

Uniwersytet Śląski
Wydział Nauk Ścisłych i Technicznych
Kierunek: Informatyka
Specjalność: Inżynieria Systemów Informatycznych

Armand Pajor
Numer albumu: 320051

Zabezpieczanie i autoryzacja dostępu —
implementacja oprogramowania
kluczów sprzętowych

Praca Dyplomowa
Inżynierska

praca dyplomowa
napisana pod kierunkiem
dr Magdaleny Tkacz

Sosnowiec 2021

Słowa kluczowe: bezpieczeństwo informacji, klucze sprzętowe, kryptologia, Windows, uwierzytelnienie, autoryzacja.

Oświadczenie autora pracy

Ja, niżej podpisany:

imię (imiona) i nazwisko: Armand Pajor
autor pracy dyplomowej pt. "Zabezpieczanie i autoryzacja dostępu —
implementacja oprogramowania kluczy sprzętowych"

Numer albumu: 320051

Student Wydziału Nauk Ścisłych i Technicznych Uniwersytetu Śląskiego w Katowicach
kierunku studiów: Informatyka
specjalności: Inżynieria Systemów Informatycznych

Oświadczam, że ww. praca dyplomowa:

- została przygotowana przeze mnie samodzielnie¹,
- nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 r.
o prawie autorskim i prawach pokrewnych (tekst jednolity Dz. U. z 2006 r. Nr 90,
poz. 631, z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- nie zawiera danych i informacji, które uzyskałem w sposób niedozwolony,
- nie była podstawą nadania dyplomu uczelni wyższej lub tytułu zawodowego ani
mnie, ani innej osobie.

Oświadczam również, że treść pracy dyplomowej zamieszczonej przeze mnie
w Archiwum Prac Dyplomowych jest identyczna z treścią zawartą w wydrukowanej
wersji pracy.

**Jestem świadomy odpowiedzialności karnej za złożenie fałszywego
oświadczenia.**

Data

Podpis autora pracy

¹uwzględniając merytoryczny wkład promotora (w ramach prowadzonego seminariu dyplomowego)

Spis treści

Spis treści	1
1 Wstęp	2
1.1 Cel i główne założenia projektu	2
1.2 Zakres pracy	3
1.3 Konwencja edytorska	3
1.4 Zawartość CD	3
2 Bezpieczeństwo i cyberbezpieczeństwo informacji	4
2.1 Bezpieczeństwo informacji — Triada CIA	4
2.1.1 Metody uwierzytelniania i autoryzacji	7
2.2 Kryptologia	13
2.2.1 Historia kryptografii i kryptoanalizy	14
2.3 Cyberbezpieczeństwo	17
2.3.1 Kryptografia asymetryczna	17
2.3.2 Symetryczne algorytmy blokowe	23
2.4 Przegląd istniejących rozwiązań	26
2.4.1 Aplikacja KeyLock	27
2.4.2 Aplikacja Predator	28
2.4.3 Aplikacja Rohos Logon Key	29
2.4.4 Podsumowanie przeglądu istniejących rozwiązań	30
3 Projekt i implementacja	31
3.1 Prototypowanie i wybór technologii	31
3.2 Założenia projektowe	33
3.3 Wymagania funkcjonalne i niefunkcjonalne projektu	34
3.4 Dokumentacja deweloperska	36
3.5 Dokumentacja techniczna i użytkownika	45
4 Podsumowanie	57
Literatura	59

Wstęp

Najcenniejszym zasobem naszych urządzeń elektronicznych są nasze dane. Ostatnimi czasy zbyt często można wyczytać w nagłówkach wiadomości, że nastąpił wyciek danych, które nie zostały odpowiednio zabezpieczone. Przyczyną przede wszystkim jest brak nawyków do ich ochrony. Wiele ludzi nie przejmuje się ich utratą, do momentu, gdy doświadczają tego na własnej skórze. Od bezpowrotnej utraty zdjęć najbliższych z domowego komputera, przez kradzież tożsamości, do milionowych strat korporacji z winy jednego pracownika. Takie mogą być konsekwencje zaniedbywania bezpieczeństwa informacji.

1.1 Cel i główne założenia projektu

Celem projektu jest implementacja darmowego oprogramowania klucza sprzętowego, które umożliwiłoby wykorzystanie urządzeń pamięci masowej USB w celu zabezpieczenia plików użytkownika. Istniejące produkty, omawiane w sekcji 2.4, wykorzystują urządzenia pamięci masowej w podobnym celu, lecz oferują jedynie zewnętrzne zabezpieczenie komputera. W przypadku ich użycia, pliki zapisane na dysku nie są chronione, jeśli uzyska się dostęp bezpośrednio do nich. Dzięki szyfrowaniu danych przez aplikację autora pracy, pozostaną one zabezpieczone nawet w razie fizycznej kradzieży dysku. Tworzenie klucza sprzętowego będzie polegało na zamieszczeniu specjalnego pliku w urządzeniu pamięci masowej. W ten sposób aplikacja autora pracy będzie rozpoznawała zaufane urządzenie pamięci masowej, które nazywane będzie kluczem sprzętowym. Użytkownik będzie mógł wybrać, które pliki zabezpieczyć. Dostęp do zabezpieczonych plików będzie możliwy jedynie po wcześniejszym umieszczeniu klucza sprzętowego w porcie USB komputera z aktywną aplikacją autora. Ponadto program musi posiadać funkcję odzyskania dostępu do danych, w razie utraty lub uszkodzenia klucza sprzętowego. Proces ten będzie wymagał weryfikacji, czy dany użytkownik jest tym, za kogo się podaje.

Podczas tworzenia projektu (opisanego w rozdziale 3), autor zgłębi wiedzę z zakresu bezpieczeństwa informacji, metod autoryzacji i uwierzytelnienia, kryptologii, cyberbezpieczeństwa oraz programowania w języku **C#** z wykorzystaniem przetwarzania współbieżnego.

1.2 Zakres pracy

W rozdziale pt. "Bezpieczeństwo i cyberbezpieczeństwo informacji" opisano założenia **Triady CIA**, jak i metody uwierzytelniania oraz autoryzacji. Kolejno opisywana jest historia rozwoju kryptologii. Następnym poruszonym zagadnieniem jest cyberbezpieczeństwo, w ramach którego, opisywane są algorytmy kryptografii asymetrycznej oraz symetrycznej. Ostatnim tematem rozdziału jest przedstawienie i omówienie istniejących rozwiązań, które w podobny sposób wykorzystują urządzenia pamięci masowej w celu ochrony komputera użytkownika.

W rozdziale "Projekt i implementacja" omówiony zostaje proces tworzenia aplikacji **PAAK**: jej założenia projektowe, wymagania funkcjonalne i niefunkcjonalne oraz znajduje się dokumentacja deweloperska, techniczna i użytkownika.

1.3 Konwencja edytorska

Na potrzeby pracy przyjęto następującą konwencję edytorską:

- Terminy anglojęzyczne: *kursywa*.
- Nazwy własne: **czcionka pogrubiona**.
- Definicje: **czcionka pogrubiona z kursywą**.
- Terminy wprowadzone: *czcionka pochylona*.
- Kod programu:

```
1 string przykładowyTekst = "Kod programu";
2 Console.WriteLine(przykładowyTekst);
```

- Elementy interfejsu: Czcionka bezszeryfowa.

1.4 Zawartość CD

Na zewnętrznym nośniku danych w formie płyty CD znajduje się:

- oprogramowanie wykonane w ramach niniejszej pracy o nazwie **PAAK**,
- praca inżynierska w wersji identycznej z przedstawioną,
- nagranie wideo przedstawiające działanie aplikacji.

Pracę inżynierską w formie elektronicznej, nagranie wideo ukazujące działanie aplikacji oraz aplikację **PAAK** można pobrać z repozytorium **GitHub** pod adresem <https://github.com/Armand98/paak>. Nagranie znajduje się w katalogu "screencast".

Bezpieczeństwo i cyberbezpieczeństwo informacji

Bezpieczeństwo oraz cyberbezpieczeństwo informacji są niepodważalnie bardzo istotnymi aspektami dzisiejszego świata. Wiele ludzi przenosi swoje życie do Internetu, w którym dzielą się oni swoimi przeżyciami. Internet wykorzystywany jest również do wykonywania bardzo ważnych operacji, które wymagają zachowania poufności podczas przesyłania danych, jak i podczas ich późniejszego przechowywania. Bez zadbania o bezpieczeństwo informacji, wiele osób byłoby narażonych na straty.

2.1 Bezpieczeństwo informacji — Triada CIA

Bezpieczeństwo informacji polega na ochronie danych poprzez ograniczanie ryzyka związanego z nimi. Aby takie ograniczenie ryzyka wprowadzić, należy omówić trzy funkcje bezpieczeństwa, które są uznawane za fundamentalne:

- poufność,
- integralność (spójność),
- dostępność.

Funkcje te są ze sobą ścisłe powiązane i przedstawiane jako **Triada CIA**.

Triada CIA [Walkowski(2020)]

Triada CIA ma ułatwić wprowadzanie polityk bezpieczeństwa dla organizacji. Na jej podstawie można zauważać zależności dotyczące użycia funkcji bezpieczeństwa opisanych poniżej.



Rysunek 2.1: Triada CIA. Źródło: [Walkowski(2020)]

Poufność to właściwość danych, wskazująca obszar, w którym te dane nie powinny być dostępne lub ujawniane nieuprawnionym osobom, procesem lub innym podmiotom [PKN(2002)].

Zapewnienie poufności informacji ma zagwarantować, iż informacje pozostaną prywatne w obrębie danej grupy odbiorców i nikt spoza tej grupy nie uzyska do nich dostępu. Efekt ten może być osiągnięty przy użyciu szyfrowania i adekwatnego systemu kontroli dostępu, który jest mechanizmem określającym uprawnienia dostępu.

Integralność to właściwość polegająca na tym, że dane nie zostały wcześniej zmienione lub zniszczone w nieautoryzowany sposób [PKN(2002)].

Integralność danych ma zagwarantować spójność informacji w nich zawartych. Tym samym, zapewnia ono gwarancję, iż dane nie ulegną modyfikacji lub destrukcji w trakcie ich przesyłu przez nieautoryzowane osoby. Przykładem mechanizmów, które są wykorzystywane do zapewnienia integralności danych są sumy kontrolne, systemy kontroli dostępu oraz kopie zapasowe monitorowanych danych.

Dostępność to właściwość danych lub zasobów polegająca na tym, że mogą one być dostępne i wykorzystywane na żądanie uprawnionej jednostki [PKN(2002)].

Zapewnienie dostępności do danych ma zagwarantować, że w założonym czasie, osoba posiadająca odpowiednie uprawnienia, może uzyskać niezawodny dostęp do żądanych zasobów informacji.

Zależność, o której była mowa na początku sekcji, polega na ograniczeniach wynikających z zastosowania wyżej wymienionych funkcji bezpieczeństwa. Wierzchołki trójkąta przedstawiającego triadę **CIA** [rys. 2.1] oznaczają skrajnie wysoki priorytet dwóch z trzech funkcji bezpieczeństwa.

- Zapewnienie poufności i integralności na wysokim poziomie spowoduje, iż dostępność do zasobów będzie ograniczona.
- Poufność zostanie zaniedbana, jeśli zapewni się wysoki stopień integralności oraz dostępności.
- Integralność zostanie naruszona, jeśli za priorytetowe funkcje bezpieczeństwa potraktuje się dostępność oraz poufność.

W związku z wyżej wskazaną zależnością, należy zachowywać równowagę między wierzchołkami triady **CIA**. W przypadku, gdy system informatyczny wymaga szczególnej ochrony, świadomie rezygnuje się z jednej właściwości na rzecz dwóch pozostałych o podwyższonych standardach, które mają największe znaczenie w danym przypadku. Najtrudniejszym wyzwaniem inżynierów bezpieczeństwa jest zapewnienie by system był jednocześnie wygodny jak i bezpieczny.

Uwierzytelnianie, autoryzacja i kontrola dostępu [Singh(2020)]

Kolejnym bardzo ważnym aspektem bezpieczeństwa informacji jest uwierzytelnianie, autoryzacja i kontrola dostępu. W nomenklaturze anglojęzycznej nazywane jest ono **AAA** (*ang. Authentication, Authorization, Accounting*).

- Uwierzytelnianie jest procesem weryfikującym czy dana osoba jest tą, za którą się podaje. Składa się na to kilka etapów:
 - uwierzytelnienie przez wiedzę — sprawdzenie co dana osoba wie (np. login, hasło, pin),
 - uwierzytelnienie przez posiadanie — sprawdzenie co dana osoba posiada (np. klucz, zbliżeniowa karta dostępu),
 - uwierzytelnienie przez charakterystykę — sprawdzenie czy dana osoba jest fizycznie tą, za którą się podaje (np. biometria).
- Autoryzacja jest procesem przyznającym uprawnienia odpowiedniego poziomu dla uwierzytelnionych użytkowników systemu.
- Kontrola dostępu jest mechanizmem, który monitoruje aktywność użytkownika systemu.

Podział ten został zaprojektowany, by umożliwić dynamiczną konfigurację procesu uwierzytelnienia i autoryzacji. Zastosowanie wymienionych elementów systemu dostępu ma zagwarantować, iż tylko osoby posiadające odpowiednie uprawnienia zdołają uzyskać dostęp do wybranych zasobów informacji.

Można również zauważać pewnego rodzaju korelację **Triady CIA** z **AAA**, polegającą na założeniu, iż **Triada CIA** jest modelem abstrakcyjnym, gdy **AAA** jest rzeczywistą i praktyczną implementacją mechanizmów bezpieczeństwa informacji.

2.1.1 Metody uwierzytelniania i autoryzacji

System operacyjny, serwis internetowy czy archiwum plików może posiadać wymóg użycia hasła. Nie wymagają dodatkowego sprzętu, a wprowadzane są za pomocą tego samego interfejsu, który służy do komunikacji systemu lub programu z użytkownikiem. Niestety czas pokazał, że natura ludzka jest silniejsza od racjonalnego podejścia. Zaczęto wykorzystywać hasła krótkie, o małej złożoności i łatwe do odgadnięcia przez nieautoryzowane osoby. Jest to spowodowane ilością haseł, które musimy pamiętać w ramach codziennych czynności związanych z komputerem: hasło do systemu operacyjnego, skrzynki pocztowej, sklepów internetowych, mediów społecznościowych i wielu więcej. Narodowy Instytut Standaryzacji i Technologii (ang. *NIST*) wyznaczył wymagań, które hasła powinny spełniać [NIST(2017)]:

1. Hasło powinno być o minimalnej długości 8 znaków, jeśli zostało wymyślone przez użytkownika. Dopuszcza się użycie hasła o długości 6 znaków, jeśli zostało wygenerowane losowo.
2. Hasło musi być unikalne i nie może zostać wykorzystane w wielu systemach lub serwisach.
3. Hasło musi być przechowywane w sposób uniemożliwiający jego odtworzenie.

Oczywiście wręcz niemożliwym byłoby zapewnienie, iż każdy użytkownik wykorzysta jedno hasło wyłącznie w jednym miejscu. Dlatego wymusza się na użytkownikach spełnienie podpunktu pierwszego, aby korzystano z silniejszych haseł. Wiele serwisów wymaga hasła o długości co najmniej 6 znaków oraz skorzystania z co najmniej jednego znaku ze zbiorów:

- liter małych alfabetu,
- liter wielkich alfabetu,
- cyfr,
- znaków specjalnych.

Tak niewielka zmiana hasła sprawia, iż jego odgadnięcie staje się znacznie bardziej trudne do wykonania. Dzieje się tak za sprawą rosnącej entropii dla hasła.

Entropia hasła [Buchanan(2018)]

Entropia w teorii informacji to średnia ilość informacji, przypadająca na znak symbolizujący zajście zdarzenia z pewnego zbioru. Zdarzenia w tym zbiorze mają przypisane prawdopodobieństwa wystąpienia.

$$H(x) = \sum_{i=1}^n p(i) \log_r \frac{1}{p(i)} = - \sum_{i=1}^n p(i) \log_r p(i),$$

gdzie $p(i)$ - prawdopodobieństwo zajścia zdarzenia i .

Wzór ten określa również entropię dla danego tekstu, jeżeli:

- $p(i)$ - prawdopodobieństwo wystąpienia danego znaku (obliczone na podstawie histogramu),
- n - liczba wszystkich niepowtarzalnych znaków,
- i - zmienna iterująca,
- r - podstawa logarytmu równa 2, ponieważ mowa o bitach, jako o jednostce entropii.

Na jego podstawie można obliczyć entropię hasła, która będzie najprostszą miarą jego siły ze względu na ścisły związek z ilością możliwych kombinacji potrzebnych do odgadnięcia hasła. Obliczenie maksymalnej liczby kombinacji potrzebnych do złamania hasła można określić za pomocą wzoru

$$2^k,$$

gdzie k to liczba bitów entropii hasła. Im więcej bitów entropii w haśle, tym trudniej je złamać. Siła hasła powinna być dobrana na podstawie wcześniejszej analizy ryzyka. Uznaje się, że bezpieczne hasło posiada od 29 do 97 bitów entropii. W sytuacji, gdy chcemy obliczyć entropię jednego znaku występującego w haśle, wzór ulega uproszczeniu, ze względu na równe prawdopodobieństwo wystąpienia każdego znaku. W takiej sytuacji, wzór określający entropię jednego znaku w określonym zbiorze, przybierze następującą postać:

$$H(x) = \log_2(n),$$

gdzie n to liczba wszystkich niepowtarzalnych znaków w danym zbiorze. Dla przykładu obliczymy entropię jednego znaku w zbiorze małych liter alfabetu łacińskiego, których w alfabetie międzynarodowym jest 26. Zatem:

$$H(26) = \log_2(26) \approx 4.7$$

Znając liczbę bitów entropii na znak w podanym zbiorze możemy teraz obliczyć, ile będzie wynosiła entropia całego hasła o podanej długości. W tym celu przemnożymy liczbę bitów entropii przez długość hasła. Przyjmijmy, iż hasło składa się z 8 małych liter alfabetu. Otrzymujemy następujące równanie:

$$8 \cdot 4.7 = 37.6$$

Z obliczeń wynika, iż hasło o długości 8 znaków, które składa się wyłącznie z małych liter alfabetu łacińskiego posiada 37.6 bitów entropii. Można jednak takie hasło znacząco wzmacnić przez jego drobną zmianę polegającą na wykorzystaniu również wielkich liter alfabetu, cyfr i znaków specjalnych. Zbiór wszystkich możliwych znaków wzrasta wtedy do 94, na którą to liczbę składają się elementy 4 zbiorów:

- zbiór 26 małych liter alfabetu łacińskiego,
- zbiór 26 wielkich liter alfabetu łacińskiego,
- zbiór 10 cyfr od 0 do 9,
- zbiór 32 znaków specjalnych.

