Małgorzata Górska, Jarosław Molendowski

Eksperymentalne wykorzystanie wybranych algorytmów steganograficznych oraz kryptograficznych – stanowisko laboratoryjne

JEL: L62 DOI: 10.24136/atest.2019.145
Data zgłoszenia: 05.04.2019 Data akceptacji: 26.06.2019

Głównym zadaniem omówionym w niniejszej publikacji było przygotowanie stanowiska laboratoryjnego do badania poufności przetwarzanej informacji. W pierwszej części artykułu przedstawiono porównanie możliwości dwóch programów o licencji otwartej – narzędzi służących do szyfrowania, realizujących szyfrowanie z wykorzystaniem szyfru blokowego AES. W części drugiej opisane zostało działanie dwóch programów stworzonych w C++ na potrzeby stanowiska laboratoryjnego. Pierwszy realizuje szyfrowanie klasyczne, przestawieniowe, drugi do szyfrowania używa operacji XOR.

Słowa kluczowe: informacja, przetwarzanie, szyfrowanie.

Wstęp

Pierwsze informacje o potrzebie zapewniania bezpieczeństwa informacji pochodzą z czasów cesarstwa rzymskiego. Już wtedy żołnierze wiedzieli, że najważniejszym elementem wojny jest zaskoczenie, które mogło zapewnić jedynie bezpieczne przekazywanie informacji o planach i rozkazach. Sposoby ukrywania tajnych wiadomości od tamtych czasów były stale modyfikowane i udoskonalane. Równolegle rozwijały się również nauki o łamaniu szyfrów. Duży postęp w rozwoju szyfrowania przyniosły wojny światowe – wtedy stopień zaawansowania szyfrów a z drugiej strony skuteczność działania jednostek zajmujących się deszyfrowaniem decydowały o zwycięstwie.

Rewolucja w szyfrowaniu nastąpiła wraz z pojawieniem się komputerów. W dzisiejszych czasach nie da się wyobrazić sobie życia bez szyfrów zapewniających bezpieczeństwo kont bankowych, transakcji internetowych, czy danych przechowywanych na serwerach rządowych. Bardzo zaawansowane i skomplikowane szyfry stosowane przez wojsko ukrywają informacje, które w rękach nieodpowiedniej osoby mogą stać się niebezpieczną bronią.

W artykule przedstawiając porównanie możliwości dwóch często obecnie wykorzystywanych, darmowych programów służących do szyfrowania – VeraCrypt oraz Boxcryptor realizujących szyfrowanie z wykorzystaniem szyfru blokowego AES, pominięto rozważania teoretyczne dotyczące steganografii, kryptografii oraz kryptoanalizy. Zagadnienia historyczne jak i opis szyfrów stosowanych współcześnie zostały niejednokrotnie dokładnie pisane w wielu książkach i innych publikacjach [1,2,4,6,11].

1.Praktyczne wykorzystanie kryptografii

1.1. Skuteczny algorytm szyfrowania AES

Współcześnie istnieje wiele programów realizujących szyfrowanie. Dzięki nim użytkownicy nie muszą zagłębiać się w struktury i kody szyfrów czy dokonywać skomplikowanych obliczeń matematycznych. Aplikacje te opierają się na różnych algorytmach szyfrowania w związku z czym poziom bezpieczeństwa jest bardzo zróżnicowany. Jednymi z najbardziej popularnych są te wykorzystujące algorytm Rijndael lub inaczej AES [5,7,10].

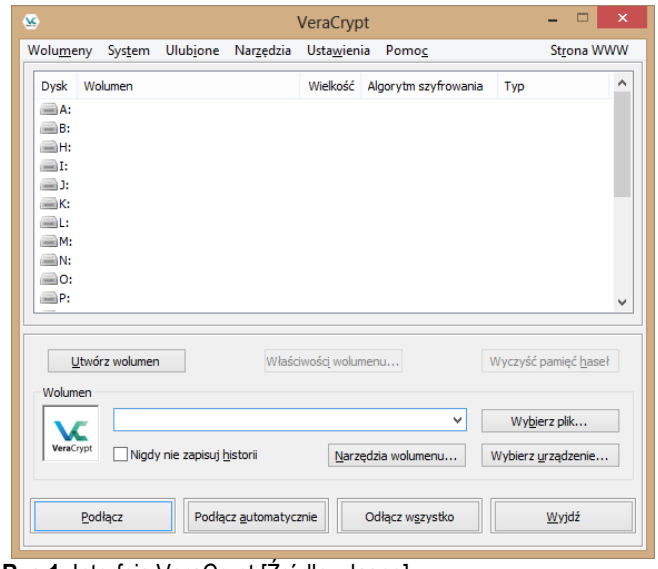
W 1997 roku Amerykański Instytut Normalizacji ogłosił konkurs na utworzenie narodowego standardu szyfrowania nazywanego AES. W konkursie mogły brać udział tylko algorytmy symetryczne, zapewniające wysoki poziom bezpieczeństwa, któremu nie może zagrażać ujawnienie sposobu działania algorytmu.

