$

nie dąży do nieskończoności:

$$\lim_{n \to \infty} = z_n \neq \infty \tag{9.3}$$

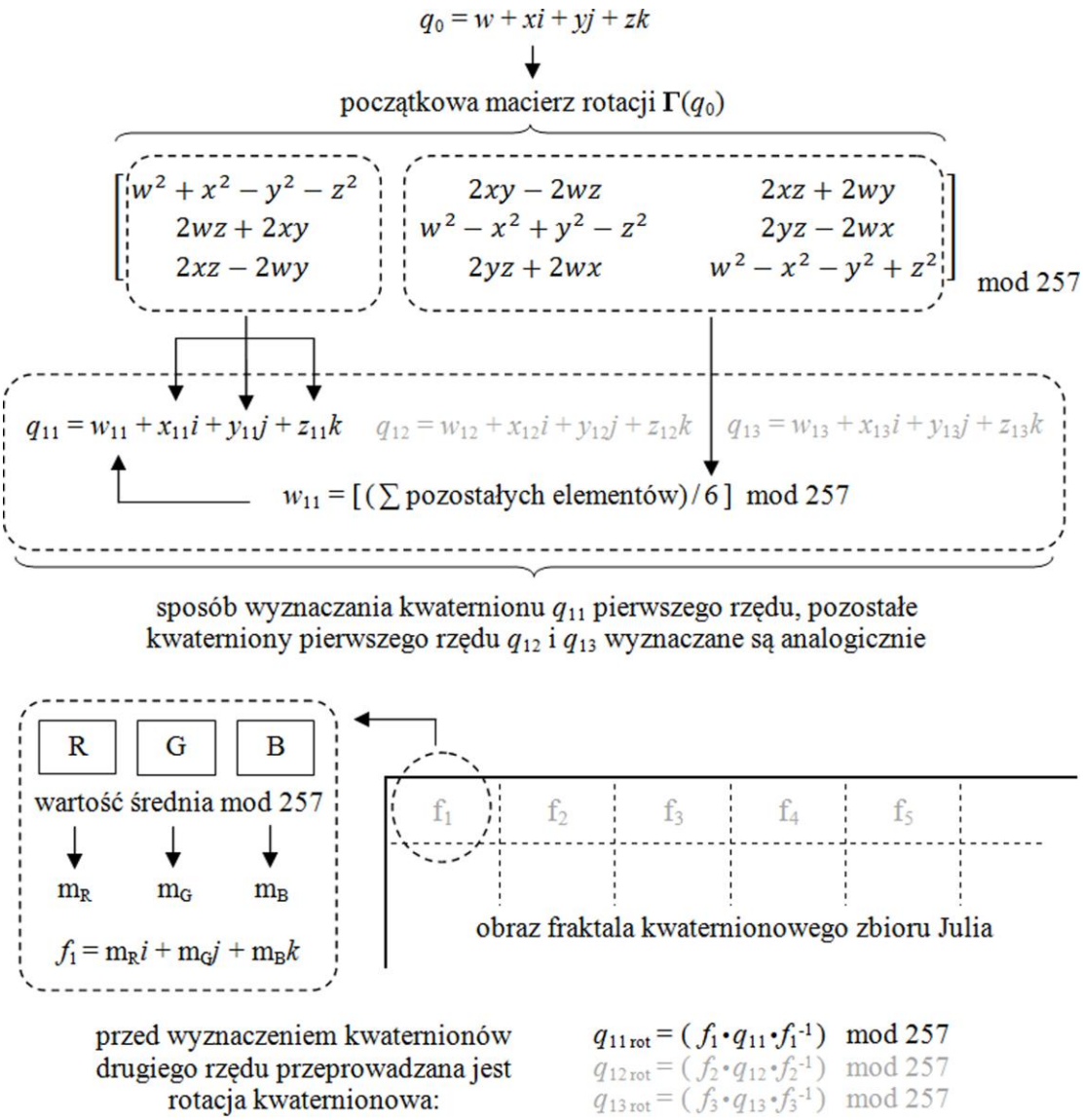
gdzie c to liczba zespolona będąca parametrem zbioru, określająca jego wygląd (Kudrewicz 1993). W przypadku kwaternionowych zbiorów Julia punkt startowy wyznaczany jest w przestrzeni trójwymiarowej (1, i, j) dla ustalonej przestrzeni k, c natomiast przyjmuje postać kwaternionu.



Rys. 12. Przykładowe fraktale kwaternionowego zbioru Julia: (a) liczba iteracji = 12, kwaternion c = 0.0882 + 0.1251i – 0.7555j + 0.1552k, wartość kontrolna = 16, bez płaszczyzny przekroju; (b) liczba iteracji = 10, kwaternion c = –0.5 + 0.6i + 0.4j + 0.5k, wartość kontrolna = 16, płaszczyzna przekroju k = 0.0945; (c) liczba iteracji = 20, kwaternion c = 0.1372 + 0.5i – 0.5154j – 0.1454k, wartość kontrolna = 5, płaszczyzna przekroju k = 0.2662; (d) liczba iteracji = 40, kwaternion c = –0.5784 + 0.1153i – 0.6112j + 0.1223k, wartość kontrolna = 20, bez płaszczyzny przekroju.
Źródło: Opracowanie własne.

Zachowanie funkcji w zbiorze Julia określane jest jako chaotyczne. Dowolnie małe zaburzenie może spowodować drastyczne zmiany w ciągu iterowanych wartości (Kudrewicz 1993).

Postać fraktala kwaternionowego zbioru Julia zależna jest od szeregu parametrów inicjalizujących takich jak: liczba iteracji, wartości współczynników kwaternionu *c*, wartości kontrolnej przy określaniu zbieżności punktów startowych oraz płaszczyzny przekroju dla wyświetlanego fraktala. Przykładowe fraktale kwaternionowego zbioru Julia, wygenerowane za pomocą generatora Quat (Quat 1.20), zostały przedstawione na Rys. 12. Quat jest generatorem fraktali, za pomocą którego możliwe jest wyświetlenie kwaternionowych zbiorów Julia poprzez nadanie stałej wartości dla czwartego wymiaru.



Rys. 13. Zmodyfikowany algorytm generacji kluczy szyfrujących, wykorzystujący fraktale kwaternionowego zbioru Julia.
Źródło: Opracowanie własne.

Fraktale kwaternionowego zbioru Julia wykorzystywane są w celu modyfikacji algorytmu generacji kluczy szyfrujących przedstawionego w rozdziale 6.5. Zaproponowana modyfikacja przedstawiona została na Rys. 13. Sposób wyznaczania kwaternionów wyższych rzędów jest podobny do tego, który został przedstawiony w rozdziale 6.5. Zasadnicza różnica widoczna jest przy określaniu współczynnika *w* dla kolejnych kwaternionów wyższych rzędów. Aby wyznaczyć wartość współczynnika *w* dla danego kwaternionu, należy obliczyć modularną (mod 257) wartość średnią elementów macierzy rotacji, pochodzących z kolumn, niewykorzystanych przy wyznaczeniu wartości współczynników *x*, *y* i *z* dla rozpatrywanego kwaternionu. Wyznaczone kwaterniony pierwszego rzędu (kwaterniony q_{11}, q_{12}, q_{13}) poddawane są modularnej (mod 257) rotacji kwaternionowej odpowiednio wokół kwaternionów f_1, f_2 i f_3.

Kwaterniony f_i otrzymywane są z trójkanałowego obrazu fraktala kwaternionowego zbioru Julia. Ich współczynnik rzeczywisty *w* jest zawsze równy zero, natomiast współczynniki wektorowe *x*, *y*, *z* przyjmują modularną (mod 257) uśrednioną wartość pikseli reprezentujących odpowiednio kanały R, G i B dla wyciętego fragmentu obrazu fraktala. Liczba fragmentów obrazu fraktala (Rys. 13.) jest uzależniona od planowanej wielkości przestrzeni kluczy (liczby kwaternionów wyższych rzędów), gdyż dla każdego następnego rzędu operacja przedstawiona na Rys. 13. jest powtarzana z wykorzystaniem kolejnych kwaternionów f_i.

W przeciwieństwie do dowolnego trójkanałowego obrazu RGB, fraktal kwaternionowego zbioru Julia nie musi być przesyłany w całości. Wystarczy podać jego parametry inicjalizujące, ograniczając w ten sposób wielkość informacji potrzebnych do wygenerowania zadanej przestrzeni kluczy.

9.2. Zasada działania obustronnego generatora klucza

Model wytwarzania par tajnych kluczy dla strony nadającej i dla strony odbierającej zaproponowany w niniejszej rozprawie, nawiązuje do modelu przedstawionego w pracy (Anand i in. 2009). Jego celem jest realizacja założeń kryptografii symetrycznej bez udziału strony trzeciej, odpowiedzialnej za dostarczanie kluczy symetrycznych. Tak więc, po obu komunikujących się stronach muszą być zaimplementowane mechanizmy zapewniające wygenerowanie par takich samych kluczy.

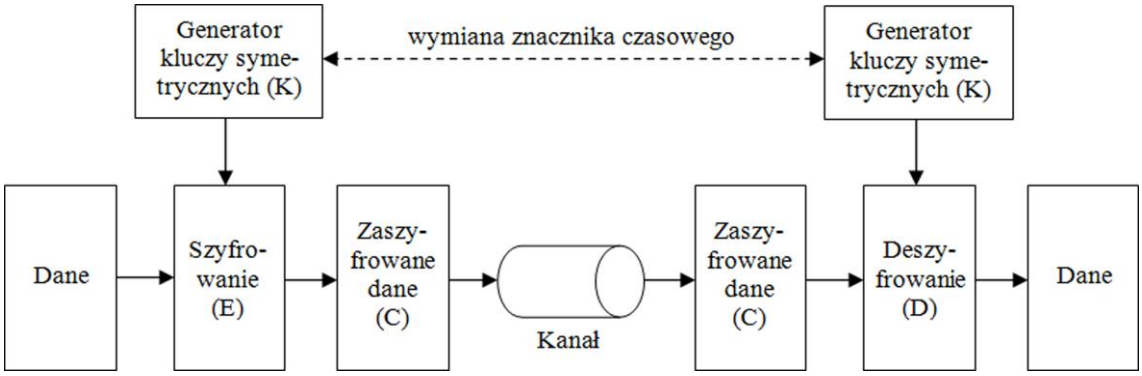
Dla algorytmów generacji kluczy szyfrujących, opisanych w rozdziałach 6.5. (wersja podstawowa) i 9.1. (wersja rozszerzona z kwaternionowym zbiorem Julia), potraktujmy kwaterniony-klucze q_i wybranego, końcowego rzędu, jako odpowiedniki kluczy symetrycznych w kryptografii symetrycznej. Kwaternionowy system kryptograficzny odpowiadałby za wyznaczenie kluczy symetrycznych ustalonego rzędu, według procedury opisanej w rozdziale 6.5. lub 9.1. Aby utworzyć kwaternion danego rzędu, konieczne jest wcześniejsze obliczenie kwaternionów rzędów niższych, co oznacza rozpoczęcie pracy obliczeniowej od kwaternionu rzędu zerowego, czyli od tzw. kwaternionu początkowego, dla którego ustalamy wartości czterech jego współczynników:

$$q_0 = w + xi + yj + zk \qquad \rightarrow \qquad (w,\ x,\ y,\ z) \tag{9.4}$$

To, jakich wartości współczynników kwaternionu początkowego *w*, *x*, *y*, *z* użyjemy, będzie miało wpływ na postaci kwaternionów wyższych rzędów. Należy więc zadbać, aby ich wybór był identyczny dla obu stron. Dodatkowo, w przypadku zastosowania rozszerzonego algorytmu generacji kluczy (rozdział 9.1.), po obu komunikujących się stronach należy ustalić parametry inicjalizujące potrzebne do wygenerowania fraktali kwaternionowego zbioru Julia. Do parametrów tych należą: liczba iteracji, współczynniki kwaternionu *c*, wartość kontrolna oraz położenie płaszczyzny przekroju.

W obrębie jednej sesji, przesyłane dane można zabezpieczyć pewną liczbą kwaternionów-kluczy q_i, wyznaczoną na podstawie wybranego kwaternionu początkowego q_0. Nic jednak nie stoi na przeszkodzie, by w obrębie jednej sesji wygenerować kilka przestrzeni kluczy, wykorzystując kilka różnych kwaternionów początkowych. Decydując się na częste zmiany kluczy, uodparniamy przesyłane dane na potencjalne ataki z zewnątrz. Należy jednak pamiętać, że zbyt częsta generacja kluczy może być przyczyną wprowadzenia zauważalnych opóźnień.

Na Rys. 14. przedstawiono strukturę rozpatrywanego w tym rozdziale modelu obustronnego generatora klucza, wykorzystującego znaczniki czasowe (ang. *timestamp*) w celu ustalenia wartości inicjalizujących, potrzebnych do wygenerowania zadanej przestrzeni kluczy symetrycznych.



Rys. 14. Model realizujący kryptografię symetryczną z obustronną generacją kluczy.
Źródło: Opracowanie własne na podstawie artykułu Anand i in. (2009).

Oznaczmy przez *M* dane, które chcemy przesłać do odbiorcy przez niezabezpieczony przed niepowołanym dostępem kanał transmisyjny. Dane te są szyfrowane za pomocą generowanych w czasie rzeczywistym, kluczy symetrycznych *K*. Uzyskujemy w ten sposób zaszyfrowane kwaternionowo dane *C*, które następnie przesłane są do strony odbiorczej. Sposób przeprowadzania szyfrowania kwaternionowego został przedstawiony w rozdziale 8.3. W ogólności, proces szyfrowania można w tym przypadku opisać wzorem:

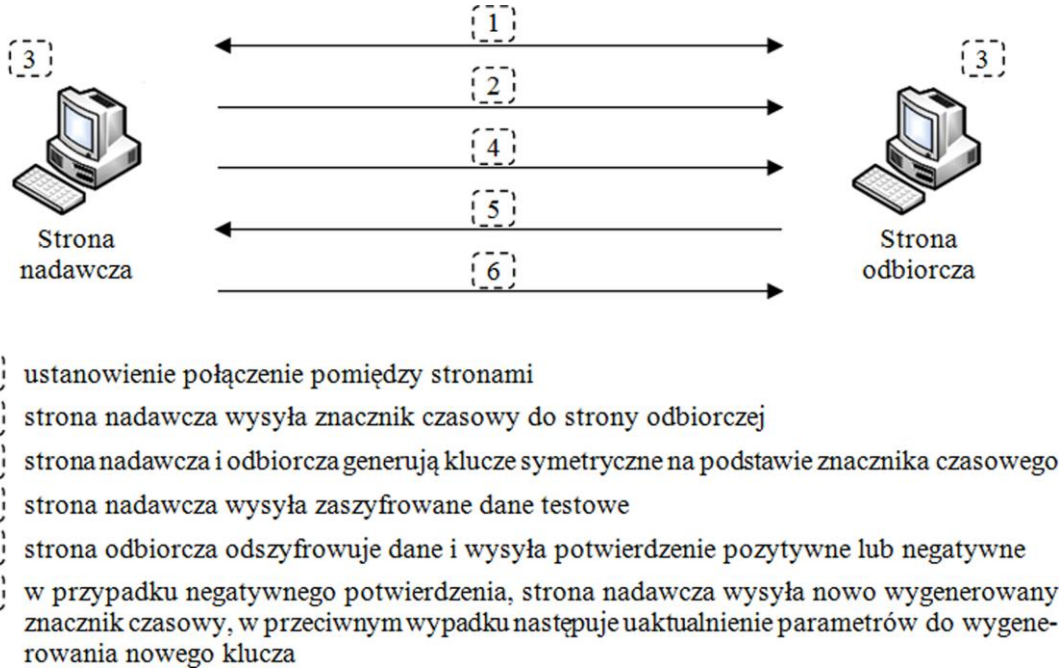
$$\mathrm{E}_K(M) = C \tag{9.5}$$

Zaszyfrowane dane *C*, otrzymane po stronie odbiorczej, są następnie poddawane deszyfracji za pomocą wygenerowanych niezależnie, w czasie rzeczywistym, identycznych, z użytymi po stronie nadawczej, kluczy symetrycznych (kwaternionów-kluczy wyższego rzędu q_i). Taki zabieg pozwala odtworzyć dane *M*, które nadano przez

stronę nadawczą. Zapisuje się to następująco:

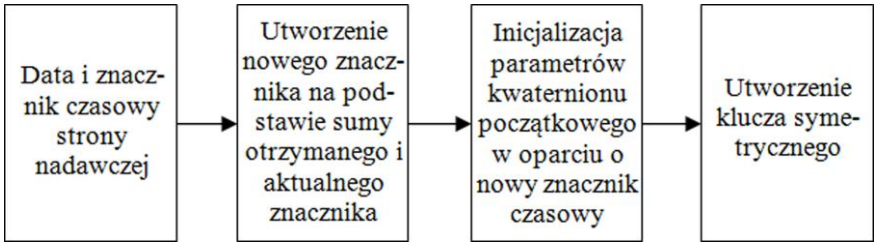
$$\mathrm{D}_K(C) = M \tag{9.6}$$

Zgodnie z (Anand i in. 2009) operacje przeprowadzane dla takiego modelu można podzielić na cztery fazy. Pierwszą z nich jest fazą nawiązywania połączenia. Strona chcąca wykonać transmisję danych musi nawiązać połączenie ze stroną odbiorczą za pośrednictwem protokołu *TLS* (ang. *Transport Layer Security*). Podczas ustanawiania połączenia strona nadawcza powinna wysłać stronie odbiorczej aktualną datę i obliczony po swojej stronie znacznik czasowy. Na tej podstawie strona odbiorcza będzie w stanie wyznaczyć nowy znacznik czasowy na potrzeby wzajemnego uwierzytelnienia. Dokładny opis wymiany znaczników czasowych został przedstawiony w (Anand i in. 2009).



Rys. 15. Operacje wymiany znaczników czasowych dla omawianego modelu.
Źródło: Opracowanie własne na podstawie artykułu Anand i in. (2009).

Faza druga jest fazą generacji kluczy symetrycznych po obu stronach. Do wygenerowania kluczy symetrycznych wykorzystany zostanie algorytm generacji kluczy przedstawiony w rozdziale 6.5. lub 9.1. Po zakończeniu pierwszej fazy, otrzymany nowy znacznik czasowy jest przechowywany w bazie danych hosta i jest wykorzystywany m.in. w celu przeprowadzenia uwierzytelnienia. Wykorzystuje się go do określenia parametrów kwaternionu początkowego q_0, rzędu szyfrowania oraz dodatkowych parametrów inicjalizujących potrzebnych do wygenerowania trójkanałowego obrazu RGB fraktala kwaternionowego zbioru Julia. Na tym etapie następuje wygenerowanie identycznej przestrzeni kluczy symetrycznych po obu stronach i realizacja procesu szyfrowania kwaternionowego.



Rys. 16. Operacje przeprowadzane przez generator kluczy symetrycznych.
Źródło: Opracowanie własne na podstawie artykułu Anand i in. (2009).

Faza trzecia to przeprowadzenie szyfrowania kwaternionowego, które zostało opisane w rozdziale 8.3.

Czwartą i ostatnią fazą jest faza potwierdzenia. Została ona wprowadzona, aby rozwiązać problem zmiany klucza, poprzez ciągłe wysyłanie znaczników czasowych. Strona odbiorcza wysyłać będzie pozytywne bądź negatywne potwierdzenie na każdy odebrany blok danych, bądź na daną sesję. Po utworzeniu nowego znacznika czasowego strona nadawcza wysyła zaszyfrowane dane testowe do strony odbiorczej, aby sprawdzić zgodność znacznika czasowego po obu stronach. Jeżeli odszyfrowanie powiedzie się, to strona odbiorcza wysyła pozytywne potwierdzenie. Po odebraniu takiego potwierdzenia przez stronę nadawczą obie strony dokonują jednoczesnej aktualizacji nowego znacznika czasowego poprzez dodanie wartości rzędu ostatniego szyfrowania kwaternionowego do poprzedniego nowego znacznika czasowego. Zapisuje się to następująco:

$$znacznik\_czasowy_{nowy} = znacznik\_czasowy_{nowy\_poprzedni} + rzad_{ostatniego\_szyfrowania} \tag{9.7}$$

Zamiast wartości rzędu można podać inną zmienną. Przykładowo może to być liczba iteracji potrzebnych do wygenerowaniu fraktala kwaternionowego zbioru Julia (przy wyborze rozszerzonego algorytmu generacji kluczy symetrycznych - rozdział 9.1.).

W przypadku, gdy strona odbiorcza nie odszyfruje danych testowych, wysyła stronie nadawczej negatywne potwierdzenie. Po otrzymaniu takiego potwierdzenia, strona nadawcza wygeneruje kolejny znacznik czasowy dla rozpoczęcia nowej sesji wymiany znaczników czasowych.

Główną zaletą omawianego modelu jest brak wymiany kluczy w kanale transmisyjnym. Jedynie przed rozpoczęciem komunikowania się, obie strony muszą znać kwaternion początkowy. Takie rozwiązanie pozwala w znaczącym stopniu zwiększyć bezpieczeństwo przesyłanych danych. Zaproponowany model pozostaje wrażliwy jedynie na ataki typu *brute-force* (Anand i in. 2009). Ponieważ liczba używanych kluczy jest zmienna i dodatkowo są one zmieniane w czasie rzeczywistym dla każdego bloku danych, zatem proces ewentualnego dekryptażu przez osoby niepożądane wymagać będzie zarówno dużo czasu, jak i dużych nakładów obliczeniowych. Wykorzystanie algorytmu szyfrowania kwaternionowego przedstawionego w rozdziale 8.3. oraz rozszerzonego algorytmu generacji kluczy symetrycznych (rozdział 9.1.), może przyczynić się do skutecznego utrudnienia procesu

kryptoanalizy, biorąc pod uwagę czas życia przesyłanych wiadomości.

Zaproponowany model pozytywnie realizuje każdą z wymienionych w rozdziale 2. cech. Jego porównanie z innymi modelami zostało przedstawione w tabeli 1. wzorowanej na (Anand i in. 2009).

Tabela 1. Porównanie różnych modeli kryptograficznych.

cecha \ model	szyfrowanie symetryczne	szyfrowanie asymetryczne	podpis cyfrowy	rozważany model
poufność	tak	tak	nie	tak
uwierzytelnienie	nie	nie	tak	tak
integralność	nie	nie	tak	tak
niezaprzeczalność	nie	nie	tak	tak

Poufność w zaproponowanym modelu realizowana jest poprzez szyfrowanie bloków danych różnymi kluczami jednorazowego użytku. Co więcej, dla danego bloku danych przewidzianych może być kilka kluczy, w zależności od wybranego rzędu szyfrowania kwaternionowego. Taki zestaw kluczy jest unikatowy dla każdego, rozpatrywanego bloku danych, co zapewnia wysoki stopień poufności z punktu widzenia transmisji całości danych.

Uwierzytelnienie w rozpatrywanym modelu związane jest z generowaniem w czasie rzeczywistym znaczników czasowych. Strona odbierająca znacznik czasowy weryfikuje go z wersją przechowywaną w swojej bazie danych. Jeżeli osoba niepożądana przeprowadziłaby atak na znacznik czasowy, to z uwagi na zaproponowany proces potwierdzania każdego wysłanego bloku danych, istnieje możliwość zidentyfikowania nieuwierzytelnionego użytkownika.

Zapewnienie integralności danych stanowi problem w przypadku systemów symetrycznych. Duża liczba kluczy, wykorzystywanych w zaproponowanym modelu transmisji danych, utrudnia przeprowadzenie wszelkich modyfikacji na przechwyconych danych. Jeżeli atakujący zna schemat szyfrowania kwaternionowego, to bardzo trudne jest utworzenie kwaternionów wyższych rzędów bez znajomości wartości parametrów przechowywanych w bazie danych użytkownika. Co więcej, naturalnym jest założenie, że atakujący nie dysponuje informacją o wielkości zastosowanego rzędu, determinującej liczbę potencjalnych kluczy do zaszyfrowania pojedynczego bloku danych.