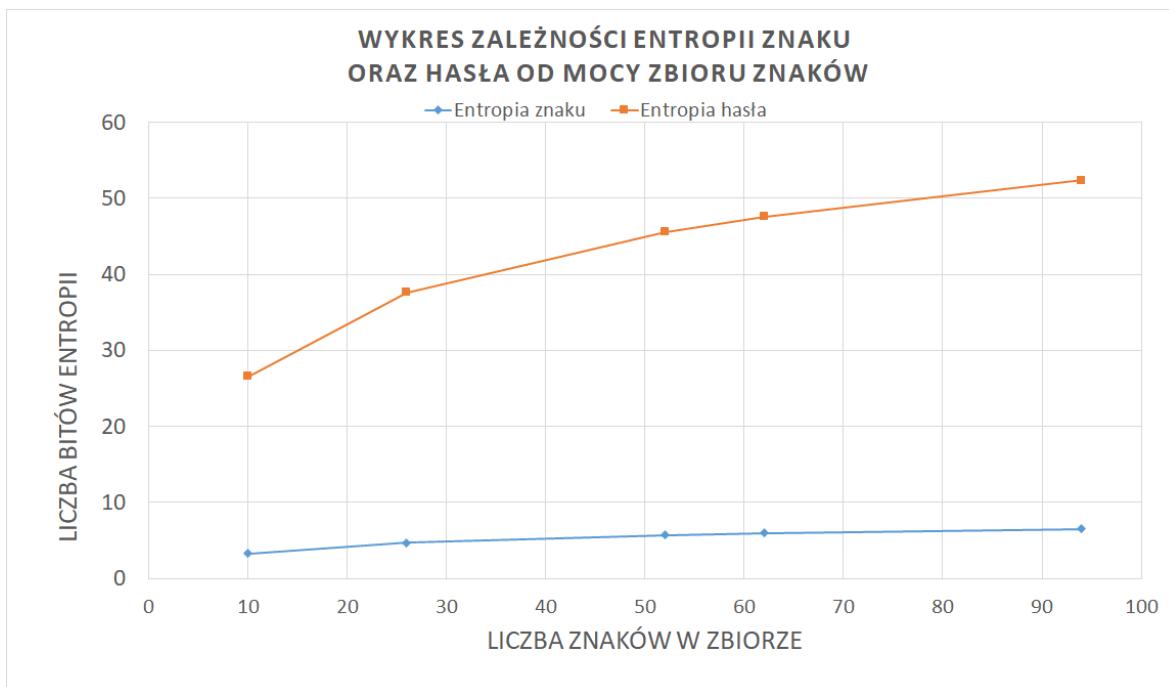
Entropia dla jednego znaku w zbiorze o 94 elementach wynosić będzie:

$$H(94) = \log_2(94) \approx 6.55$$

W takim przypadku, entropia dla hasła o długości 8 znaków wyniesie:

$$8 \cdot 6.55 = 52.4$$

Liczba bitów entropii wzrosła z 37.6 bitów do aż 52.4 bita. Poniżej porównano czas potrzebny na złamanie tych dwóch haseł o wyliczonej entropii. Dla uproszczenia obliczeń, zaokrąglono liczbę bitów entropii w dół, przyjmując dla przypadku pierwszego 37 bitów, a dla przypadku drugiego 52 bity entropii.



Rysunek 2.2: Wykres zależności entropii znaku oraz hasła od mocy zbioru znaków, z którego składa się hasło [Źródło własne].

Atak siłowy

Atak siłowy (*ang. brute-force*) polega na sprawdzeniu każdej możliwej wariacji znaków z danego niepowtarzalnego zbioru znaków w celu odgadnięcia hasła. Metoda ta jest najmniej efektywna, czasochłonna, ale skuteczna przy dużej ilości czasu na jej przeprowadzenie lub podczas posiadania gwarancji, iż odgadywane hasło nie jest zbyt złożone.

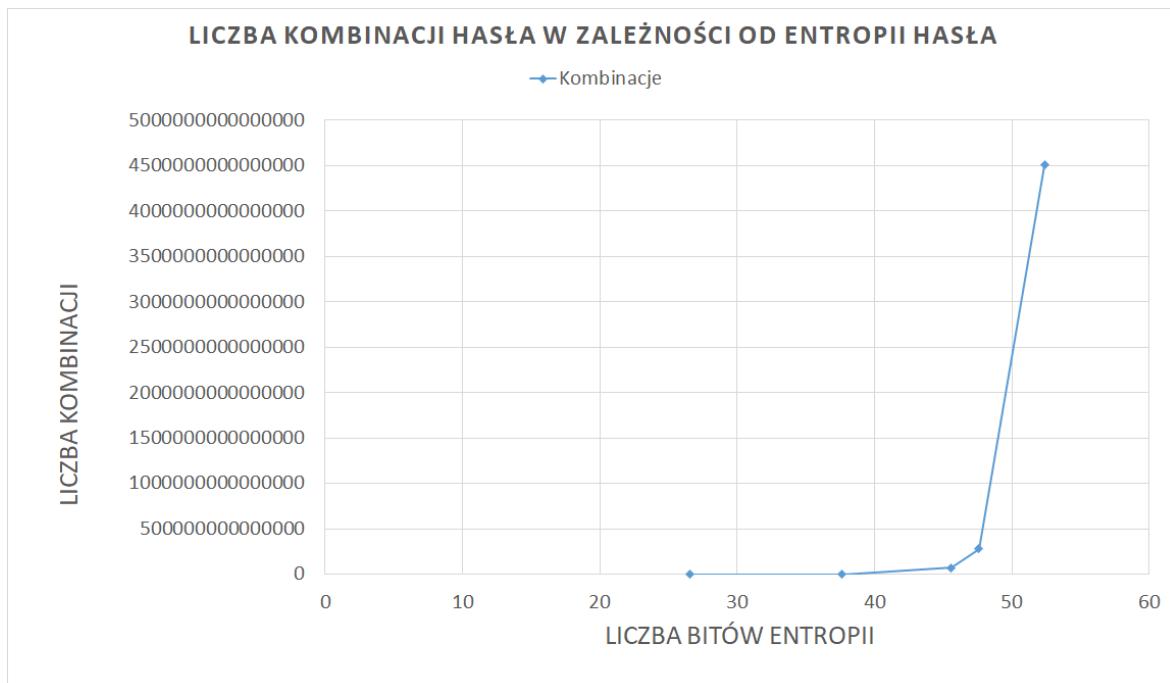
Dla przykładu przyjmijmy, że komputer będzie sprawdzał 1 mln kombinacji na sekundę.

1. Liczba kombinacji dla hasła o entropii 37 bitów to:

$$2^{37} = 137\,438\,953\,472$$

2. Natomiast liczba kombinacji dla hasła o entropii 52 bitów to:

$$2^{52} = 4\,503\,599\,627\,370\,496$$



Rysunek 2.3: Wykres zależności kombinacji hasła od liczby bitów entropii [Źródło własne].

Dzieląc liczbę kombinacji przez liczbę operacji na sekundę otrzymamy liczbę sekund potrzebną na złamanie hasła w pesymistycznym przypadku, gdy ostatnia sprawdzana kombinacja okaże się hasłem.

1. Przypadek dla hasła o 8 znakach ze zbioru małych liter alfabetu łacińskiego:

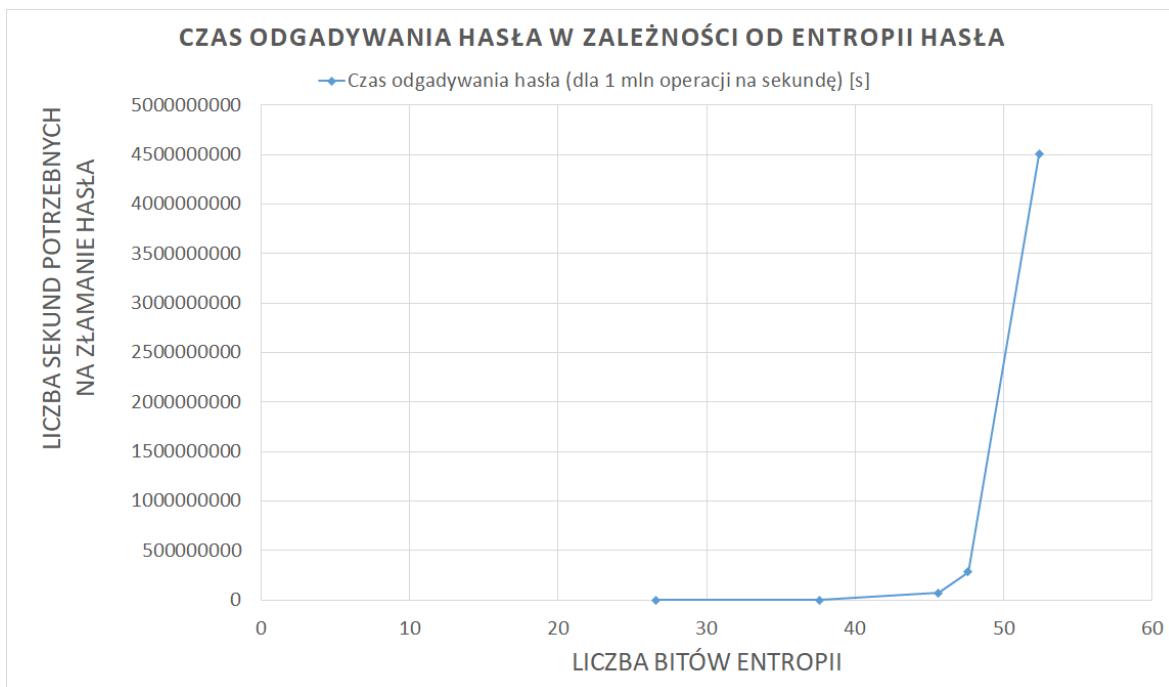
$$\frac{137\,438\,953\,472}{1\,000\,000} \approx 137\,439 \text{ sec}$$

W przypadku hasła składającego się wyłącznie z małych liter alfabetu łacińskiego czas ten wynosi około 137 439 sekund, a jest to w przybliżeniu 1 dzień i 14 godzin.

2. Przypadek dla hasła o 8 znakach ze zbioru 94 znaków:

$$\frac{4\ 503\ 599\ 627\ 370\ 496}{1\ 000\ 000} \approx 4\ 503\ 599\ 627 \text{ sec}$$

W tym przypadku, potencjalny czas potrzebny na złamanie hasła znacząco wzrósł i wynosi prawie 52 125 dni, czyli ponad 142 lata.



Rysunek 2.4: Wykres zależności czasu odgadnięcia hasła od liczby bitów entropii [Źródło własne].

Wykres 2.2 i 2.4 pokrywają się i obrazują, jak liczba bitów entropii jest ściśle powiązana z ilością kombinacji hasła. Ten przykład ukazuje jak bezpieczeństwo hasła o tej samej długości może się znacząco różnić w zależności od użytych znaków.

Klucze sprzętowe

Pomimo, iż hasła są podstawowym sposobem uwierzytelnienia użytkowników, istnieją rozwiązania, które potrafią być dużo bardziej bezpieczne i wygodniejsze w codziennym użytkowaniu.

Klucz sprzętowy to fizyczne urządzenie, które może zostać wykorzystane do uwierzytelnienia dwuskładnikowego lub jako zastąpienie tradycyjnego hasła w procesie logowania. Istnieją klucze prefabrykowane, które stworzone zostały tylko w konkretnym celu, lecz istnieje możliwość wykorzystania dowolnego urządzenia pamięci masowej z odpowiednim oprogramowaniem. Użycie kluczy sprzętowych jest zaawansowanym mechanizmem bezpieczeństwa, który ma zapewnić wyższy poziom ochrony danych i jednocześnie

pozostać prosty w użyciu. Tradycyjne systemy logowania z użyciem hasła wymagają zapamiętywania używanych haseł. Klucz sprzętowy jest pozbawiony wymogu zapamiętywania haseł, gdyż wystarczy się nim posłużyć w procesie logowania.



Rysunek 2.5: Klucz sprzętowy **YubiKey** firmy **Yubico** [Yubico(2021a)].

Wygodnym i bezpiecznym rozwiązaniem zabezpieczenia danych jest **YubiKey**. Ten sprzętowy klucz bezpieczeństwa pozwala zabezpieczyć konto m.in. w systemie operacyjnym **Windows**, na platformie **Gmail**, **Facebook**, **Instagram**, **Twitter** i innych portalach, które oferują tego rodzaju zabezpieczenie [Yubico(2021b)]. Wystarczy umieścić go w porcie USB i przeprowadzić prostą konfigurację. Od jej ukończenia możemy korzystać z **YubiKey** do logowania się w różnych serwisach. Takie rozwiązanie jest bardzo wygodne dla użytkownika i posiada wiele zalet:

- Zabezpieczenie sprzętowe jest niezwykle trudne do obejścia.
- Klucze są małe i poręczne, a ich użycie jest również bardzo proste.
- Pozwala zmniejszyć liczbę pamiętanych haseł.
- Nie zachęca do korzystania z haseł prostych, które można łatwo złamać.
- Pozwala wyeliminować prawdopodobieństwo użycia tego samego hasła kilkukrotnie.

W czasie, gdy zapora sieciowa zabezpiecza komputer przed intruzami z sieci zewnętrznej, klasyczny komputer, poza loginem do systemu operacyjnego, nie posiada żadnej ochrony. Rozwiązaniem może okazać się odpowiednie wykorzystanie klucza sprzętowego, które zapewni kolejną warstwę zabezpieczeń. Istnieją rozwiązania (opisywane w rozdziale 2.4), które programowo odcinają połączenie komputera ze wszystkimi urządzeniami wejścia oraz wyjścia, dzięki którym interakcja nieautoryzowanego użytkownika z komputerem będzie niemożliwa. Dopiero po podłączeniu do niego odpowiedniego klucza sprzętowego, możliwe staje się dalsze korzystanie z niego. Nawet w sytuacji fizycznej

kradzieży komputera, zabezpieczenie swoich plików z użyciem klucza sprzętowego w połączeniu z odpowiednim oprogramowaniem szyfrującym, uniemożliwi odczytanie skradzionych plików.

2.2 Kryptologia

Kontrola dostępu, procesy uwierzytelniania oraz autoryzacji 2.1 stanowią zewnętrzną barierę przed nieautoryzowanym dostępem do informacji. Ich zadaniem jest zapewnienie, iż wybrane informacje będą poza zasięgiem nieautoryzowanych osób. Niestety istnieją sytuacje, które sprzyjają przechwyceniu informacji, np. podczas ich przesyłu. Aby nie ujawnić treści przesyłanych informacji, powinny być one niemożliwe do odczytu przez jednostki nieposiadające odpowiednich przywilejów. W celu zabezpieczania danych, by nie zostały one zinterpretowane przez osoby niepowołane, wykorzystuje się kryptografię.

Kryptologia to nauka o bezpiecznym przekazywaniu i przechowywaniu informacji, zazwyczaj w sekretnej postaci [Simmons(1999)].

Kryptologia, należąca do gałęzi matematyki oraz informatyki, składa się z dwóch działów:

Kryptologia
=
Kryptografia + Kryptoanaliza

Kryptografia polega na tworzeniu szyfrów oraz szyfrowaniu wiadomości, tzn. ukryciu treści semantycznej przez zmianę formy syntaktycznej wiadomości. Innymi słowy, jest to proces zamiany tekstu jawnego na szyfrogram, którego nie da się odczytać bez uprzedniego odszyfrowania wiadomości z użyciem klucza [Dictionary(2021b)].

Klucz w kryptologii to tajna informacja, która jest niezbędna w procesie szyfrowania i deszyfrowania wiadomości. Można go również określić jako wartość wejściową, która wpływa bezpośrednio na działanie algorytmu szyfrującego i otrzymywany w wyniku jego działania szyfrogram wyjściowy [Staśkiewicz(2021)].

Drugą składową kryptologii jest kryptoanaliza, która polega na łamaniu szyfrów, tzn. odczytywaniu zaszyfrowanej informacji bez znajomości klucza [Dictionary(2021a)].

Ze względu na zamienne stosowanie terminów "metod szyfrowania" oraz "systemów szyfrowania" w różnych źródłach, autor na potrzeby pracy przyjmuje następujące definicje wyżej wymienionych pojęć:

Metoda szyfrowania jest implementacją algorytmu kryptograficznego obejmującego operacje szyfrowania oraz deszyfrowania danych wedle konkretnego schematu.

Metodą szyfrowania można nazwać algorytm kryptograficzny jak np. **RSA** lub **AES**.

System szyfrowania to system nadzędny odpowiadający za prawidłowe funkcjonowanie jednej lub kilku metod szyfrowania.

Systemem szyfrowania można nazwać fragment aplikacji, odpowiedzialny za wykorzystanie jednego lub wielu algorytmów szyfrowania w celu zabezpieczenia plików użytkownika.