W konkursie zwyciężył Rijandael. Zostało to uzasadnione tym, że był on odporny na wszelkie znane próby złamania czy ataku, okazał się również najszybszym ze wszystkich kandydatów na wszystkich badanych platformach. Ponadto wykorzystuje on minimalne ilości pamięci ROM i RAM co pozwala na stosowanie go w pracy z mikrokontrolerami czy z wymagającymi aplikacjami [3].

Poniżej porównano działanie dwóch programów do szyfrowania wykorzystujących algorytm AES. Analizowane programy to Boxcryptor i Veracrypt – które są udostępnione w darmowych wersjach.

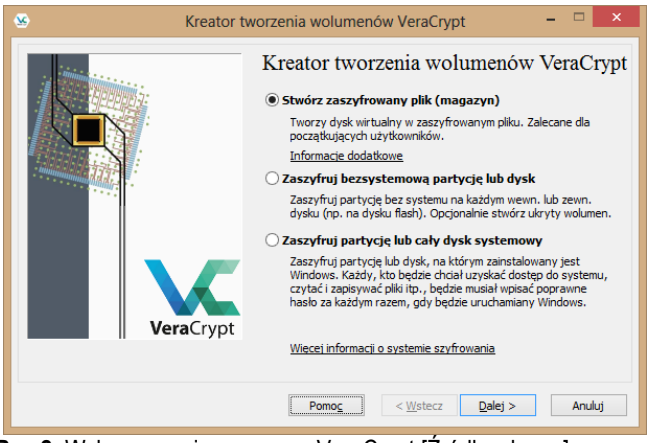
1.2.Sekwencje generowania szyfrowanego pliku w VeraCrypt

Interfejs VeraCrypt jest bardzo przejrzysty i prosty w obsłudze. Wszystkie potrzebne elementy są widoczne, nie ma nadmiaru niepotrzebnej grafiki (Rys.1).



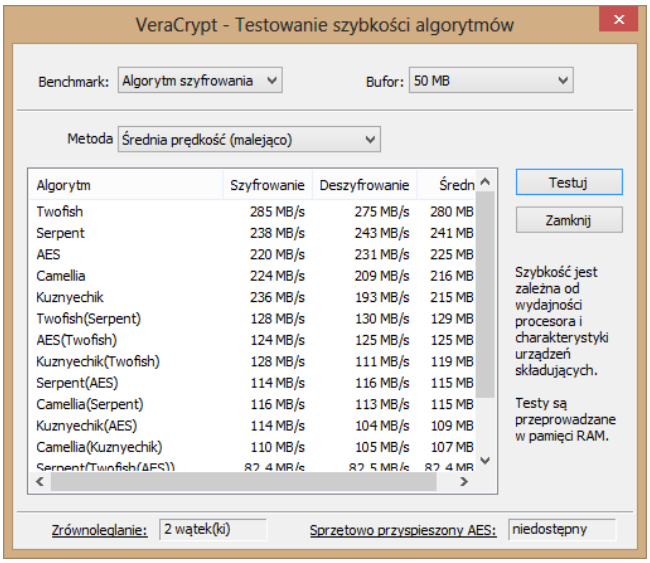
Rys.1. Interfejs VeraCrypt [Źródło własne]

Program ma zaawansowane możliwości. Pozwala na tworzenie zaszyfrowanych folderów, szyfrowanie pojedynczych partycji anawet dysków systemowych. Ma również możliwość szyfrowania pamięci zewnętrznych (Rys.2).



Rys.2. Wybrane opcje programu VeraCrypt [Źródło własne]

Jedną z opcji aplikacji jest testowanie szybkości algorytmów. Ukazuje on, który algorytm będzie działał najszybciej przy danej wielkości szyfrowanego folderu (Rys.3).

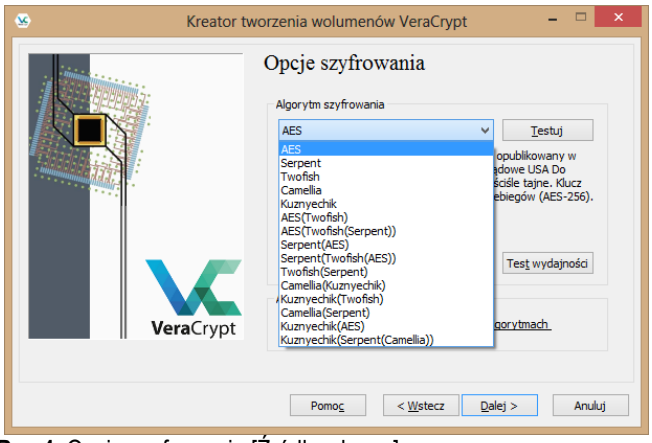


Rys.3. Testowanie szybkości algorytmów szyfrowania w VeraCrypt [Źródło własne]

W celu wygenerowania zaszyfrowanego wolumenu po wybraniu opcji *stwórz zaszyfrowany plik* w kreatorze wolumenów pojawiają się dwie opcje. Należy dokonać wyboru między tworzeniem widocznego prostego folderu lub wolumenu ukrytego co pozwala dodatkowe zabezpieczanie danych.

Kolejnym krokiem w tworzeniu ukrytego pliku jest wybór lokalizacji. Istotną cechą jest to, że jeśli jako lokalizacja zostanie wybrany folder istniejący nie będzie on zaszyfrowany lecz usunięty i zamieniony wolumenem szyfrowanym.