Niezaprzeczalność w zaproponowanym modelu oparta jest na wykorzystaniu kryptografii czasu rzeczywistego i znacznika czasowego nadawcy, dlatego też nadawca nie może zaprzeczyć wysłania wiadomości. Wykorzystywany protokół *TLS* podczas fazy nawiązywania połączenia, odpowiada za uwierzytelnienie nadawcy za pośrednictwem znaczników czasowych. Ponieważ znacznik czasowy nie jest generowany w sposób losowy u nadawcy, zatem problem zaprzeczenia jest rozwiązany. Wszelkie procesy inicjalizujące połączenie są niewidoczne dla użytkownika, a generowane przez nie raporty (logi), są przechowywane w bezpiecznym miejscu, co

pozwala na zapewnienie niezaprzeczalności nadawcy.

Zaproponowany model może być wykorzystany w sposób elastyczny. Można ograniczyć formę stosowanych zabezpieczeń, skupiając się na zaimplementowaniu kwaternionowego algorytmu szyfrującego (rozdział 8.3.) bez obustronnego mnożenia macierzowego wraz z podstawowym algorytmem generacji kluczy symetrycznych (rozdział 6.5.). Dla uzyskania maksymalnego bezpieczeństwa, algorytm szyfrowania kwaternionowego należy zaimplementować wraz z obustronnym mnożeniem macierzowym i wykorzystać rozszerzony algorytm generacji kluczy (rozdział 9.1.).

Dodatkowo, model ten obsługuje transmisję danych pomiędzy dwoma użytkownikami jak i pomiędzy większą ich liczbą (Anand i in. 2009). Do bezpiecznego przesyłu wykorzystać możemy zarówno dane o małej, jak i dużej objętości, zaczynając od numerów kart kredytowych, a kończąc na VoD (ang. *Video on Demand*). W przypadku transmisji pomiędzy wieloma użytkownikami, każda komunikująca się strona powinna przeprowadzić wymianę znaczników czasowych i komunikować się między sobą na osobnych portach. Omawiany model może być też bezpiecznie zaimplementowany jako oprogramowanie, bądź też jako dedykowany sprzęt dla komunikujących się użytkowników, dodatkowo ograniczając możliwość przeprowadzenia potencjalnych ataków na zaszyfrowane dane.

10. ZASTOSOWANIA

Przeznaczenie kwaternionowego systemu kryptograficznego to nie tylko zabezpieczanie danych multimedialnych. Zaproponowany model może być z powodzeniem wykorzystany w sieciach DICOM dla zabezpieczania danych medycznych oraz jako narzędzie do osadzania fingerprintów w odszyfrowanym obrazie.

10.1. Zastosowanie medyczne

Rozwój nauki i technologii zaowocował cyfryzacją przemysłu medycznego. Obrazy medyczne przetwarzane w postaci cyfrowej stały się normą obowiązującą w każdym współczesnym szpitalu. Jest to szczególnie mocno uwidocznione w dziedzinie diagnostyki i leczenia pacjentów. Jednakże, wraz z zastosowaniem nowych rozwiązań i technologii pojawiają się nowe problemy. Wykorzystując obrazy cyfrowe, nie można pominąć zagadnień związanych z ich transmisją czy udostępnianiem, co pociąga za sobą problem dotyczący bezpieczeństwa danych. Dla zapewnienia bezpiecznej transmisji i wymiany danych opracowano medyczny standard DICOM (ang. *Digital Image and Communication On Medicine*). Standard DICOM, opracowany przez ACR/NEMA (ang. *American College of Radiology* / *National Electrical Manufacturers Association*), zapewnia ujednolicenie wymiany i interpretacji danych medycznych reprezentujących lub związanych z obrazami diagnostycznymi w medycynie. Dane w formacie DICOM mają dużą objętość, wymagają specjalistycznego sprzętu komputerowego i oprogramowania oraz łączy o wysokiej przepustowości. Duża objętość plików DICOM przekłada się na wysoką jakość obrazu.

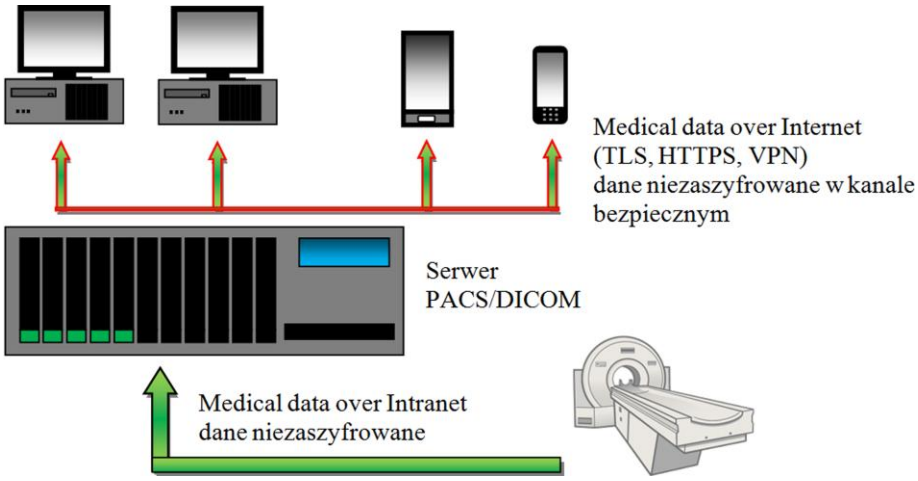
Kwestia bezpieczeństwa, oferowanego przez standard DICOM, zwykle sprowadzana jest do indywidualnego wdrożenia wybranych mechanizmów zabezpieczających. Obecnie, w celu zabezpieczania danych medycznych w sieciach DICOM stosuje się mechanizmy wykorzystujące szyfrowanie algorytmem AES lub 3DES (NEMA 2008). Wadą aktualnego rozwiązania jest znaczący czas przetwarzania potrzebny na zaszyfrowanie i odszyfrowanie obrazów typu DICOM. Z tego też powodu zauważyć można ciągle rosnące zapotrzebowanie na architekturę, która oferowałaby nie tylko bezpieczeństwo, ale także szybki dostęp do danych medycznych.

10.1.1. Sieć DICOM

Rozwój nowych technologii i wdrażanie nowych rozwiązań w dziedzinie medycyny coraz silniej uwidacznia problem związany z bezpiecznym przesyłaniem i przechowywaniem poufnych informacji medycznych. Dla standardu DICOM kwestia bezpieczeństwa nie jest pomijana (NEMA 2008, Pianykh 2008), jednakże stosowane rozwiązania zwykle polegają na zabezpieczaniu transmisji danych do użytkownika (lekarza/pacjenta) (OsiriX 2009, 2010).

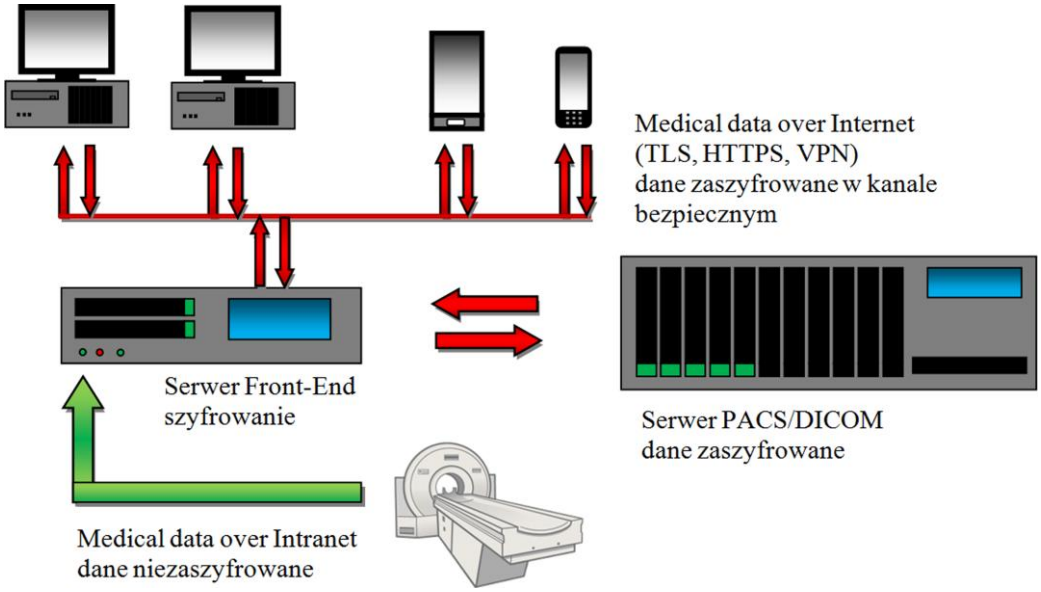
W przykładowej sieci DICOM (Rys. 17.) wykorzystuje się protokoły TLS, HTTPS i VPN (OsiriX 2009, 2010) w celu zabezpieczenia transmisji i dostępu do danych. Dane nie są szyfrowane, lecz przesyłane kanałem bezpiecznym. Dodatkowo, istnieje

możliwość zaszyfrowania przesyłanych danych algorytmem AES bądź 3DES, jednakże operacje te dodatkowo wydłużają czas potrzebny na dostęp do danych.



Rys. 17. Przykładowa sieć DICOM.
Źródło: Opracowanie własne.

Przez sieć wewnętrzną, zapewniającą połączenie między serwerem DICOM i specjalistycznym sprzętem medycznym (Rys. 17.), przesyłane są dane w postaci niezaszyfrowanej. Serwer DICOM natomiast, przechowujący dane medyczne, odpowiada za realizację połączeń zewnętrznych, przez co narażony jest na różnego rodzaju ataki (np. atak Man-in-the-middle).



Rys. 18. Zmodyfikowana sieć DICOM, wyposażona w serwer typu front-end.
Źródło: Opracowanie własne.

Model zaproponowany w niniejszej rozprawie pozwala na zabezpieczenie zarówno sieci wewnętrznej jak i zabezpieczenie serwera DICOM, przechowującego dane w

postaci niezaszyfrowanej. Standardową sieć DICOM należy wzbogacić o dodatkowy serwer tzw. serwer typu front-end, odpowiedzialny za przechwytywanie, szyfrowanie (wg. algorytmu przedstawionego w rozdziale 8.3.) i przekazywanie danych medycznych do serwera DICOM bez ich uprzedniego zapisywania na dodatkowych nośnikach (Fig. 18.). Wykorzystanie serwera typu front-end pozwala na wyeliminowanie podstawowej wady standardowej sieci DICOM, tj. wyeliminowanie dostępu do niezabezpieczonych danych na serwerze DICOM (Philips 2012).

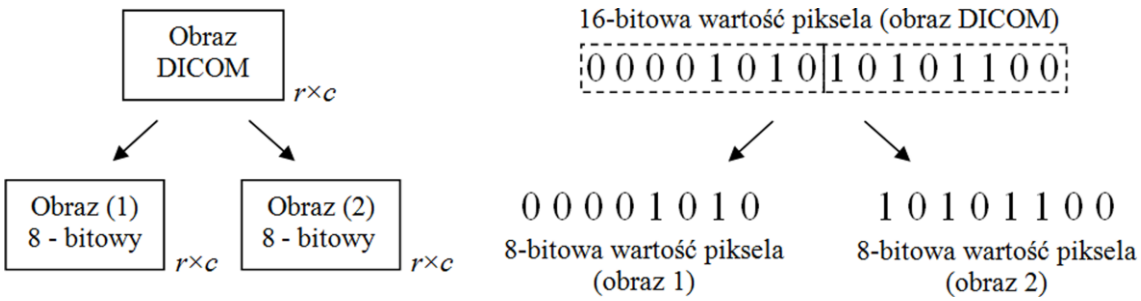
Dodatkowo, zaproponowany serwer front-end jest odpowiedzialny za kontrolę dostępu do danych, nawiązując bezpieczne połączenie (TLS, HTTPS) z wybranym użytkownikiem i przekazując mu zaszyfrowaną treść. W przypadku obsługi użytkowników wykorzystujących połączenie VPN, serwer font-end umożliwiałby także dekryptaż zaszyfrowanych danych. Takie rozwiązanie pozwala na dostęp do danych nawet w przypadku urządzeń (smartfonów czy tabletów) niezdolnych do przeprowadzenia szybkiej deszyfracji.

Model zmodyfikowanej sieci DICOM wprowadza nie tylko dodatkowy poziom bezpieczeństwa, oparty na szyfrowaniu kwaternionowym, lecz dodatkowo umożliwia szybki dostęp do danych dla użytkowników mobilnych. Przedstawiona komunikacja między serwerem, sprzętem medycznym i użytkownikami końcowymi nie wykracza poza zakres już zdefiniowanej komunikacji w standardzie DICOM (NEMA 2008). Zaproponowany model nie wymaga więc gruntownych modyfikacji w aktualnie wykorzystywanym modelu standardowej sieci DICOM.

10.1.2. Szyfrowanie obrazu DICOM

Obiekt w formacie DICOM jest wieloatrybutowy. Składa się z elementów takich jak m.in. imię, nazwisko, identyfikator pacjenta, a także ze specjalnego atrybutu, który zawiera dane obrazu wyrażone w pikselach (NEMA 2008). W każdym obrazie DICOM można więc wyróżnić część informacyjną (tekstową) oraz część graficzną 16-bitowego obrazu w odcieniach szarości.

Szyfrowanie obrazu DICOM (zarówno części informacyjnej jak i graficznej) sprowadza się do zastosowania algorytmu przedstawionego w rozdziale 8.3. Znaki tekstowe części informacyjnej obrazu DICOM zapisywane są w postaci decymalnej wg. standardu ASCII i grupowane w celu utworzenia sześciu macierzy kwadratowych. Otrzymane macierze stanowią współczynniki kwaternionów L_0 i R_0 (rozdział 8.3.).



Rys. 19. Binarna dekompozycja obrazu DICOM.
Źródło: Opracowanie własne.

Szyfrowanie części graficznej obrazu DICOM polega na dekompozycji 16-bitowego obrazu o rozmiarze $r \times c$ na dwa obrazy 8-bitowe o rozmiarach $r \times c$ każdy (Rys. 19.). Dwa 8-bitowe obrazy traktowe są jako dane wejściowe i zapisywane w postaci sześciu macierzy kwadratowych. Utworzone macierze stanowią współczynniki kwaternionów L_0 i R_0 (rozdział 8.3.). Dekompozycja części graficznej obrazu DICOM jest konieczna z uwagi na wykorzystywaną arytmetykę modularną (mod 257). Efekt szyfrowania obrazów DICOM został przedstawiony i omówiony w rozdziale 11.3.

10.2. Zastosowanie dla fingerprintingu

Multimedia znajdują coraz szersze zastosowanie w usługach interaktywnych, takich jak wideorozmowa lub wideokonferencja, ale również w usługach rozsiewczych, takich jak wideo na żądanie lub telewizja internetowa. Treści multimedialne są obecnie przesyłane na wiele sposobów w zależności od ich przeznaczenia, które obecnie sięgają od zastosowań rozrywkowych, aż po biznesowe.

Bezpośrednim zagrożeniem dla cyfrowego rynku multimediów jest piractwo, czyli zjawisko nielegalnego kopiowania i udostępniania treści multimedialnych bez zgody autorów. Istnieją dwie uzupełniające się metody ochrony multimediów oraz praw autorskich (Liu i in. 2005, Liu i Zhao 2004). Pierwszą metoda jest szyfrowanie, które zapewnia, że tylko zarejestrowani użytkownicy, posiadający odpowiednie klucze deszyfrujące, będą w stanie odszyfrować przesyłane treści multimedialne i w rezultacie będą mogli z nich korzystać. Niestety, szyfrowanie nie jest wystarczającym zabezpieczeniem, gdyż po deszyfracji użytkownik mający dostęp do multimediów może je ponownie udostępnić bez zgody autora, łamiąc tym samym prawa autorskie. Druga metoda to tzw. cyfrowy odcisk palca (ang. *digital fingerprinting*), będący rozszerzeniem cyfrowego znaku wodnego (ang. *digital watermarking*). Zarówno watermarking jak i fingerprinting są technikami osadzania dodatkowych danych w treści multimedialnej. W przypadku watermarkingu, dodatkowe dane (watermarki) służą do identyfikacji źródła pochodzenia, czyli wydawcy multimediów. W przypadku fingerprintingu, dodatkowe dane (fingerprinty) stanowią informacje o użytkownikach, którym udostępniono daną treść multimedialną. Watermarking stosowany jest w celu zachowania praw autorskich przez faktycznego właściciela danej treści multimedialnej, podczas gdy fingerprinting wykorzystywany jest do wykrywania nieuczciwych użytkowników i pociągania ich do odpowiedzialności karnej.

Fingerprinty są osadzane w multimediach w taki sposób, aby pozostały niezauważalne dla człowieka. Jest to możliwe dzięki osadzaniu fingerprintów w wartościach pikseli obrazu (Wang i Pearmain 2004, Chang i in. 2006), w dziedzinie dyskretnej transformaty kosinusowej DCT (ang. *Discrete Cosine Transform*) (Ahmed i in. 1974, Kundur i Karthnik 2004, Wang i Pearmain 2004, Suthaharan i in. 2000), w dziedzinie transformaty falkowej (Maity i in. 2007). Analiza fingerprintu w nielegalnie udostępnionej kopii pozwoli na identyfikację pirata.

10.2.1. Algorytm kwaternionowy dla fingerprintingu

Modyfikacja jednego z pierwszych algorytmów szyfrowania kwaternionowego, opartego na rotacjach kwaternionowych bloków danych zaimplementowanych w trybie CBC (Dzwonkowski i Rykaczewski 2012), pozwala wprowadzić niezauważalne dla ludzkiego oka błędy w odszyfrowanym obrazie. Pojawienie się błędów po stronie deszyfrującej umożliwiło ich wykorzystanie jako unikatowego fingerprintu osoby odszyfrowującej daną treść multimedialną.

Zastosowany dla fingerprintingu algorytm szyfrujący jest zmodyfikowaną wersją algorytmu zaproponowanego w pracach (Nagase i in. 2004a, 2004b, 2005). W pracach autorstwa Nagase i in. przedstawiono algorytm ECB (rozdział 8.1.1.) szyfrujący bloki danych wykorzystując rotację kwaternionową wyrażoną macierzowo (6.6). Algorytm wykorzystany do osadzania fingerprintów w odszyfrowanym obrazie wykorzystuje tryb szyfrowania CBC (rozdział 8.1.2.) i oparty jest na rotacji kwaternionowej bloków danych, zapisanej za pomocą mnożenia kwaternionowego (6.10).

Implementacja trybu CBC wykonana została następująco. Przed rotacją k-tego kwaternionu B, wykonywana jest suma binarna wszystkich elementów tworzących komponenty (x, y, z) k-tego kwaternionu B i $(k\text{-}1)$-ego zaszyfrowanego kwaternionu B_{rot}. Dla pierwszego kroku wymagany jest losowy kwaternion inicjalizujący B_{ini}. Wielkość komponentów kwaternionu B_{ini} musi być zgodna z wielkością komponentów kwaternionów B. Wynikiem przeprowadzonej sumy binarnej jest k-ty kwaternion B_{mod}, który jest następnie szyfrowany zgodnie z zasadą (6.10). Kwaternionem wynikowym przeprowadzonego szyfrowania jest k-ty kwaternion B_{rot} (Rys. 20.).

Poelementowe sumowanie binarne należy zaimplementować w taki sposób, aby możliwe było zsumowanie wartości zmiennoprzecinkowych. Ponieważ omawiany algorytm nie uwzględnia operacji modularnych podczas szyfrowania kwaternionowego, zatem otrzymywane wartości komponentów zaszyfrowanych kwaternionów B_{rot} są postaci zmiennoprzecinkowej.

Tabela 2. Wartości specjalne dla liczb zmiennoprzecinkowych pojedynczej precyzji na podstawie standardu IEEE-754.

Wartość specjalna	Bit znaku	Bity wykładnika	Bity mantysy
NaN	x	111..111	1xx..xxx
SNaN	x	111..111	0xx..xxx
Zero	x	000..000	000..000
Nieskończoność	x	111..111	000..000
Nieznormalizowana	x	000..000	xxx..xxx

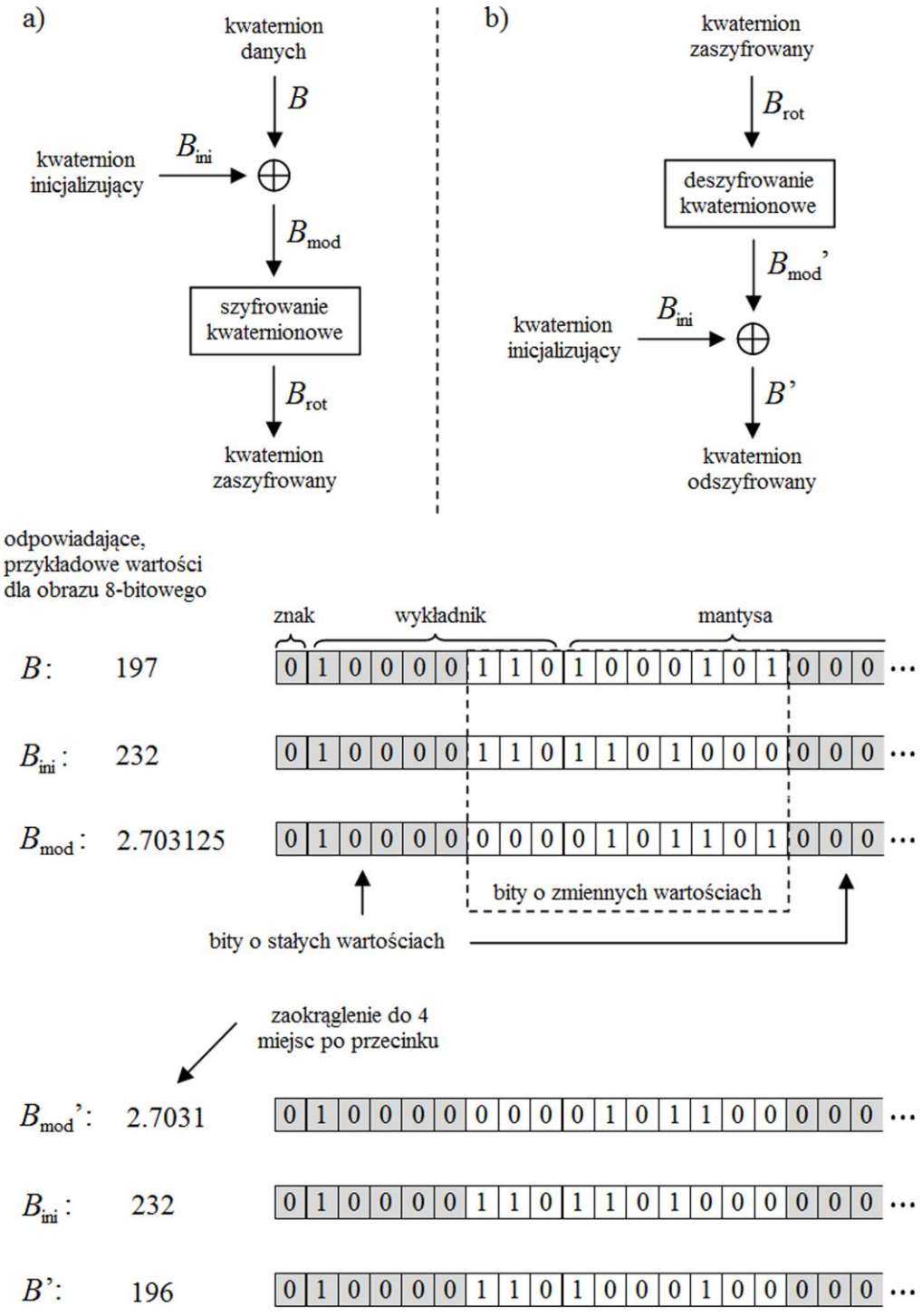
Zgodnie ze standardem IEEE-754, liczba zmiennoprzecinkowa, pojedynczej precyzji, składa się z 1-ego bitu znaku, 8-iu bitów wykładnika i 23-ech bitów mantysy, co daje szerokość słowa równą 32 bity. Poelementowe sumowanie binarne liczb zmiennoprzecinkowych nie może być przeprowadzane dla każdego bitu, gdyż istnieje ryzyko uzyskania wyjątku w postaci liczby specjalnej (Kahan 1997) (Tabela 2.). Oprócz wyjątków należy także uwzględnić zakres wartości dla liczby zmiennoprzecinkowej.