2.2.1 Historia kryptografii i kryptoanalizy

Historia kryptografii sięga czasów starożytnego Egiptu, gdzie Egipcjanie wykorzystywali ją w codziennym życiu. W tamtych czasach, forma w jakiej była ona używana, znaczaco odbiega od kryptografii, którą znamy dzisiaj. Służyła ona podmianom hieroglifów w pismach egipskich co miało na celu poprawienie ich lingwistycznego wyglądu, przybranie formy ozdobnej lub zagadkowej. Najstarszy znany przykład takiego szyfrowania wiadomości znaleziono w grobowcu egipskiego nomarcha (wysoko postawionego urzędnika) Khnumhotepa II, który żył w około 1900 roku p.n.e. Za to pierwszym udokumentowanym szyfrowaniem, które miało na celu utajenie informacji, była transkrypcja z Mezopotamii, pochodząca z około 1500 roku p.n.e.. W tamtym czasie pewien pisarz zaszyfrował formułę szkliwa garncarskiego, która była używana do produkcji glinianych tabliczkach. Od tamtej pory coraz częściej wykorzystywano kryptografię do szyfrowania informacji. Na przestrzeni kolejnych 1300 lat rozwój kryptografii był bardzo powolny. W II w. p.n.e. grecki historyk Polibiusz opracował system szyfrowania oparty na tablicy przyporządkowującej każdej literze parę cyfr (por. Tab 2.1).

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I/J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

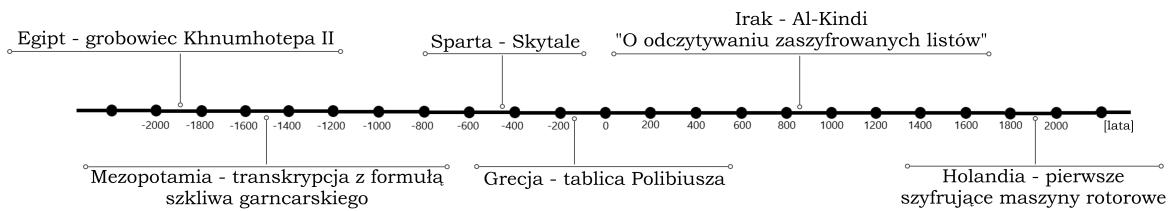
Tabela 2.1: Powyższa tabela przedstawia Tablicę Polibiusza [Harbowski(2014)]

Tablica ta stała się podstawą wielu systemów szyfrowania. Za pomocą przekształcania liter w liczby można było rozwinać szyfrowanie o dalsze przekształcanie powstałych liczb przez użycie obliczeń lub funkcji matematycznych.

Metody szyfrowania znane ze starożytności można podzielić na dwie kategorie - przez przestawianie i podstawianie. W przypadku przestawiania zmieniano szyk liter w zdaniach. W ten sposób powstawał anagram. Przykładem pierwszego znanego urządzenia szyfrującego, które opierało się na metodzie przestawiania, była spartańska **Skytale** z V w. p.n.e. Miała ona kształt pręta o określonej grubości, na który nadawca nawijał skórzany pas lub wąski pasek pergaminu. Wiadomość była pisana wzduż pręta, po czym odwijano pas, na którym widać było tylko pozornie bezsensowną sekwencję liter lub ich fragmentów. Po przekazaniu "listu" do adresata, ten mógł odczytać wiadomość przy użyciu pręta o takiej samej grubości jak pręt nadawcy. Metoda ta była bardzo wygodna, gdyż cała procedura szyfrowania i deszyfrowania składała się z prostych manualnych czynności. Dodatkowo, taki list był bardzo prosty do ukrycia ze względu na swoją formę.

Metoda podstawiania była bardziej popularna. Polegała na podstawieniu za litery tekstu jawnego innych liter bądź symboli. Za przykład może posłużyć tutaj szyfr Cezara, który zyskał miano najsłynniejszego algorytmu szyfrującego czasów starożytnych. Według zapisków, jego twórcą był Juliusz Cezar. Algorytm podmieniał każdą literę na inną, położoną o pewną liczbę miejsc dalej w alfabetie, która to liczba była nazywana kluczem. Adresat musiał znać klucz i po otrzymaniu zaszyfrowanej wiadomości, podmienić каждą literę w szyfrogramie o liczbę miejsc określoną w kluczu, ale tym razem w przeciwnym kierunku.

Szyfry, które przyporządkowują każdej literze alfabetu jawnego dokładnie jedną literę, kombinację cyfr lub symbol to *szyfry monoalfabetyczne*. Istnieją również *szyfry polialfabetyczne*, które są połączeniem wielu szyfrów monoalfabetycznych. Stosuje się w nich wiele tajnych alfabetów do szyfrowania kolejnych znaków, a alfabetów są cyklicznie zmieniane.



Rysunek 2.6: Kryptologia na przestrzeni lat - oś czasu [Źródło własne].

Na przestrzeni lat rozwijała się również inna gałąź kryptologii nazywana kryptoanalizą. Jej kolebką były państwa arabskie. To tam najlepiej opanowano sztukę lingwistyki i statystyki. Przy ich pomocy tworzono metody łamania szyfrów monoalfabetycznych. Najstarszy znany zapis dotyczący kryptoanalizy znajduje się w pracy Al-Kindiego z IX wieku. Był to arabski uczony, filozof, teolog, matematyk i lekarz. W swojej pracy pt. "O odczytywaniu zaszyfrowanych listów", którą odnaleziono w 1987 roku w Archiwum Ottomańskim w Stambule, Al-Kindi zawiera szczegółowe rozważania na temat statystyki i fonetyki oraz składni języka arabskiego, podając przy tym opracowaną przez siebie technikę poznawania tajnego pisma. Tak przedstawiał się jego pomysł:

"Jeden sposób na odczytanie zaszyfrowanej wiadomości, gdy wiemy, w jakim języku została napisana, polega na znalezieniu innego tekstu w tym języku, na tyle długiego, by zajął mniej więcej jedną stronę, i obliczeniu, ile razy występuje w nim każda litera. Literę, która występuje najczęściej, będziemy nazywać "pierwszą", następną pod względem częstości występowania "drugą" i tak dalej, aż wyczerpiemy listę wszystkich liter w próbce jawnego tekstu."

Następnie bierzemy tekst zaszyfrowany i również klasyfikujemy użyte w nim symbole. Znajdujemy najczęściej występujący symbol i zastępujemy go wszędzie "pierwszą" literą z próbki jawnego tekstu. Drugi najczęściej występujący symbol zastępujemy "drugą" literą, następny "trzecią" i tak dalej, aż wreszcie zastąpimy wszystkie symbole w zaszyfrowanej wiadomości, którą chcemy odczytać" [Singh(2001)].

Opisana metoda znana jest jako analiza częstości i w dalszym ciągu stanowi podstawową technikę kryptoanalytyczną.

Szyfrujące Maszyny Rotorowe [Kowalczyk(2020e)]

Na początku XX wieku wynaleziono pierwsze elektro-mechaniczne szyfrujące maszyny rotorowe. Pozwalały na stosowanie bardziej złożonych algorytmów szyfrowania, od tych, które stosowano ręcznie. Oficjalnie uznaje się, iż pierwszymi osobami, które stworzyły pierwsze szyfrujące maszyny rotorowe, byli oficerowie marynarki wojennej Holandii, Theo A. van Hengel i R.P.C. Spengler. Szyfrujące maszyny rotorowe, które przypominały klasyczne maszyny do pisania, były wyposażone w ruchome dyski nazywane rotorami. Szyfry wykorzystywane w takich urządzeniach były szyframi polialfabetycznymi, wykorzystującymi wieloalfabetowe szyfry podstawieniowe.

Metoda działania takich maszyn była z założenia prosta i opierała się na metodzie podstawiania. Prosta maszyna podstawieniowa, zależnie od użytego wewnątrz mechanizmu, ma przypisany pewien znak wyjściowy do danego znaku wejściowego. Taka maszyna wykorzystuje monoalfabetyczny szyfr podstawieniowy. Po dodaniu do niej rotora, będzie on obracał się o pewien kąt po każdym naciśnięciu klawisza. Taka modernizacja pozwala na użycie duże silniejszego szyfru. W celu odszyfrowania treści, odbiorca musi użyć maszyny wyposażonej w taki sam mechanizm wewnętrzny.

2.3 Cyberbezpieczeństwo

Powstanie komputerów znacząco zmieniło dotychczasowe sposoby szyfrowania. Proces szyfrowania i deszyfrowania uległ znaczniemu skróceniu, a dzięki temu można było wdrażać coraz to bardziej skomplikowane i złożone algorytmy. Komputery nie posiadały ograniczeń elektro-mechanicznych maszyn szyfrujących. Pozwalały one na symulację dowolnie złożonej maszyny szyfrującej, której fizyczna konstrukcja byłaby niemożliwa do wykonania. Ostatnia, najważniejsza zmiana, jaka nastąpiła dzięki zastosowaniu komputerów, dotyczyła poziomu szyfrowania. Do tej pory szyfrowanie odbywało się na poziomie liter alfabetu. Komputery, ze względu na swoją konstrukcję, która składała się z elektronicznych przełączników, operowały na liczbach binarnych. Należało przejść z szyfrowania liter i znaków na szyfrowanie ciągów zer i jedynek. Dlatego w latach sześćdziesiątych opracowano kod **ASCII**. Pozwalał on z łatwością powiązać liczbowe kody liter oraz znaków z ich postacią binarną, co umożliwiało ich zapis w komputerze. Gdy dane były w postaci binarnej, można było zastosować metodę szyfrowania, która znacząco nie różniła się od tego procesu z czasów zanim powstały komputery. Nadal podstawową metodą było przedstawianie elementów zapisanej wiadomości, by pozornie nie miała ona sensu - z tą różnicą, że podstawowym elementem, na którym dokonuje się szyfrowania, jest bit, a nie znak.

Kryptologia komputerowa najszybciej rozwijała się w Stanach Zjednoczonych. Powstało tam wiele systemów kryptograficznych, ale ze względu na obowiązujące wtedy amerykańskie prawo, pojawiła się konieczność ustalenia powszechnie obowiązującego standardu szyfrowania. Standardem, w tym przypadku, nazywa się wspólnie ustalone normy określające podstawowe wymagania dotyczące kryptografii [PWN(1999)]. Pierwszym takim standardem był **DES** (ang. *Data Encryption Standard*), obowiązujący od 1976 roku i utrzymany jako standard do roku 2001. **DES** był szyfrem blokowym, co oznacza, że za dane wejściowe przyjmuje on bloki danych o określonej długości, a na wyjściu podaje bloki kryptogramu o tej samej długości. Podstawową jednostką przetwarzania nie są pojedyncze bity czy bajty, a całe bloki danych. Algorytm ten należy do grupy algorytmów symetrycznych blokowych [Harbowski(2014)].

2.3.1 Kryptografia asymetryczna

Największą wadą algorytmów symetrycznych jest klucz. Sprawia on wiele problemów, gdyż należy odpowiednio zadbać o jego poufność. W praktyce stanowi to nie lada wyzwanie. Zaczęto rozważać całkowicie nowe podejście, które miałyby umożliwić

dystrybucję klucza bez obawy o niepowołane odszyfrowanie kryptogramu. Taka myśl przyświecała człowiekowi, który przyczynił się do rozwoju całkowicie nowej grupy algorytmów asymetrycznych. Był to Whitfield Diffie. Jego koncepcja opierała się na założeniu dotąd uważanym za technicznie niemożliwe do zrealizowania. Klucz szyfrujący miał być powszechnie dostępny, przy zachowaniu bezpieczeństwa kryptogramu. Do tej pory obowiązywała niepodważalna zasada kryptografii - zasada tajności klucza, który służył do szyfrowania i deszyfrowania wiadomości. Diffie zaprojektował system oparty o dwa klucze szyfrujące. Taka para kluczy miała być tworzona dla każdego użytkownika systemu. Pierwszy z kluczy (klucz publiczny) miał służyć do szyfrowania wiadomości, a drugi (klucz prywatny) do jej odszyfrowywania. System mógł też działać w drugą stronę. Wiadomość zaszyfrowana kluczem prywatnym, mogła zostać odszyfrowana jedynie przy użyciu przypisanego do niego klucza publicznego. Taka innowacja pozwoliła na rozwój kryptografii w całkowicie nowym, nieznanym dotąd, kierunku. Dzięki takiej kombinacji możliwe stawało się uwierzytelnianie nadawcy wiadomości elektronicznej, gdyż tylko osoba posiadająca klucz prywatny mogła zaszyfrować dokument tak, aby dało się go odczytać przy użyciu klucza publicznego.

Sama teoria kluczy nie wystarczała i konieczne stało się opracowanie odpowiednich podstaw matematycznych, możliwych do zaimplementowania w językach programowania. Do rozwiązania tego problemu przyczynił się współpracownik Diffiego, którym był Marty Hellman. Diffie i Hellman wspólnie opracowali system matematyczny oparty na funkcji jednokierunkowej. Funkcja jednokierunkowa ma tą zależność, że jest łatwa do obliczenia, lecz trudna do odwrócenia. System ten znany jest obecnie jako algorytm Diffiego-Hellmana. Rozwiązał on nie tylko problem dystrybucji klucza, ale zapoczątkował technologię elektronicznego uwierzytelniania użytkowników.

Algorytm Diffiego-Hellmana [Kowalczyk(2020d)]

Algorytm opracowany przez Diffiego i Hellmana służy do ustalania wspólnego tajnego klucza przy użyciu publicznych środków komunikacji. Jego działanie jest zazwyczaj prezentowane na przykładzie wymiany informacji pomiędzy dwoma użytkownikami, Alicją i Bobem. W celu wynegocjowania wspólnego sekretnego klucza, należy wykonać poniższe kroki:

1. Alicja i Bob uzgadniają liczbę pierwszą p oraz bazę g .
2. Alicja wybiera sekretną liczbę całkowitą a , a następnie wysyła Bobowi liczbę $A = g^a \text{ mod } (p)$.
3. Bob wybiera sekretną liczbę całkowitą b , a następnie wysyła Alicji liczbę $B = g^b \text{ mod } (p)$.
4. Alicja oblicza liczbę $k = B^a \text{ mod } (p)$.
5. Bob oblicza tę samą liczbę $k = A^b \text{ mod } (p)$.
6. Od tego momentu Alicja i Bob współzielą sekretną liczbę k .

Otrzymana liczba k jest bardzo duża. Z założenia, liczby a oraz b mają przynajmniej po 100 cyfr, a liczba pierwsza p ma przynajmniej 300 cyfr. Liczba k może być użyta jako klucz symetryczny do szyfrowania całej dalszej komunikacji między Alicją i Bobem. Po ustaleniu klucza symetrycznego, Alicja i Bob usuwają wszelkie ślady po swoich wartościach a oraz b . Dzięki temu mają pewność, iż nikt nie wykradnie jednej z liczb w celu uzyskania tego samego klucza.

Przykład obliczeń

Dla uproszczenia obliczeń pominięte zostanie wymaganie dotyczące wielkości liczb początkowych.

1. Alicja i Bob uzgadniają liczbę pierwszą $p = 11$ oraz bazę $g = 5$.
2. Alicja wybiera sekretną liczbę całkowitą $a = 8$, a następnie wysyła Bobowi liczbę $A = g^a \text{ mod } (p)$.
 - $A = 5^8 \text{ mod } (11)$
 - $A = 390\,625 \text{ mod } (11)$
 - $A = 4$
3. Bob wybiera sekretną liczbę całkowitą $b = 6$, a następnie wysyła Alicji liczbę $B = g^b \text{ mod } (p)$.
 - $B = 5^6 \text{ mod } (11)$
 - $B = 15\,625 \text{ mod } (11)$
 - $B = 5$
4. Alicja oblicza liczbę $k = B^a \text{ mod } (p)$.
 - $k = 5^8 \text{ mod } (11)$
 - $k = 390\,625 \text{ mod } (11)$
 - $k = 4$
5. Bob oblicza tę samą liczbę $k = A^b \text{ mod } (p)$.
 - $k = 4^6 \text{ mod } (11)$
 - $k = 4\,096 \text{ mod } (11)$
 - $k = 4$
6. Od tego momentu Alicja i Bob współdzielą sekretną liczbę $k = 4$.

Zastosowanie

Poza ustaleniem wspólnego klucza symetrycznego, algorytm Diffiego-Hellmana może zostać również wykorzystany do szyfrowania z użyciem klucza publicznego lub prywatnego. Dla przykładu, Alicja mogłaby wysłać wiadomość do Boba, która byłaby zaszyfrowana jego kluczem publicznym, którym byłby zestaw liczb g, p oraz $(g^a \bmod (p))$. Aby Alicja mogła wysłać sekretną wiadomość, musi uprzednio wylosować liczbę b i wysłać Bobowi liczbę $(g^b \bmod (p))$ w postaci jawnej. Alicja może już zaszyfrować wiadomość z użyciem klucza symetrycznego $((g^a)^b \bmod (p))$ i wysłać ją do Boba. Tylko on będzie w stanie określić wartość liczby b i odszyfrować wiadomość.

Podsumowanie

Odkrycie Diffiego oraz Hellmana przyczyniło się do dalszego rozwoju kriptografii asymetrycznej. Mimo możliwego zastosowania ich algorytmu w celu szyfrowania wiadomości, szukano bardziej efektywnych rozwiązań, które zapewniałyby jeszcze większy poziom bezpieczeństwa.

Algorytm RSA [Kowalczyk(2020f)], [Wałaszek(2020)]

Kryptosystem z kluczem publicznym został rozwinięty przez trzech naukowców z uniwersytetu w Stanford - Rona Rivesta, Adi Shamira i Leonarda Adlemana. Przełomowego odkrycia dokonał Rivest. Stworzył własny system obliczeń, który miał zastąpić algorytm Diffiego-Hellmana. System ten bazuje na problemie rozkładu dużych liczb na czynniki pierwsze. Algorytm Rivesa i jego współpracowników został opatentowany pod nazwą **RSA**. Jest on obecnie jednym z najbardziej popularnych algorytmów szyfrowania z kluczem publicznym.

Generowanie klucza publicznego oraz prywatnego

Para kluczy algorytmu **RSA** może być wygenerowana przy użyciu poniższego algorytmu:

1. Wybieramy dwie różne losowe liczby pierwsze p oraz q .
2. Obliczamy $n = p \cdot q$. Liczba n jest modułem obu kluczy.
3. Obliczamy wartość funkcji Eulera dla n :

$$\varphi(n) = \varphi(p) \cdot \varphi(q) = (p - 1) \cdot (q - 1)$$

4. Wybieramy liczbę całkowitą e , która jest większa od 1 i mniejsza niż poprzednio wyliczona wartość $\varphi(n)$. Obie liczby powinny być względnie pierwsze, tzn. takie, których największym wspólnym dzielnikiem jest jeden. Liczba e jest wykładnikiem klucza publicznego.
5. Wyznaczamy taką liczbę d , że: $d \cdot e = 1 \bmod (\varphi(n))$. Liczba d jest wykładnikiem klucza prywatnego.