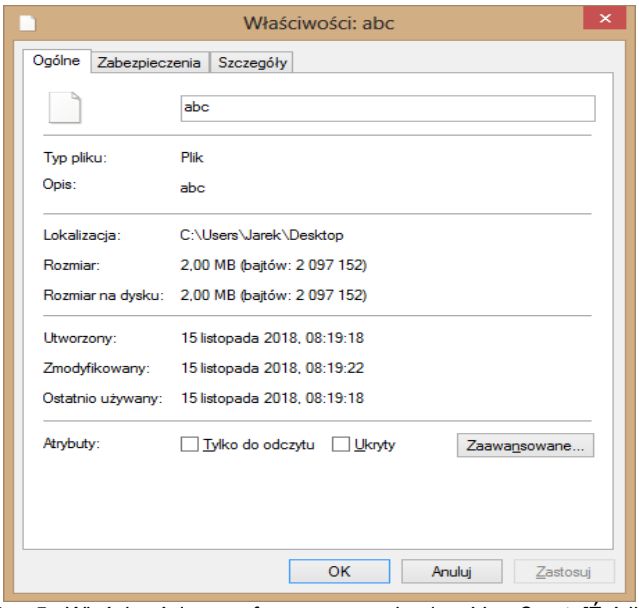
Następnie pojawia się okno wyboru szyfrowania (rys.4). Jest to najważniejszy element działania VeraCrypt. Różni się on od innych programów tym, że poza możliwością wyboru pojedynczego algorytmu szyfrującego, może tworzyć szyfr mieszany, łączący możliwości dwóch, lub trzech algorytmów.



Rys.4. Opcje szyfrowania [Źródło własne]

Po dokonaniu kolejnego kroku jakim jest wybór wielkości pliku należy wybrać hasło. Veracrypt wyróżnia się spośród konkurencji tym, że daje możliwość wykorzystania tak zwanych plików-kluczy. Jest to dodatkowe zabezpieczenie, które przy próbie wykradnięcia danych poza wpisaniem hasła, wymaga wskazania odpowiedniego pliku.

Ostatnim etapem jest wybór systemu plików, oraz stworzenie poprzez dokonywanie losowych ruchów myszką dodatkowego klucza, który poprawia jakość szyfrowania. Wolumen zapisuje się jako plik bez rozszerzenia(Rys.5).



Rys.5. Właściwości zaszyfrowanego woluminu VeraCrypt [Źródło własne]

Otwieranie zaszyfrowanego folderu odbywa się poprzez wybranie odpowiednich opcji w głównym oknie. Program wymaga podania hasła oraz ewentualnych plików-kluczy. Następnie montuje zaszyfrowany folder jako wirtualną partycję.

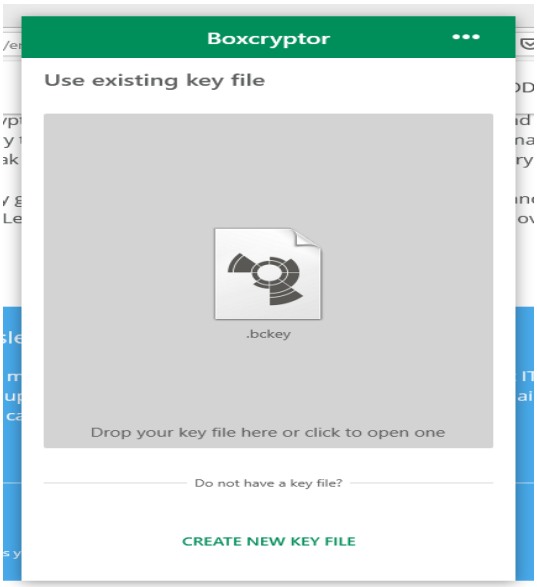
Opcją programu, która poprawia bezpieczeństwo jest tworzenie folderów ukrytych. Folder ukryty w rzeczywistości składa się z dwóch zaszyfrowanych folderów. Pierwszy – Zewnętrzny jest widoczny na ekranie komputera podobnie jak zwykły plik programu VeraCrypt, drugi zostaje ukryty. Każdy z nich ma swoje hasło. Procedura otwarcia jest taka sama jak w przypadku zwykłego folderu, różnica pojawia się dopiero przy wpisywaniu hasła, program otwiera tylko ten folder, którego hasło zostaje podane. Jest to ochrona przed atakiem fizycznym, gdy np. użytkownik zostaje zmuszony do podania hasła do zaszyfrowanego folderu.

Procedura szyfrowania partycji i dysków niesystemowych jest taka sama jak w przypadku zwykłych folderów. Szyfrowanie dysku systemowego wymaga wybrania dysku, oraz systemu, który ma być zaszyfrowany lub ukryty. Skutkiem działania programu jest zmuszenie użytkownika do wpisania hasła zawsze gdy uruchamiany jest system.

1.3.Przebieg operacji szyfrowania w Boxcryptor

Boxcryptor jest zaawansowanym programem pozwalającym na szyfrowanie plików zarówno w systemie lokalnym jak i w chmurze.