Nawiązując do pracy (Kahan 1997) zakres liczby zmiennoprzecinkowej pojedynczej precyzji wynosi $1.2 \cdot 10^{-38} – 3.4 \cdot 10^{38}$.

Zaproponowany w tym rozdziale algorytm dla fingerprintingu pozwala na uzyskanie niezauważalnych dla ludzkiego oka błędów w odszyfrowanym obrazie. Otrzymywane po stronie odbiorczej błędy traktowane są jako unikatowe fingerprinty użytkowników dokonujących deszyfracji. W celu zrozumienia zasady powstawania błędów po stronie odbiorczej rozważmy następujący przykład. Na Rys. 20. przedstawiono tylko pojedynczą parę odpowiadających sobie elementów, pochodzących z komponentów kwaternionów B i B_{ini}. W celu przeprowadzenia binarnego sumowania podanych wartości, należy najpierw zapisać je w ich binarnej postaci dla liczb zmiennoprzecinkowych pojedynczej precyzji (wg. standardu IEEE-754). W wyniku operacji XOR (sumowania binarnego) otrzymamy najprawdopodobniej liczbę zmiennoprzecinkową (w tym przypadku wartość odpowiadającego elementu, pochodzącego z komponentu kwaternionu B_{mod}). Zbiór uzyskanych w ten sposób elementów stanowi komponenty (x, y, z) kwaternionu B_{mod}, który jest następnie szyfrowany kwaternionowo, przesyłany do odbiorcy i tam deszyfrowany. Podczas deszyfracji otrzymamy dokładnie te same wartości w komponentach kwaternionu B_{mod}' co w komponentach kwaternionu B_{mod}. Taka implementacja pozwala na bezbłędne odszyfrowywanie przesyłanych treści, a więc nie posiada zastosowania dla fingerprintingu. Jednakże, w przypadku zaokrąglenia wszystkich liczb zmiennoprzecinkowych znajdujących się w komponentach kwaternionu B_{mod}' uzyskamy możliwość wprowadzenia błędów podczas deszyfracji. Na Rys. 20. przedstawiono zaokrąglenie do 4 miejsc po przecinku, co prowadzi do uzyskania obrazu wynikowego o wartościach innych niż oryginalne i różniących się maksymalnie o ±3.

Aby wyeliminować ryzyko pojawienia się wartości specjalnej (Tabela 2.) podczas sumowania binarnego, w binarnej reprezentacji liczb zmiennoprzecinkowych zastosowany został podział na bity o stałych i zmiennych wartościach (Rys. 20.). Operacja XOR przeprowadzana jest wyłącznie dla części o zmiennych bitach, która zawiera 3 bity wykładnika i 7 bitów mantysy. Pozostałe bity są niezmienne i posiadają stałe wartości. Tak dobrana liczebność zmiennych bitów (na których przeprowadzana jest operacja XOR) jest niezbędna do przedstawienia największej dopuszczalnej wartości w obrazach 8-bitowych, wartości 255. Pozwoli to także na wyeliminowanie wartości specjalnych podczas sumowania binarnego, jak również rozwiąże problem związany z wymaganym zakresem wartości dla liczb zmiennoprzecinkowych pojedynczej precyzji. Przeprowadzenie operacji XOR na wybranym zakresie zmiennych bitów pozwoli na uzyskanie wartości zmiennoprzecinkowych z zakresu 2-510 z dokładnością do 6 miejsc po przecinku. Niestety, nie będzie możliwe uzyskanie wartości mniejszych od 2, dlatego też piksele o wartościach 0 lub 1 w obrazie oryginalnym odszyfrowane będę jako piksele o wartości 2. Taki błąd leży w dopuszczalnym, przyjętym zakresie ±3.

Zwiększenie dokładności do 6 miejsc po przecinku dla wartości kwaternionu B_{mod}' wyeliminuje całkowicie błędy po stronie odbiorczej. Zmniejszenie dokładności do 3 miejsc po przecinku wygeneruje błędy z zakresu ±7. Dokładność ustawiona na 2 miejsca po przecinku wygeneruje błędy z zakresu ±15, natomiast w przypadku zmniejszenia dokładności do 1 miejsca po przecinku będzie można zaobserwować błędy duże o wielkości z zakresu ±478.



Rys. 20. Pojawienie się błędu podczas deszyfracji, zobrazowane dla pojedynczych elementów kwaternionów przy wykorzystaniu reprezentacji binarnej dla liczb zmiennoprzecinkowych wg standardu IEEE-754: (a) strona nadawcza, (b) strona odbiorcza.

Źródło: Opracowanie własne.

Pojawiający sie zakres błędów wynika bezpośrednio z zastosowanego podziału na bity o stałych i zmiennych wartościach dla reprezentacji binarnej liczb zmiennoprzecinkowych. Przykładowo, wykonując po stronie nadawczej operację XOR dla dwóch liczb 134 (B) i 249 (B_{ini}) otrzymamy liczbę zmiennoprzecinkową z dokładnością do 6 miejsc po przecinku, tj. liczbę 3.984375. Następnie, po zastosowaniu po stronie odbiorczej zaokrąglenia do 1 miejsca po przecinku, otrzymamy liczbę zmiennoprzecinkową równą 4.0. Reprezentacja binarna dla liczby zmiennoprzecinkowej 4.0 wykorzystuje tylko pojedynczy bit w części wykładnika dla przedstawionego zakresu zmiennych bitów. W reprezentacji binarnej liczby 3.984375 wszystkie bity mantysy dla przedstawionego zakresu zmiennych bitów posiadają wartość 1. Różnica w postaciach binarnych liczb 4.0 i 3.984375 jest znaczna, tak więc podczas sumowania binarnego liczby 4.0 z liczbą 249 (B_{ini}), otrzymamy po stronie odbiorczej wartość 498. Na tej podstawie błąd wartości odszyfrowanej względem wartości oryginalnej wynosi 498 - 134 = 364.

Wpływ błędów na obraz odszyfrowany oraz wydobyte fingerprinty zostały przedstawione w rozdziale 11.4.

11. BADANIA I TESTY

Rozdział 11 poświęcony jest badaniom i testom omawianych w rozprawie kwaternionowych algorytmów szyfrujących. W rozdziale 11.1. przedstawiono wpływ trybów szyfrowania na efekt rotacji kwaternionowej. Algorytm zmodyfikowanej sieci Feistela (rozdział 8.3.), zwany dalej QFC (ang. *Quaternion Feistel Cipher*) został przetestowany dla dwóch różnych implementacji:

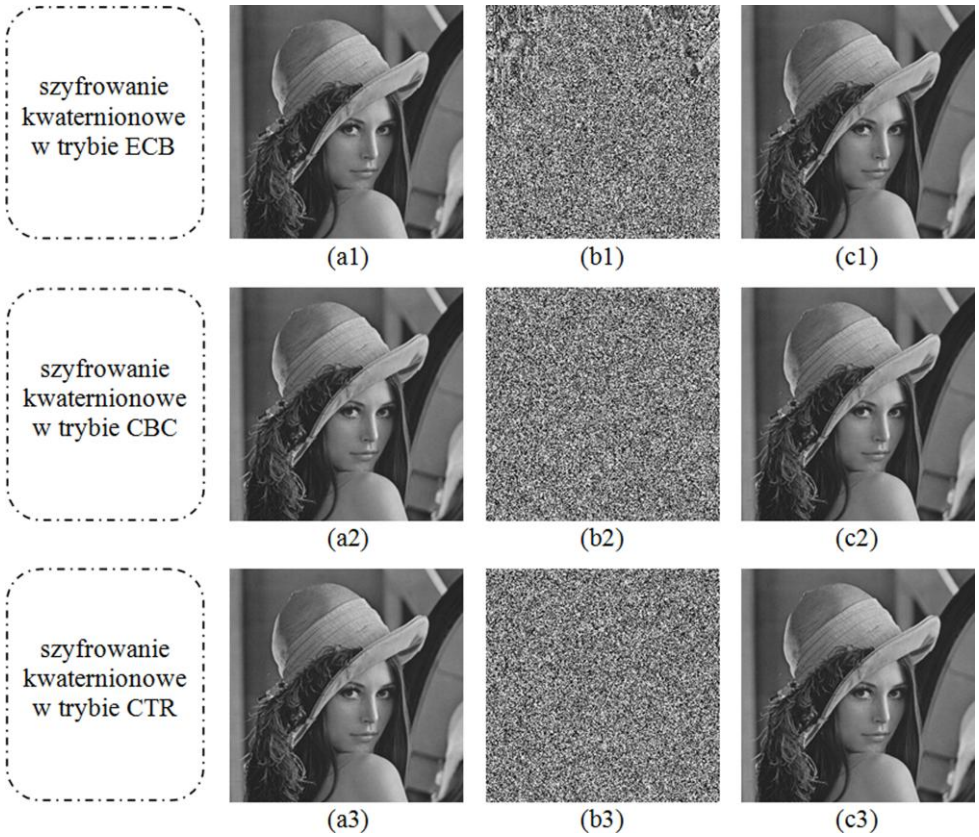
- rozszerzonej S-QFC (ang. *Secure-QFC*), przetestowanej w rozdziale 11.2.
- podstawowej F-QFC (ang. *Fast-QFC*), przetestowanej w rozdziale 11.3.

Dodatkowo, w rozdziale 11.3., algorytm F-QFC został przetestowany pod kątem zastosowania medycznego w szyfrowaniu obrazów DICOM (rozdział 10.1.). W rozdziale 11.4. przedstawiono wyniki testów algorytmu kwaternionowego dla fingerprintingu, omówionego w rozdziale 10.2.

11.1. Wpływ trybów szyfrowania na rotację kwaternionową

W celu zbadania właściwości rotacji kwaternionowej, zastosowano modularne (modulo 257) szyfrowanie kwaternionowe wg. zasady (6.10) i przykładu z Rys. 5. Obraz wejściowy w odcieniach szarości o rozmiarze 243×243 px został podzielony na 9 bloków, każdy o rozmiarze 81×81 px. Każdy z 9-ciu bloków został zaszyfrowany zgodnie z zasadą, obowiązującą dla trzech trybów szyfrowania ECB, CBC i CTR (rozdział 8.1.). Efekt szyfrowania został przedstawiony na Rys. 21.

Rotacja kwaternionowa 9-ciu bloków danych, przeprowadzana jest z wykorzystaniem innego kwaternionu-klucza dla każdego bloku. Przestrzeń 9-ciu kluczy szyfrujących została wygenerowana wykorzystując podstawowy algorytm generacji kluczy (rozdział 6.5.).



Rys. 21. Efekt modularnego szyfrowania kwaternionowego: (a) obraz oryginalny, (b) obraz zaszyfrowany, (c) obraz odszyfrowany.
Źródło: Opracowanie własne.

Na podstawie Rys. 21. można zauważyć, że tylko dla trybów szyfrowania CBC i CTR otrzymać można szyfrogramy pozbawione widocznych artefaktów. Dla szyfrowania w trybie ECB widoczne są niewielkie artefakty. Przy szyfrowaniu danych, podzielonych na mniejsze bloki zalecane jest więc wykorzystanie trybu CBC lub CTR. Tryb szyfrowania CBC lub CTR może zostać z powodzeniem zaimplementowany w algorytmie szyfrującym QFC przedstawionym w rozdziale 8.3.

11.2. Algorytm S-QFC

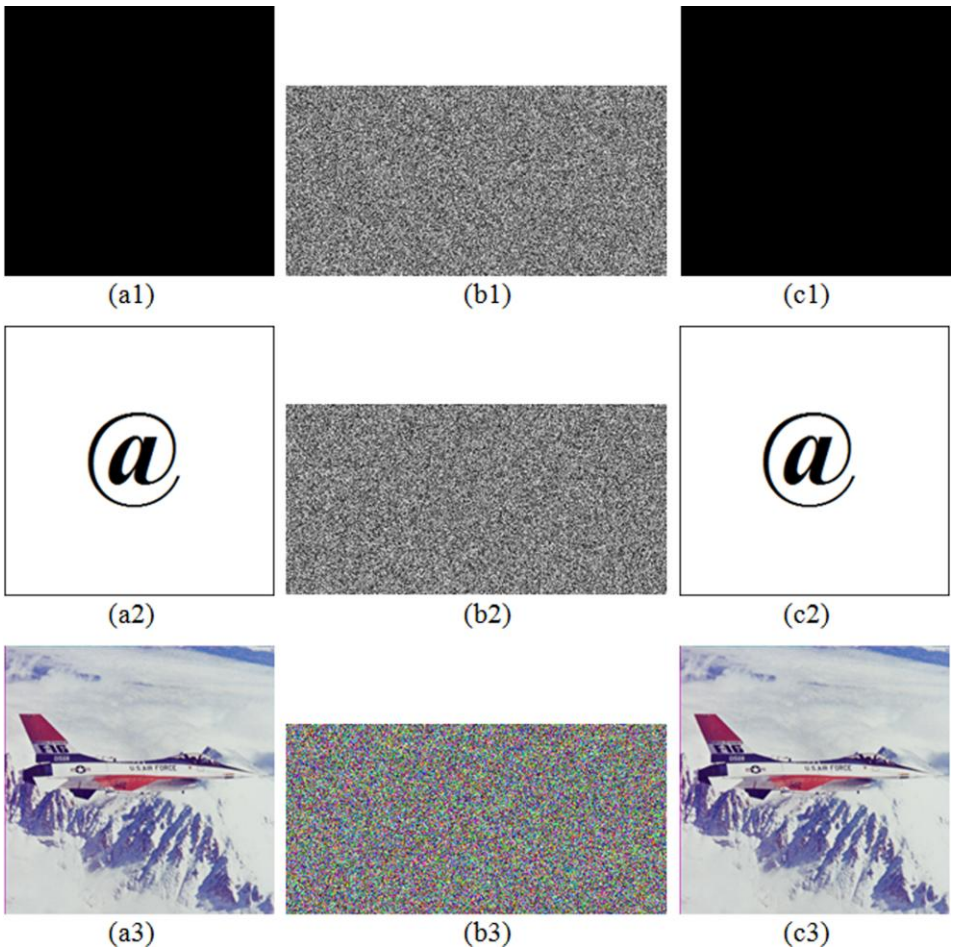
Algorytm zmodyfikowanej sieci Feistela (rozdział 8.3.) w wersji S-QFC wykorzystuje zmodyfikowany algorytm generacji kluczy szyfrujących oparty na kwaternionowych zbiorach Julia (rozdział 9.1.) oraz uwzględnia obustronne mnożenie macierzowe podczas szyfrowania i deszyfracji (rozdział 8.3.1.). Implementacja S-QFC zachowuje właściwości bezstratne - odszyfrowane dane są identyczne z danymi wejściowymi.

Algorytm S-QFC jest przeznaczony do szyfrowania danych 8-bitowych. W przypadku danych wejściowych o większej liczbie bitów stosowany jest podział na kilka części 8-bitowych. Przykład takiego podziału został przedstawiony w rozdziale 10.1.2.

11.2.1. Szyfrowanie tekstów jawnych o różnej postaci

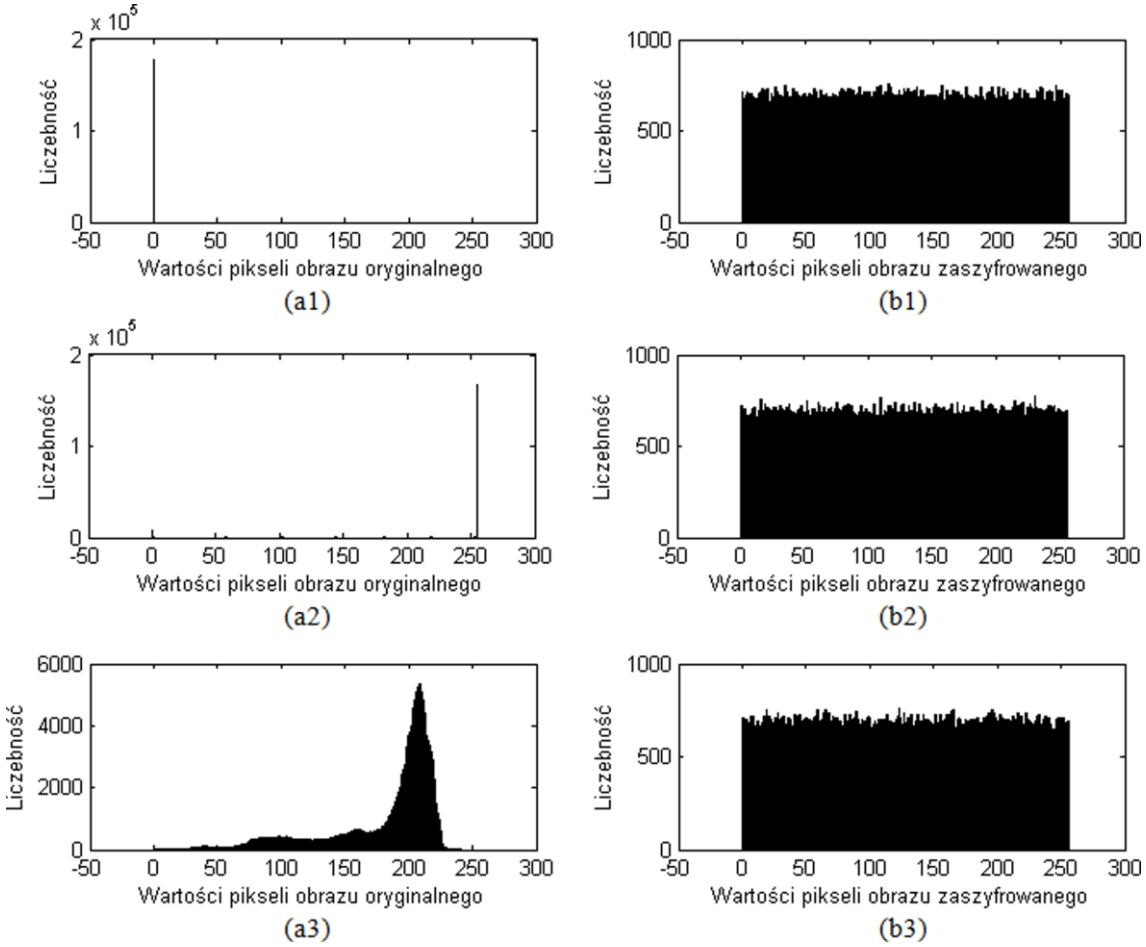
Przykład szyfrowania i deszyfracji różnych obrazów 8-bitowych dla algorytmu S-QFC został przedstawiony na Rys. 22., natomiast odpowiadające tym obrazom wartości pikseli przedstawiono na Rys. 23. W eksperymencie wykorzystano 9 rund algorytmu S-QFC z mnożeniem macierzowym co 3 rundy. Dla uproszczenia, wielkość zastosowanych w eksperymencie obrazów wejściowych jest równa 243×243 px. Wielkość obrazów zaszyfrowanych jest równa 172×344 px (wg. zasady $m \times 2m$ przedstawionej w rozdziale 8.3.).

Zaszyfrowaniu algorytmem S-QFC poddane zostały trzy różne obrazy. Obraz jednolitej czerni o wartościach pikseli równych 0, obraz symbolu @ na białym tle o wartościach pikseli równych 0 i 255 oraz obraz kolorowy RGB przedstawiający myśliwiec F16. Niezależnie od postaci obrazu wejściowego, szyfrowanie algorytmem S-QFC pozwala na uzyskanie szyfrogramów pozbawionych cech wspólnych z obrazem oryginalnym.



Rys. 22. Efekt szyfrowania algorytmem S-QFC: (a) obraz oryginalny, (b) obraz zaszyfrowany, (c) obraz odszyfrowany.
Źródło: Opracowanie własne.

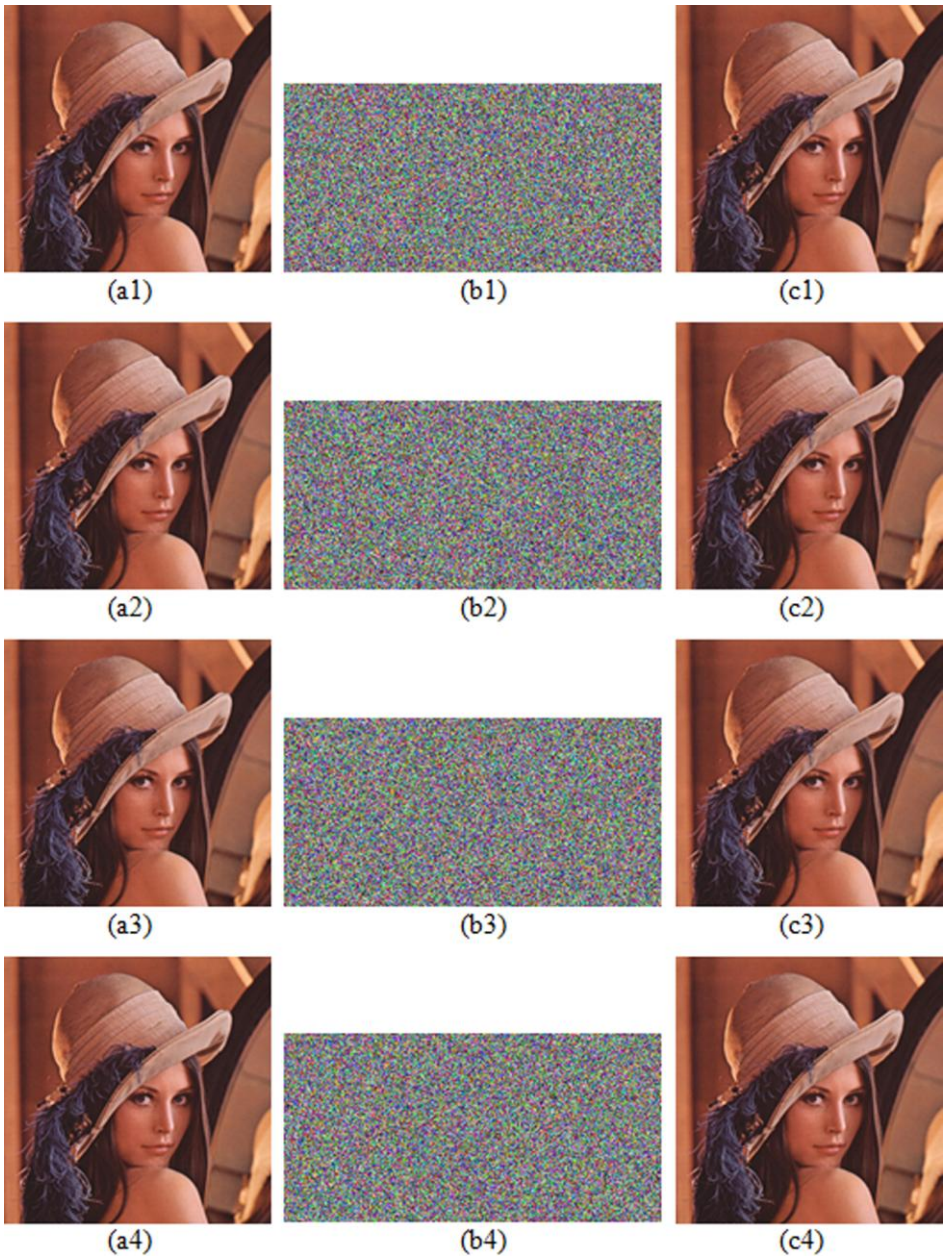
Odpowiednie histogramy dla szyfrogramów z Rys. 22. zostały przedstawione na Rys. 23. Histogramy dla obrazu myśliwca F16 zostały wykreślone z uwzględnieniem wszystkich 3 kanałów RGB. Jak można zauważyć, rozkłady wartości pikseli dla wszystkich obrazów zaszyfrowanych są w przybliżeniu równomierne.