Klucz publiczny jest parą liczb (e, n) , a klucz prywatny to para liczb (d, n) . Zgodnie z założeniem algorytmu, wszystkie liczby związane z kluczem prywatnym muszą być trzymane w tajemnicy.

Przykład obliczeń

Dla zobrazowania działania algorytmu, wykonane zostaną obliczenia dla kluczy.

1. Wybieramy dwie różne losowe liczby pierwsze, niech $p = 13$ i $q = 11$.

2. Obliczamy $n = p \cdot q$.

- $n = 13 \cdot 11$
- $n = 143$

3. Obliczamy wartość funkcji Eulera dla $n = 143$:

$$\varphi(n) = \varphi(p) \cdot \varphi(q) = (p - 1) \cdot (q - 1)$$

$$\varphi(143) = \varphi(13) \cdot \varphi(11) = (13 - 1) \cdot (11 - 1)$$

$$\varphi(143) = 12 \cdot 10$$

$$\varphi(143) = 120$$

4. Wybieramy liczbę całkowitą e , która będzie względnie pierwsza z $\varphi(n) = 120$.

Niech $e = 7$, ponieważ $NWD(120, 7) = 1$.

5. Wyznaczamy taką liczbę d , że: $d \cdot e = 1 \bmod (\varphi(n))$.

$$d \cdot 7 \bmod 120 = 1$$

Liczba d ma być odwrotnością modulo $\varphi(n)$ liczby e , czyli warunek ten spełnia $d = 103$.

6. Otrzymujemy klucz publiczny $(7, 143)$ oraz klucz prywatny $(103, 143)$.

Szyfrowanie

Po wygenerowaniu kluczy, można wykorzystać klucz publiczny do zaszyfrowania wiadomości. Wiadomość musi zostać zamieniona na liczby naturalne, które muszą należeć do przedziału $(0; n)$. Dla przykładu, zaszyfrowana zostanie mała litera "a", której przypisana liczba, zgodnie z kodowaniem **ASCII**, wynosi 97. Zakładamy, iż nadawca oraz odbiorca ustalili, że wysyłać będą do siebie znaki, które zamienią na ich reprezentację dziesiętną w tym systemie kodowania znaków. Niech $a = 97$, wtedy:

$$c = a^e \bmod (n)$$

$$c = 97^7 \bmod (143)$$

$$c = 80\ 798\ 284\ 478\ 113 \bmod (143)$$

$$c = 59$$

Liczba c to zaszyfrowana liczba a . W kodzie **ASCII**, znak kodowany liczbą 59 to średnik. Taki znak można wysłać do adresata.

Deszyfrowanie

Po odebraniu przez adresata znaku średnika, można zamienić go na jego liczbową reprezentację w ustalonym kodowaniu i uzyskać liczbę $c = 59$. Teraz można przystąpić do procesu deszyfrowania, z wykorzystaniem klucza prywatnego $(103, 143)$.

$$t = c^d \bmod (n)$$

$$t = 59^{103} \bmod (143)$$

Liczba do wyliczenia jest zbyt duża, by można byłoby ją policzyć w klasyczny sposób. Jednak adresata interesuje jedynie reszta z dzielenia tej liczby przez 143. Można rozłożyć potęgę na iloczyn składników o wykładnikach równych kolejnym potęgom liczby dwa:

$$59^{103} \bmod (143) = 59^{64+32+4+2+1} \bmod (143)$$

$$(59^{64} \bmod (143)) \cdot (59^{32} \bmod (143)) \cdot (59^4 \bmod (143)) \cdot (59^2 \bmod (143)) \cdot (59 \bmod (143))$$

$$59^1 \bmod (143) = 59$$

$$59^2 \bmod (143) = 3481 \bmod (143) = 49$$

$$59^4 \bmod (143) = (59^2 \bmod (143))^2 = 49^2 \bmod (143) = 113$$

$$59^8 \bmod (143) = (59^4 \bmod (143))^2 = 113^2 \bmod (143) = 42$$

$$59^{16} \bmod (143) = (59^8 \bmod (143))^2 = 42^2 \bmod (143) = 48$$

$$59^{32} \bmod (143) = (59^{16} \bmod (143))^2 = 48^2 \bmod (143) = 16$$

$$59^{64} \bmod (143) = (59^{32} \bmod (143))^2 = 16^2 \bmod (143) = 113$$

Do wyliczenia potęgi, adresat bierze tylko te reszty, które występują w sumie potęg liczby 2:

$$t = 59^{103} \bmod (143)$$

$$t = (113 \cdot 16 \cdot 113 \cdot 49 \cdot 59) \bmod (143)$$

$$t = 590\,642\,864 \bmod (143) = 97$$

Adresat odszyfrował liczbę i po zakodowaniu jej zgodnie z **ASCII**, odczytuje małą literę "a".

2.3.2 Symetryczne algorytmy blokowe

Mimo wielu zalet algorytmu **RSA**, posiada on również znaczącą wadę. Szyfrowanie jest mało wydajne i w przypadku dużych plików, może ono zajść dużo czasu. Znaczająco wyższą efektywnością wykażą się tutaj symetryczne algorytmy blokowe, które operują na partiach danych nazywanych blokami.

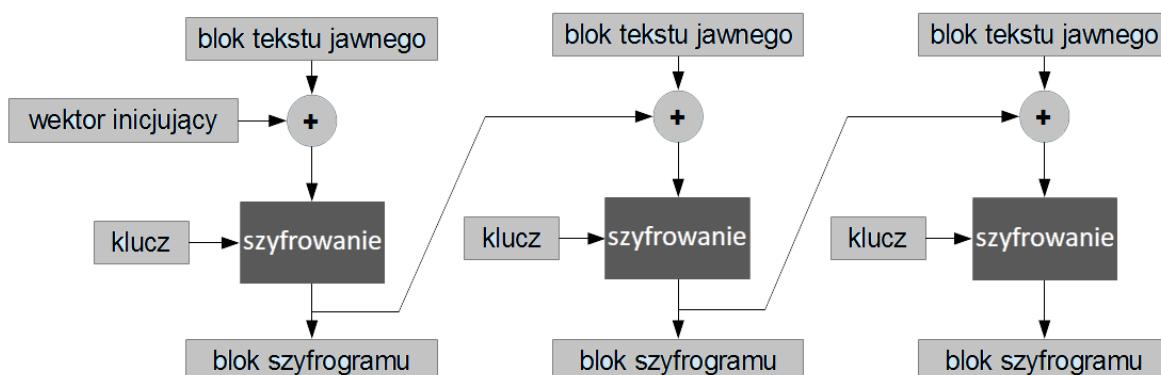
Bloksy szyfrowane są iteracyjnie. Dane wejściowe dzielone są na bloki równej długości i każdy z nich jest przetwarzany z wykorzystaniem symetrycznego klucza. Długość bloku danych, długość klucza i wykonywane operacje zależą od wybranego algorytmu.

Szyfry blokowe używają funkcji efektywnych i deterministycznych. Główną operacją wykonywaną na blokach jest permutacja ich zawartości w sposób pseudolosowy. Odszyfrowanie danych polega na wykonaniu permutacji odwrotnych, które powinny być porównywalnie efektywne.

Algorytmy blokowe posiadają różne tryby wiązania bloków. W ramach pracy zostanie omówiony tryb **CBC** (ang. *Cipher-Block Chaining*), który został wykorzystany w algorytmie **AES** podczas implementacji aplikacji **PAAK** omawianej w rozdziale 3.

Tryb CBC algorytmu blokowego

Tryb ten został opracowany przez firmę **IBM** w 1976 roku. Do każdego kolejnego bloku jawnego dodawany jest XOR poprzednio otrzymanego bloku szyfrogramu. Wynik tej operacji jest szyfrowany w klasyczny sposób. Taki algorytm powoduje, iż każdy kolejny blok szyfrogramu zależy od poprzedniego. Dla pierwszego bloku informacji jawnych, XOR jest dodawany do losowego wektora inicjującego, który jest takiej samej długości jak długość bloku danych.



Rysunek 2.7: Schemat szyfrowania w trybie CBC [Kowalczyk(2020b)].

W sytuacji, gdy długość wiadomości nie jest wielokrotnością długości pojedynczego bloku, ostatni blok wymaga uzupełnienia brakujących bajtów w celu zapełnienia bloku.

Dopełnianie bloków [Kowalczyk(2020c)]

Proces ten polega na dodawaniu uzgodnionych wartości na końcu przesłanych wiadomości. Użycie standardu dopełniania musi być jasne, gdyż odbiorca musi określić,

gdzie kończy się oryginalna wiadomość. Wszystkie standardy działają w podobny sposób i opisują jakie wartości powinny być dodawane do wiadomości w celu uzupełnienia ostatniego bloku danych. Poniżej opisano niektóre dopełnienia wykorzystywane w algorytmach blokowych:

- Dopełnianie bitowe polega na dołączeniu pojedynczego bitu 1 do wiadomości i uzupełnieniu reszty brakujących bitów zerami.
- Dopełnianie **TBC** (ang. *Trailing bit complement*) jest zależne od ostatniego bitu danych. Jeśli ostatnim bitem danych jest zero, to bity dopełnienia będą jedynkami, a jeśli ostatnim bitem danych jest jedynka, to dopełnienie będzie składać się z samych zer.
- Dopełnienie **PKCS5** i **PKCS7** uzupełnia blok bajtami, w przeciwnieństwie do poprzednio wymienionych standardów. Oba standardy określają, iż każdy bajt dopełnienia ma wartość równą ilości wszystkich dodanych bajtów dopełnienia.

Ustalenie dopełnienia jest bardzo ważne i pozwala szyfrować oraz deszyfrować dane, których długość nie jest podzielna bez reszty przez rozmiar ustalonego bloku. W momencie deszyfrowania, algorytm oczekuje, że na końcu szyfrowanej wiadomości znajdzie się jakieś dopełnienie. W sytuacji, gdy ilość szyfrowanych danych jest wielokrotnością wielkości bloku, dodawany jest blok końcowy zawierający wyłącznie dopełnienie. Brak dodatkowego bloku z samym dopełnieniem wywołalby błąd podczas deszyfrowania.

Bezpieczeństwo symetrycznych algorytmów blokowych [Point(2020)]

Atak słownikowy

Atak słownikowy jest podobny do ataku siłowego, lecz zamiast sprawdzać każdą kombinację hasła, sprawdzane są poszczególne wyrazy pochodzące z przygotowanego słownika. Dane pochodzące z listy wykradzionych haseł wykazały, iż bardzo dużo osób używa haseł przewidywalnych, prostych do zapamiętania i zdecydowanie zbyt słabych [Marino(2020)]. Wyżej opisana metoda może zostać wykorzystana do ataku na algorytmy blokowe.

Pomimo, iż rozmiar bloku nie ma bezpośredniego przełożenia na siłę algorytmu, to istnieją pewne założenia, których należy się trzymać podczas wyboru rozmiaru bloku dla algorytmu szyfrującego. Należy rozważyć przypadki dla zbyt małego bloku oraz zbyt dużego.

Założymy, że blok posiada n bitów. W takim przypadku, tekst jawnego zapisany w bitach składa się z 2^n wariacji. W przypadku zastosowania małego rozmiaru bloku, jeżeli atakujący odkryje tekst jawnego jednego bloku, który jest powiązany z innymi blokami danych zaszyfrowanych tym samym algorytmem, może on przeprowadzić atak słownikowy oparty o pary tekstu jawnego i szyfrogramu. Tym sposobem może odszyfrować pozostałe zaszyfrowane bloki w oparciu o ten słownik. Rozwiązaniem jest zastosowanie większego bloku, który spowoduje, że rozmiar słownika do przeprowadzenia ataku musi być znacznie większy. Nie można jednak stosować bardzo dużych bloków, gdyż takie rozwiązanie będzie bardzo nieefektywne w działaniu. Liczba informacji

szyfrowanych może być mniejsza od rozmiaru dużego bloku i musi zostać uzupełniona dopełnieniem przed zastosowaniem szyfrowania. Najlepszym rozwiązaniem jest stosowanie bloków będących wielokrotnością 8 bitów, czyli będą to bloki składające się z całkowitej liczby bajtów. Najpopularniejsze rozmiary bloków używanych w szyfrach blokowych to 64, 128, 192, 256 bitów.

W przeciwnieństwie do rozmiaru bloków, długość klucza użytego do zaszyfrowania danych ma duży wpływ na ich bezpieczeństwo. Im dłuższy klucz, tym trudniej go odgadnąć i bezpiecznejsze są zaszyfrowane nim informacje. W przypadku niektórych symetrycznych algorytmów blokowych, długość klucza wpływa na liczbę operacji wykonywanych w procesie szyfrowania, co bezpośrednio przekłada się na silniejsze szyfrowanie. Uznaje się, iż bezpieczny klucz algorytmu posiada 256 bitów długości, chociaż do niedawna powszechnie używano kluczy o długości 128 bitów.

AES (ang. *Advanced Encryption Standard*) [Kowalczyk(2020a)],
[Piwowarczyk(2012)]

Rijndael to oryginalna nazwa algorytmu opublikowanego w 1997 roku, który został zaprojektowany przez belgijskich kryptologów Vincenta Rijmena i Joana Daemena. Był jednym z wielu algorytmów zgłoszonych do konkursu organizowanego przez **NIST**, w którym chciano wyłonić następcę algorytmu **DES** (ang. *Data Encryption Standard*), którego oryginalna nazwa brzmiała **Lucifer**. Przestał on spełniać wymagania bezpieczeństwa ze względu na rosnącą moc obliczeniową komputerów. W 1997 roku rozszyfrowano wiadomość zabezpieczoną algorymem **DES** bez znajomości klucza. Operacja ta zajęła 3 dni obliczeń przy użyciu najnowszego w tamtych czasach komputera wartego 250 000 dolarów. Zaczęto szukać jego następcy i zwycięskim algorymem w konkursie okazał się **Rijndael**. Algorytm ten został przyjęty jako standard federalny Stanów Zjednoczonych w 2002 roku i od tamtej pory nazywany jest **AES**.

Wybór nowego standardu **AES** trwał 5 lat nieustannej oceny zgłoszonych algorytmów. Odporność na ataki kryptoanalityczne nie była jedynym wyznacznikiem odpowiedniego algorytmu. Brano pod uwagę kryteria takie, jak:

- Ogólne bezpieczeństwo oceniane przez liczną grupę kryptologów.
- Implementacja programowa, która miała zapewnić wysoką wydajność algorytmu na różnych platformach.
- Implementacja sprzętowa związana z wydajnością algorytmu oraz samym rozmiarem programu.
- Niskie użycie pamięci operacyjnej w celu użycia algorytmu na urządzeniach o bardzo ograniczonej pamięci.
- Atak implementacyjny, który miał potwierdzić odporność na ataki na fizyczną implementację algorytmu.
- Procesy szyfrowania i deszyfrowania, które miały mieć jak najmniejszą różnicę między sobą pod kątem zapotrzebowania na pamięć operacyjną.

- Sprawność zarządzania kluczami, które oceniało złożoność tego procesu.
- Użycie zrównoleglej wykonywanych operacji w celu przetwarzania równoległego poprawiającego ogólną wydajność algorytmu.
- Wszechstronność i uniwersalność oznaczające elastyczność przyjmowanych przez algorytm parametrów, obsługa nietypowych długości bloków, kluczy oraz możliwość sterowania ilością rund szyfrowania wewnątrz algorytmu.

Mimo, iż **Rijndael** okazał się mniej bezpieczny od algorytmu **Serpent**, który zajął drugie miejsce w konkursie na nowy standard rządu federalnego, to był od niego szybszy i zyskał miano **AES**. Dokładny opis algorytmu jest opisany w źródle [Kowalczyk(2020a)] oraz w dokumentacji [Joan Daemen(1999)].

Podsumowanie

Zdobyta wiedza z zakresu bezpieczeństwa informacji 2.1, procesów uwierzytelnienia i autoryzacji 2.1, kryptologii 2.2 i wybranych algorytmów szyfrowania, omawianych w sekcjach 2.3.2 i 2.3.1, pozwoliła autorowi na przygotowanie założeń projektowych 3.2 tworzonej aplikacji **PAAK**, omawianej w rozdziale 3, wraz z doborem adekwatnych algorytmów kryptografii w celu zapewnienia wysokiego poziomu zabezpieczeń.

W następnej sekcji omówione zostaną istniejące rozwiązania, które wykorzystują urządzenia pamięci masowej w podobny sposób, w jaki ma je wykorzystywać aplikacja autora pracy.