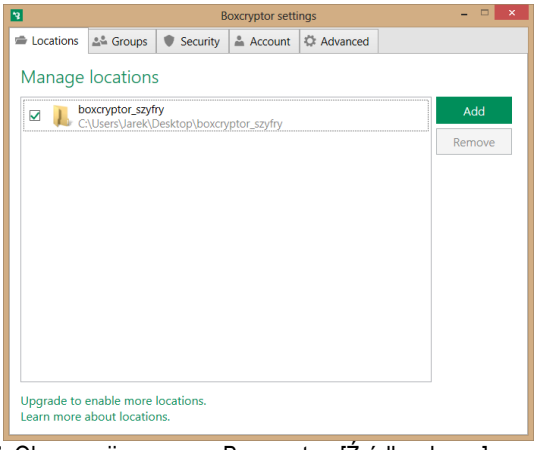
Ważną zaletą programu jest konieczność wyboru pliku-klucza który będzie dodatkowym zabezpieczeniem (Rys.6) jeśli taki plik nie istnieje program sam go stworzy. Następnie program prosi o podanie hasła. Jeśli hasło lub plik-klucz zaginie nie ma możliwości odzyskania dostępu do zaszyfrowanych plików.



Rys. 6. Wybór plików kluczy w Boxcryptor [Źródło własne]

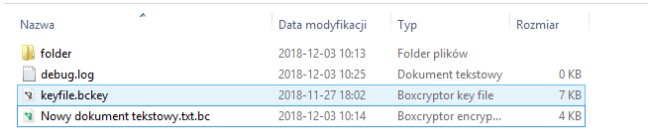
Istotną zaletą ze strony użytkownika jest krótkie pokazanie możliwości programu, oraz instrukcja szyfrowania plików – generowane w programie.

Po przedstawieniu instrukcji ukazuje się okno ustawień programu (Rys.7). W tym kroku dochodzi do wyboru folderu, w którym będą znajdowały się zaszyfrowane pliki. Okno pozwala również na zarządzanie hasłami oraz kontem użytkownika (w przypadku szyfrowania w chmurze).



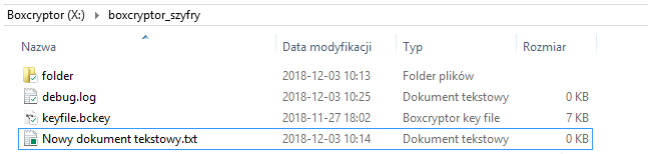
Rys. 7. Okno opcji programu Boxcryptor. [Źródło własne]

Szyfrowanie jest mało skomplikowane i dużo szybsze niż w VeraCrypt. Operacja polega na wybraniu pliku znajdującego się w folderze wybranym w poprzednim kroku, następnie należy kliknąć na niego prawym przyciskiem myszy, po czym wybrać opcję *boxcryptor/encrypt*.



Rys. 8. Zmiana rozszerzenia pliku w Boxcryptor [Źródło własne]

Po zaszyfrowaniu plik zmienia rozszerzenie na .bc (Rys. 8). Dostęp do pliku jest możliwy dopiero po zalogowaniu w boxcryptor, wtedy rozszerzenie powraca do oryginalnej formy(Rys.9).



Rys. 9. Dostęp do zaszyfrowanego pliku Boxcryptor [Źródło własne]

Odszyfrowanie przebiega w taki sam sposób jak szyfrowanie – na końcu zamiast opcji *encrypt* należy wybrać *decrypt,* po czym zawartość pliku może ponownie odczytać każdy.

1.4.Podsumowanie – porównanie analizowanych programów szyfrujących

Program Boxcryptor, podobnie jak VeraCrypt korzysta z bardzo bezpiecznego algorytmu AES w wersji 256 bitowej, jednak VeraCrypt daje możliwość użycia wielu innych zaawansowanych szyfrów oraz kombinacji złożonych z dwóch lub trzech algorytmów co daje znacznie większe bezpieczeństwo.

Veracrypt dla każdego pliku nadaje osobne hasło, co sprawia, że udany, pojedynczy atak pozwala na odszyfrowanie tylko jednego kontenera. Poważną wadą Boxcryptor jest przypisanie jednego hasła do otwierania wszystkich zakodowanych plików na danej platformie.

Zaletami Boxcryptor w porównaniu do VeraCrypt są: szyfrowanie plików w chmurze, oraz wersja na urządzenia mobilne. Przewagą VeraCrypt jest opcja szyfrowania całych partycji, partycji systemowej a nawet dysku, co poprawia zabezpieczenia całego komputera. Tworzenie zaszyfrowanych plików i folderów w obu aplikacjach nie jest bardzo skomplikowanie.

Ważną kwestią jest aspekt ekonomiczny – VeraCrypt jest darmowym narzędziem otwarto-źródłowym nie ma ograniczeń co do wielkości i ilości szyfrowanych folderów. Boxcryptor w darmowej wersji ogranicza się do jednego folderu o maksymalnej wielkości 2GB, wykupienie wersji płatnej znacznie poprawia możliwości programu.