Rys. 23. Histogramy 3 różnych obrazów zaszyfrowanych algorytmem S-QFC: (a) histogram obrazu oryginalnego, (b) histogram obrazu zaszyfrowanego.
Źródło: Opracowanie własne.

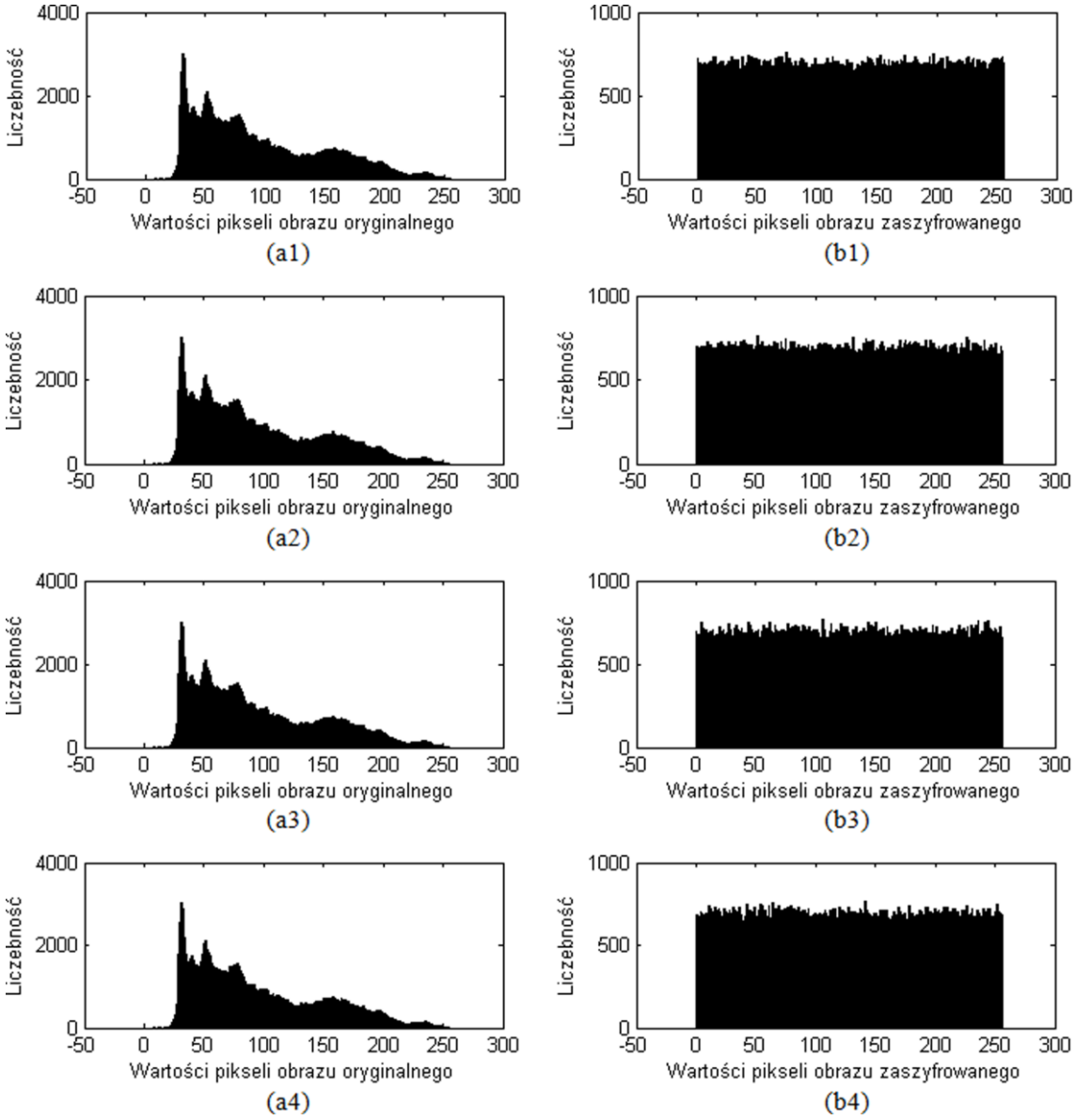
11.2.2. Szyfrowanie dla różnej liczby rund

Kolejny eksperyment dotyczy zbadania wpływu liczby rund w algorytmie S-QFC na jakość uzyskiwanych szyfrogramów. Każda runda algorytmu szyfrowana jest innym kluczem rundowym, a zatem większa liczba rund zwiększa złożoność algorytmu. Dodatkowo co każdą trzecią rundę wykonywane jest mnożenie macierzowe szyfrowanych danych przez macierz klucza **K**. Wynik eksperymentu został zobrazowany na Rys. 24. W eksperymencie zastosowano 3, 9, 27 i 81 rund algorytmu. Testy przeprowadzono na kolorowym obrazie Leny o rozmiarze 243×243 px.



Rys. 24. Wpływ liczby rund w algorytmie S-QFC na szyfrogram: (a) obraz oryginalny, (b1) obraz zaszyfrowany dla 3 rund, (b2) obraz zaszyfrowany dla 9 rund, (b3) obraz zaszyfrowany dla 27 rund, (b4) obraz zaszyfrowany dla 81 rund, (c) obraz odszyfrowany.
Źródło: Opracowanie własne.

Odpowiednie histogramy dla szyfrogramów z Rys. 24. zostały przedstawione na Rys. 25. Histogramy dla każdego obrazu zostały wykreślone z uwzględnieniem wszystkich 3 kanałów RGB. We wszystkich czterech przypadkach, wartości pikseli obrazów zaszyfrowanych opisane są przybliżonym rozkładem równomiernym. Na tej podstawie, dla kolejnych eksperymentów jako zadowalającą liczbę rund algorytmu S-QFC przyjęto liczbę 9.



Rys. 25. Histogramy obrazów zaszyfrowanych algorytmem S-QFC przy zastosowaniu różnej liczebności rund: (a) histogram obrazu oryginalnego, (b1) histogram obrazu zaszyfrowanego dla 3 rund, (b2) histogram obrazu zaszyfrowanego dla 9 rund, (b3) histogram obrazu zaszyfrowanego dla 27 rund, (b4) histogram obrazu zaszyfrowanego dla 81 rund.
Źródło: Opracowanie własne.

11.2.3. Testy losowości

Algorytm S-QFC został dodatkowo poddany testom losowości przy wykorzystaniu darmowej aplikacji CrypTool (CrypTool 2015) oraz pakietu Diehard (Marsaglia 1995). Wyniki testów losowości dla 9-cio rundowego algorytmu S-QFC zostały przedstawione w Tabeli 3. i 4.

Tabela 3. Wyniki testów losowości aplikacji CrypTool dla algorytmu S-QFC.

Test	Wynik	Test	Wynik
Frequency	pass	Serial	pass
Poker	pass	Long-Run	pass
Runs	pass	Mono-Bit	pass

Aplikacja CrypTool umożliwia zbadanie losowości zaszyfrowanych danych wykorzystując zbiór testów NIST (ang. *National Institute of Standards and Technology*). Jak widać w Tabeli 3., dane zaszyfrowane 9-cio rundowym algorytmem S-QFC pomyślnie przeszły wszystkie testy.

Tabela 4. Wyniki testów losowości pakietu Diehard dla algorytmu S-QFC.

Test	*p*-value	Test	*p*-value
Birthday spacings	0.531	The 3Dsphere	0.829
Binary rank	0.480	Up-down runs	0.404
Parking lot	0.642	Craps	0.516
OPERM5	0.871	Minimum distance	0.235
Overlapping sums	0.799		

Wyniki testów losowości pakietu Diehard dla danych zaszyfrowanych 9-cio rundowym algorytmem S-QFC zostały przedstawione w Tabeli 4. Testy Chi-Square, dostępne w pakiecie Diehard, charakteryzuje parametr *p*-value, który przyjmuje wartości od 0 do 1. Jeżeli dany test zwróci wartość *p*-value równą dokładnie 0 lub 1, to test uznaje się za niezdany (Marsaglia 1995). Najbardziej pożądaną wartością *p*-value jest wartość 0.5, jednakże nawet wartości bardzo bliskie 0 lub 1 interpretuje się pozytywnie, jako pomyślne przejście testu (Marsaglia 1995).

Pakiet Diehard wykorzystywany jest głównie jako narzędzie do badania właściwości generatorów losowych RNG (ang. *Random Number Generator*) i w niniejszej pracy został zastosowany opcjonalnie do sprawdzenia losowości szyfrogramów. Głównym wyznacznikiem właściwości losowych dla zaszyfrowanych danych są testy NIST (przeprowadzone za pomocą aplikacji CrypTool).

11.2.4. Efekt lawinowy

W kolejnym eksperymencie zbadano efekt lawinowy dla 9-cio rundowego algorytmu S-QFC szyfrującego kolorowy obraz Leny. W tym celu dokonano zmiany 1 bitu w: pojedynczym kluczu rundowym q_i, w kluczu inicjalizującym q_0, w pojedynczej macierzy klucza **K** oraz w tekście jawnym.

Po zmianie wartości 1 bitu w postaci binarnej pojedynczego, dowolnego klucza rundowego q_i w 9-cio rundowym algorytmie S-QFC otrzymano szyfrogram, którego postać binarna różni się w przybliżeniu o 50% od postaci binarnej oryginalnego

szyfrogramu (bez modyfikacji).

Modyfikacja 1 bitu postaci binarnej klucza inicjalizującego q_0 w 9-cio rundowym algorytmie S-QFC daje szyfrogram, którego postać binarna różni się w przybliżeniu o 50% od postaci binarnej oryginalnego szyfrogramu (bez modyfikacji).

Modyfikując 1 bit postaci binarnej macierzy klucza **K** w 9-cio rundowym algorytmie S-QFC otrzymano szyfrogram, którego postać binarna różni się w przybliżeniu o 50% od postaci binarnej oryginalnego szyfrogramu (bez modyfikacji).

Modyfikując 1 bit postaci binarnej tekstu jawnego w 9-cio rundowym algorytmie S-QFC otrzymano szyfrogram, którego postać binarna różni się w przybliżeniu o 50% od postaci binarnej oryginalnego szyfrogramu (bez modyfikacji). Właściwość ta jest charakterystyczna tylko dla implementacji S-QFC z uwagi na zastosowanie obustronnego mnożenia macierzowego, gdzie każdy element szyfrowanego tekstu jawnego jest wyrażony za pomocą wszystkich pozostałych elementów (rozdział 8.3.1.).

Dodatkowo, rozważmy przykład zobrazowany na Rys. 26. Do zaszyfrowania kolorowego obrazu Leny wykorzystano 9-cio rundowy algorytm S-QFC. Każda runda algorytmu szyfrowana jest innym kluczem. Zakładamy, że potencjalny haker dysponuje informacją dotyczącą binarnej reprezentacji macierzy klucza **K** oraz 8 kluczy rundowych. Postać binarna 9-tego klucza rundowego, którym dysponuje haker, różni się jedynie 1 bitem względem postaci binarnej klucza oryginalnego.

(a) (b)

Rys. 26. Deszyfracja błędnym kluczem przeprowadzona dla algorytmu S-QFC: (a) obraz oryginalny, (b) obraz odszyfrowany przy zastosowaniu innego klucza rundowego (różnica na 1 bicie względem klucza oryginalnego).
Źródło: Opracowanie własne.

Wynik deszyfracji danych przy wykorzystaniu kluczy hakera został przedstawiony na Rys. 26b. Haker, pomimo posiadanej wiedzy nt. kluczy szyfrujących, nie zdoła poprawnie odszyfrować danych.

11.2.5. Szybkość obliczeniowa

W ostatnim eksperymencie porównano szybkość obliczeniową 9-cio rundowego algorytmu S-QFC z algorytmem AES. W tym celu wykorzystano implementację AES-ECB dla środowiska symulacyjnego MATLAB, autorstwa prof. Jörg J. Buchholz (Buchholz 2001), pracującą w dokładnie takich samych warunkach co przedstawiony w rozprawie algorytmy S-QFC. Zastosowanie implementacji referencyjnych dla algorytmu AES, wykonanych w innym środowisku niż MATLAB nie przedstawiłoby

miarodajnego porównania.

Obydwa algorytmy przetestowano na tej samej maszynie tj. Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz, 16 GB RAM, 64-bit Win 7 Pro SP1; środowisko symulacyjne: MATLAB. Wyniki testów dla obrazów kolorowych o różnej wielkości zostały przedstawione w Tabeli 5. Wartości oczekiwane i odpowiadające im przedziały ufności wyznaczono na podstawie puli 20 symulacji, przeprowadzonych indywidualnie dla każdego obrazu. Z uwagi na małą liczbę przeprowadzonych symulacji zastosowano rozkład *t*-Studenta (Armitage i in. 2008). Wartości oczekiwane, przedstawione w Tabeli 5., są estymacją wartości oczekiwanej μ, wyznaczonej na podstawie wzoru (Armitage i in. 2008):

$$\mathrm{P}\left(\overline{X} - t_\alpha \frac{S}{\sqrt{N-1}} < \mu < \overline{X} + t_\alpha \frac{S}{\sqrt{N-1}}\right) = 1-\alpha \tag{11.1}$$

gdzie $\overline{X}$ to wartość oczekiwana z próby, S to odchylenie standardowe z próby, N to wielkość próby (20 symulacji), t_α to wartość odczytana z tablicy *t*-Studenta dla N-1 stopni swobody, natomiast 1-α to współczynnik ufności równy 95%.

Tabela 5. Szybkość obliczeniowa dla algorytmów AES-ECB i S-QFC, wyznaczona dla 3 obrazów kolorowych o różnej wielkości.

	243×243 px		512×512 px		1024×1024 px	
AES-ECB	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$
Inicjalizacja [s]	6.259e-01	5.693e-03	6.606e-01	5.329e-02	6.965e-01	1.327e-01
Szyfrowanie [s]	7.065e+01	2.167e-01	3.139e+02	2.455e+00	1.214e+03	1.616e+01
Deszyfracja [s]	1.007e+02	3.616e-01	4.478e+02	4.656e+00	1.733e+03	2.106e+01
Całkowity czas [s]	1.720e+02	-	7.624e+02	-	2.949e+03	-
S-QFC	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$
Inicjalizacja [s]	5.979e-02	8.488e-03	6.586e-02	8.855e-03	1.055e-01	8.398e-03
Szyfrowanie [s]	1.347e+00	4.373e-03	9.065e+00	7.649e-02	1.076e+02	2.944e-01
Deszyfracja [s]	1.345e+00	2.683e-03	8.945e+00	8.876e-02	1.078e+02	3.776e-01
Całkowity czas [s]	2.753e+00	-	1.807e+01	-	2.156e+02	-

Czas inicjalizacji to czas potrzebny na wyznaczenie wszystkich parametrów inicjalizujących (wartości współczynników, wektorów, macierzy itp.), wymaganych do przeprowadzenia szyfrowania i deszyfracji. Dla algorytmu AES parametry inicjalizujące to: tablica podstawień S-Box i jej odwrotność, 16-bajtowy klucz startowy i jego rozszerzenie do klucza rundowego, macierz współczynników wielomianu i jej

odwrotność. Dla algorytmu S-QFC parametry inicjalizujące to: współczynniki klucza inicjalizującego q_0, wielkość rzędu rotacji do wygenerowania zadanej przestrzeni kluczy rundowych, parametry inicjalizujące do wygenerowania fraktala kwaternionowego zbioru Julia (rozdział 9.1.), macierz klucza **K** i jej odwrotność.

Na podstawie przedstawionych w Tabeli 5. wyników, można zaobserwować jedną z największych zalet algorytmu QFC - bardzo duża szybkość obliczeniowa z uwagi na zastosowanie rotacji kwaternionowych. Większa liczba rund dla algorytmu S-QFC nieznaczne pogorszy uzyskaną wydajność, ale jednocześnie zapewni wyższy poziom bezpieczeństwa z uwagi na zastosowanie większej przestrzeni kluczy.

Zauważalna różnica w pomierzonych czasach wydajności obydwu algorytmów wynika przede wszystkim z faktu, że algorytm AES przeprowadza szyfrowanie małych fragmentów obrazu, dla wszystkich 3 kanałów osobno. Algorytm S-QFC szyfruje jednorazowo cały obraz, dla wszystkich 3 kanałów jednocześnie (Rys. 9.).

Dodatkowo, szyfrowanie algorytmem AES może zająć jeszcze więcej czasu, gdyż na ogół algorytm ten implementowany jest w mniej wydajnym lecz bezpieczniejszym trybie, takim jak AES-CBC, AES-CTR, AES-OFB czy AES-OCB.

11.3. Algorytm F-QFC

Algorytm zmodyfikowanej sieci Feistela (rozdział 8.3.) w wersji F-QFC wykorzystuje podstawowy algorytm generacji kluczy szyfrujących (rozdział 6.5.) bez obustronnego mnożenia macierzowego podczas szyfrowania i deszyfracji. Implementacja F-QFC zachowuje właściwości bezstratne - odszyfrowane dane są identyczne z danymi wejściowymi.

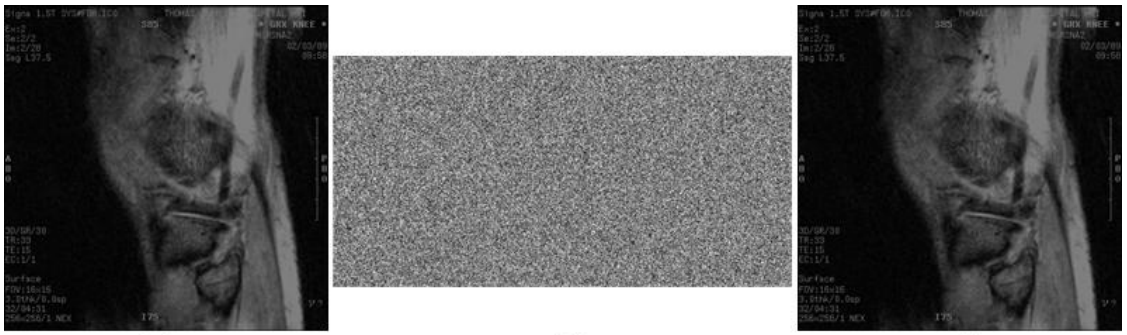
Algorytm F-QFC jest przeznaczony do szyfrowania danych 8-bitowych (tak jak S-QFC idealnie nadaje się do szyfrowania obrazów kolorowych RGB). W nawiązaniu do rozdziału 10.1. oraz pracy (Dzwonkowski i in. 2015), algorytm F-QFC przetestowany został pod kątem zastosowania medycznego w szyfrowaniu 16-bitowych obrazów DICOM.

11.3.1. Szyfrowanie obrazu DICOM

W celu zaszyfrowania 16-bitowego obrazu DICOM należy dokonać podziału danych wejściowych na dwa 8-bitowe obrazy według schematu przedstawionego w rozdziale 10.1.2.

Przykład szyfrowania i deszyfracji pojedynczego obrazu 8-bitowego przy wykorzystaniu algorytmu F-QFC został przedstawiony na Rys. 27. Przykład szyfrowania części tekstowej obrazu DICOM został przedstawiony na Rys. 28.

W eksperymencie wykorzystano 9 rund algorytmu F-QFC. Dla uproszczenia, wielkość zastosowanego w eksperymencie obrazu wejściowego jest równa 512×512 px. Wielkość obrazu zaszyfrowanego jest równa 363×726 px (wg. zasady $m \times 2m$ przedstawionej w rozdziale 8.3.).



(a) (b) (c)

Rys. 27. Szyfrowanie i deszyfracja obrazu DICOM: (a) obraz oryginalny, (b) obraz zaszyfrowany, (c) obraz odszyfrowany.
Źródło: Opracowanie własne / Computer Assisted Radiology and Surgery 11th International Symposium and Exhibition, Berlin, June 25-28, 1997

```
                    Filename: 'C:\Users\oem\Desktop\IM1'
                FileModDate: '28-maj-2014 14:42:11'
                    FileSize: 525442
                      Format: 'DICOM'
                FormatVersion: 3
                       Width: 512
                      Height: 512
                    BitDepth: 16
                   ColorType: 'grayscale'
FileMetaInformationGroupLength: 186
      MediaStorageSOPClassUID: '1.2.840.10008.5.1.4.1.1.7'
  MediaStorageSOPInstanceUID: '1.2.840.113619.2.1.1.318790346.551
           TransferSyntaxUID: '1.2.840.10008.1.2.1'
     ImplementationClassUID: '1.2.840.113619.6.36'
  ImplementationVersionName: '1_2_5'
SourceApplicationEntityTitle: 'DTIR'
```

(a)

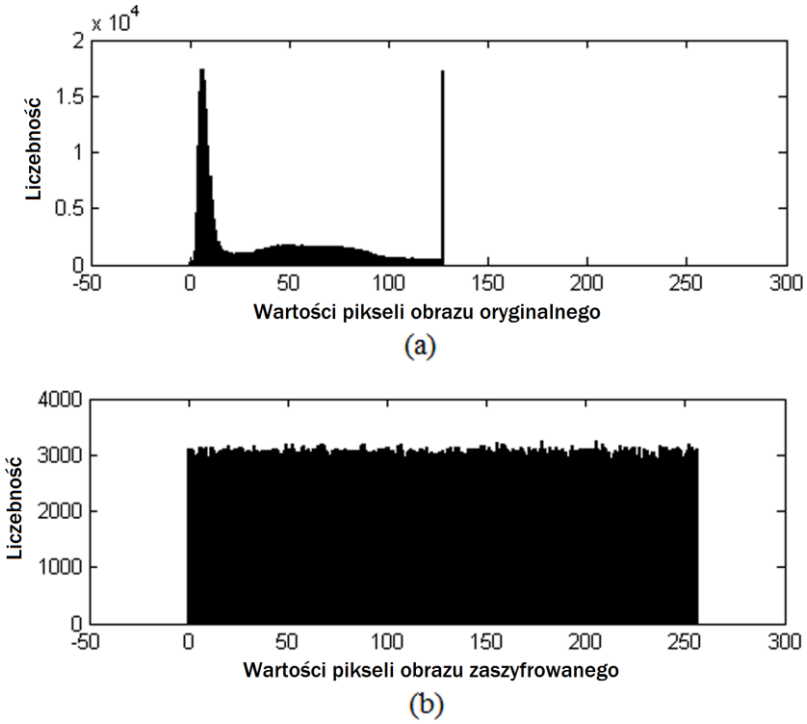
```
G)]-ç $l TýH¦  Á[K$âm  ç ß ËL   á vˆéI   =)T°ý    â
îeÎm  Aácë÷  X B    îf <Bv 1¾ ;) ÂÉ   @ Ó   ~
¦
Gq_)Ç` ] kq¦ ITKBm# O  9<  ©I P  3*     I     5Ë Î
 µ©  ¦  k ¸ KuL   &yTí&
VC~  j  qM ü  &/|   ÂS;
6 ¤ ×q á s˜kkm& U'¬$  # Q-=   g öB   X   VÅ  #ö Ötz
   l ß V rRú  J 1"¦Q +<) qá     a2 /   V ÓzÇ  _°ô
 Q ¨ÝÓ P
  ¶)l= ¦  U'Up   Ô)      ˆ \  @ß!  aë;_ó¶

 \aß@ @ÂBß   9(Ú7 J 11      ´ Ý  .iS  Á = ) ß7 Ép M
J(  _   F  y D˜°íZÖ Qb±] z  R;¦´.  *Í&\ (]j  ]+ N` y
   ox× x)p P.  Js°< ×«D¨» Ü aÅ·   2  ßq×M öX V  ) (
 Ú×   D bˆ Î O  ¦  W    ÝÅéU~±<Q Ë:  ¸ó   IG   ób¾
```

(b)

Rys. 28. Szyfrowanie części tekstowej obrazu DICOM: (a) część tekstowa oryginalna, (b) część tekstowa zaszyfrowana.
Źródło: Opracowanie własne / Computer Assisted Radiology and Surgery 11th International Symposium and Exhibition, Berlin, June 25-28, 1997

Odpowiadające histogramy dla obrazu DICOM z Rys. 27. zostały przedstawione na Rys. 29. Podobnie jak w przypadku szyfrowania algorytmem S-QFC, wartości pikseli obrazu zaszyfrowanego mają w przybliżeniu rozkład równomierny.