2.4 Przegląd istniejących rozwiązań

Najbardziej znane oprogramowania, które wykorzystują urządzenia pamięci masowej USB w celu zabezpieczenia komputera to:

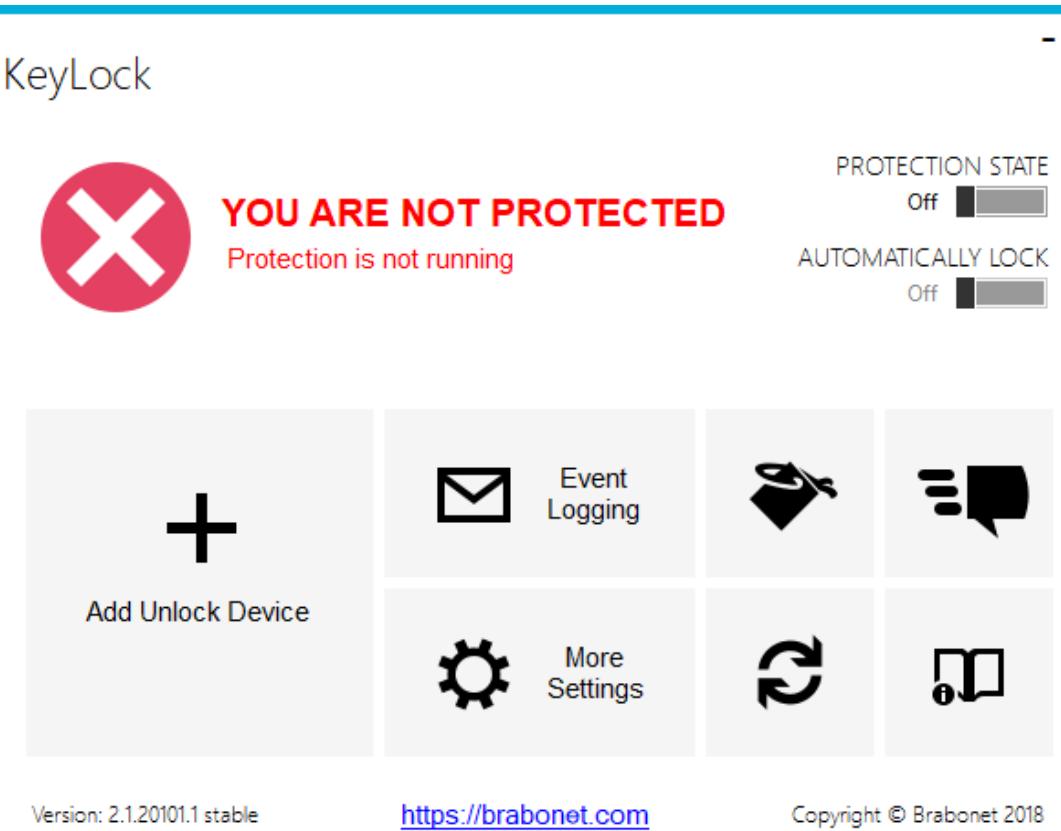
1. **KeyLock**,
2. **Predator**,
3. **Rohos Logon Key**.

Wszystkie wymienione aplikacje spełniają podstawowe wymaganie - ochronę komputera przed nieautoryzowanym dostępem z wykorzystaniem urządzenia pamięci masowej. Metoda działania każdego programu jest podobna i polega na programowym odłączeniu urządzeń wejścia i wyjścia komputera z jednoczesnym wylogowaniem użytkownika. Z tego powodu wskazane zostaną dodatkowe funkcje, które każdy z wymienianych programów posiada.

Autor pracy pobrał wszystkie omawiane aplikacje i zainstalował je lokalnie na swoim komputerze. Było to możliwe, gdyż wszystkie aplikacje posiadają bezpłatną wersję testową, która jest aktywna przez 2 tygodnie od daty instalacji. Wykonane zrzuty ekranu omawianych aplikacji zostały wykonane przez autora pracy.

2.4.1 Aplikacja KeyLock

Oprogramowanie **KeyLock** stworzyła firma **Brabonet**, która zajmuje się tworzeniem oprogramowania na systemy **Windows** oraz **Android**. Interfejs graficzny użytkownika jest nowoczesny, intuicyjny oraz minimalistyczny.



Rysunek 2.8: Interfejs ekranu głównego aplikacji **KeyLock** [Źródło własne].

Zabezpieczenie komputera wiąże się również z zablokowaniem menadżera zadań w celu uniemożliwienia wyłączenia procesu odpowiedzialnego za program. Posiada on również szereg funkcji dodatkowych:

1. prowadzenie rejestru logów z działania pracy programu,
2. nagrywanie wiadomości głosowych dla osób chcących uzyskać dostęp do komputera,
3. powiadomienia e-mail w razie próby uzyskania dostępu do komputera przez osobę nieautoryzowaną,
4. automatyczną aktualizację oprogramowania,
5. wygenerowanie nowego klucza,
6. import kluczów,
7. eksport kluczów,

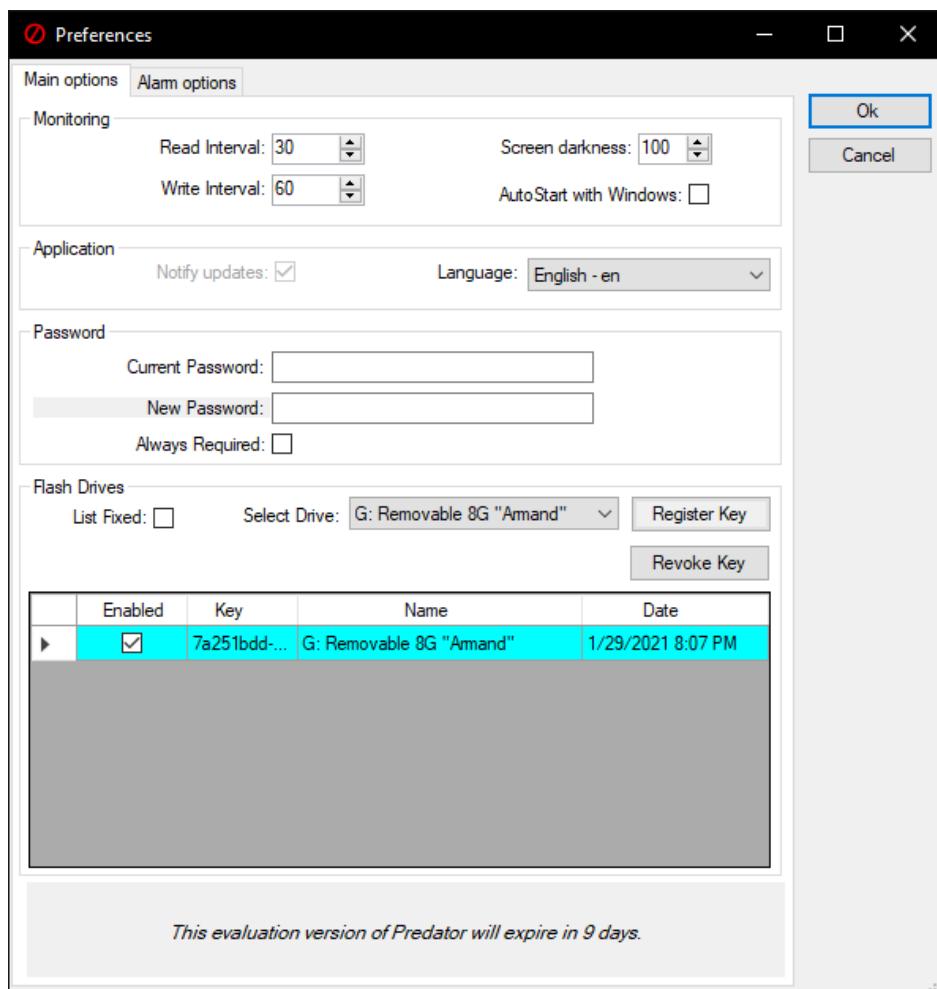
8. reset programu do ustawień początkowych,
9. przypisanie flagi domyślnego użytkownika, które skutkuje pominięciem podawana loginu podczas logowania.

Wszystkie informacje zostały zaczerpnięte ze strony producenta oprogramowania [Brabonet(2020)].

2.4.2 Aplikacja Predator

Program **Predator** posiada podstawową wersję darmową oraz płatną wersję profesjonalną. Omówiona zostanie wersja darmowa. Zgodnie z tym co podaje producent, oprogramowanie pomaga ograniczać czas, który dzieci spędzają na korzystaniu z komputera. Dla każdego urządzenia - klucza można:

1. adefiniować porę dnia kiedy użytkownik posiadający klucz może korzystać z komputera,
2. ustalić zasady korzystania z komputera dla każdego dnia tygodnia,
3. wybrać metodę zablokowania komputera (wylogowanie, wyłączenie).



Rysunek 2.9: Interfejs ekranu głównego aplikacji **Predator** [Źródło własne].

Ponadto program posiada bogatą dokumentację, która wyjaśnia wiele aspektów z działania programu. Producent zadbał również o stronę internetową z sekcją najczęściej zadawanych pytań, na które podaje odpowiedzi. Program posiada wiele dodatkowych funkcji, które pokrywają się z funkcjami poprzednio omawianego programu, ale niektóre z nich zasługują na wyróżnienie:

1. zabezpieczenie komputera przed nieautoryzowanymi osobami w przypadku uruchomienia systemu w trybie awaryjnym,
2. ochrona wielu komputerów z wykorzystaniem jednego klucza,
3. ochrona wielu kont systemowych na jednym komputerze (dla każdego konta można przypisać osobne ustawienia),
4. automatyczna zmiana kodu bezpieczeństwa zapisywanego na urządzeniu pamięci masowej USB w celu zapobiegnięcia ryzyka włamania wynikającego z wykonania kopii klucza,
5. zabezpieczenie przed wyłączeniem programu z poziomu menadżera zadań.

Wszystkie informacje zostały zaczerpnięte ze strony producenta oprogramowania [Predator(2020)].

2.4.3 Aplikacja Rohos Logon Key

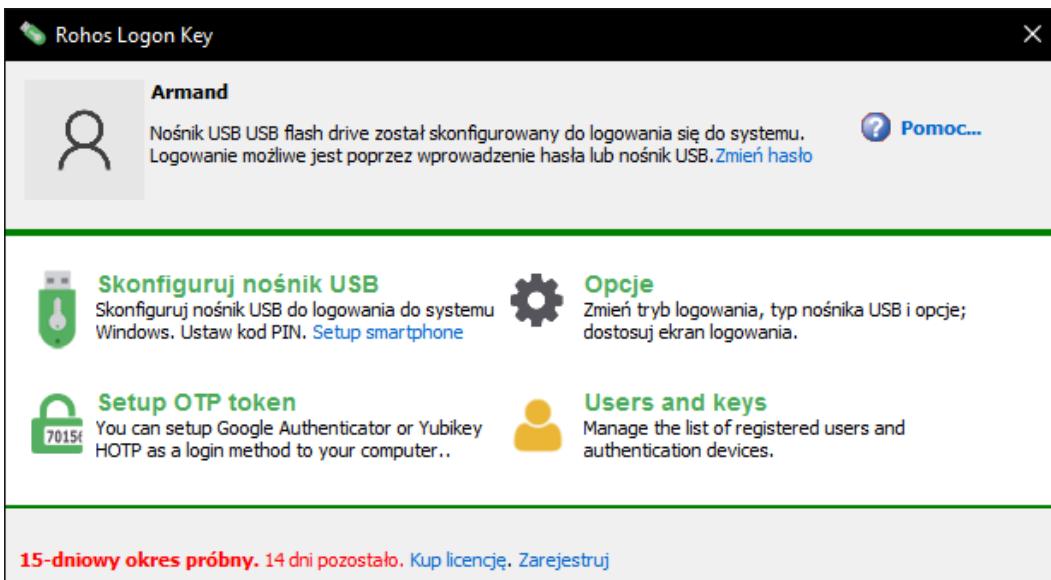
Rohos Logon Key to oprogramowanie stworzone przez firmę **SAFEJKA S.R.L.** Aplikacja w odróżnieniu od swoich poprzedników może korzystać również z systemu *Google Authenticator*. **Rohos Logon Key** jest oprogramowaniem wykorzystującym uwierzytelnienie dwuskładnikowe. Poza wykorzystaniem uwierzytelnienia **Google**, współpracuje ono z technologiami, takimi jak:

- RDIF (ang. *Radio-frequency identification*) - systemem zdalnej identyfikacji radiowej,
- OTP (ang. *One-time password*) - hasło jednorazowe, które zmienia się po każdym użyciu,
- U2F (ang. *Universal 2nd Factor*) - otwarty standard, który umacnia i upraszcza uwierzytelnienie dwuskładnikowe.

Pozostałe podstawowe funkcje programu:

- Zabezpieczenie przed obejściem programu przez uruchomienie systemu w trybie awaryjnym.
- Wykorzystanie jednego klucza w wielu komputerach.
- Uwierzytelnienie konta za pomocą zdalnego pulpitu.
- Możliwość wykorzystania wielu kluczy na jednym komputerze.

- Zapisywanie logów autoryzacji w plikach lub za pomocą systemu **Windows Event Log**.
- Możliwość ustawienia własnego obrazu na ekranie logowania.
- Opcjonalne ukrycie ikony logowania (pozwala to na ukrycie występowania systemu uwierzytelnienia dwuskładnikowego).



Rysunek 2.10: Interfejs ekranu głównego aplikacji **Rohos Logon Key** [Źródło własne].

Program wykorzystuje szyfrowanie **AES-256**, które zapewnia bezpieczeństwo hasła przechowywanego w kluczu, którym jest pendrive. Wszystkie informacje zostały zaczerpnięte ze strony producenta oprogramowania [Rohos(2020)].

2.4.4 Podsumowanie przeglądu istniejących rozwiązań

Wszystkie wymienione programy blokują dostęp do komputera w bardzo podobny sposób. Po wyjęciu urządzenia pamięci masowej USB, które program traktuje jako klucz sprzętowy, system staje się niemożliwy do normalnego użytkowania. Programowo odłączone zostają urządzenia wejścia oraz wyjścia, by uniemożliwić jakiekolwiek operacje w systemie. Zostaje dodana nowa opcja dla panelu logowania, która umożliwia podpięcie klucza zamiast wpisywania standardowego hasła.

Wszystkie rozwiązania zapewniają ochronę całego komputera. Stanowi to pierwszą linię zabezpieczeń, która ma uniemożliwić jego nieautoryzowane użycie. Dane użytkowników nie są jednak chronione przez głębsze systemy zabezpieczeń. Pliki pozostają zapisane na dysku w formie jawnej i są narażone na ataki innego typu, np. fizyczna kradzież dysku, w celu późniejszego odzyskania danych przy użyciu specjalistycznych technik. W rozdziale 3, poświęconym aplikacji **PAAK**, zostanie zaprezentowane podejście odmienne do zaprezentowanych powyżej rozwiązań, które będzie stanowiło uzupełnienie brakujących zabezpieczeń na głębszym ich poziomie.

Projekt i implementacja

W tym rozdziale, autor przyjmuje, iż *klucz sprzętowy* oznaczać będzie wykorzystywane przez aplikację **PAAK** urządzenie pamięci masowej, które zostało przypisane w aplikacji jako urządzenie zaufane.

Pomysł na aplikację **PAAK** (ang. *Pendrive as a Key*) powstał w wyniku konsultacji z promotorem oraz z pomysłu autora. Aplikacja jest odpowiedzią na rosnącą świadomość użytkowników komputera w kwestii bezpieczeństwa danych. Po przeprowadzeniu analizy podobnych rozwiązań, opisanej w rozdziale 2.4, widocznie brakuje produktu, który uplasuje się między aplikacjami do ochrony komputera jako stacji roboczej, a oprogramowaniem szyfrującym dyski. Cieszący się popularnością produkt firmy **Yubico** wymaga wkładu finansowego i nie trafia dokładnie w segment ochrony plików lokalnych. Aplikacja **PAAK** ma zapewnić ochronę tylko poszczególnych plików, wybranych przez użytkownika.

3.1 Prototypowanie i wybór technologii

W celu zrealizowania głównej funkcji programu, aplikacja musi odczytywać informacje o urządzeniu pamięci masowej w sposób programowy. W tym celu należało poszukać technologii, która w najbardziej przystępny sposób pozwoli na implementację tego wymagania. Pierwszym kandydatem technologicznym był język **Python**. Ten język programowania wysokiego poziomu posiada rozbudowany pakiet bibliotek standardowych oraz liczne moduły stworzone przez społeczność użytkowników. Dodatkowo, jest to język interpretowany, a co za tym idzie, dostępność interpreterów tego języka pozwala uruchamiać programy na wielu systemach operacyjnych. Taki wybór miałby gwarantować elastyczność projektu i późniejsze stworzenie wersji wieloplatformowej.

Prototypowanie rozpoczęło się od prostej aplikacji napisanej w języku **Python**, która miała rozpoznawać podpinane urządzenia pamięci masowej do komputera. Ze względu na budowę systemów operacyjnych, które pełnią rolę pośredniczącą pomiędzy aplikacjami, a urządzeniami fizycznymi podłączonymi do komputera, odczyt danych o urządzeniach pamięci masowej musiał się odbywać poprzez system operacyjny.

Aby wykonywać zapytania do systemu operacyjnego, aplikacja napisana w języku **Python** wymaga dodatkowej biblioteki, która umożliwia takie operacje. Taką biblioteką

pozwalającą na komunikację z urządzeniami podłączonymi przez port USB jest **PyUSB**. Do jej prawidłowego działania wymagana jest jedna z wymienionych bibliotek w systemie operacyjnym: **libusb 0.1**, **libusb 1.0** lub **OpenUSB**. System **Windows 10**, pod którym rozpoczęto pracę nad aplikacją **PAAK**, nie posiada żadnej z wymienionych bibliotek.