2.Projekt stanowiska laboratoryjnego

2.1.Autorski program SzyfrCezara

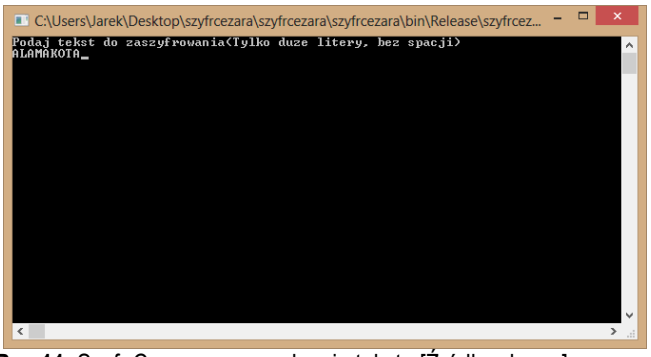
Program o nazwie SzyfrCezara (Rys. 10), stworzony na potrzeby stanowiska laboratoryjnego w C++, wykorzystuje dwie biblioteki iostream i string. Biblioteka iostream odpowiada za operacje wejścia/wyjścia. Biblioteka string opisuje ciągi znaków [13,14].

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string kod;
    int p;
    int a = 0;
    cout << "Podaj tekst do zaszyfrowania(Tylko duze litery, bez spacji)" << endl;
    cin >> kod;

    cout << "Podaj wartosc przesuniecia" << endl;
    cin >> p;


    do
    {
        if(kod[a]>=91-p && kod[a]<=90)
        {
            kod[a]=kod[a]-26+p;
        }
        else
        {
            kod[a]=kod[a]+p;
        }
        a++;
    }
    while (a !=kod.size());
    cout << kod << endl;