Rys. 29. Histogramy dla obrazu DICOM: (a) histogram obrazu oryginalnego, (b) histogram obrazu zaszyfrowanego.
Źródło: Opracowanie własne.

11.3.2. Testy losowości

Algorytm F-QFC został dodatkowo poddany testom losowości przy wykorzystaniu darmowej aplikacji CrypTool (CrypTool 2015) oraz pakietu Diehard (Marsaglia 1995). Wyniki testów losowości dla 9-cio rundowego algorytmu F-QFC zostały przedstawione w Tabeli 6. i 7.

Tabela 6. Wyniki testów losowości aplikacji CrypTool dla algorytmu F-QFC.

Test	Wynik	Test	Wynik
Frequency	pass	Serial	pass
Poker	pass	Long-Run	pass
Runs	pass	Mono-Bit	pass

Aplikacja CrypTool umożliwia zbadanie losowości zaszyfrowanych danych wykorzystując zbiór testów NIST (ang. *National Institute of Standards and Technology*). Na podstawie Tabeli 6., dane zaszyfrowane 9-cio rundowym algorytmem F-QFC pomyślnie przeszły wszystkie testy.

Tabela 7. Wyniki testów losowości pakietu Diehard dla algorytmu F-QFC.

Test	*p*-value	Test	*p*-value
Birthday spacings	0.559	The 3Dsphere	0.914
Binary rank	0.501	Up-down runs	0.369
Parking lot	0.552	Craps	0.580
OPERM5	0.722	Minimum distance	0.858
Overlapping sums	0.848		

Wyniki testów losowości pakietu Diehard dla danych zaszyfrowanych 9-cio rundowym algorytmem F-QFC zostały przedstawione w Tabeli 7. Testy Chi-Square, dostępne w pakiecie Diehard, charakteryzuje parametr *p*-value, który przyjmuje wartości od 0 do 1. Jeżeli dany test zwróci wartość *p*-value równą dokładnie 0 lub 1, to test uznaje się za niezdany (Marsaglia 1995). Najbardziej pożądaną wartością *p*-value jest wartość 0.5, jednakże nawet wartości bliskie 0 lub 1 interpretuje się pozytywnie, jako pomyślne przejście testu (Marsaglia 1995).

11.3.3. Efekt lawinowy

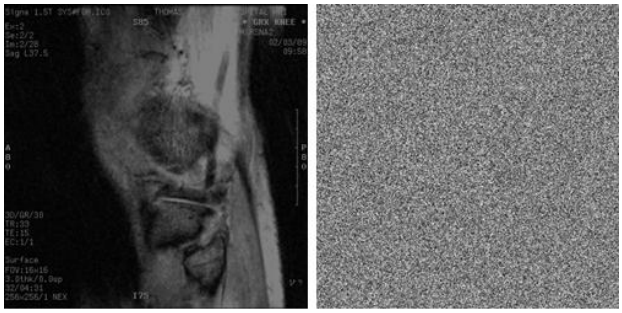
W kolejnym eksperymencie zbadano efekt lawinowy dla 9-cio rundowego algorytmu F-QFC przy szyfrowaniu obrazu DICOM. W tym celu dokonano zmiany 1 bitu w: pojedynczym kluczu rundowym q_i, w kluczu inicjalizującym q_0, oraz w tekście jawnym.

Po zmodyfikowaniu 1 bitu w postaci binarnej pojedynczego, dowolnego klucza rundowego q_i w 9-cio rundowym algorytmie F-QFC otrzymano szyfrogram, którego postać binarna różni się w przybliżeniu o 50% od postaci binarnej oryginalnego szyfrogramu (bez modyfikacji).

Modyfikując 1 bit postaci binarnej klucza inicjalizującego q_0 w 9-cio rundowym algorytmie F-QFC otrzymano szyfrogram, którego postać binarna różni się w przybliżeniu o 50% od postaci binarnej oryginalnego szyfrogramu (bez modyfikacji).

Modyfikując 1 bit postaci binarnej tekstu jawnego w 9-cio rundowym algorytmie F-QFC otrzymano szyfrogram, którego postać binarna jest bardzo zbliżona do postaci binarnej oryginalnego szyfrogramu (bez modyfikacji) - brak efektu lawinowego.

Dodatkowo, rozważmy przykład zobrazowany na Rys. 30. Do zaszyfrowania obrazu DICOM wykorzystano 9-cio rundowy algorytm F-QFC. Każda runda algorytmu szyfrowana jest innym kluczem. Zakładamy że potencjalny haker dysponuje informacją dotyczącą binarnej reprezentacji 8 kluczy rundowych. Postać binarna 9-tego klucza rundowego, którym dysponuje haker, różni się jedynie 1 bitem względem postaci binarnej klucza oryginalnego.



(a) (b)

Rys. 30. Deszyfracja błędnym kluczem przeprowadzona dla algorytmu F-QFC: (a) obraz oryginalny, (b) obraz odszyfrowany przy zastosowaniu innego klucza rundowego (różnica na 1 bicie względem klucza oryginalnego).
Źródło: Opracowanie własne.

Deszyfracja danych przy wykorzystaniu kluczy hakera została przedstawiona na Rys. 30b. Haker, pomimo posiadanej wiedzy nt. kluczy szyfrujących, nie zdoła poprawnie odszyfrować danych.

11.3.4. Szybkość obliczeniowa

W ostatnim eksperymencie porównano szybkość obliczeniową 9-cio rundowego algorytmu F-QFC z algorytmem AES. W tym celu wykorzystano implementację algorytmu AES-ECB autorstwa prof. Jörg J. Buchholz (Buchholz 2001). Obydwa algorytmy przetestowano na tej samej maszynie tj. Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz, 16 GB RAM, 64-bit Win 7 Pro SP1; środowisko symulacyjne: MATLAB. Wyniki testów dla obrazów DICOM o różnej wielkości zostały przedstawione w Tabeli 8. Wartości oczekiwane i odpowiadające im przedziały ufności wyznaczono na podstawie puli 20 symulacji, przeprowadzonych indywidualnie dla każdego obrazu. Z uwagi na małą liczbę przeprowadzonych symulacji zastosowano rozkład *t*-Studenta (Armitage i in. 2008). Wartości oczekiwane, przedstawione w Tabeli 8., są estymacją wartości oczekiwanej μ, wyznaczonej na podstawie wzoru (11.1).

Tabela 8. Szybkość obliczeniowa dla algorytmów AES-ECB i F-QFC, wyznaczona dla 3 obrazów DICOM o różnej wielkości.

	512×512 px		1024×1024 px		2048×2048 px	
AES-ECB	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$
Inicjalizacja [s]	6.294e-01	2.999e-02	6.394e-01	1.634e-02	6.316e-01	2.007e-02
Szyfrowanie [s]	1.002e+02	1.561e-01	4.148e+02	1.387e+00	1.647e+03	1.129e+01
Deszyfracja [s]	1.446e+02	2.276e-01	6.010e+02	2.061e+00	2.380e+03	1.723e+01
Całkowity czas [s]	2.454e+02	-	1.016e+03	-	4.028e+03	-

F-QFC	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$	$\overline{X}$	$\frac{t_\alpha S}{\sqrt{N-1}}$
Inicjalizacja [s]	2.516e-03	2.894e-04	2.600e-03	3.433e-04	2.615e-03	3.356e-04
Szyfrowanie [s]	9.662e-01	1.166e-03	3.757e+00	2.889e-03	1.467e+01	6.498e-03
Deszyfracja [s]	9.620e-01	2.167e-04	3.758e+00	1.899e-03	1.474e+01	4.919e-03
Całkowity czas [s]	1.931e+00	-	7.518e+00	-	2.941e+01	-

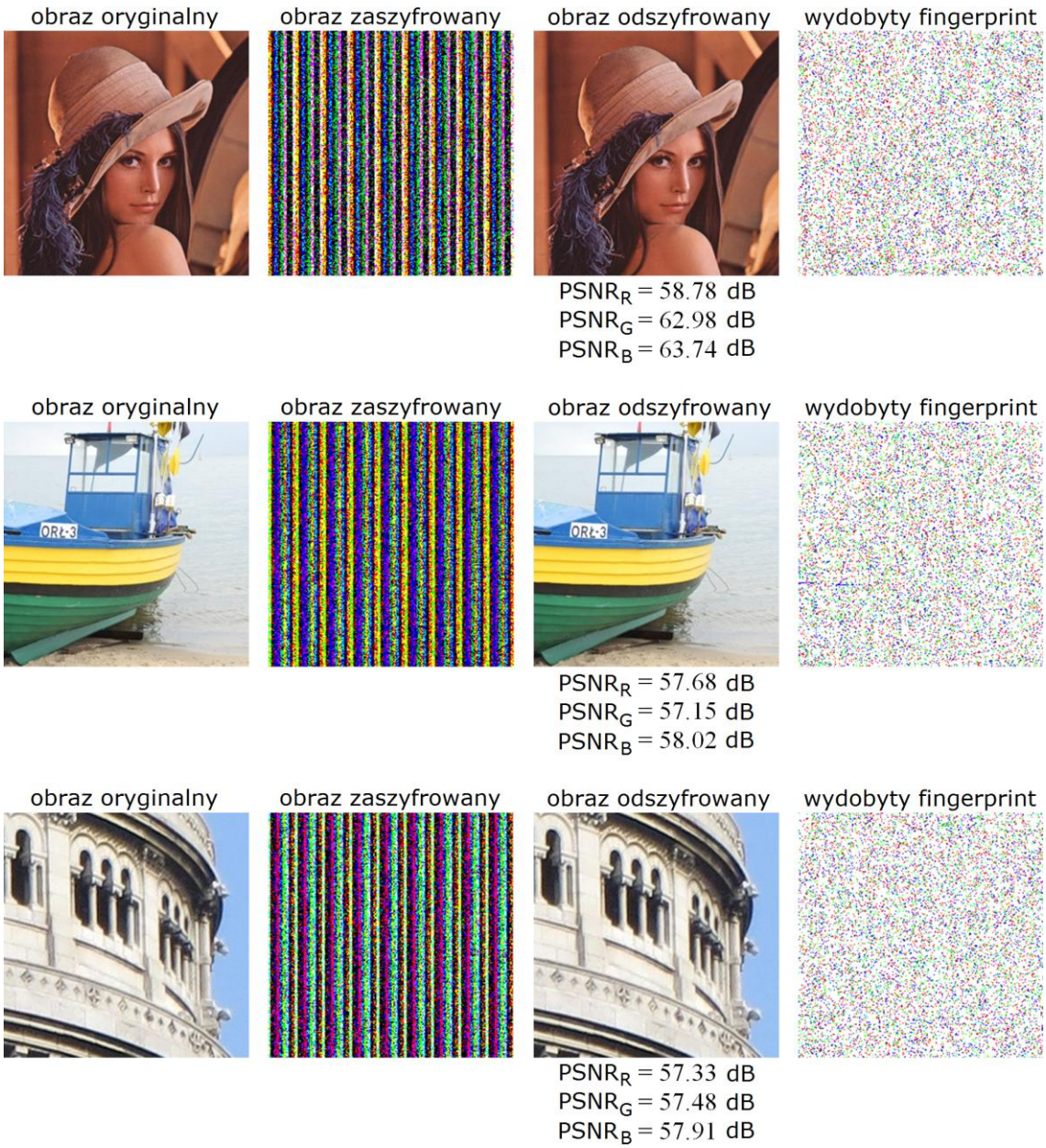
Czas inicjalizacji to czas potrzebny na wyznaczenie wszystkich parametrów inicjalizujących (wartości współczynników, wektorów, macierzy itp.), wymaganych do przeprowadzenia szyfrowania i deszyfracji. Dla algorytmu AES parametry inicjalizujące to: tablica podstawień S-Box i jej odwrotność, 16-bajtowy klucz startowy i jego rozszerzenie do klucza rundowego, macierz współczynników wielomianu i jej odwrotność. Dla algorytmu F-QFC parametry inicjalizujące to: współczynniki klucza inicjalizującego q_0 oraz wielkość rzędu rotacji do wygenerowania zadanej przestrzeni kluczy rundowych.

Na podstawie przedstawionych w Tabeli 8. wyników, można zaobserwować jeszcze większą szybkość przetwarzania danych niż w przypadku algorytmu S-QFC. Większa liczba rund dla algorytmu F-QFC nieznaczne pogorszy uzyskaną wydajność ale jednocześnie zapewni wyższy poziom bezpieczeństwa, z uwagi na zastosowanie większej przestrzeni kluczy.

11.4. Algorytm dla fingerprintingu

Algorytm kwaternionowy omówiony w rozdziale 10.2. oraz pracach (Dzwonkowski i Rykaczewski 2012, Czaplewski i in. 2014) został przetestowany pod kątem zastosowania dla fingerpintingu. Omawiany algorytm to jedna z pierwszych implementacji szyfrowania kwaternionowego. Algorytm nie realizuje koncepcji sieci Feistela, wykonuje jedynie pojedyncze rotacje kwaternionowe, przeprowadzane dla danych podzielonych na mniejsze fragmenty. Algorytm wykorzystuje tryb łączenia bloków CBC podczas szyfrowania i deszyfracji.

W nawiązaniu do opisu w rozdziale 10.2. algorytm pozwala na wprowadzenie niewykrywalnych dla ludzkiego oka błędów w odszyfrowanym obrazie. Uzyskiwane błędy wykorzystywane są jako unikatowy fingerprint, identyfikujący osobę odszyfrowującą dane. Przykład osadzania fingerprintów w odszyfrowanych obrazach został przedstawiony na Rys. 31. W przykładzie zastosowano zaokrąglenie do 4 miejsc po przecinku dla strony deszyfrującej. Uzyskane wartości PSNR dla odszyfrowanych obrazów świadczą o wizualnej niewykrywalności osadzonych fingerprintów.



Rys. 31. Przykład osadzania fingerprintów dla trzech różnych obrazów kolorowych, przy wykorzystaniu zaokrąglenia do 4 miejsc po przecinku dla strony deszyfrującej. *Źródło: Opracowanie własne*.

Wartości PSNR, informujące o jakości wizualnej odszyfrowanego obrazu, zostały wyznaczone wykorzystując wzór:

$$\mathrm{PSNR} = 10\log_{10}\frac{255^2}{\frac{1}{M\cdot N}\sum_{m=1}^{M}\sum_{n=1}^{N}(d_{m,n}-x_{m,n})^2}\ [\mathrm{dB}] \qquad (11.2)$$

gdzie PSNR to szczytowy stosunek sygnału do szumu dla pojedynczego kanału obrazu kolorowego, $d_{m,n}$ to wartość piksela dla m-tego wiersza i n-tej kolumny w odszyfrowanym obrazie, $x_{m,n}$ to wartość piksela dla m-tego wiersza i n-tej kolumny w obrazie oryginalnym, $M \times N$ to wielkość obrazu.

Wysoka wartość PSNR oznacza akceptowalną jakość odszyfrowanego obrazu z osadzonym fingerprintem. Typowe wartości PSNR dla obrazów 8-bitowych z kompresją stratną zawarte są w przedziale 30-50 dB (Welstead 1999, Barni 2006).

Poziom niewykrywalności fingerprintów zbadano dla trzech różnych obrazów kolorowych RGB, przy zastosowaniu różnych wartości zaokrąglenia po stronie deszyfrującej. Tabela 9. przedstawia liczby uzyskanych błędów oraz wartości PSNR dla każdego przypadku. Testy przeprowadzone zostały dla obrazów kolorowych wielkości 243×243 px.

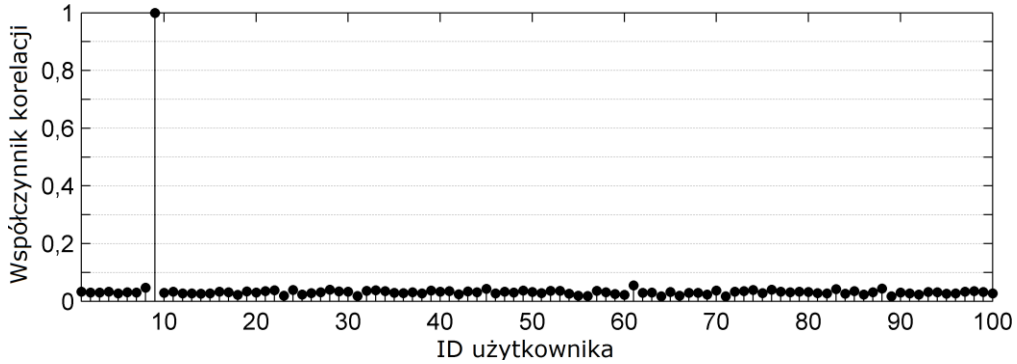
Tabela 9. Wartości PSNR oraz liczby uzyskanych błędów przy zastosowaniu różnych wartości zaokrąglenia podczas deszyfracji dla 3 obrazów kolorowych RGB.

	Zaokrąglenie do 1 miejsca po przecinku	Zaokrąglenie do 2 miejsc po przecinku	Zaokrąglenie do 3 miejsc po przecinku	Zaokrąglenie do 4 miejsc po przecinku	Zaokrąglenie do 5 miejsc po przecinku
Obraz Lena	$PSNR_R$ = 24.93 dB $PSNR_G$ = 28.61 dB $PSNR_B$ = 27.42 dB $\Delta_{(0,3>}$ = 56971 (32%) $\Delta_{(3,7>}$ = 3444 (2%) $\Delta_{(7,15>}$ = 1479 (1%) Δ_{15+} = 2149 (1%)	$PSNR_R$ = 49.52 dB $PSNR_G$ = 53.34 dB $PSNR_B$ = 54.28 dB $\Delta_{(0,3>}$ = 34355 (19%) $\Delta_{(3,7>}$ = 907 (1%) $\Delta_{(7,15>}$ = 144 (0%) Δ_{15+} = 0 (0%)	$PSNR_R$ = 53.80 dB $PSNR_G$ = 57.29 dB $PSNR_B$ = 58.41 dB $\Delta_{(0,3>}$ = 22722 (13%) $\Delta_{(3,7>}$ = 322 (0%) $\Delta_{(7,15>}$ = 0 (0%) Δ_{15+} = 0 (0%)	$PSNR_R$ = 58.78 dB $PSNR_G$ = 62.98 dB $PSNR_B$ = 63.74 dB $\Delta_{(0,3>}$ = 11675 (7%) $\Delta_{(3,7>}$ = 0 (0%) $\Delta_{(7,15>}$ = 0 (0%) Δ_{15+} = 0 (0%)	$PSNR_R$ = 64.52 dB $PSNR_G$ = 69.58 dB $PSNR_B$ = 70.58 dB $\Delta_{(0,3>}$ = 4900 (3%) $\Delta_{(3,7>}$ = 0 (0%) $\Delta_{(7,15>}$ = 0 (0%) Δ_{15+} = 0 (0%)
Obraz Łódź	$PSNR_R$ = 27.23 dB $PSNR_G$ = 26.38 dB $PSNR_B$ = 27.28 dB $\Delta_{(0,3>}$ = 51076 (29%) $\Delta_{(3,7>}$ = 4362 (2%) $\Delta_{(7,15>}$ = 1869 (1%) Δ_{15+} = 2951 (2%)	$PSNR_R$ = 48.32 dB $PSNR_G$ = 48.86 dB $PSNR_B$ = 48.93 dB $\Delta_{(0,3>}$ = 37269 (21%) $\Delta_{(3,7>}$ = 1450 (1%) $\Delta_{(7,15>}$ = 291 (0%) Δ_{15+} = 0 (0%)	$PSNR_R$ = 52.90 dB $PSNR_G$ = 52.81 dB $PSNR_B$ = 52.96 dB $\Delta_{(0,3>}$ = 23270 (13%) $\Delta_{(3,7>}$ = 572 (0%) $\Delta_{(7,15>}$ = 0 (0%) Δ_{15+} = 0 (0%)	$PSNR_R$ = 57.68 dB $PSNR_G$ = 57.15 dB $PSNR_B$ = 58.02 dB $\Delta_{(0,3>}$ = 14037 (8%) $\Delta_{(3,7>}$ = 0 (0%) $\Delta_{(7,15>}$ = 0 (0%) Δ_{15+} = 0 (0%)	$PSNR_R$ = 64.15 dB $PSNR_G$ = 63.56 dB $PSNR_B$ = 64.04 dB $\Delta_{(0,3>}$ = 5819 (3%) $\Delta_{(3,7>}$ = 0 (0%) $\Delta_{(7,15>}$ = 0 (0%) Δ_{15+} = 0 (0%)
Obraz Wieża	$PSNR_R$ = 26.32 dB $PSNR_G$ = 27.17 dB $PSNR_B$ = 27.59 dB $\Delta_{(0,3>}$ = 51852 (29%) $\Delta_{(3,7>}$ = 4306 (2%) $\Delta_{(7,15>}$ = 1828 (1%) Δ_{15+} = 2804 (2%)	$PSNR_R$ = 49.49 dB $PSNR_G$ = 48.74 dB $PSNR_B$ = 48.96 dB $\Delta_{(0,3>}$ = 34087 (19%) $\Delta_{(3,7>}$ = 1326 (1%) $\Delta_{(7,15>}$ = 262 (0%) Δ_{15+} = 0 (0%)	$PSNR_R$ = 52.48 dB $PSNR_G$ = 52.61 dB $PSNR_B$ = 52.86 dB $\Delta_{(0,3>}$ = 24677 (14%) $\Delta_{(3,7>}$ = 639 (0%) $\Delta_{(7,15>}$ = 0 (0%) Δ_{15+} = 0 (0%)	$PSNR_R$ = 57.33 dB $PSNR_G$ = 57.48 dB $PSNR_B$ = 57.91 dB $\Delta_{(0,3>}$ = 13860 (8%) $\Delta_{(3,7>}$ = 0 (0%) $\Delta_{(7,15>}$ = 0 (0%) Δ_{15+} = 0 (0%)	$PSNR_R$ = 65.34 dB $PSNR_G$ = 66.41 dB $PSNR_B$ = 65.98 dB $\Delta_{(0,3>}$ = 3608 (2%) $\Delta_{(3,7>}$ = 0 (0%) $\Delta_{(7,15>}$ = 0 (0%) Δ_{15+} = 0 (0%)

Na podstawie Tabeli 9., gdzie wartość $\Delta_{(0,3>}$ to liczba błędów o wielkości z zakresu (0;3> pikseli, $\Delta_{(3,7>}$ to liczba błędów o wielkości z zakresu (3;7> pikseli, $\Delta_{(7,15>}$ to liczba błędów o wielkości z zakresu (7;15> pikseli, Δ_{15+} to liczba błędów przekraczających wielkość 15 pikseli. Symbol Δ przedstawia sumaryczne błędy dla wszystkich trzech kanałów reprezentacji RGB. Wartości procentowe, podane w nawiasie, informują o liczbie przekłamanych pikseli w odniesieniu do liczby pikseli całego obrazu. Przy założeniu, że niewykrywalność osadzonego fingerprintu uzyskiwana jest dla PSNR równego co najmniej 50 dB (Welstead 1999, Barni 2006), najdłuższy, osadzony fingerprint uzyskuje się przy zastosowaniu zaokrąglenia do 3 miejsc po przecinku.

Fingerprinty osadzone w obrazach identyfikują poszczególnych użytkowników, gdyż każdy z nich odszyfrowuje otrzymane dane unikatowym kluczem. W celu

skutecznego wykrywania nielegalnych użytkowników (piratów), każdy fingerprint powinien posiadać wysoki współczynnik autokorelacji i niski współczynnik korelacji skrośnej z fingerprintami innych użytkowników. Wartości współczynników korelacji dla przykładowego, 9-tego użytkownika zostały przedstawione na Rys. 32. W eksperymencie wykorzystano kolorowy obraz Leny o rozmiarze 243×243 px, zaszyfrowany i odszyfrowany (unikatowym kluczem) dla 100 użytkowników.