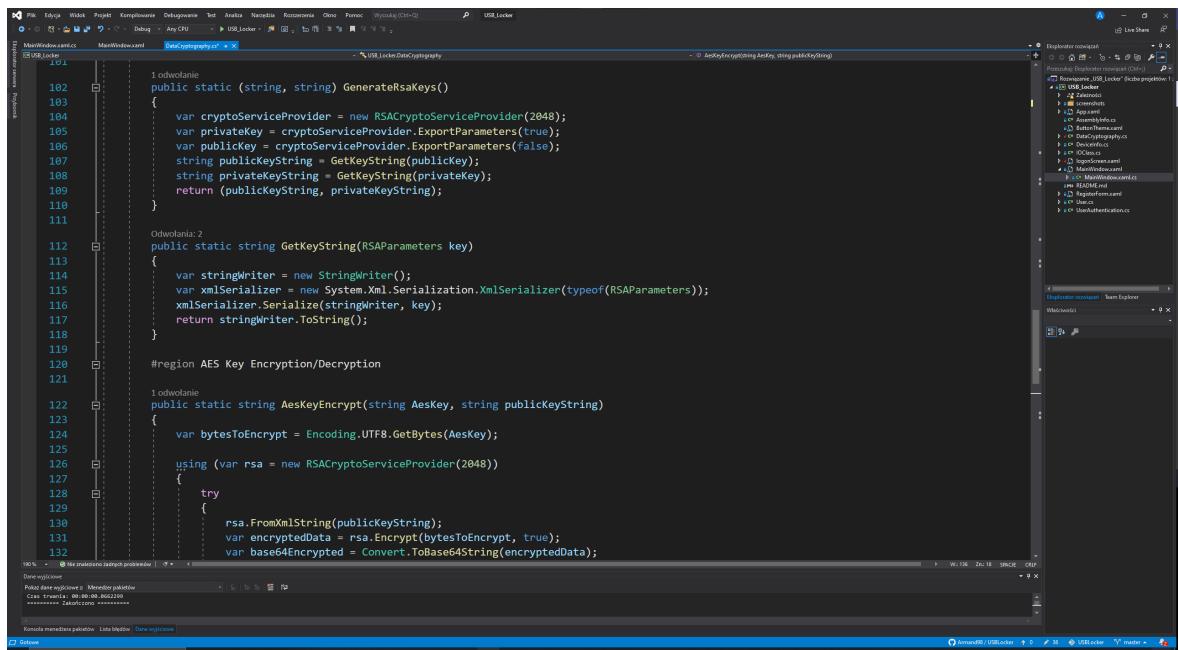
Autor pracy wykonał liczne próby instalacji ww. bibliotek, lecz ich dokumentacje nie precyzovaly procesu instalacji w celu zapewnienia ich odpowiedniego funkcjonowania. Zostały przeszukane liczne fora internetowe w poszukiwaniu rozwiązania problemu instalacji brakujących bibliotek, lecz znalezione potencjalne rozwiązania nie znajdowały zastosowania dla przypadku autora pracy.

Z racji liczniego upływu czasu na szukanie rozwiązania tego problemu, zaczęto szukać innej technologii, która pozwoliłaby na sprawne wykonanie aplikacji. W celu statystycznego ograniczenia potencjalnych problemów w procesie implementacji rozwiązania, autor zdecydował się przeanalizować język **C#**, pod kątem wykorzystania go do pozyskiwania informacji o urządzeniach podłączanych do komputera za pośrednictwem portów USB.

C# to wieloparadygmatowy język programowania wysokiego poziomu. Światło dzienne ujrzał pod sam koniec XX wieku. Stworzony został dla firmy **Microsoft**. Przez 20 lat rozwoju od daty publikacji, język ten został znacząco rozbudowany o m.in.:

- Typy generyczne w wersji 2.0.
- Wyrażenie lambda w wersji 3.0.
- Słowa kluczowe "await" i "async" w wersji 5.0, które znacząco ułatwiają programowanie asynchroniczne.
- Możliwość tworzenia funkcji lokalnych (definiowanych wewnątrz innych funkcji) w wersji 7.0.

C# pozwala również na wykorzystanie bibliotek umożliwiających wykorzystanie algorytmów kryptografii symetrycznej oraz asymetrycznej, które zostały użyte podczas tworzenia aplikacji.



Rysunek 3.1: Interfejs środowiska Visual Studio [Źródło własne].

Do procesu tworzenia aplikacji **PAAK** i zrealizowania wszystkich założeń projektowych, wykorzystany został język **C#**. Za środowisko programistyczne posłużyło **Visual Studio 2019**, widoczne na rysunku 3.1.

3.2 Założenia projektowe

Projekt wymagał podzielenia go na mniejsze cele. Takie podejście pozwala oddzielić od siebie problemy, których rozwiązania częściowe pozwolą uzyskać wynik końcowy. Tak oto wyodrębniono poszczególne etapy projektowania aplikacji.

1. Odczytanie informacji urządzeń pamięci masowej USB
2. Projekt interfejsu użytkownika
3. Wykonywanie operacji współbieżnych
4. Szyfrowanie i deszyfrowanie danych

Odczytywanie informacji urządzeń pamięci masowej USB

Głównym zadaniem programu jest ochrona danych w oparciu o szyfrowanie z użyciem klucza sprzętowego jakim jest *pendrive* USB. W tym celu program rozpoznaje podpięte urządzenia pamięci masowej i pozyskuje o nich informacje. Urządzenia wyświetlane są w formie listy, w której widnieje nazwa urządzenia oraz przypisana mu nazwa.

Projekt interfejsu użytkownika

Technologia **C# WPF** (ang. *Windows Presentation Foundation*) to struktura interfejsu użytkownika, która tworzy aplikacje klienckie dla komputerów stacjonarnych

[Microsoft(2020)]. Pozwala ona na tworzenie interfejsów użytkownika znanych z technologii webowych. Takie rozwiązanie pozwala na niestandardowy wygląd aplikacji oraz lepsze spersonalizowanie pod kątem praktycznym. Interfejs został również zrealizowany wedle zaleceń dotyczących tworzenia dostępnych serwisów internetowych **WCAG** (ang. *Web Content Accessibility Guidelines*) [W3C(2018)].

Wykonywanie operacji współbieżnych

Dzisiejsze komputery są wyposażone w procesory wielordzeniowe i wielowątkowe. Zastosowanie rozwiązań wielowątkowych pozwala na dużo efektywniejsze wykorzystanie zasobów komputera w celu dokonywania obliczeń. Dzięki takiemu rozwiązaniu, program może wykonywać obliczenia i operacje na przeznaczonym do tego wątku procesora. Wątek główny aplikacji oczekuje na dalsze polecenia użytkownika. W efekcie program nie zostaje zawieszony na czas trwania operacji. Programowe tworzenie wątków jest jednak bardzo kosztowne i z tego powodu została opracowana biblioteka **Task Parallel**. Umożliwia ona tworzenie zadań jako procesów równoległych. Zadanie jest niezależną jednostką, która jest uruchamiana w ramach konkretnego programu. Takie rozwiązanie jest bardziej wydajne pod kątem wykorzystania zasobów systemowych oraz zapewnia większą kontrolę nad kodem niż ma to miejsce w przypadku użycia wątków. Zadanie samodzielnie decyduje o ilości używanych wątków w sposób dynamiczny w środowisku uruchomieniowym.

Szyfrowanie i deszyfrowanie danych

Ochrona plików użytkownika zrealizowana jest z wykorzystaniem sprawdzonych i bezpiecznych metod szyfrowania. Omówione we wprowadzeniu algorytmy kriptografii asymetrycznej [**RSA** 2.3.1] oraz symetrycznej [**AES** 2.3.2] stanowią silny system szyfrowania, który odpowiednio wykorzystany, zapewnia wysoki poziom bezpieczeństwa plików.

Algorytm **AES** odpowiada za efektywne szyfrowanie i deszyfrowanie plików użytkownika. Klucz algorytmu **AES** jest chroniony przez szyfrowanie algorymem **RSA**. Klucz prywatny, służący do odszyfrowania symetrycznego klucza algorytmu **AES**, znajduje się w pamięci klucza sprzętowego. Takie rozwiązanie pozwala na bezpieczne przechowywanie symetrycznego klucza dla algorytmu **AES** w plikach lokalnych użytkownika w postaci zaszyfrowanej.

3.3 Wymagania funkcjonalne i niefunkcjonalne projektu

Autor wyznaczył następujące wymagania funkcjonalne:

1. Logowanie do aplikacji z użyciem loginu oraz hasła.
2. Rejestracja nowego użytkownika z wykorzystaniem formularza rejestracji, który zawiera:

- imię,
- nazwisko,
- login,
- hasło,
- data urodzenia,
- lista rozwijana z pytaniami bezpieczeństwa,
- pole tekstowe z odpowiedzią na pytanie bezpieczeństwa.

3. Ciągłe monitorowanie i wyświetlanie w liście podłączonych urządzeń USB:

Program stale sprawdza czy podłączono urządzenie USB i czy jest ono wykorzystywany kluczem sprzętowym. Jeśli nie jest, to wyświetla go w sekcji wyboru urządzeń do utworzenia nowych kluczy, a jeśli nim jest, to odblokowuje dostęp do zabezpieczonych danych.

4. Identyfikacja urządzenia pamięci masowej:

Program odczytuje dane urządzenia, takie jak:

- a) model urządzenia,
- b) numer seryjny,
- c) pojemność,
- d) nazwa wolumenu.

Na podstawie powyższych parametrów program rozpoznaje podłączone urządzenia, sprawdza czy klucz prywatny jest zgodny i pozwala uzyskać dostęp, gdy rozpozna zaufany identyfikator.

5. Dodanie urządzenia USB do listy zaufanych urządzeń: Takie działanie wiąże się z wykonaniem kilku operacji:

- a) W pierwszej kolejności wygenerowana zostaje para kluczy dla algorytmu **RSA**. Klucz publiczny zostaje przypisany do konta użytkownika, a klucz prywatny zostaje zapisany w ukrytym pliku na urządzeniu zewnętrznym.
- b) Klucz **AES** użytkownika zostaje zaszyfrowany z użyciem algorytmu **RSA** przy użyciu klucza publicznego. Tak zaszyfrowany klucz **AES** zostaje przypisany do danych zalogowanego użytkownika.
- c) Jeżeli wszystkie poprzednie operacje zostaną poprawnie wykonane, to dane identyfikacyjne urządzenia zostają przypisane do konta użytkownika, a on sam zostaje poinformowany o pozytywnym wyniku zakończenia procesu tworzenia klucza sprzętowego.

6. Eksplorator plików do zabezpieczenia:

Umożliwia wybór plików oraz folderów, które mają zostać zabezpieczone przez aplikację.

7. Usunięcie klucza sprzętowego:

Sprawdza istniejące relacje klucza sprzętowego z plikami i usuwa je, jeśli występują. Usuwany jest klucz prywatny **RSA** z klucza sprzętowego i jego dane są usuwane z pamięci programu.

8. Szyfrowanie danych:

Zabezpieczenie danych będzie polegało na ich zaszyfrowaniu algorytmem **AES**.

9. Deszyfrowanie danych:

Po rozpoznaniu klucza sprzętowego nastąpi automatyczne odszyfrowanie danych, które zostały powiązane z tym kluczem sprzętowym.

10. Tworzenie profili użytkowników:

Program może być używany przez wiele użytkowników. Każdy z nich posiada swój profil, który powiązany jest z zabezpieczonymi danymi i kluczem sprzętowym.

11. Moduł odzyskiwania danych:

Pozwala na odzyskanie danych w razie utraty lub uszkodzenia klucza sprzętowego. Składa się z:

- a) Formularza z pytaniem bezpieczeństwa i odpowiedzią na nie oraz hasłem użytkownika
- b) Funkcji wywołującej deszyfrowanie danych w wyniku podania poprawnych odpowiedzi na pytania uwierzytelniające

Postawione wymagania niefunkcjonalne:

1. Interfejs graficzny użytkownika zgodny ze standardem **WCAG** w kwestii kontrastu użytych kolorów.
2. Program funkcjonuje lokalnie na komputerze użytkownika.
3. Docelowy system operacyjny to **Windows 10**.

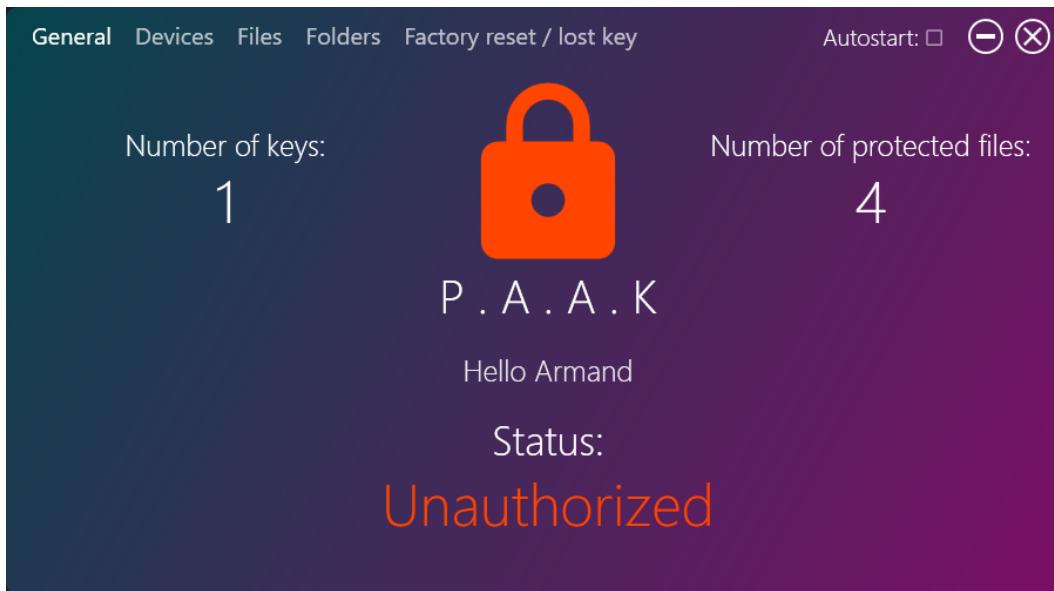
3.4 Dokumentacja deweloperska

Niniejsza sekcja zawiera dokumentację deweloperską, przedstawiającą technologiczne aspekty aplikacji **PAAK**.

Interfejs użytkownika

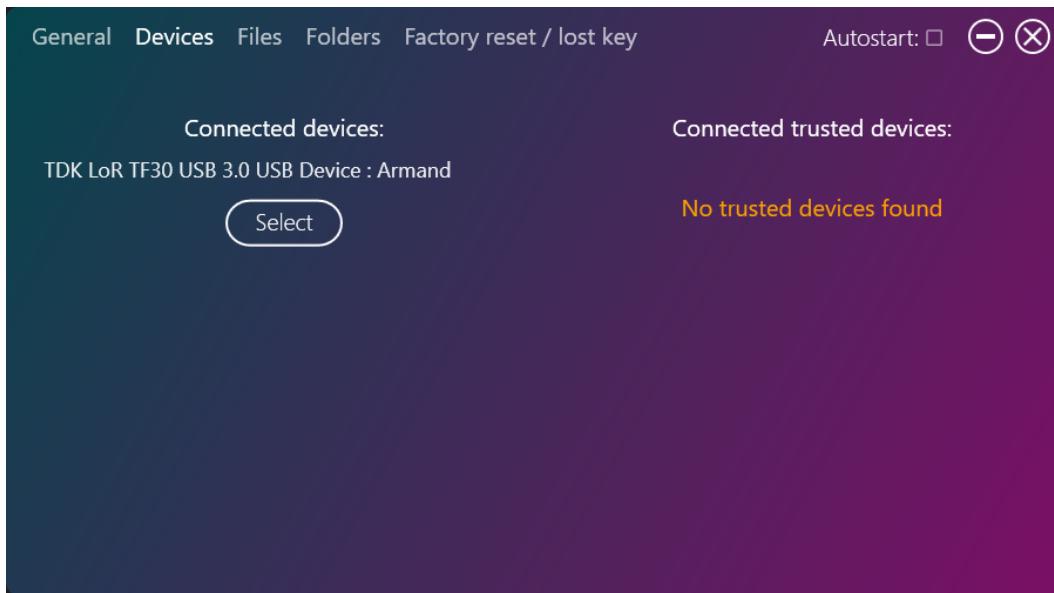
Głównym założeniem przy projektowaniu interfejsu użytkownika było utrzymanie minimalizmu. Użytkownik, który jest przytłoczony ilością dostępnych opcji potrafi się zniechęcić i wybrać inny produkt, który będzie schludny, prosty w obsłudze i przyjemny dla oka. Dokładny opis działania programu został opisany w dokumentacji użytkownika 3.5. Program główny został podzielony na 5 widoków, gdzie każdy widok jest oddzielnym zbiorem elementów interakcji z użytkownikiem. Są to:

- Widok informacji ogólnych:



Rysunek 3.2: Widok informacji ogólnych zawierający dane dotyczące statusu autoryzacji, chronionych plików oraz liczby kluczy. Status autoryzacji odnosi się do zalogowanego użytkownika. Informuje on o posiadanym kluczu sprzętowym. Jeżeli użytkownik posiada klucz sprzętowy i jest on podpięty do komputera, to aplikacja autoryzuje użytkownika, przyznając mu dostęp do chronionych plików. [Źródło własne].

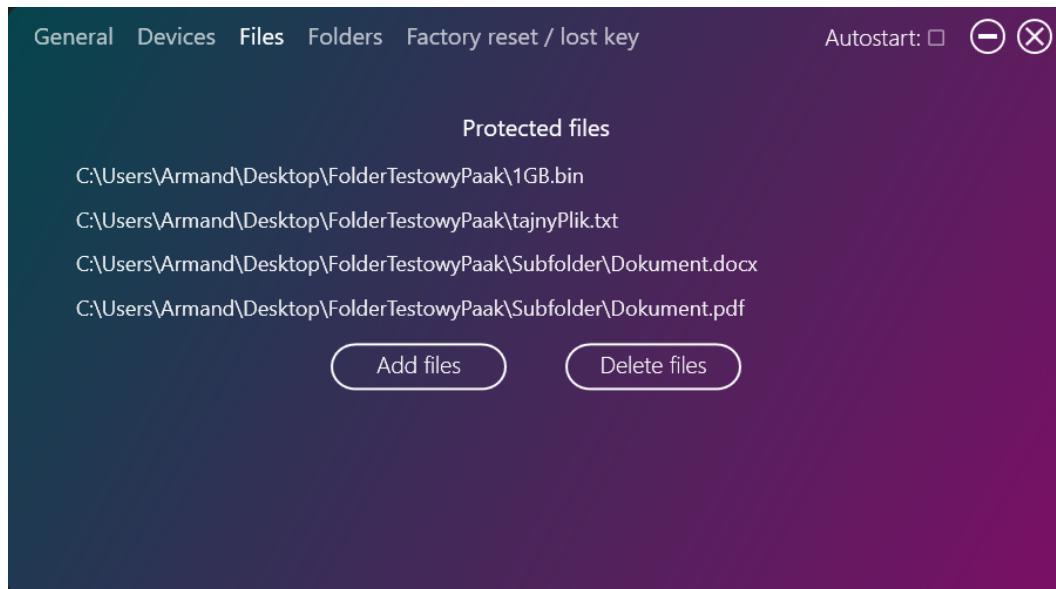
- Widok urządzeń:



Rysunek 3.3: Widok urządzeń zawierający listę urządzeń podłączonych do komputera oraz listę podpiętych kluczy sprzętowych [Źródło własne].