    return 0;
}
```

Rys. 10. Listing programu SzyfrCezara [Źródło własne]

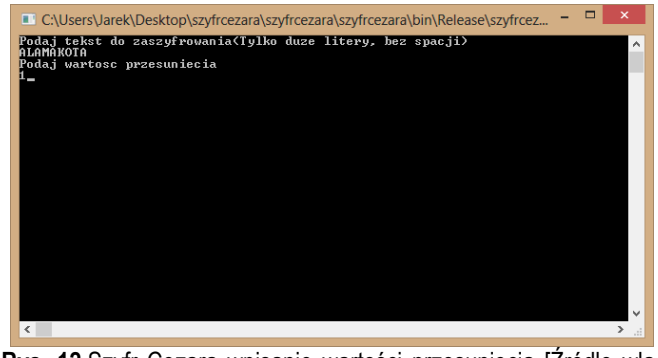
Program zawiera trzy zmienne: Zmienną *kod* typu string, zmienną *p* typu int oraz zmienną *a* typu int o przypisanej wartości 0.

Program na początku działania z wykorzystaniem polecenia cout wypisuje komunikat *Podaj tekst do zaszyfrowania(Tylko duze litery, bez spacji)* (Rys. 11). Po czym przypisuje z wykorzystaniem polecenia *cin* tekst wprowadzony przez użytkownika do zmiennej *kod*.



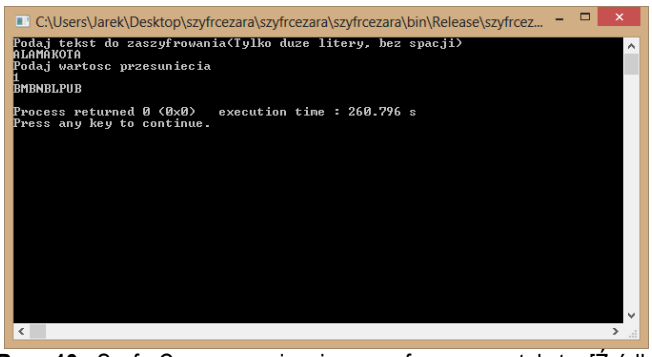
Rys.11. Szyfr Cezara wprowadzenie tekstu [Źródło własne]

Kolejnym krokiem jest wypisanie komunikatu *Podaj wartość przesunięcia* oraz przypisanie wartości przesunięcia poszczególnych liter do zmiennej *p* (Rys. 12).



Rys. 12.Szyfr Cezara wpisanie wartości przesunięcia [Źródło własne]

Następnie program rozpoczyna wykonanie pętli do while. Pętla zawiera w sobie instrukcję warunkową if. Na początku pętla sprawdza jakiej wartości w kodzie ASCII odpowiada znak o numerze *a* z wczytanego ciągu *kod*. Jeśli wartość znaku jest większa lub równa 91 – *p* (wartość przesunięcia) oraz jednocześnie mniejsza lub równa 90 to pętla najpierw odejmie od tej wartości 26, po czym doda wartość przesunięcia. Na przykład dla wpisanej w programie litery Z, która w kodzie ASCII ma wartość 90 oraz przesunięcia 2 program najpierw wykona obliczenie 90-26 po czym do wyniku doda 2 otrzymując numer 66, któremu w kodzie ASCII odpowiada litera B. Jeśli wartość danego znaku nie spełnia początkowego warunku, pętla doda wartość przesunięcia *p* do tego znaku. Po zakończeniu jednej pętli ta sama operacja zostaje wykonana na kolejnym znaku z ciągu. Pętla będzie wykonywana dopóki wartość *a* będzie równa lub mniejsza od liczby znaków w ciągu. Po zakończeniu pracy pętli program wypisuje z wykorzystaniem polecenia cout wartość zmiennej kod po przesunięciu, która jest zaszyfrowanym tekstem (Rys. 13).

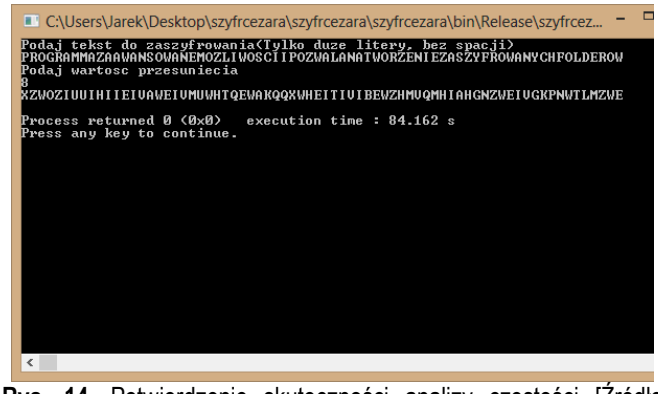


Rys. 13. Szyfr Cezara wypisanie zaszyfrowanego tekstu [Źródło własne]

Skuteczną metodą łamania Szyfru Cezara jest analiza częstości, która pozwala na odczytanie zaszyfrowanego tekstu w bardzo prosty sposób. Do wykorzystania analizy częstości wymagana jest znajomość języka, w którym zakodowany jest tekst. Analiza polega na odnalezieniu najczęściej występujących liter w tekście i porównaniu ich z najczęściej występującymi literami w danym języku. Dla szyfru Cezara wystarczające jest odnalezienie jednej, najczęściej występującej litery, następnie sprawdzenie o ile miejsc znajduje się dalej od litery A. Ostatnim krokiem jest cofnięcie pozostałych liter tekstu o tyle miejsc, o ile została przesunięta litera A i sprawdzenie czy odszyfrowany w ten sposób tekst ma sens. W celu zobrazowania metody zaszyfrowano pewien tekst. Wynikiem szyfrowania jest następujący ciąg znaków:

XZWOZIUUIHIIEIVMUWHTQEWAKQQXWHEITIVIBEWZHMV QMHIAHGNZWEIVGKPNWTLMZWE.

Analiza częstości wykazała, że najczęściej występującą literą, która pojawiła się aż 10 razy jest *I*. Różnica w kodzie ASCII między literą A i I wynosi 8. Należy więc wszystkie litery cofnąć o osiem miejsc w kodzie ASCII oraz dodać odpowiednie przerwy między wyrazami. Wynikiem deszyfrowania jest tekst *PROGRAM MA ZAAWANSOWANE MOŻLIWOŚCI I POZWALA NA TWORZENIE ZASZYFROWANYCH FOLDERÓW.*



Rys. 14. Potwierdzenie skuteczności analizy częstości [Źródło własne]

Dla potwierdzenia wyniku dokonano ponownego zaszyfrowania tekstu z przesunięciem o osiem. Porównanie szyfrogramów udowodniło skuteczność metody (Rys. 14).

2.2.Autorski program XORCrypt

Program XORCrypt jest autorskim programem szyfrującym. Do szyfrowania stosuje operację logiczną XOR (Rys. 15). Program został stworzony w języku C++ i wykorzystuje 3 biblioteki – *cstdio* zajmuje się obsługą operacji wejścia/wyjścia na strumieniach, *cstring* zajmuje się obsługą łańcuchów a iostream operacjami wejścia/wyjścia [8,9,12].

Funkcja *xorowanie* zawiera trzy argumenty szyfrowany, zaszyfrowany oraz klucz.

```
#include <cstdio>
#include <cstring>
#include<iostream>
using namespace std;
bool xorowanie( const char *klucz, const char * szyfrowany, const char *zaszyfrowany)
{
    int dlugosc = strlen(klucz);
    int bit, xorowany;
    FILE *do_szyfrowania = fopen(szyfrowany, "rb");
    FILE *wynik = fopen(zaszyfrowany, "wb");
    if(do_szyfrowania == NULL)
        return false;
    else
        for (int nr = 0; bit != EOF; nr++)
        {
            if (nr >= dlugosc)
                nr=0;
            bit = fgetc(do_szyfrowania);
            xorowany = bit^klucz[nr];
            fputc(xorowany, wynik);
        }