Rys. 32. Współczynniki korelacji pomiędzy fingerprintem 9-tego użytkownika i fingerprintami pozostałych użytkowników.
Źródło: Opracowanie własne.

Współczynnik korelacji dla fingerprintu wydobytego z nielegalnej kopii i fingerprintu zwykłego użytkownika wyznacza się wykorzystując wzór:

$$r = \frac{\sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{c=1}^{3}(p_{m,n,c} - \overline{p})(f_{m,n,c} - \overline{f})}{\sqrt{\left(\sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{c=1}^{3}(p_{m,n,c} - \overline{p})^2\right)\left(\sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{c=1}^{3}(f_{m,n,c} - \overline{f})^2\right)}} \tag{11.3}$$

gdzie r to współczynnik korelacji dla fingerprintów obrazów kolorowych, $p_{m,n,c}$ to wartość fingerprintu osadzonego w pikselu m-tego wiersza, n-tej kolumny i c-tego kanału (dla reprezentacji RGB) pirackiej kopii, $f_{m,n,c}$ to wartość fingerprintu osadzonego w pikselu m-tego wiersza, n-tej kolumny i c-tego kanału (dla reprezentacji RGB) zweryfikowanej, legalnej kopii użytkownika, $\overline{p}$ to wartość średnia wszystkich pikseli kopii pirackiej, $\overline{f}$ to wartość średnia wszystkich pikseli zweryfikowanej, legalnej kopii użytkownika, $M \times N$ to wielkość obrazu.

Dodatkowe informacje dotyczące fingerprintingu dla omawianego algorytmu kwaternionowego zostały przedstawione w pracy (Czaplewski i in. 2014).

12. KRYPTOANALIZA

Niniejszy rozdział dotyczy zagadnień kryptoanalizy przeprowadzonej dla algorytmów QFC. W tym celu zbadano wielkość przestrzeni kluczy algorytmów, czas potrzebny na złamanie algorytmów atakiem brute force, postaci kluczy inicjalizujących, generujących słabe klucze rundowe oraz wykazano odporność algorytmu S-QFC na atak ze znanym tekstem jawnym.

Zastosowane w rozprawie, modularne algorytmy generacji kluczy pozwalają na uzyskanie kwaternionów kluczy o współczynnikach będących wartościami z zakresu od 0 do 256 (rozdział 8.2.). Kryptoanalityk musi zatem sprawdzić wszystkie kombinacje ośmiu bitów (liczby od 0 do 255) oraz 1 kombinację dziewięciu bitów (liczba 256). Dla uproszczenia, na potrzeby kryptoanalizy, przyjęto zakres od 0 do 255 by pozostać w 8-bitowej reprezentacji binarnej.

12.1. Badanie przestrzeni kluczy

Przestrzeń kluczy uzyskana za pomocą podstawowego algorytmu generacji kluczy (rozdział 6.5.) i zastosowana w algorytmie F-QFC sprowadza się do wyznaczenia n kluczy rundowych (kwaternionów kluczy wyższych rzędów) na podstawie kwaternionu początkowego q_0.

Przyjmijmy, dla przykładu, że implementacja algorytmu F-QFC posiada 9 rund, gdzie każda runda szyfrowana jest unikatowym kluczem. Zastosowanie podstawowego algorytmu generacji kluczy dla drugiego rzędu wygeneruje 9 unikatowych kwaternionów kluczy (Rys. 4.). Każdy klucz wyższego rzędu posiada 3 różne współczynniki x, y, z (współczynnik $w = 0$), będące liczbami całkowitymi z zakresu od 0 do 255. Każdy współczynnik może być przedstawiony w postaci liczby 8-io bitowej. Na tej podstawie, liczność przestrzeni kluczy można zapisać jako:

$$2^{8\cdot 3n} = 2^{24n} \cong 10^{2.4\cdot 3n} = 10^{7.2n} \tag{12.1}$$

gdzie n to liczba kwaternionów kluczy, potrzebnych do zaszyfrowania każdej rundy algorytmu F-QFC - w naszym przypadku $n = 9$. Załóżmy, że kryptoanalityk zna zastosowaną liczbę kluczy rundowych oraz dysponuje sprzętową implementacją algorytmu F-QFC, dla której czas szyfrowania/deszyfracji niewielkich danych wynosi 10^{-7} sekund. Czas potrzebny na odszyfrowanie danych metodą brutalną (sprawdzając wszystkie kombinacje kluczy) wynosi:

$$\begin{aligned} &10^{7.2n}\cdot 10^{-7}\ [s] \cong \\ &\cong 3.17\cdot 10^{7.2n-15}\ [lat] \end{aligned} \tag{12.2}$$

Podstawiając do równości (12.2) $n = 9$, otrzymamy szacowany czas sprawdzenia wszystkich kluczy równy $3.17\cdot 10^{49.8}$ lat. Załóżmy jednak, że kryptoanalityk dysponuje pełną wiedzą na temat zastosowanego algorytmu generacji kluczy. Zna zastosowany w algorytmie rząd rotacji i musi jedynie sprawdzić wszystkie kombinacje

współczynników *w*, *x*, *y*, *z* kwaternionu początkowego q_0, z którego wygeneruje klucze rundowe. Liczność przestrzeni kluczy do sprawdzenia w tym przypadku można zapisać jako:

$$2^{8\cdot 4} = 2^{32} \cong 10^{2.4\cdot 4} = 10^{9.6} \tag{12.3}$$

Przyjmijmy dodatkowo, że czas potrzebny do wygenerowania zadanej przestrzeni kluczy na podstawie klucza początkowego q_0 jest bardzo mały i w tym przypadku pomijalny. Czas potrzebny na odszyfrowanie danych metodą brutalną (sprawdzając wszystkie kombinacje klucza q_0) wynosi:

$$\begin{gathered} 10^{9.6} \cdot 10^{-7}\ [s] \cong \\ \cong 6.6\ [\text{min}] \end{gathered} \tag{12.4}$$

Zatem wykorzystując znajomość algorytmu generacji kluczy, kryptoanalityk może w przeciągu 6 minut sprawdzić całą przestrzeń zastosowanych kluczy rundowych (zakładając pomijalnie mały czas generacji kluczy wyższych rzędów oraz przybliżony czas szyfrowania/deszyfracji równy 10^{-7} sekundy). Dowodzi to, że podstawowy algorytm generacji kluczy wpływa w sposób niekorzystny na ogólny poziom bezpieczeństwa oferowany przez algorytm F-QFC.

Rozważmy teraz przestrzeń kluczy dla rozszerzonego algorytmu generacji kluczy (rozdział 9.1.), zastosowanego w algorytmie S-QFC. Przyjmijmy, dla przykładu, że implementacja algorytmu S-QFC posiada 9 rund, gdzie każda runda szyfrowana jest unikatowym kluczem. Mnożenie macierzowe wykonywane jest co 3 rundy, przy zastosowaniu pojedynczej macierzy klucza **K** o minimalnym rozmiarze 5×5 (25 liczb całkowitych z zakresu od 0 do 255). Zastosowanie rozszerzonego algorytmu generacji kluczy dla drugiego rzędu wygeneruje 9 unikatowych kwaternionów kluczy (Rys. 13.). Każdy klucz wyższego rzędu posiada 4 różne współczynniki *w*, *x*, *y*, *z*, będące liczbami całkowitymi z zakresu od 0 do 255. Każdy współczynnik może być przedstawiony w postaci liczby 8-io bitowej. Na tej podstawie, liczność przestrzeni kluczy można zapisać jako:

$$2^{8\cdot(m^2+4n)} \cong 10^{2.4\cdot(m^2+4n)} \tag{12.5}$$

gdzie *m* to rozmiar kwadratowej macierzy klucza **K**, *n* to liczba kwaternionów kluczy, potrzebnych do zaszyfrowania każdej rundy algorytmu S-QFC. Dla uproszczenia w zapisie (12.5) nie uwzględniono wymagania odwracalności macierzy **K** (macierz **K** musi być macierzą nieosobliwą). W praktyce wartość m^2 zostanie pomniejszona o liczbę możliwych reprezentacji macierzy **K**, dla których wyznacznik tej macierzy przyjmuje wartość 0.

Załóżmy analizę danych o małych rozmiarach, dla których minimalny rozmiar macierzy **K** wynosi $m = 5$ oraz, że liczba zastosowanych rund algorytmu S-QFC wynosi $n = 9$. Załóżmy też, że kryptoanalityk zna zastosowaną liczbę kluczy rundowych oraz dysponuje sprzętową implementacją algorytmu S-QFC, dla której czas szyfrowania/deszyfracji niewielkich danych wynosi 10^{-7} sekund. Czas potrzebny na

odszyfrowanie danych metodą brutalną (sprawdzenie wszystkich kombinacji kluczy) wynosi przy tych założeniach:

$$10^{2.4\cdot(m^2+4n)}\cdot 10^{-7}\ [s]\cong \\ \cong 3.17\cdot 10^{2.4\cdot(m^2+4n)-15}\ [lat] \tag{12.6}$$

Podstawiając do równości (12.6) $m = 5$ oraz $n = 9$, otrzymamy szacowany czas sprawdzenia wszystkich kluczy równy $3.17\cdot 10^{131.4}$ lat. Załóżmy jednak, że kryptoanalityk zna zasadę wyznaczania kwaternionów wyższych rzędów, wie że musi sprawdzić wszystkie kombinacje współczynników *w*, *x*, *y*, *z* kwaternionu początkowego q_0, z którego wygeneruje klucze rundowe oraz wszystkie kombinacje współczynników *x*, *y*, *z* (współczynnik $w = 0$) kwaternionów *f*, uzyskanych z obrazu fraktala. Współczynniki kwaternionów *f* są liczbami całkowitymi z zakresu od 0 do 255. Liczebność *l* kwaternionów *f* wyznaczana jest na podstawie wzoru (12.7). Dla uproszczenia przyjmijmy liczbę rund *n* algorytmu S-QFC jako wielokrotność 3.

$$l = \sum_{i=1}^{\log_3 n} 3^i \tag{12.7}$$

Na tej podstawie, liczność przestrzeni kluczy do sprawdzenia można zapisać jako:

$$2^{8\cdot(m^2+4+3l)} \cong 10^{2.4\cdot(m^2+4+3l)} \tag{12.8}$$

Przyjmijmy, że czas potrzebny do wygenerowania zadanej przestrzeni kluczy na podstawie klucza początkowego q_0 i kwaternionów *f* jest bardzo mały i w tym przypadku pomijalny. Czas potrzebny na odszyfrowanie danych metodą brutalną (sprawdzając wszystkie kombinacje klucza q_0 i kwaternionów *f*) wynosi:

$$10^{2.4\cdot(m^2+4+3l)}\cdot 10^{-7}\ [s]\cong \\ \cong 3.17\cdot 10^{2.4\cdot(m^2+4+3l)-15}\ [lat] \tag{12.9}$$

Podstawiając do równości (12.9) $m = 5$ oraz $l = 12$ (wyznaczone na podstawie wzoru (12.7) dla $n = 9$) otrzymamy szacowany czas sprawdzenia wszystkich kluczy równy $3.17\cdot 10^{141}$ lat. Załóżmy jednak, że kryptoanalityk zna zasadę wyznaczania kwaternionów wyższych rzędów oraz dysponuje narzędziem do zobrazowania kwaternionowych zbiorów Julia. W tym przypadku kryptoanalityk musi sprawdzić wszystkie kombinacje współczynników *w*, *x*, *y*, *z* kwaternionu początkowego q_0, z którego wygeneruje klucze rundowe oraz wszystkie kombinacje parametrów inicjalizujących, potrzebnych do wygenerowania obrazu fraktala. Do parametrów tych należą: liczba iteracji (liczba całkowita 8-io bitowa), współczynniki kwaternionu *c* (4 liczby zmiennoprzecinkowe; reprezentacja 32-u bitowa), liczba kontrolna (liczba całkowita 8-io bitowa) oraz płaszczyzna podziału (liczba zmiennoprzecinkowa; reprezentacja 32-u bitowa). Na tej podstawie, liczność przestrzeni kluczy do sprawdzenia można zapisać jako:

$$2^{8\cdot(m^2+4+1+4\cdot4+1+4)} \cong 10^{2.4\cdot(m^2+4+1+4\cdot4+1+4)} \tag{12.10}$$

Przyjmijmy, że czas potrzebny do wygenerowania zadanej przestrzeni kluczy na podstawie klucza początkowego q_0 i obrazu fraktala jest bardzo mały i w tym przypadku pomijalny. Czas potrzebny na odszyfrowanie danych metodą brutalną (sprawdzając wszystkie kombinacje klucza q_0 i parametrów inicjalizujących) wynosi:

$$\begin{gathered} 10^{2.4\cdot(m^2+4+1+4\cdot4+1+4)} \cdot 10^{-7}\ [s] \cong \\ \cong 3.17\cdot10^{2.4\cdot(m^2+26)-15}\ [lat] \end{gathered} \tag{12.11}$$

Podstawiając do równości (12.11) $m = 5$ otrzymamy szacowany czas sprawdzenia wszystkich kluczy równy $3.17\cdot10^{107.4}$ lat. Zatem, dla algorytmu S-QFC, nawet przy pełnej znajomości rozszerzonego algorytmu generacji kluczy, sprawdzenie zastosowanej przestrzeni kluczy jest praktycznie niemożliwe (zakładając pomijalnie mały czas generacji kluczy wyższych rzędów oraz przybliżony czas szyfrowania/deszyfracji równy 10^{-7} sekundy).

Oczywistym jest, że wyniki takich obliczeń nie mogą w sposób jednoznaczny zadecydować o bezpieczeństwie proponowanego rozwiązania. Przedstawione w niniejszym rozdziale operacje szacowania rozmiaru przestrzeni kluczy i przeliczania ich liczby na czas obliczeń, wykonane zostały głównie w celu porównania algorytmów generacji kluczy dla implementacji F-QFC i S-QFC. Dużo bardziej istotne od liczności przestrzeni kluczy dla każdego algorytmu kryptograficznego jest możliwość istnienia tzw. słabych kluczy. Temu zagadnieniu poświęcony jest rozdział 12.2.

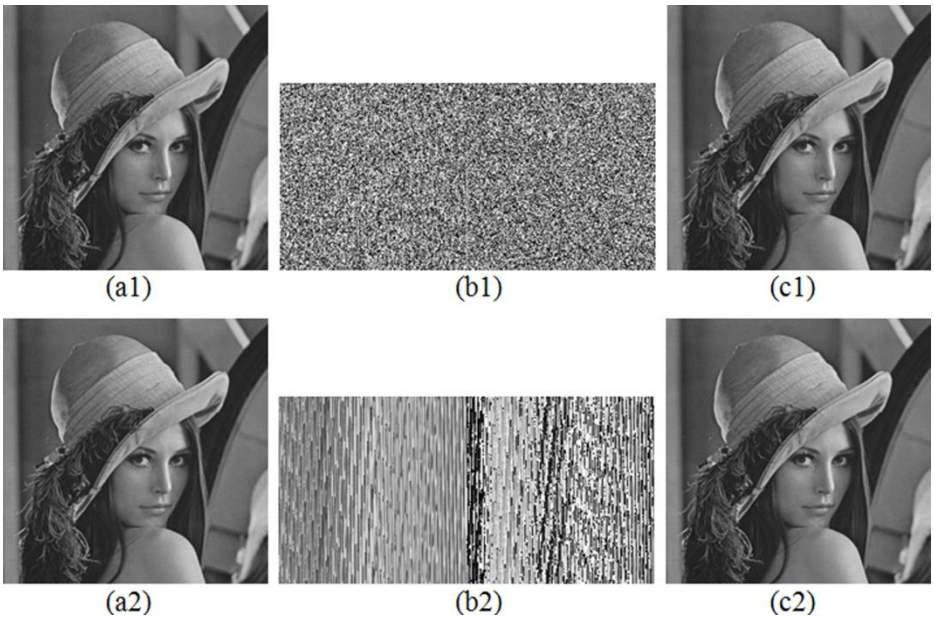
12.2. Badanie słabych kluczy

Klucz słaby to klucz kryptograficzny, dla którego algorytm kryptograficzny nie zachowuje w pełni swoich właściwości, co w konsekwencji może prowadzić do uproszczonego procesu kryptoanalizy.

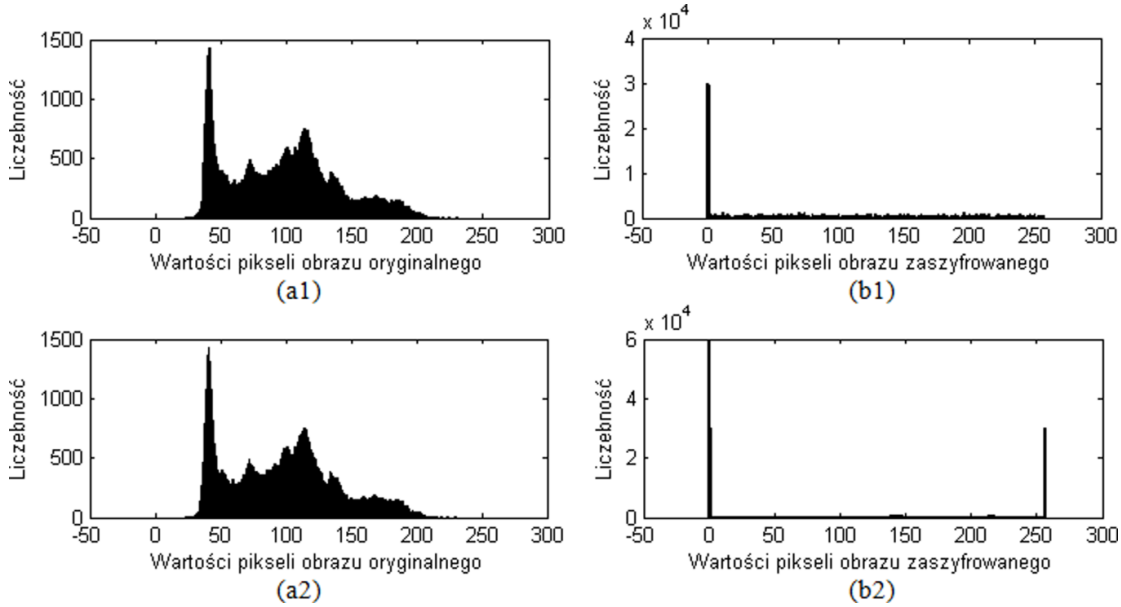
Algorytmy F-QFC i S-QFC podczas operacji szyfrowania i deszyfrowania wykorzystują klucze rundowe q_i (kwaterniony wyższych rzędów, które są wyznaczane za pomocą, odpowiednio dla algorytmu F-QFC i algorytmu S-QFC, podstawowego i rozszerzonego algorytmu generacji kluczy). Klucze rundowe w obu przypadkach obliczane są w oparciu o kwaternion inicjalizujący q_0 (rzędu zerowego). Problem słabych kluczy należy rozpocząć od analizy klucza inicjalizującego, za pomocą którego, w sposób pośredni, mamy wpływ na postaci kluczy rundowych. Ponieważ obydwa algorytmy generacji kluczy wykorzystują arytmetykę modularną, zatem postać klucza inicjalizującego o zerowych współczynnikach: $q_0 = (0, 0, 0, 0)$, wygeneruje błąd, ponieważ obliczona z niego macierz rotacji będzie macierzą zerową i tym samym macierzą osobliwą. Dodatkowo, niemożliwe będzie obliczenie kwaternionu odwrotnego.

Dla algorytmu F-QFC można wyróżnić pięć charakterystycznych postaci kwaternionu inicjalizującego, dla których otrzymamy słabe klucze rundowe (tzn. kwaterniony, których przynajmniej dwa współczynniki są zerowe). Dla takich kluczy

rundowych nie uzyskamy rozkładu równomiernego dla zaszyfrowanych danych, a w niektórych przypadkach widoczne będą artefakty dla zobrazowania szyfrogramu.



Rys. 33. Efekt szyfrowania algorytmem F-QFC przy zastosowaniu słabych kluczy rundowych: (a) obraz oryginalny, (b) obraz zaszyfrowany, (c) obraz odszyfrowany (bez błędów).
Źródło: Opracowanie własne.



Rys. 34. Histogramy obrazów zaszyfrowanych algorytmem F-QFC przy zastosowaniu słabych kluczy rundowych: (a) histogram obrazu oryginalnego, (b) histogram obrazu zaszyfrowanego.
Źródło: Opracowanie własne.

Na Rys. 33. przedstawiono efekt szyfrowania algorytmem F-QFC przy zastosowaniu słabych kluczy rundowych. Na Rys. 34. przedstawiono histogramy wartości pikseli obrazów zaszyfrowanych z Rys. 33.

Postaci kwaternionu inicjalizującego, dla których otrzymamy słabe klucze rundowe dla algorytmu F-QFC są następujące: (wartości a i b to dowolne liczby całkowite z zakresu od 1 do 256 (z uwagi na zastosowanie w algorytmie generacji kluczy arytmetyki modularnej o module 257) oraz $a \neq b$).

Pierwsza postać odpowiada sytuacji, w której trzy dowolne współczynniki kwaternionu inicjalizującego przyjmują wartość 0, np. $q_0 = (0, 0, 0, a)$. Dodatkowo, dla tej postaci należy uwzględnić wszystkie możliwe kombinacje położenia współczynników kwaternionu inicjalizującego - jest ich $4!/(3!\cdot 1!) = 4$.

Druga postać odpowiada sytuacji, w której dwa dowolne współczynniki kwaternionu inicjalizującego przyjmują wartość 0, a pozostałe dwa posiadają tę samą wartość, różną od 0, np. $q_0 = (0, 0, a, a)$. Dodatkowo, dla tej postaci należy uwzględnić wszystkie możliwe kombinacje położenia współczynników kwaternionu inicjalizującego - jest ich $4!/(2!\cdot 2!) = 6$.

Trzecia postać odpowiada sytuacji, w której dwa dowolne współczynniki kwaternionu inicjalizującego przyjmują wartość 0, a pozostałe dwa posiadają różne względem siebie wartości, różne od 0, np. $q_0 = (0, 0, a, b)$. Dodatkowo, dla tej postaci należy uwzględnić wszystkie możliwe kombinacje położenia współczynników kwaternionu inicjalizującego - jest ich $4!/(2!\cdot 1!\cdot 1!) = 12$.

Czwarta postać odpowiada sytuacji, w której dwa dowolne współczynniki kwaternionu inicjalizującego przyjmują wartość a, a pozostałe dwa współczynniki wartość b, np. $q_0 = (a, a, b, b)$. Dodatkowo, dla tej postaci należy uwzględnić wszystkie możliwe kombinacje położenia współczynników kwaternionu inicjalizującego - jest ich $4!/(2!\cdot 2!) = 6$.

Piąta postać odpowiada sytuacji, w której wszystkie współczynniki kwaternionu inicjalizującego posiadają tę samą wartość - $q_0 = (a, a, a, a)$.

Na tej podstawie, sumaryczna liczba wszystkich wartości współczynników kwaternionu inicjalizującego, dla których można wyznaczyć słabe klucze rundowe jest następująca. Zaczynając od sprawdzenia wszystkich możliwych wartości parametru a, należy uwzględnić wszystkie kombinacje dla postaci pierwszej, drugiej i piątej:

$$\begin{gathered} a \rightarrow 256 \text{ możliwych wartości} \\ 256\cdot(4+6+1) = 256\cdot 11 = 2816 \end{gathered} \tag{12.12}$$

Dla pojedynczej wartości a (uwzględniając postaci pierwszą, drugą i piątą), możemy uzyskać 2816 różnych kluczy inicjalizujących, generujących słabe klucze rundowe. Przechodząc do sprawdzenia wszystkich możliwych wartości parametrów a i b, należy uwzględnić wszystkie kombinacje dla postaci trzeciej i czwartej:

$$\begin{gathered} a, b \rightarrow \frac{256!}{(256-2)!} = 255\cdot 256 \quad \text{możliwych wartości} \\ 255\cdot 256\cdot(12+6) = 1175040 \end{gathered} \tag{12.13}$$

Dla obydwu wartości a i b (uwzględniając postaci trzecią i czwartą), możemy uzyskać 1175040 różnych kluczy inicjalizujących, generujących słabe klucze rundowe.