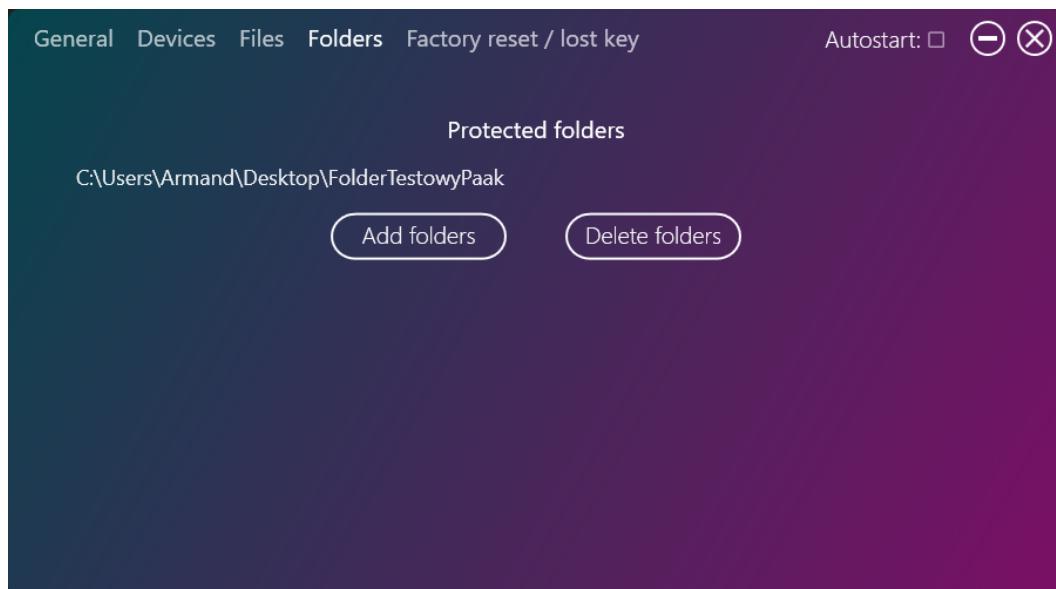
Lista urządzeń jest stale aktualizowana by w czasie rzeczywistym wyświetlać realny status podłączonych urządzeń. W razie odpięcia urządzenia, zostaje ono natychmiast usunięte z listy. Odpowiada za to wyspecjalizowane zadanie bazujące na wątkach, które obsługuje takie zdarzenia.

- Widok plików:



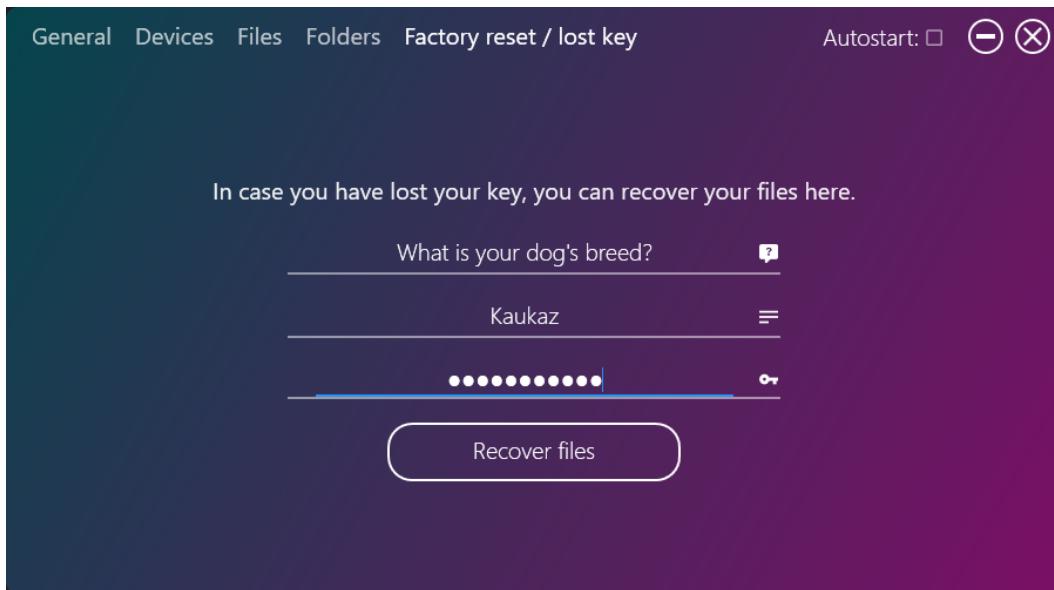
Rysunek 3.4: Widok plików zawiera listę plików, które mają być chronione przez program [Źródło własne].

- Widok Folderów:



Rysunek 3.5: Widok folderów zawiera listę folderów, które mają być chronione przez program [Źródło własne].

- Widok odzyskiwania danych:



Rysunek 3.6: Widok odzyskiwania danych zawiera formularz z pytaniem bezpieczeństwa i miejscem na udzielenie odpowiedzi na to pytanie oraz miejscem na podanie hasła. Poniżej znajduje się przycisk zatwierdzający operację odzyskiwania danych w sytuacji chęci resetu aplikacji lub utraty klucza sprzętowego [Źródło własne].

Odczytywanie listy urządzeń podłączonych do komputera

Jedną z podstawowych funkcji programu jest ciągłe sprawdzanie czy użytkownik podłączył do komputera urządzenie pamięci masowej. Funkcja ta wykorzystuje zadanie, które wykonuje operacje asynchronicznie względem głównego wątku aplikacji. Dzięki takiemu zabiegowi, główny interfejs aplikacji może nieustannie przyjmować polecenia użytkownika. Poniżej znajduje się fragment kodu źródłowego aplikacji, który odpowiada za utworzenie zadania nasłuchującego na zdarzenie podpięcia urządzenia pamięci masowej do komputera.

```

1 var loopTask = Task.Run(() =>
2 {
3     while(true)
4     {
5         Task.Delay(1000);
6         UpdateConnectedDevices();
7         if(DeviceListChanged)
8         {
9             if (ConnectedTrustedDevices != null)
10            {
11                if (ConnectedTrustedDevices.Count.Equals(1))
12                {
13                    AuthorizationStatus = true;
14                    if (RsaPrivateKey.Equals(string.Empty))
15                    {

```

```

16         RsaPrivateKey =
17             IOClass.ReadPrivateKeyFromDeviceToString(
18                 ConnectedTrustedDevices[0].Path);
19     }
20 }
21 else
22 {
23     AuthorizationStatus = false;
24 }
25 }
26
27     uiSyncContext.Post((s) =>
28     {
29         UpdateDevicesStatus();
30     }, null);
31 }
32 }
33 });

```

Szyfrowanie i deszyfrowanie danych

Głównym celem aplikacji jest ochrona danych użytkownika. Takie wymaganie zostało zrealizowane z użyciem szyfrowania algorytmami **RSA** i **AES**, które zostały opisane w sekcjach 2.3.1 i 2.3.2. Implementacja algorytmów szyfrowania oparta została o przestrzeń nazw **System.Security.Cryptography**, z której zostały wykorzystane klasy:

- **RNGCryptoServiceProvider**,
- **RijndaelManaged**,
- **Rfc2898DeriveBytes**,
- **CryptoStream**,
- **RSACryptoServiceProvider**.

RNGCryptoServiceProvider

Klasa **RNGCryptoServiceProvider** implementuje kryptograficzny generator liczb losowych. W aplikacji **PAAK** została ona wykorzystana do wygenerowania soli używanej w algorytmie szyfrującym **AES**.

```

1 public static byte[] GenerateSalt()
2 {
3     byte[] data = new byte[32];
4     using (RNGCryptoServiceProvider rgnCryptoServiceProvider =
5         new RNGCryptoServiceProvider())

```

```
5     {
6         rgnCryptoServiceProvider.GetBytes(data);
7     }
8     return data;
9 }
```

RijndaelManaged

Klasa **RijndaelManaged** zawiera implementację algorytmu symetrycznego **Rijndael**, który jest szerzej znany pod nazwą **AES**. W aplikacji **PAAK** algorytm ten wykorzystywany jest do szyfrowania i deszyfrowania plików użytkownika. Poniżej znajduje się listing metody szyfrującej pliki użytkownika.

```
1 public static string FileEncrypt(string inputFile, string
2     password)
3 {
4     byte[] salt = GenerateSalt();
5     byte[] passwords = Encoding.UTF8.GetBytes(password);
6     RijndaelManaged AES = new RijndaelManaged
7     {
8         KeySize = 256,
9         BlockSize = 128,
10        Padding = PaddingMode.PKCS7
11    };
12    var key = new Rfc2898DeriveBytes(passwords, salt, 50000);
13    AES.Key = key.GetBytes(AES.KeySize / 8);
14    AES.IV = key.GetBytes(AES.BlockSize / 8);
15    AES.Mode = CipherMode.CBC;
16    using (FileStream fsCrypt = new FileStream(inputFile +
17        ".aes", FileMode.Create))
18    {
19        fsCrypt.Write(salt, 0, salt.Length);
20        using (CryptoStream cs = new CryptoStream(fsCrypt,
21            AES.CreateEncryptor(), CryptoStreamMode.Write))
22        {
23            using (FileStream fsIn = new FileStream(inputFile,
24                FileMode.Open))
25            {
26                byte[] buffer = new byte[1048576];
27                int read;
28                while ((read = fsIn.Read(buffer, 0,
29                    buffer.Length)) > 0)
30                {
31                    cs.Write(buffer, 0, read);
32                }
33            }
34        }
35    }
36 }
```

```

30     }
31     File.Delete(inputFile);
32     return inputFile + ".aes";
33 }
```

Metoda tworzy obiekt klasy **RijndaelManaged** o nazwie **AES**. Przypisane zostają do niego następujące parametry:

- wielkość klucza o wartości 256 bitów,
- wielkość bloku szyfrowanych danych o wartości 128 bitów (jest to standardowa wielkość bloku dla algorytmu **AES**),
- klucz oraz wektor inicjalizujący dla algorytmu zostaje podany z użyciem obiektu klasy **Rfc2898DeriveBytes**, która implementuje funkcję wyprowadzania klucza,
- tryb dopełniania bloków w standardzie **PKCS7** (opisany w sekcji 2.3.2), który zdefiniowany jest w **RFC 5652** [Housley(2009)],
- tryb pracy szyfru blokowego polegający na wiązaniu bloków zaszyfrowanych **CBC**. W tym trybie każdy zaszyfrowany blok jest zależny od poprzedniego.

CryptoStream

Klasa **CryptoStream** definiuje strumień, który łączy strumienie danych z transformacjami kryptograficznymi. Dla obiektu strumienia podawany jest enkryptor lub dekryptor **AES**, w zależności od kierunku działania algorytmu kryptograficznego. Poniżej znajduje się listing metody deszyfrującej pliki użytkownika.

```

1 public static string FileDecrypt(string inputFileName, string
                                  password)
2 {
3     byte[] passwords = Encoding.UTF8.GetBytes(password);
4     byte[] salt = new byte[32];
5     string outputFileName = string.Empty;
6     using (FileStream fsCrypt = new FileStream(inputFileName,
9         FileMode.Open))
7     {
8         fsCrypt.Read(salt, 0, salt.Length);
9         RijndaelManaged AES = new RijndaelManaged
10        {
11            KeySize = 256,
12            BlockSize = 128
13        };
14        var key = new Rfc2898DeriveBytes(passwords, salt, 50000);
15        AES.Key = key.GetBytes(AES.KeySize / 8);
16        AES.IV = key.GetBytes(AES.BlockSize / 8);
17        AES.Padding = PaddingMode.PKCS7;
18        AES.Mode = CipherMode.CBC;
```

```

19     outputFileName = inputFileName.Substring(0,
20         inputFileName.LastIndexOf("."));
21     using (CryptoStream cryptoStream = new
22         CryptoStream(fsCrypt, AES.CreateDecryptor(),
23         CryptoStreamMode.Read))
24     {
25         using (FileStream fsOut = new FileStream(outputFileName,
26             FileMode.Create))
27         {
28             int read;
29             byte[] buffer = new byte[1048576];
30             while ((read = cryptoStream.Read(buffer, 0,
31                 buffer.Length)) > 0)
32             {
33                 fsOut.Write(buffer, 0, read);
34             }
35         }
36     }
37     File.Delete(inputFileName);
38     return outputFileName;
39 }
```

RSACryptoServiceProvider

Klasa **RSACryptoServiceProvider** zapewnia implementację asymetrycznego algorytmu **RSA**. W aplikacji **PAAK** algorytm ten został wykorzystany do szyfrowania oraz deszyfrowania klucza algorytmu **AES**. Pierwszym etapem użycia obiektu klasy **RSACryptoServiceProvider** jest wygenerowanie pary kluczy. W konstruktorze obiektu podana została długość klucza. Poniższy listing prezentuje omawianą metodę zwracającą parę kluczy dla algorytmu **RSA**.

```

1 public static (string, string) GenerateRsaKeys()
2 {
3     var cryptoServiceProvider = new
4         RSACryptoServiceProvider(2048);
5     var privateKey =
6         cryptoServiceProvider.ExportParameters(true);
7     var publicKey =
8         cryptoServiceProvider.ExportParameters(false);
9     string publicKeyString = GetKeyString(publicKey);
10    string privateKeyString = GetKeyString(privateKey);
11    return (publicKeyString, privateKeyString);
12 }
```

Algorytmy kryptograficzne wykorzystane w aplikacji PAAK

Wykorzystanie kryptografii zostanie opisane iteracyjnie względem kolejno wykonywanych operacji przez aplikację, począwszy od utworzenia klucza sprzętowego z wybranego przez użytkownika podpiętego urządzenia pamięci masowej USB. Klucz algorytmu **AES**, podlegający szyfrowaniu, został wygenerowany w momencie rejestracji konta użytkownika.

- Tworzenie klucza sprzętowego:
 1. Wygenerowanie pary kluczy dla algorytmu **RSA**.
 2. Zaszyfrowanie klucza **AES** algorytmem **RSA** z wykorzystaniem klucza publicznego.
 3. Zapisanie klucza publicznego **RSA** oraz zaszyfrowanego klucza **AES** w danych profilu użytkownika.
 4. Zapisanie klucza prywatnego **RSA** na wybranym urządzeniu pamięci masowej w postaci ukrytego pliku i w pamięci podręcznej programu.
- Wyjęcie klucza sprzętowego z komputera:
 1. Utworzenie nowego zadania asynchronicznego, które odpowiada za proces szyfrowania danych.
 2. Odszyfrowanie klucza **AES** z użyciem prywatnego klucza **RSA**.
 3. Dla każdego pliku, wybranego przez użytkownika, uruchamiana jest metoda szyfrująca algorytmem **AES** z wykorzystaniem odszyfrowanego klucza.
 4. Wyczyszczenie sektora pamięci odpowiedzialnego za przechowywanie klucza **AES** w jawnej postaci.
 5. Wyczyszczenie sektora pamięci zawierającego klucz prywatny **RSA**.
- Włożenie klucza sprzętowego do komputera:
 1. Pobranie klucza prywatnego **RSA** z podłączonego klucza sprzętowego.
 2. Utworzenie nowego zadania asynchronicznego, które odpowiada za proces deszyfrowania danych.
 3. Odszyfrowanie klucza **AES** z użyciem prywatnego klucza **RSA**.
 4. Dla każdego zaszyfrowanego pliku, który jest powiązany z kluczem oraz użytkownikiem, uruchamiana jest metoda deszyfrująca algorytmu **AES** z wykorzystaniem odszyfrowanego klucza.
 5. Wyczyszczenie sektora pamięci odpowiedzialnego za przechowywanie klucza **AES** w jawnej postaci.

Wykorzystane technologie i narzędzia

Do stworzenia aplikacji **PAAK** wykorzystano następujące narzędzia i technologie:

- **C# WPF .NET**,
- **Visual Studio 2019 Community**,
- **Material Design Themes v3.2.0** —James Willock,
- **MahApps.Metro.IconPacks.Material v4.5.0** —Jan Karger,
- **Newtonsoft.Json v12.0.3** — James Newton-King,
- **System.Linq v4.3.0** — Microsoft,
- **System.Management v4.7.0** — Microsoft.

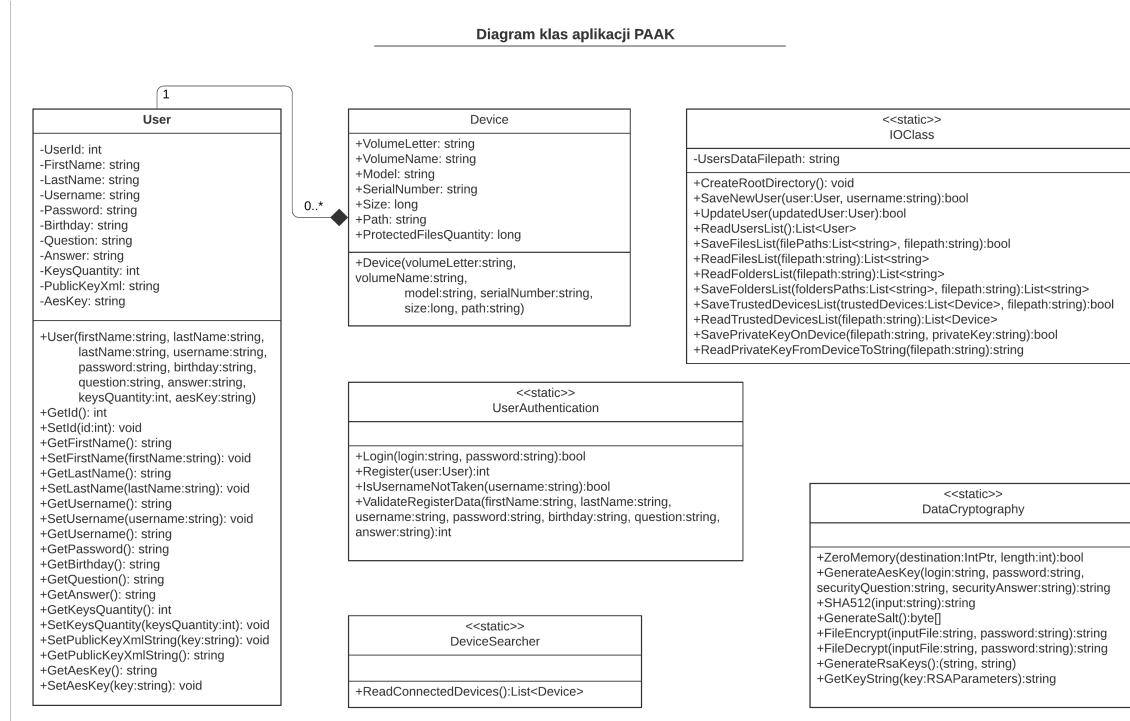
3.5 Dokumentacja techniczna i użytkownika

Rozdział ten zawiera dokumentację techniczną, która posiada diagram klas, instrukcje pozwalające na uruchomienie aplikacji, oraz dokumentację użytkownika, która zawiera informacje dotyczące korzystania z aplikacji **PAAK**.