        return true;
}
int main (int dane, char *nrd[])
{
    if (dane>=3)
    {
        if (xorowanie(nrd[1], nrd[2], nrd[3]))
            cout<<"Poprawnie zaszyfrowano plik"<<endl;
        else
            cout<<("Blad w szyfrowaniu")<<endl;
    }
    else
    {
        cout<<("Wpisz dane w nastepujacej kolejnosci element do zaszyfrowania wynik szyfrowania klucz")<<endl;
    }
    return 0;
}
```

Rys. 15. Listing autorskiego programu XORCrypt [Źródło własne]

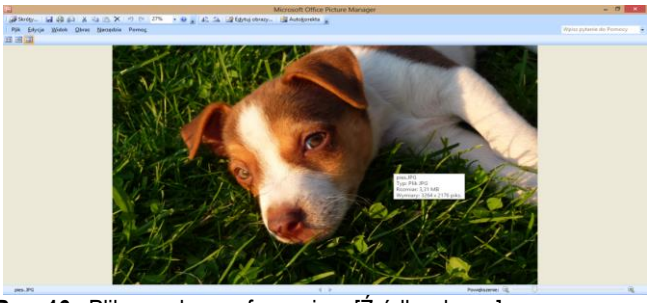
Szyfrowany – jest to plik, który będzie zaszyfrowany, zaszyfrowany – tu należy wpisać nazwę pliku, który będzie wynikiem szyfrowania, klucz- jest to klucz którym użytkownik chce się posłużyć w operacji szyfrowania. Funkcja korzysta z czterech zmiennych typu int: bit, xorowany, długość, której wartość to długość wpisanego klucza oraz nr, która ma na początku przypisaną wartość 0.

Program otwiera plik *szyfrowany*, w postaci binarnej, oraz tworzy nowy plik *zaszyfrowany*, który będzie wynikiem szyfrowania. Następnie inicjuje działanie pętli for. Pętla zawiera instrukcję warunkową, która sprawdza czy zmienna nr nie przekroczyła długości klucza, jeśli jej wartość jest większa lub równa wielkości klucza program przypisuje jej wartość 0 – pozwala to na wykonywanie operacji xor z kluczem dowolnej długości. Pętla na początku wczytuje znaki z otwartego pliku do zaszyfrowania, następnie wykonuje operacje xor na wczytanym znaku oraz kolejnym znaku klucza. Im większa długość klucza, tym lepsza skuteczność programu. Na końcu pętla zapisuje powstały w ten sposób znak do pliku będącego wynikiem operacji szyfrowania.

Funkcja main sprawdza na początku, czy użytkownik podał trzy argumenty, jeśli nie to funkcja wypisuje komunikat o poprawnej kolejności podawania danych, jeśli tak to instrukcja warunkowa sprawdza poprawność wykonania programu. Jeśli program został wykonany poprawnie wypisuje komunikat o udanym szyfrowaniu, jeśli nie, wypisuje informacje o błędzie.

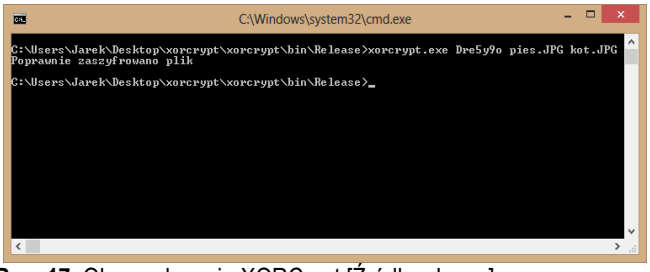
Szyfrowanie typu xor jest symetryczne, oznacza to, że do szyfrowania oraz odszyfrowania wykorzystuje się taki sam klucz. Program można więc wykorzystać do odszyfrowania. Aby uruchomić program należy wpisać w oknie polecenia nazwę programu wraz z rozszerzeniem, następnie jako argumenty wpisać klucz ,nazwę pliku do zaszyfrowania, po niej nazwę pliku wynikowego. Program nie nadpisuje pliku, który ma zaszyfrować, lecz tworzy jego zaszyfrowaną kopię.

W celu przedstawienia działania programu przeprowadzone zostało szyfrowanie obrazu pies.JPG (Rys. 16).



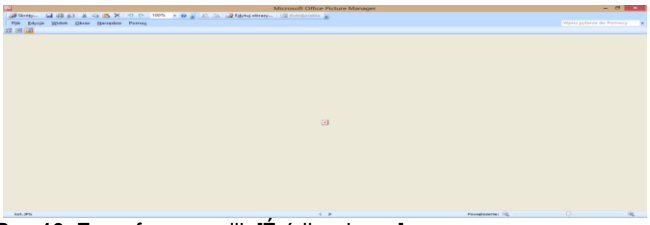
Rys. 16. Plik przed zaszyfrowaniem [Źródło własne]

Pierwszym krokiem było otwarcie okna polecenia w folderze, w którym znajdował się program, następnie wpisanie polecenia xorcrypt.exe Dre5y9o pies.JPG kot.JPG (Rys. 17).



Rys. 17. Okno polecenia XORCrypt [Źródło własne]

Zastosowany klucz spełniał podstawowe wymogi bezpieczeństwa. Po udanym szyfrowaniu program utworzył plik o nazwie kot.JPG, którego nie dało się odczytać (Rys. 18).



Rys.18. Zaszyfrowany plik [Źródło własne]

W celu zapewnienia dobrego bezpieczeństwa należy stosować długie klucze, nie będące nazwami własnymi ani wyrazami, które można odnaleźć w słowniku. Ważne też by nie były to wyrażenia, które mogą kojarzyć się z danym użytkownikiem. Klucz idealny powinien być bardzo długi i być ciągiem przypadkowych znaków – kombinacją liter różnej wielkości i cyfr.

Kluczy oczywiście nie należy zapisywać w postaci cyfrowej, ani udostępniać przypadkowym osobom.

Wnioski

Współcześnie większość czynności realizowanych z użyciem urządzeń elektronicznych zabezpieczana jest szyfrowaniem. Odpowiada ono między innymi za bezpieczeństwo wszystkich transakcji internetowych realizowanych na całym świecie. Każdego dnia podejmowane są próby łamania algorytmów szyfrujących w celu zdobycia pieniędzy lub tajnych danych. Jednocześnie trwają nieustanne prace nad stworzeniem coraz lepszych szyfrów, które będą w stanie oprzeć się wszelkim nowym metodom kryptoanalizy.