Podsumowując, liczba różnych kluczy inicjalizujących, generujących słabe klucze rundowe dla algorytmu F-QFC, w oparciu o dane z równań (12.12) i (12.13), wynosi 1177856.

Liczba wszystkich możliwych kluczy inicjalizujących dla algorytmów QFC wynosi $257^4 - 1$ (należy pominąć reprezentację zerowych współczynników tj. $q_0 = (0, 0, 0, 0)$). Procentowy udział kwaternionów inicjalizujących, generujących słabe klucze rundowe, w puli wszystkich możliwych kwaternionów inicjalizujących dla algorytmu F-QFC wynosi:

$$\frac{1177856}{257^4 - 1} \cong 0.027\% \tag{12.14}$$

Algorytm F-QFC jest wyposażony w funkcję sprawdzającą możliwość wygenerowania słabych kluczy rundowych w oparciu o wartości współczynników kwaternionu inicjalizującego.

Dla algorytmu S-QFC, nie istnieje postać klucza inicjalizującego, która mogłaby wygenerować słabe klucze rundowe (tzn. kwaterniony, których przynajmniej dwa współczynniki są zerowe). Taka właściwość wynika bezpośrednio z rozszerzonego algorytmu generacji kluczy, dla którego można zaobserwować właściwości "korygujące" zerowe wartości współczynników kwaternionu inicjalizującego.

Można to prześledzić na następującym przykładzie, w oparciu o schemat rozszerzonego algorytmu generacji kluczy - Rys. 13. (rozdział 9.1.).

Załóżmy przykładowo, że kwaternion inicjalizujący dla algorytmu S-QFC jest postaci $q_0 = (0, a, 0, 0)$. Wyznaczona z niego początkowa macierz rotacji będzie macierzą diagonalną. Przy wyznaczaniu kwaternionów rzędu pierwszego zaobserwujemy, że już tylko 2 współczynniki posiadają wartość zero: $q_{11} = (w_{11}, x_{11}, 0, 0)$, $q_{12} = (w_{12}, 0, y_{12}, 0)$, $q_{13} = (w_{13}, 0, 0, z_{13})$. Kontynuując ten proces dla kolejnych rzędów, liczba zerowych współczynników dla następnych kwaternionów będzie maleć. Dodatkowo, zaraz po wyznaczeniu kwaternionów rzędu pierwszego, liczba zerowych współczynników dla tych kwaternionów zostanie całkowicie wyeliminowana z uwagi na obrót wokół kwaternionów fraktalowych f_i.

Kwestię słabych kluczy dla algorytmów QFC można dodatkowo zbadać rozpatrując reprezentację binarną otrzymywanych szyfrogramów, w celu zbadania istnienia takich samych szyfrogramów dla różnych kluczy rundowych. Niestety, w przypadku algorytmów QFC, takie podejście jest w praktyce trudne do zrealizowania z uwagi na brak ustalonej wielkości bloku danych, poddawanego operacjom szyfrowania i deszyfracji. Sprawdzenie wszystkich możliwości jest zadaniem bardzo czasochłonnym i z tego też powodu zostało pominięte w niniejszej rozprawie. Mimo to, problem słabych kluczy stanowi bardzo istotne zagadnienie, które będzie zbadane podczas dalszych prac nad algorytmami szyfrowania kwaternionowego.

12.3. Atak ze znanym tekstem jawnym

Obustronne mnożenie macierzowe danych przez macierz klucza **K** (rozdział 8.3.1.), realizowane dla algorytmu S-QFC, zapewnia odporność na atak ze znanym tekstem jawnym. Właściwość ta, w sposób ogólny, opisowy, została przedstawiona w pracy (Sastry i Kumar 2012). W tym rozdziale przedstawiono dokładne matematyczne uzasadnienie.

Załóżmy, że kryptoanalityk zna liczbę rund algorytmu S-QFC oraz postać wszystkich kluczy rundowych. Potrzebuje jedynie wyznaczyć macierz klucza **K**. Kryptoanalityk dysponując parami tekst jawny i odpowiadający mu tekst zaszyfrowany, postanawia przeprowadzić atak ze znanym tekstem jawnym. Kryptoanalityk może wyznaczyć przynajmniej 3 równania z macierzą klucza **K** - równania (8.4), (8.5) i (8.6) przedstawione w rozdziale 8.3.1. Przykładowo, jeżeli algorytm S-QFC posiada 9 rund i mnożenie przez macierz klucza **K** wykonywane jest co 3 rundy, gdzie macierz klucza pozostaje niezmieniona, kryptoanalityk może uzyskać 9 równań dla macierzy klucza. Dla uproszczenia rozważmy 2 równania dla macierzy klucza **K**:

$$\mathbf{R}_1' = \mathbf{K} \cdot \mathbf{R}_1 \cdot \mathbf{K}^{-1} \tag{12.15}$$

$$\mathbf{R}_2' = \mathbf{K} \cdot \mathbf{R}_2 \cdot \mathbf{K}^{-1} \tag{12.16}$$

gdzie $\mathbf{R}_1$ i $\mathbf{R}_2$ to fragmenty tekstu jawnego, a $\mathbf{R}_1'$ i $\mathbf{R}_2'$ to odpowiednie fragmenty tekstu zaszyfrowanego. Równania (12.15) i (12.16) można przekształcić do postaci (12.17) i (12.18) mnożąc je prawostronnie przez macierz klucza **K**:

$$\mathbf{R}_1' \cdot \mathbf{K} = \mathbf{K} \cdot \mathbf{R}_1 \tag{12.17}$$

$$\mathbf{R}_2' \cdot \mathbf{K} = \mathbf{K} \cdot \mathbf{R}_2 \tag{12.18}$$

Z równania (12.18) można obliczyć macierz **K**, mnożąc go lewostronnie przez macierz $\mathbf{R}_2'^{-1}$ (pod warunkiem, że macierz $\mathbf{R}_2'$ jest macierzą nieosobliwą) i następnie podstawić do równania (12.17) uzyskując:

$$\mathbf{R}_1' \cdot \mathbf{R}_2'^{-1} \cdot \mathbf{K} \cdot \mathbf{R}_2 = \mathbf{K} \cdot \mathbf{R}_1 \tag{12.19}$$

Mnożąc prawostronnie równanie (12.19) przez macierz $\mathbf{R}_1^{-1}$ (pod warunkiem, że macierz $\mathbf{R}_1$ jest macierzą nieosobliwą) otrzymamy:

$$\mathbf{R}_1' \cdot \mathbf{R}_2'^{-1} \cdot \mathbf{K} \cdot \mathbf{R}_2 \cdot \mathbf{R}_1^{-1} = \mathbf{K} \tag{12.20}$$

Wykonując mnożenie macierzy $\mathbf{R}_1'$ i $\mathbf{R}_2'^{-1}$ (zapisując wynik jako macierz **X**) oraz mnożenie macierzy $\mathbf{R}_2$ i $\mathbf{R}_1^{-1}$ (zapisując wynik jako macierz **Y**), przekształcimy równanie (12.20) do postaci:

$$\mathbf{X} \cdot \mathbf{K} \cdot \mathbf{Y} = \mathbf{K} \quad \rightarrow \quad \mathbf{X} \cdot \mathbf{K} = \mathbf{K} \cdot \mathbf{Y}^{-1} \tag{12.21}$$

Widoczne jest, że atak ze znanym tekstem jawnym w każdym przypadku prowadzi do równania macierzowego $\mathbf{A}\cdot\mathbf{K} = \mathbf{K}\cdot\mathbf{B}$, gdzie niewiadomą jest macierz $\mathbf{K}$. Równanie takie posiada pojedyncze rozwiązanie wtedy i tylko wtedy, gdy macierze $\mathbf{A}$ i $\mathbf{B}$ nie posiadają wspólnych wartości własnych (Gantmacher 1959). Ponieważ dla macierzy $\mathbf{R}_1'$ i $\mathbf{R}_1$ z równania (12.17) zachodzi zależność (12.15), macierze te są macierzami podobnymi (Gantmacher 1959). Macierze podobne posiadają ten sam wielomian charakterystyczny, a w konsekwencji te same wartości własne o identycznych krotnościach algebraicznych i geometrycznych oraz te same postaci Jordana (Gantmacher 1959). Na tej podstawie można stwierdzić, że pojedyncze rozwiązanie dla równania (12.17), gdzie niewiadomą jest macierz klucza $\mathbf{K}$, nie istnieje.

W przypadku podobieństwa macierzy $\mathbf{R}_1'$ i $\mathbf{R}_1$, równanie (12.17) nie posiada pojedynczego rozwiązania, ale istnieją rozwiązania wielokrotne. Liczba możliwych rozwiązań dla równania (12.17) uzależniona jest od zbioru M wartości dowolnych: s_1, s_2, ..., s_M określających postać macierzy $\tilde{\mathbf{K}}$ (Gantmacher 1959):

$$\mathbf{K} = \mathbf{U}\cdot\tilde{\mathbf{K}}\cdot\mathbf{V}^{-1} \tag{12.22}$$

gdzie macierze $\mathbf{U}$ i $\mathbf{V}$ związane są odpowiednio z postaciami Jordana $\mathbf{J}_{\mathbf{R}_1'}$ i $\mathbf{J}_{\mathbf{R}_1}$ odpowiednio macierzy $\mathbf{R}_1'$ i $\mathbf{R}_1$:

$$\mathbf{R}_1' = \mathbf{U}\cdot\mathbf{J}_{\mathbf{R}_1'}\cdot\mathbf{U}^{-1} \qquad \mathbf{R}_1 = \mathbf{V}\cdot\mathbf{J}_{\mathbf{R}_1}\cdot\mathbf{V}^{-1} \tag{12.23}$$

Struktura macierzy $\tilde{\mathbf{K}}$ uzależniona jest od macierzy $\mathbf{K}_{\alpha\beta}$ (α = 1, 2, ..., u; β = 1, 2, ..., v), gdzie wartości u i v oznaczają odpowiednio liczbę wartości własnych macierzy $\mathbf{R}_1'$ i $\mathbf{R}_1$ (Gantmacher 1959). Ponieważ macierze $\mathbf{R}_1'$ i $\mathbf{R}_1$ są podobne, zatem $u = v$. Uwzględniając postać kwadratową $N\times N$ wszystkich macierzy w równaniu (12.17), liczba M wartości dowolnych dla macierzy $\tilde{\mathbf{K}}$ jest z zakresu od N (12.24) do N^2 (12.25):

$$\tilde{\mathbf{K}} = \begin{bmatrix} s_1 & s_2 & s_3 & \cdots & s_N \\ 0 & s_1 & s_2 & & \\ 0 & 0 & s_1 & & \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & & \cdots & s_1 \end{bmatrix}_{N\times N} \tag{12.24}$$

$$\tilde{\mathbf{K}} = \begin{bmatrix} s_1 & s_2 & \cdots & s_N \\ s_{N+1} & s_{N+2} & & s_{2N} \\ & & & \\ \vdots & \vdots & \ddots & \vdots \\ s_{(N-1)N+1} & & \cdots & s_{N^2} \end{bmatrix}_{N\times N} \tag{12.25}$$

Przypadek (12.24) zachodzi w sytuacji, gdy macierze $\mathbf{R}_1'$ i $\mathbf{R}_1$ posiadają pojedynczą wartość własną o krotności N. Przypadek (12.25) zachodzi w sytuacji, gdy macierze $\mathbf{R}_1'$

i $\mathbf{R}_1$ posiadają N wartości własnych o krotności 1 każda. W załączniku 1 przedstawiono przykład wyznaczania macierzy $\tilde{\mathbf{K}}$ na podstawie znajomości macierzy $\mathbf{R}_1$' i $\mathbf{R}_1$.

Ponieważ w równaniu (12.15) zastosowano modularne mnożenie macierzowe (modulo 257), wartości dowolne dla macierzy $\tilde{\mathbf{K}}$ są całkowitoliczbowe z zakresu od 0 do 256. Uwzględniając przypadek (12.24) i (12.25), liczba możliwych macierzy **K**, wyznaczonych z równania (12.22), zawarta jest w przedziale:

$$\left\langle 257^{N};\, 257^{N^2} \right\rangle \tag{12.26}$$

Kryptoanalityk badając postaci macierzy $\mathbf{R}_1$' i $\mathbf{R}_1$ będzie mógł określić strukturę macierzy $\tilde{\mathbf{K}}$. Chcąc przeprowadzić atak ze znanym tekstem jawnym w celu poznania macierzy klucza **K**, w najlepszym przypadku (12.24) kryptoanalityk będzie musiał sprawdzić 257^N wariantów macierzy $\tilde{\mathbf{K}}$. W przypadku najgorszym (12.25) kryptoanalityk będzie zmuszony przeprowadzić atak brutalny na postać macierzy $\tilde{\mathbf{K}}$ (musi sprawdzić wszystkie możliwe kombinacje elementów dla macierzy wielkości $N \times N$, co jest równoważne z atakiem brutalnym na postać macierzy klucza **K**).

Przedstawione wyżej rozważania zostały wykorzystane w publikacji prezentującej odporność algorytmu S-QFC na znane ataki kryptoanalityczne (Dzwonkowski i Rykaczewski 2017: *Secure Quaternion Feistel Cipher (S-QFC) Resistant to Known Plaintext Attack* - w przygotowaniu).

13. PODSUMOWANIE

Problemem badawczym, którego dotyczyła niniejsza rozprawa, jest ochrona danych cyfrowych. Sposób zabezpieczania przesyłanych informacji zwykle zależny jest od rodzaju danych i ich przeznaczenia. Do najważniejszych metod ochrony bezpieczeństwa transmitowanych danych należą metody kryptograficzne, jednakże zastosowanie mechanizmów szyfrujących i deszyfrujących w sposób istotny wpływa na całkowity czas dostępu do danych. Konieczne staje się więc stosowanie wydajnych algorytmów kryptograficznych, szczególnie w przypadku transmisji danych mniejszej wagi (dane o treści multimedialnej) i danych dużego rozmiaru (dane medyczne).

Celem pracy było zaprojektowanie wydajnego i bezpiecznego, symetrycznego systemu kryptograficznego w celu zabezpieczania danych multimedialnych i medycznych. Cel pracy został zrealizowany poprzez przedstawienie w rozprawie nowych, oryginalnych rozwiązań:

- programowej implementacji szyfrowania kwaternionowego, opartej na rotacjach kwaternionowych przeprowadzanych z wykorzystaniem algebry kwaternionowej;
- implementacji trybów szyfrowania kwaternionowego: ECB, CBC i CTR;
- implementacji arytmetyki modularnej dla operacji kwaternionowych;
- algorytmów zmodyfikowanej sieci Feistela: w wersji podstawowej (algorytm F-QFC) i w wersji rozszerzonej (algorytm S-QFC);
- algorytmów generacji kluczy szyfrujących: w wersji podstawowej i w wersji rozszerzonej, wykorzystującej kwaternionowe zbiory Julia;
- obustronnego generatora kluczy symetrycznych.

Przedstawione w rozprawie symetryczne algorytmy F-QFC i S-QFC, bazują na sieci Feistela i wykorzystują modularne rotacje kwaternionowe. Algorytmy te realizują szyfrowanie i deszyfrację danych dowolnego typu i rozmiaru. Przeprowadzone testy ukazują bardzo dobrą losowość uzyskiwanych szyfrogramów, jak również dużą szybkość obliczeniową (nawet stukrotnie większą w porównaniu ze standardowym algorytmem AES-ECB, zaimplementowanym w tym samym środowisku).

Algorytm F-QFC, wykorzystujący podstawowy algorytm generacji kluczy i nie realizujący obustronnego mnożenia macierzowego przez macierz klucza **K**, jest najszybszą implementacją szyfrowania kwaternionowego. Koncepcja algorytmu F-QFC nie jest oparta na wysokiej złożoności obliczeniowej, dlatego też algorytm ten jest mniej bezpieczny od zmodyfikowanego, i wolniejszego algorytmu S-QFC.

Algorytm S-QFC, podobnie jak algorytm F-QFC, wykorzystuje arytmetykę modularną w ciele kwaternionów (operacje na liczbach Lipschitza). Algorytm S-QFC cechuje mniejsza wydajność obliczeniowa (w porównaniu z algorytmem F-QFC) z uwagi na zastosowanie zmodyfikowanego algorytmu generacji kluczy, wykorzystującego kwaternionowe zbiory Julia oraz zastosowanie modularnego mnożenia macierzowego przez macierz klucza **K**. Wyniki przeprowadzonych testów i analiz świadczą o wysokiej odporności algorytmu na atak brutalny i atak ze znanym tekstem jawnym. Z tego też powodu algorytm S-QFC jest idealnym rozwiązaniem dla zabezpieczania danych o dużej ważności.

Obydwa oryginalne algorytmy F-QFC i S-QFC spełniają założenia kryptografii symetrycznej, dla której operacje inicjalizujące oraz dystrybucja kluczy symetrycznych realizowane są przez stronę trzecią. Dla omawianych algorytmów szyfrujących można wykorzystać, przedstawiony w rozdziale 9.2., algorytm obustronnego generatora kluczy symetrycznych. Dla takiego rozwiązania parametry inicjalizujące, potrzebne do wygenerowania kluczy symetrycznych, ustalane są po stronie użytkowników, bez udziału strony trzeciej, co znacznie zwiększa bezpieczeństwo takiego rozwiązania.

Do realizacji osiągnięć przedstawionych w rozprawie (m.in. implementacji algorytmów F-QFC i S-QFC) doprowadziły prace nad publikacjami (Dzwonkowski i Rykaczewski 2011, 2012, 2013a, 2013b, 2014, 2015a (JCR), 2015b, 2016; Czaplewski i in. 2013, 2014a (JCR), 2014b), które stanowią oryginalny dorobek autora niniejszej rozprawy.

Podsumowując osiągnięte w rozprawie wyniki, należy stwierdzić, że obszar badań dotyczących zastosowania liczb hiperzespolonych w kryptografii jest teoretycznie i praktycznie nieograniczony. Tak więc modyfikacja i rozszerzenie, przedstawionego w niniejszej pracy, szyfrowania kwaternionowego jest tematem dalszych prac i badań. Planowane jest m.in. rozszerzenie algorytmów kryptograficznych (wraz z dokładną analizą) do innych niż kwaternionowa algebr Clifforda, takich jak algebra bikwaternionów, oktonionów i sedenionów, zastosowanie jako wartości elementów kwaternionów elementów ciała Galois rzędu 2^8 (podobnie, jak jest to stosowane w algorytmie AES) oraz zastosowanie dodatkowych operacji zwiększających odporność nowych algorytmów na ataki kryptoanalityczne, przy jednoczesnym zachowaniu ich wysokiej wydajności obliczeniowej.

BIBLIOGRAFIA

Ahmed N., Natarajan T., Rao K.R., 1974. *Discrete Cosine Transform*, IEEE Trans. Comput., C-32, s. 90-93

Anand R., Bajpai G., Bhaskar V., 2009. *Real-Time Symmetric Cryptography using Quaternion Julia Set*, IJCSNS 2009, s. 20-26

Armitage P., Berry G., Matthews J.N.S., 2008. *Statistical Methods in Medical Research*, Chichester: John Wiley & Sons

Barni M., 2006. *Fractal Image Compression*, w: Doc. and image compression, s. 168-169

Biham E., Shamir A., 1990. *Differential Cryptanalysis of DES-like Cryptosystems*, Advances in Cryptology - CRYPTO '90. Springer-Verlag. s. 2–21

Bilski T., Bucholc K., Chmiel K., Grocholewska-Czuryło A., Idzikowska E., Janicka-Lipska I., Stokłosa J., 2013. *Bezpieczeństwo kryptograficzne przesyłania i gromadzenia informacji*, w: Nowoczesne systemy łączności i transmisji danych na rzecz bezpieczeństwa: Szanse i zagrożenia, red. A. R. Pach, Z. Rau, M. Wągrowski, Warszawa: Wolters Kluwer Polska SA, s. 21-52

Buchholz J.J., 2001. *Matlab Implementation of the Advanced Encryption Standard*, dostępne na: http://buchholz.hs-bremen.de/aes/aes.htm

Chang C.C., Hu Y.S., Lu T.C., 2006. *A watermarking-based image ownership and tampering authentication scheme*. Pattern Recognit. Lett. 27 (5), s. 439-446

Christensen C., 2005. *Finding Multiplicative Inverses Modulo n*, Cryptology Notes, dostępne na: http://www.nku.edu/~christensen/section%206%20appendix%20euclidean%20algorithm.pdf

Conway J.H., Smith D.A., 2004. *On Quaternions and Octonions: Their Geometry, Arithmetic, and Symmetry*, A K Peters, Limited, s. 1-18

Coppersmith D., Shamir A., 1997. *Lattice attacks on NTRU*. EUROCRYPT, s. 52-61

CrypTool, 2015. *An open-source Windows program for cryptography and cryptanalysis*, dostępne na: https://www.cryptool.org/en/

Czaplewski B., Dzwonkowski M., Rykaczewski R., 2013. *Digital fingerprinting based on quaternion encryption for image transmission*, Telecommun. Rev. + Telecommun. News, 8-9, s. 792-798

Czaplewski B., Dzwonkowski M., Rykaczewski R., 2014. *Digital fingerprinting for color images based on the quaternion encryption scheme*, Pattern Recognit. Lett., tom 46, s. 11–19

Czaplewski B., Dzwonkowski M., Rykaczewski R., 2014. *Digital fingerprinting based on quaternion encryption scheme for gray-tone images*, J. Telecommun. Inf. Technol., 2, s. 3-11

Doukhnitch E., Chefranov A.G., Mahmoud A., 2013. *Encryption Schemes with Hyper-Complex Number Systems and Their Hardware-Oriented Implementation*, w: Theory and Practice of Cryptography Solutions for Secure Information Systems, IGI Global, s. 110-132, doi:10.4018/978-1-4666-4030-6

Doukhnitch E., Ozen E., 2011. *Hardware-oriented algorithm for quaternion valued matrix decomposition*, IEEE Transactions on Circuits and Systems II, Express Briefs, 58 (4), s. 225–229, doi:10.1109/TCSII.2011.2111590

DRM, 2003. Digital Rights Management - Final Report, dostępne na: https://web.archive.org/web/20080308232149/http://ec.europa.eu/enterprise/ict/policy/doc/drm.pdf

Dzwonkowski M., Rykaczewski R., 2011. *Implementacja programowa i badanie kwaternionowego systemu kryptograficznego*, Zeszyty Naukowe WETI PG, Gdansk, tom 1, s. 205-210

Dzwonkowski M., Rykaczewski R., 2012. *A New Quaternion Encryption Scheme for Image Transmission*, ICT Young 2012 Conf., Gdansk, s. 21-27

Dzwonkowski M., Rykaczewski R., 2013. *Quaternion Encryption Method for Image and Video Transmission*, Telecommun. Rev. + Telecommun. News, 8-9, s. 1216-1220

Dzwonkowski M., Rykaczewski R., 2013. *Kryptografia kwaternionowa dla zabezpieczania danych multimedialnych*, w: Nowoczesne systemy łączności i transmisji danych na rzecz bezpieczeństwa: szanse i zagrożenia, Wolters Kluwer Polska SA, s. 72-93

Dzwonkowski M., Rykaczewski R., 2014. *A Quaternion-based Modified Feistel Cipher for Multimedia Transmission*, Telecommun. Rev. + Telecommun. News, 8-9, s. 1177-1181

Dzwonkowski M., Papaj M., Rykaczewski R., 2015. *A New Quaternion-Based Encryption Method for DICOM Images*, IEEE Transactions on Image Processing, 24 (11), s. 4614-4622

Dzwonkowski M., Rykaczewski R., 2015. *Quaternion Feistel Cipher with an infinite key space based on quaternion Julia sets*, J. Telecommun. Inf. Technol., 4, s. 15-21

Dzwonkowski M., Rykaczewski R., 2016. *Quaternion encryption methods for multimedia transmission, a survey of existing approaches*, Telecommun. Rev. + Telecommun. News, 7, s. 668-671

Eberly D., 2010. *Quaternion Algebra and Calculus*, Geom. Tools, LLC, dostępne na: http://www.geometrictools.com/Documentation/Documentation.html

Falconer K., 1997. *Techniques in Fractal Geometry*, John Willey and Sons, ISBN 0-471-92287-0

Gantmacher F.R., 1959. *Theory of Matrices*, Chelsea, New York

Gawinecki J., Bora P., Jurkiewicz M., Kijko T., 2014. *Zastosowanie krzywych eliptycznych do konstrukcji bezpiecznych algorytmów kryptograficznych*, Studia Bezpieczeństwa Narodowego, 6, s. 61-80

Gupta I., Singh J., Chaudhary R., 2007. *Cryptanalysis of an Extension of the Hill Cipher*, Cryptologia, 31 (3), s. 246-253

Goldman R., 2009. *An Integrated Introduction to Computer Graphics and Geometric Modeling*, CRC Press, New York

Goldman R., 2011. *Understanding quaternions*, Graphical Models, 73 (2), s. 21-49

Goldwasser S., Bellare M., 2008. *Lecture Notes on Cryptography*, MIT Computer Science and Artificial Intelligence Laboratory 2008, s. 85-118

Hamilton W.R., 1844. *On quaternions, or on a new system of imaginaries in algebra*, Philosophical Magazine, edited by David R. Wilkins 2000, dostępne na: http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/OnQuat/OnQuat.pdf

Hermans J., Vercauteren F., Preneel B., 2011. *Speed records for NTRU*, Department of Electrical Engineering, dostępne na: http://homes.esat.kuleuven.be/~fvercaut/papers/ntru_gpu.pdf

Hoffstein J., Pipher J., Silverman J.H., 2008. *An Introduction to Mathematical Cryptography*. Science+Business Media, Springer

Hsiao S.-F., Delosme J.-M., 1994. *Parallel processing of complex data using quaternion and pseudo-quaternion CORDIC algorithms*, w: Proceedings of the ASAP 1994 Conference, University of California, s. 125-130

Johnson J., Kaliski B., 2013. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*, dostępne na: http://www.ietf.org/rfc/rfc3447.txt

Kaczorowski J., 2014. *Zastosowanie funkcji L w kryptologii*, w: Studia Bezpieczeństwa Narodowego, WAT, 6, s. 259-270

Kahan W., 1997. *IEEE Standard 754 for Binary Floating-Point Arithmetic*, October 1 1997, s. 1-30

Katz A., 1996. *Computational Rigid Vehicle Dynamics*, Krieger Publishing Co.

Kipnis A., Shamir A., 1999. *Cryptanalysis of the HFE public key cryptosystem by relinearization*, Lecture Notes in Computer Science 1666, s. 19-30

Koblitz N., 1987. *Elliptic curve cryptosystems*, Mathematics of Computation, 48 (177), s. 203-209

Kudrewicz J., 1993. *Fraktale i chaos. Wyd. II*. Warszawa: Wydawnictwa Naukowo-Techniczne, ISBN 83-204-1676-0

Kuipers J.B., 1999. *Quaternions and rotation sequences*. Princeton, NJ: Princeton University Press

Kundur D., Karthik K., 2004. *Video fingerprinting and encryption principles for digital rights management*. Proc. IEEE. 92 (6), s. 918-932

Kunze K., Schaeben H., 2004. *The Bingham Distribution of Quaternions and Its Spherical Radon Transform in Texture Analysis*, Mathematical Geology, 36 (8), s. 917–943

Liu K.J.R., Zhao H., 2004. *Bandwidth efficient fingerprint multicast for video streaming*, w: Proc. IEEE International Conf. on Acoust. Speech and Signal Process. (ICASSP ’04), Montreal, vol. 5, s. 849-852

Liu K.J.R., Trappe W., Wang Z.J., Wu M., Zhao H., 2005. *Multimedia fingerprinting forensics for traitor tracing*, EURASIP Book Ser. on Signal Process. and Commun. Hindawi Publishing Corporation. vol. 4.

Maity S., Kundu M., Das T., 2007. *Robust SS watermarking with improved capacity*. Pattern Recognit. Lett. 28 (3), s. 350-356

Malekian E., Zakerolhosseini A., Mashatan A., 2009. *QTRU: A Lattice Attack Resistant Version of NTRU PKCS Based on Quaternion Algebra*, dostępne na: https://eprint.iacr.org/2009/386.pdf

Marins J.L., Yun X., Bachmann E.R., McGhee R.B., Zyda M.J., 2001. *An extended kalman filter for quaternion-based orientation estimation using MARG sensors*, w: Proceedings of the 2001 IEEE/RSJ, International Conference on Intelligent Robots and Systems, s. 2003-2011

Marsaglia G., 1995. *Diehard Battery of Tests of Randomness*, dostępne na: https://wayback.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/

MathWorks, 1994-2016. *MATLAB, the language of technical computing, and Simulink, for simulation and Model-Based Design*, dostępne na: http://www.mathworks.com

Matsui M., 1993. *Linear cryptanalysis method for DES cipher*, w: Advances in Cryptology - Eurocrypt '93, LNCS 765, red. T. Helleseth, Berlin 1994, s. 386-397

Menezes A.J., Oorschot P.C., Vanstone S.A., 1996. *Block Ciphers*, w: Handbook of Applied Cryptography, CRC Press, s. 223-271

Miller V., 1985. *Use of elliptic curves in cryptography*, CRYPTO '85. Lecture Notes in Computer Science, s. 417-426

Nagase T., Koide R., Araki T., Hasegawa Y., 2004. *A new Quadripartite Public-Key Cryptosystem*, ISCIT 2004, s. 74-79

Nagase T., Komata M., Araki T., 2004. *Secure Signals Transmission Based on Quaternion Encryption Scheme*, AINA 2004, s. 1-4

Nagase T., Koide R., Araki T., Hasegawa Y., 2005. *Dispersion of Sequences for Generating a Robust Enciphering System*, ECTI 2005, s. 9-14

Nalty K., 2008. *A Quaternion Toolbox for Four Dimensional Euclidean Spacetime*, dostępne na: http://www.kurtnalty.com/QuaternionToolbox.pdf

Narayan K.L., Ibrahim S.J.A., 2013. *Optimal Cryptographic Technique to increase the Data Security*, International Journal of Electronics Communication and Computer Technology (IJECCT), 3 (2), s. 398-402

NEMA, 2008. *Digital Imaging and Communications in Medicine (DICOM) Part 15: Security and System Management Profiles*

Quat 1.20, *A 3D Fractal Generator*, dostępne na: http://www.physcip.uni-stuttgart.de/phy11733/quat_e.html

OsiriX, 2009. *Zaawansowana stacja robocza PACS i przeglądarka plików DICOM*, dostępne na: http://www.osirix.com.pl/Strony/features.html

OsiriX, 2010. *OsiriX Technical Guides*, dostępne na: http://www.osirix-viewer.com/Documentation/Guides/index.html

Philips, 2012. *CT Scanners and Workstations V2/V3 Rev. 7*, dostępne na: http://www.healthcare.philips.com/main/about/Connectivity/dicom_statements/ct_statements.wpd

Pianykh O.S., 2008. *Digital Imaging and Communications in Medicine (DICOM): A practical Introduction and Survival Guide*, Springer-Verlag

Sangwine S., Bihan N.L., 2005. *Quaternion toolbox for Matlab*, dostępne na: http://qtfm.sourceforge.net

Sastry V.U.K., Kumar K.A., 2012. *A Modified Feistel Cipher Involving Modular Arithmetic Addition and Modular Arithmetic Inverse of a Key Matrix*, International Journal of Advanced Computer Science and Applications (IJACSA 2012), 3 (7), s. 40-43

Schneier B., 1996. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Second Edition, John Wiley & Sons

Shannon C., 1949. *Communication Theory of Secrecy Systems*, Bell System Technical Journal, vol. 28, (4), s. 656-715

Shao Z., Wu J., Coatrieux J.L., Coatrieux G., Shu H., 2013. *Quaternion gyrator transform and its application to color image encryption*. 20th IEEE International Conference on Image Processing (ICIP), s. 4579-4582

Stallings W., 2006. *Cryptography and Network Security: Principles and Practices*, Fourth Edition, Prentice Hall, 2006

Suthaharan S., Kim S.W., Lee H.K., Sathananthan S., 2000. *Perceptually tuned robust watermarking scheme for digital images*. Pattern Recognit. Lett. 21 (2), s. 145-149

Wang X.L., Zhai H.C., Li Z.L., Ge Q., 2011. *Double random-phase encryption based on discrete quaternion fourier-transforms*, Optik, 122 (20), s. 1856-1859

Wang Y., Pearmain A., 2004. *Blind image data hiding based on self reference*. Pattern Recognit. Lett. 25 (15), s. 1681-1689

Welstead S.T., 1999. *Fractal and wavelet image compression techniques*. SPIE Publ., s. 155-156

Zhang F., 1997. *Quaternion and Matrices of Quaternions*, w: Linear Algebra and its Applications, Elsevier Science Inc., New York, s. 21-57

WYKAZ OZNACZEŃ

Rozdział 4

X	wiadomość do zaszyfrowania
Y	zaszyfrowana wiadomość
K	klucz tajny
E_K	funkcja szyfrująca wiadomość X kluczem tajnym K
D_K	funkcja deszyfrująca zaszyfrowaną wiadomość Y kluczem tajnym K
P	klucz publiczny; klucz jawny
E_P	funkcja szyfrująca wiadomość X kluczem publicznym P

Rozdział 5

q	kwaternion
q^*	kwaternion sprzężony
$\|\|q\|\|$	norma kwaternionu
q^{-1}	kwaternion odwrotny

Rozdział 6

P	kwaternion o części skalarnej równej 0
P_{rot}	kwaternion o części skalarnej równej 0 po rotacji
$\mathbf{P}$	wektor danych; wartości współczynników części wektorowej kwaternionu P, zapisane w postaci macierzy
$\mathbf{\Gamma}(q)$	macierz rotacji
$\mathbf{B}$	macierz danych
$\mathbf{B}_{\text{rot}}$	zaszyfrowana macierz danych
B	kwaternion danych
B_{rot}	kwaternion danych po rotacji (zaszyfrowany)
q_0	kwaternion początkowy; kwaternion rzędu zerowego
$\mathbf{\Gamma}(q_0)$	początkowa macierz rotacji; macierz rotacji rzędu zerowego

Rozdział 8

$\mathbf{B}_n$	n-ty blok danych; n-ta macierz danych
$\mathbf{B}_{n\,\text{rot}}$	n-ty zaszyfrowany blok danych (tryb ECB i CBC); n-ta zaszyfrowana macierz danych

$\mathbf{B}_{n\ \mathrm{mod}}$	n-ty blok danych otrzymany po poelementowym zsumowaniu modulo 2 bloku $\mathbf{B}_n$ i bloku $\mathbf{B}_{n\text{-}1\ \mathrm{rot}}$
IM	macierz inicjalizująca
Ctr	blok licznika
$\mathbf{K}_{n\ \mathrm{mod}}$	n-ty blok klucza otrzymany po poelementowym zsumowaniu modulo 2 bloku licznika **Ctr** z macierzą inicjalizującą **IM**
$\mathbf{K}_{n\ \mathrm{rot}}$	n-ty blok klucza otrzymany po rotacji kwaternionowej n-tego bloku $\mathbf{K}_{n\ \mathrm{mod}}$
$\mathbf{B}_{n\ \mathrm{enc}}$	n-ty zaszyfrowany blok danych (tryb CTR)
$d \times d$	rozmiar kwadratowego obrazu wejściowego
$m \times m$	rozmiar kwadratowych macierzy **L** i **R** powstałych po przekształceniu obrazu wejściowego
L	lewa macierz kwadratowa o rozmiarze $m \times m$
R	prawa macierz kwadratowa o rozmiarze $m \times m$
L	lewy kwaternion, którego współczynnikami części wektorowej są macierze **L**
R	prawy kwaternion, którego współczynnikami części wektorowej są macierze **R**
N	wartość modulo (dla arytmetyki modularnej) równa 257
B_{enc}	kwaternion zaszyfrowany
K	macierz klucza o rozmiarze $m \times m$

Rozdział 9

p	dowolny punkt o postaci liczby zespolonej (zbiór Julia); dowolny punkt o postaci kwaternionu (kwaternionowy zbiór Julia)
z_n	ciąg opisany równaniem rekurencyjnym
c	liczba zespolona będąca parametrem zbioru Julia; kwaternion będący parametrem kwaternionowego zbioru Julia
k	płaszczyzna przekroju
f	kwaternion otrzymywany z trójkanałowego obrazu fraktala kwaternionowego zbioru Julia

Rozdział 10

$r \times c$	rozmiar przykładowego obrazu DICOM
B_{ini}	kwaternion inicjalizujący
B_{mod}	kwaternion otrzymany po binarnym zsumowaniu kwaternionu danych B i kwaternionu inicjalizującego B_{ini}

B_{mod}'	kwaternion otrzymany po stronie odbiorczej po odszyfrowaniu zaszyfrowanego kwaternionu B_{rot}
B'	kwaternion danych, odszyfrowany

Rozdział 11

$\overline{X}$	wartość oczekiwana z próby
S	odchylenie standardowe z próby
N	wielkość próby
t_α	wartość z tablicy *t*-Studenta dla *N*-1 stopni swobody
1-α	współczynnik ufności
$M \times N$	wielkość obrazu wejściowego (fingerprinting)
$d_{m,n}$	wartość piksela dla *m*-tego wiersza i *n*-tej kolumny w odszyfrowanym obrazie
$x_{m,n}$	wartość piksela dla *m*-tego wiersza i *n*-tej kolumny w obrazie oryginalnym
Δ	sumaryczne błędy podczas deszyfracji dla wszystkich trzech kanałów reprezentacji RGB
r	współczynnik korelacji dla fingerprintów obrazów kolorowych
$p_{m,n,c}$	wartość fingerprintu osadzonego w pikselu *m*-tego wiersza, *n*-tej kolumny i *c*-tego kanału (dla reprezentacji RGB) pirackiej kopii
$f_{m,n,c}$	wartość fingerprintu osadzonego w pikselu *m*-tego wiersza, *n*-tej kolumny i *c*-tego kanału (dla reprezentacji RGB) zweryfikowanej, legalnej kopii użytkownika
$\overline{p}$	wartość średnia wszystkich pikseli kopii pirackiej
$\overline{f}$	wartość średnia wszystkich pikseli zweryfikowanej, legalnej kopii użytkownika

Rozdział 12

n	liczba rund w algorytmach F-QFC i S-QFC; liczba kluczy rundowych
$m \times m$	rozmiar kwadratowej macierzy klucza **K**
l	liczebność kwaternionów *f*, potrzebnych do wygenerowania kluczy rundowych dla algorytmu S-QFC
a	dowolna liczba całkowita z zakresu od 1 do 256 oraz $a \neq b$
b	dowolna liczba całkowita z zakresu od 1 do 256 oraz $a \neq b$
$\mathbf{R}_1$, $\mathbf{R}_2$	fragmenty tekstu jawnego (postać macierzowa)
$\mathbf{R}_1'$, $\mathbf{R}_2'$	fragmenty tekstu zaszyfrowanego (postać macierzowa)

$\mathbf{X}$	wynik mnożenia macierzy $\mathbf{R}_1'$ i $\mathbf{R}_2'^{-1}$
$\mathbf{Y}$	wynik mnożenia macierzy $\mathbf{R}_2$ i $\mathbf{R}_1^{-1}$
$\mathbf{U}$, $\mathbf{V}$	macierze związane odpowiednio z postaciami Jordana $\mathbf{J}_{\mathbf{R}1'}$ i $\mathbf{J}_{\mathbf{R}1}$
$\mathbf{J}_{\mathbf{R}1'}$, $\mathbf{J}_{\mathbf{R}1}$	postacie Jordana odpowiednio macierzy $\mathbf{R}_1'$ i $\mathbf{R}_1$
s_i	wartości dowolne
M	liczba wartości dowolnych s_i
$\tilde{\mathbf{K}}$	macierz pozwalająca na wyznaczenie macierzy klucza $\mathbf{K}$
$\mathbf{K}_{\alpha\beta}$	macierz definiująca strukturę macierzy $\tilde{\mathbf{K}}$
α	indeksacja wartości własnych dla macierzy $\mathbf{R}_1'$
β	indeksacja wartości własnych dla macierzy $\mathbf{R}_1$
u	liczba wartości własnych macierzy $\mathbf{R}_1'$
v	liczba wartości własnych macierzy $\mathbf{R}_1$
$N \times N$	przyjęta wielkość macierzy $\mathbf{R}_1'$, $\mathbf{R}_1$ i $\mathbf{K}$

WYKAZ SKRÓTÓW

3DES	Triple Data Encryption Standard
ACR	American College of Radiology
AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange
CBC	Cipher Block Chaining; tryb wiązania bloków zaszyfrowanych
CTR	Counter; tryb licznikowy
DCT	Discrete Cosine Transform; dyskretna transformata kosinusowa
DICOM	Digital Image and Communication On Medicine; standard medyczny dla zapewnienia bezpiecznej transmisji i wymiany danych
DQFT	Discrete Quaternion Fourier Transform; dyskretna kwaternionowa transformata Fouriera
DRM	Digital Rights Management; zarządzanie prawami cyfrowymi
ECB	Electronic Code Book; tryb elektronicznej książki kodowej
F-QFC	Fast Quaternion Feistel Cipher
HTTPS	Hypertext Transfer Protocol Secure
HW-QES	Hardware - Oriented Quaternion Encryption Scheme; sprzętowa implementacja kwaternionowego algorytmu szyfrującego
MAC	Message Authentication Code; kod uwierzytelniający wiadomość
M-QES	Quaternion Encryption Scheme Modification; zmodyfikowany kwaternionowy algorytm szyfrujący
NEMA	National Electrical Manufacturers Association
NTRU	asymetryczny, probabilistyczny system kryptograficzny
PSNR	Peak Signal-to-Noise Ratio; szczytowy stosunek sygnału do szumu
QES	Quaternion Encryption Scheme; kwaternionowy algorytm szyfrujący
QFC	Quaternion Feistel Cipher
QGT	Quaternion Gyrator Transform; kwaternionowa transformacja żyratora
QTRU	kwaternionowa modyfikacja systemu kryptograficznego NTRU
S-QFC	Secure Quaternion Feistel Cipher
TLS	Transport Layer Security
VoD	Video on Demand; wideo na żądanie
VPN	Virtual Private Network; wirtualna sieć prywatna

WYKAZ RYSUNKÓW I TABEL

Wykaz rysunków

Wykaz tabel

ZAŁĄCZNIKI

Załącznik 1: Algorytm S-QFC: Przykład obliczenia macierzy klucza **K** przy znajomości macierzy tekstu jawnego $\mathbf{R}_1$ i macierzy tekstu zaszyfrowanego $\mathbf{R}_1'$.

Dla uproszczenia w niniejszym przykładzie zastosowano macierze wielkości 3×3 z pominięciem arytmetyki modularnej. Macierze **K**, $\mathbf{R}_1'$ i $\mathbf{R}_1$ powiązane są zależnością (12.15). Załóżmy następujące postacie macierzy:

$$\mathbf{R}_1'=\begin{bmatrix} 1 & -3 & -2 \\ -1 & 1 & -1 \\ 2 & 4 & 5 \end{bmatrix} \qquad \mathbf{K}=\begin{bmatrix} 1 & 1 & 2 \\ -2 & -1 & -3 \\ 1 & -2 & 0 \end{bmatrix} \qquad \mathbf{R}_1=\begin{bmatrix} -13 & -36 & -38 \\ -6 & -12 & -15 \\ 12 & 28 & 32 \end{bmatrix} \quad \text{(Z1.1)}$$

Chcąc poznać postać macierzy **K** (12.22), należy najpierw wyznaczyć postacie Jordana macierzy $\mathbf{R}_1'$ i $\mathbf{R}_1$ (12.23). Ponieważ macierze $\mathbf{R}_1'$ i $\mathbf{R}_1$ są podobne, posiadają ten sam wielomian charakterystyczny, te same wartości własne o identycznych krotnościach i tą samą postać Jordana:

$$\mathbf{R}_1': \qquad \mathbf{U}=\begin{bmatrix} -1 & 1 & -1 \\ -1 & 0 & 0 \\ 2 & 0 & 1 \end{bmatrix} \qquad \mathbf{J}_{\mathbf{R}_1'}=\begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} \quad \text{(Z1.2)}$$

$$\mathbf{R}_1: \qquad \mathbf{V}=\begin{bmatrix} -8 & -2/3 & -5 \\ -3 & 1/2 & -2 \\ 6 & 0 & 4 \end{bmatrix} \qquad \mathbf{J}_{\mathbf{R}_1}=\begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} \quad \text{(Z1.3)}$$

Wartości własne dla macierzy $\mathbf{R}_1'$ i $\mathbf{R}_1$ to $\lambda_1 = 2$ i $\lambda_2 = 3$ o krotnościach odpowiednio 2 i 1. Macierz $\tilde{\mathbf{K}}$ składać się będzie z 4 macierzy $\mathbf{K}_{\alpha\beta}$ (α i β = 1, 2 ponieważ macierze $\mathbf{R}_1'$ i $\mathbf{R}_1$ posiadają dwie wartości własne):

$$\tilde{\mathbf{K}}=\begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix} \quad \text{(Z1.4)}$$

gdzie macierze $\mathbf{K}_{\alpha\beta}$ mają następującą postać (rozmiar macierzy określany jest na podstawie krotności wartości własnych):

$$\mathbf{K}_{11}=\begin{bmatrix} s_1 & s_2 \\ 0 & s_1 \end{bmatrix} \qquad \mathbf{K}_{12}=\begin{bmatrix} s_3 \\ 0 \end{bmatrix} \qquad \mathbf{K}_{21}=\begin{bmatrix} 0 & s_4 \end{bmatrix} \qquad \mathbf{K}_{22}=\begin{bmatrix} s_5 \end{bmatrix} \quad \text{(Z1.5)}$$

Uwzględniając postaci macierzy $\mathbf{K}_{\alpha\beta}$ (Z1.5) macierz $\tilde{\mathbf{K}}$ można zapisać jako:

$$\tilde{\mathbf{K}} = \left[\begin{array}{cc:c} s_1 & s_2 & s_3 \\ 0 & s_1 & 0 \\ \hdashline 0 & s_4 & s_5 \end{array}\right] \tag{Z1.6}$$

Sprawdzając wszystkie możliwe postaci macierzy **K** dla dowolnych wartości s_1, ..., s_5, otrzymamy ostatecznie, że dla wartości s_1=-1, s_2=-5/6, s_3=0, s_4=0, s_5=-1 macierz $\tilde{\mathbf{K}}$ po podstawieniu do wzoru (12.22) pozwoli na uzyskanie macierzy **K**, o postaci przedstawionej w (Z1.1).

Należy pamiętać, że macierz klucza **K**, występująca w algorytmie S-QFC, posiada wartości całkowitoliczbowe z zakresu od 0 do 256, z uwagi na zastosowanie arytmetyki modularnej modulo 257, zatem obliczenie modularnej macierzy $\tilde{\mathbf{K}}$ będzie operacją dużo bardziej złożoną.

Załącznik 2: Płyta CD.

Do rozprawy doktorskiej dołączona została płyta CD zawierająca wszystkie opisane w pracy algorytmy szyfrowania kwaternionowego, przykładowe obrazy, narzędzie do realizacji operacji kwaternionowych - *toolbox* kwaternionowy dla programu MATLAB oraz wersję elektroniczną rozprawy.

Dodatkowo na płycie umieszczono plik *README.txt*, w którym dokładnie opisano zawartość płyty CD oraz objaśniono proces przygotowania środowiska pracy do przeprowadzenia testów.