Niestety często najsłabszym elementem procesu szyfrowania jest człowiek, który często używa oczywistych haseł, przechowuje

hasła i klucze bez odpowiedniego zabezpieczenia, czy nawet sam tworzy luki w programie. To właśnie błędy i niedopatrzenia programistów są najczęstszą przyczyną niedoskonałości algorytmów szyfrowania.

W artykule przedstawiono opis działania zarówno dawnych jak i współczesnych wybranych metod szyfrowania stosowanych w steganografii oraz kryptografii. Opisano popularne, sprawdzone sposoby i algorytmy używane do łamania szyfrów. Przeprowadzono porównanie działania programów wybranych programów, które wykazało wady i zalety stosowania każdego z nich.

Celem stworzonego stanowiska laboratoryjnego jest zapoznanie studentów z podstawowymi zagadnieniami dotyczącymi budowy i łamania szyfrów, oraz zobrazowanie różnic między szyframi stosowanymi historycznie i współcześnie.

Przygotowane zostały dwa programy w języku C++. Stanowisko laboratoryjne obrazuje potrzeby stosowania coraz bardziej zaawansowanych szyfrów w celu zachowania poufności przetwarzanych informacji. Pokazuje słabości popularnych dawniej szyfrów przestawieniowych, daje możliwość sprawdzenia często stosowanej i bardzo skutecznej metody łamania Szyfru Cezara. Ponadto stanowisko pozwala na analizę działania bardziej zaawansowanego sposobu szyfrowania. Uświadamia również potrzebę stosowania bezpiecznych haseł i kluczy.

Bibliografia

1. Aumasson J.P., Nowoczesna kryptografia, Wydawnictwo Naukowe PWN, Warszawa 2018.
2. Bauer F.L., Sekrety kryptografii, Helion, Gliwice 2002.
3. Daemen J., Rijmen V., The Design of Rjindael AES – The Advanced Encryption Standard, Springer-Verlag, Berlin 2002.
4. Denning D.E., Kryptografia i ochrona danych, Wydawnictwa Naukowo-Techniczne, Warszawa 1992.
5. Ferguson N. Schneier B., Kryptografia w praktyce, Helion, Gliwice 2004.
6. Karbowski M., Podstawy kryptografii. Wydanie III, Helion, Gliwice 2013.
7. Kościelny C., Kurkowski M., Srebrny M., Kryptografia. Teoretyczne podstawy i praktyczne zastosowania, Wydawnictwo Polsko-Japońskiej Wyższej Szkoły Technik Komputerowych, Warszawa 2009.
8. Kutyłowski M, Strothmann W.B., Kryptografia – teoria i praktyka zabezpieczania systemów komputerowych, Oficyna Wydawnicza Read Me, Warszawa 1999.
9. Menezes A. J., van Oorschot P.C., Vanstone S. A., Kryptografia stosowana, Wydawnictwa Naukowo-Techniczne, Warszawa 2005.
10. Morosov V., Steganografia cyfrowa. Sztuka ukrywania informacji, Wydawnictwo Uniwersytetu Łódzkiego, Łódź 2013.
11. Singh S., Księga szyfrów, Albatros, Warszawa 2001.
12. Stallings W., Kryptografia i bezpieczeństwo sieci komputerowych. Matematyka szyfrów i techniki kryptologii, Helion, Gliwice 2011.
13. Stamp M., Low R.M., Applied Cryptanalysis: Breaking Ciphers In the Real World, John Wiley & Sons, Hoboken, New Jersey 2007.
14. Welchenbach M., Kryptografia w C i C++, Wydawnictwo Mikom, Warszawa 2002.
15. Wobst R., Kryptologia — budowa i łamanie zabezpieczeń, Wydawnictwo RM, Warszawa 2002.

Experimental use of selected steganographic and cryptographic algorithms - laboratory workstation

The main task discussed in this publication was to prepare a laboratory stand to investigate the confidentiality of information processed. The first part of the article presents a comparison of the possibilities of two open license programs - encryption tools that implement encryption with the use of AES block cipher. The second part describes the operation of two programs created in C++ for the needs of the laboratory workstation. The first one performs classic, changeable encryption, the second one uses XOR operation for encryption.

Keywords: information, processing, encryption.

Autorzy:

dr inż. **Małgorzata Górska** – Uniwersytet Technologiczno-Humanistyczny im. K. Pułaskiego w Radomiu, Wydział Transportu i Elektrotechniki, Instytut Automatyki i Telematyki

inż. **Jarosław Molendowski** – Uniwersytet Technologiczno-Humanistyczny im. K. Pułaskiego w Radomiu, Wydział Transportu i Elektrotechniki