Dokumentacja techniczna

Dokumentacja techniczna zawiera diagramem klas oraz instrukcje uruchomienia aplikacji i opis przeprowadzanych testów jednostkowych.

Diagram klas został przedstawiony na rysunku 3.7 zamieszczonym poniżej:

Rysunek 3.7: Diagram klas aplikacji **PAAK** [Źródło własne].

Proces instalacji został przedstawiony w punktach, które należy wykonać, by uruchomić aplikację **PAAK**.

1. Aplikacja **PAAK** wymaga zainstalowanej biblioteki **Microsoft .NET Framework 4.0**. Bibliotekę można pobrać ze strony **Microsoft** pod adresem <https://www.microsoft.com/pl-pl/download/details.aspx?id=17851>.
2. Aplikację **PAAK** należy pobrać z repozytorium **GitHub**, które znajduje się pod adresem https://github.com/Armand98/paak/tree/master/Paak_portable.
3. Pobrała aplikacja w postaci pliku wykonywalnego jest samodzielnie działającą aplikacją, której nie trzeba instalować i wystarczy uruchomić plik **paak.exe**.

Wykonanie powyższych kroków powinno skutkować uruchomieniem aplikacji **PAAK**.

Testy jednostkowe aplikacji **PAAK**

Aplikacja autora została przetestowana z użyciem testów jednostkowych dla najważniejszych metod aplikacji, które odpowiadają za szyfrowanie i deszyfrowanie klucza **AES**, jak i validację danych wejściowych do programu. Wyniki testów zostały przedstawione na rysunku 3.8.

Test	Czas trwania	Cechy
TestPaak.UnitTesting (10)	89 ms	
UnitTestPaak (10)	89 ms	
EncryptionAndDecryptionTests (2)	89 ms	
RsaEncryptedTextCanBeDecrypted	87 ms	
SHA512GivesTheSameHashForTheSameInput	2 ms	
UserAuthorizationTests (6)	< 1 ms	
RegisterDataValidationCatchesEmptyParameter	< 1 ms	
RegisterDataValidationCatchesPasswordWithWhiteSpaces	< 1 ms	
RegisterDataValidationCatchesTooWeakPassword	< 1 ms	
RegisterDataValidationCatchesUsernameIsTooShort	< 1 ms	
RegisterDataValidationCatchesWrongDateFormat	< 1 ms	
RegisterDataValidationWorksFineWithExpectedParameters	< 1 ms	
UserTests (2)	< 1 ms	
UserObjectCreatesWithParameters_GetMethodsWorksFine	< 1 ms	
UserObjectCreatesWithParameters_SerMethodsWorksFine	< 1 ms	

Rysunek 3.8: Wyniki wykonanych testów jednostkowych aplikacji **PAAK** [Źródło własne].

Kody źródłowe wszystkich testów jednostkowych są dostępne w repozytorium **GitHub** aplikacji **PAAK** pod adresem https://github.com/Armand98/paak_unitTests.

Podczas testów wykryto jednak nietypowe zachowania metody odpowiedzialnej za odczytywanie urządzeń pamięci masowej. Zdarzają się nieobsłużone wyjątki, wywoływane z obiektu klasy **ManagementObjectSearcher**, który odpowiada za zapytania systemowe odnośnie podpiętych urządzeń pamięci masowej. Błąd ten został przeanalizowany i autor nie był w stanie stwierdzić przyczyny występowania nieobsłużonych wyjątków, które pojawiają się niedeterministycznie i nie są uzależnione od konkretnej sytuacji, która wystąpi w programie. Zostały przeszukane liczne fora internetowe w poszukiwaniu rozwiązania problemu, lecz żadne zaproponowane rozwiązanie nie rozwiązywało problemu autora.

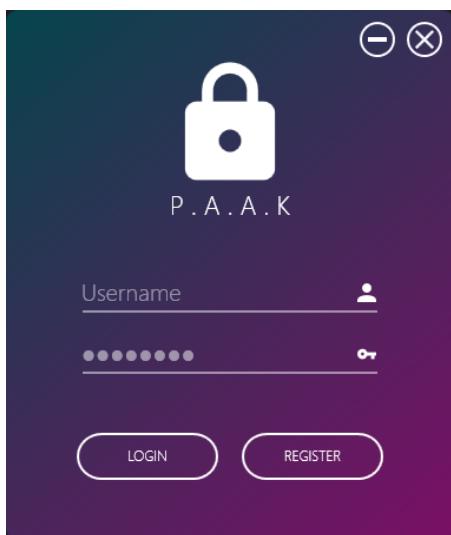
Wyjątek ten nie stanowi przeszkody w korzystaniu z aplikacji i nie powoduje uszkodzenia plików użytkownika. W momencie wystąpienia nieobsłużonego wyjątku, aplikacja wyłącza się, lecz można ją ponownie uruchomić i dalej z niej korzystać. Autor będzie analizował zaistniały problem i poszukiwał rozwiązania, które wyeliminuje tak niedeterministyczne zachowanie aplikacji **PAAK**.

Dokumentacja użytkownika

Opis działania aplikacji został podzielony na czynności, które użytkownik może wykonać.

Logowanie i rejestracja

Po uruchomieniu programu ukaże się panel logowania. Można z jego poziomu zalogować się istniejącym kontem lub przejść do panelu rejestracji, pozwala utworzyć nowe konto. Podczas rejestracji, użytkownik wypełnia formularz oraz wybiera jedno z trzech pytań bezpieczeństwa i odpowiada na nie. Zapamiętanie odpowiedzi na pytanie bezpieczeństwa będzie jedyną możliwością odzyskania danych w razie utraty klucza sprzętowego. Nie istnieje również możliwość resetu hasła ze względów bezpieczeństwa.



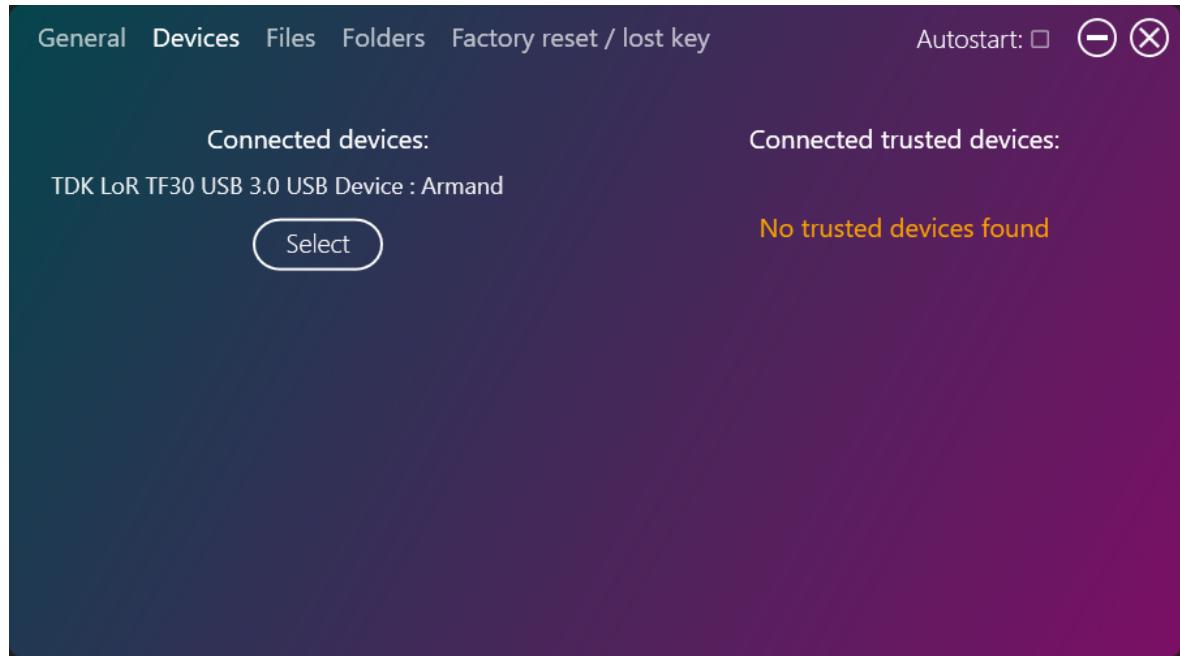
Rysunek 3.9: Panel logowania użytkownika
[Źródło własne].

The image shows the registration screen. It has a "Register Form" title at the top. It includes fields for "First name", "Last name", and "Username", each with a user icon. Below these are fields for a password (represented by a series of dots) and a birthday (with a calendar icon). A dropdown menu asks "What is your dog's breed?", followed by an "Answer" field with a list icon. In the bottom right corner is a button labeled "CREATE AN ACCOUNT". The top right corner has close and minimize buttons.

Rysunek 3.10: Panel rejestracji użytkownika
[Źródło własne].

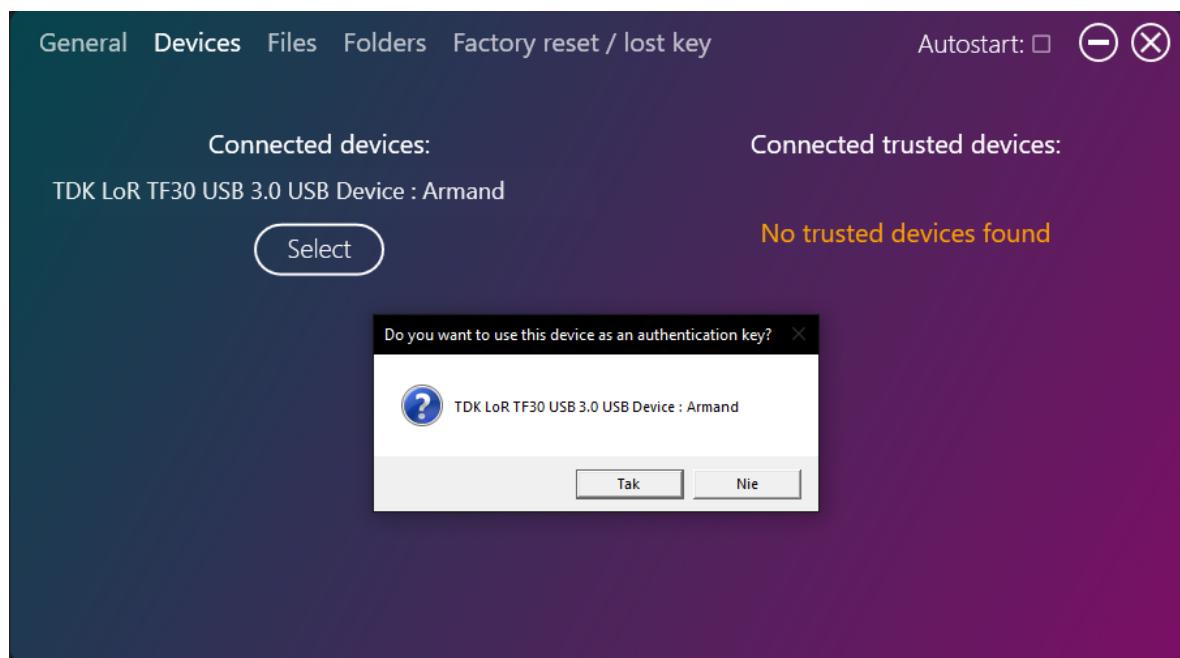
Tworzenie klucza sprzętowego

Po wpięciu urządzenia pamięci masowej do komputera oraz po zalogowaniu, użytkownik zobaczy swoje urządzenie w zakładce **Devices**.



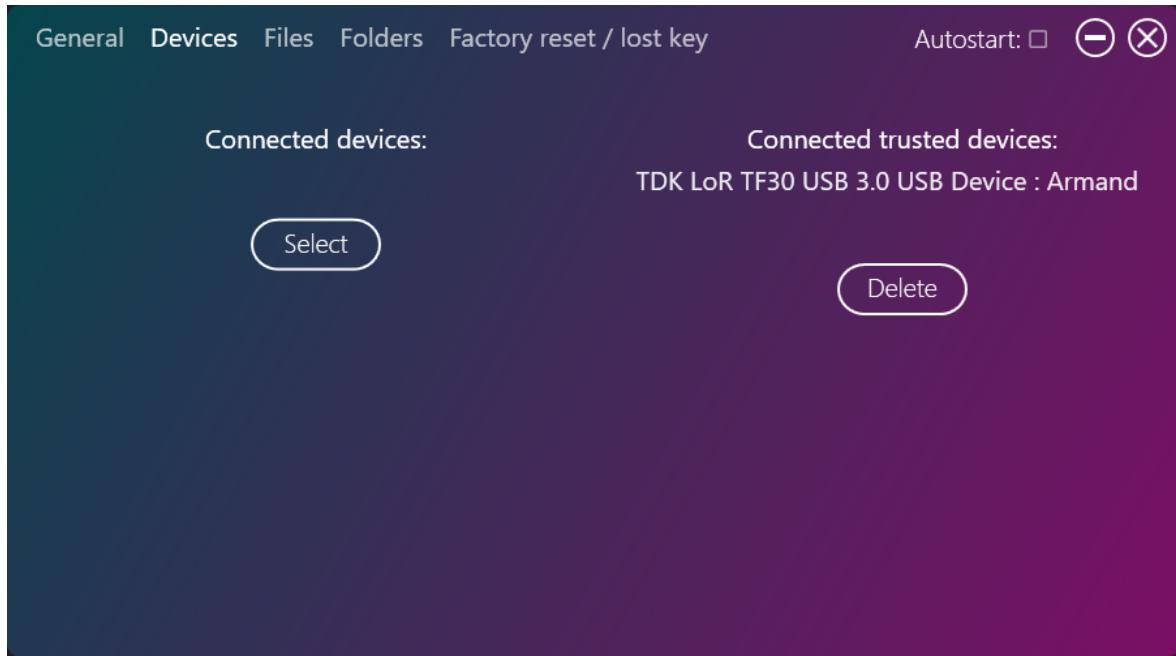
Rysunek 3.11: Lista podłączonych urządzeń pamięci masowej oraz pusta lista kluczy sprzętowych [Źródło własne].

Następnie może je wybrać i zapisać jako zaufane urządzenie, które posłuży za sprzętowy klucz dostępu do jego danych.



Rysunek 3.12: Zatwierdzanie dodawania zaufanego urządzenia [Źródło własne].

Po zatwierdzeniu, wybrane urządzenie będzie rozpoznawane jako zaufany klucz sprzętowy.



Rysunek 3.13: Klucz sprzętowy został utworzony pomyślnie [Źródło własne].

Od tego momentu wybrane urządzenie posiada ukryty plik, który zawiera informacje o kluczu prywatnym **RSA**. Zawartość pliku **PAAK_PrivateKey.xml**:

```

1  <?xml version="1.0" encoding="utf-16"?>
2  <RSAParameters
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
5          <D>XIFFfUvEFHh1nNWpYNhsFe194dMJyAkmdV6APhSHqcN3B9tT2Q5L
6              1NZv+9osVyNhEHLyo iYNE0QHywrFwIdM1EMysZrHpd07MaikI+/NAT
7              DMCB+m2yBgkxz8en8oHcAR3UDaITFXOMZf36T2D6w2ZZGknzwHykNU
8              uuPkzNkBksADr5Q6+LBmE4soEkfZazz0iY424X7On1ImLC10hl5aW7
9              6GevWvNVMA1KftA+lzglazFNis2VzzNhJMe pDuz5gpz+nLC1GSUG1o
10             vIhvXXScpnZ5FfKR9w2Iu0H6Z1/lsZxT5+PxnxJEs79BnS5vajREi/
11             9oZ7iPGN/AI1kWKrXoeOQ==</D>
12             <DP>SiMfAjN5bs+hKMG+ZL1c9CykUbbz9r8I+kfnNhT31IG6usDi9V
13             uBs0iewsVe8UTSxYhnx/QbK/IEf9QNXhbZTcsR44x82lr8wFpfG/5
14             yut8qPLhexGsylc8oSbg3rqpIhmizZjm/TFTQMdvCA9ZPEp7Z9UceQ
15             LTozzJN/OaeOM=</DP>
16             <DQ>z4C1N7Vxom28n0Pyh6Vg5NPWhg23bogk/w2hrJCix6HNkbWKft
17             hJ3aydEhScxfm63pKm95Ga49DirwRBZT5PSN6zv9mPZTS6xQemnC7u
18             9lwoltTAsdvPrcJW8oIC13ynagfd3QDFSq7ix5ARZ3dPYfJIL7TSDg
19             +lzc1LYIsKYyM=</DQ>
20             <Exponent>AQAB</Exponent>
21             <InverseQ>XaHYvtaPKs9GJs0SEE1EHBUuu/gpvkpd2+7kV7Eg6aP+
22             AuRTj+1R2y1PluYbZ+0uTv37Kx48HnUbRKIwf b4qVLjkMXODmrPce2
23             Tpiyqefg+6EI3rVm3cfuTZgvi7D966+1vw9L4UzsJecCroqVZ2TYqs
XlhXfER46IFFGSelq+I=</InverseQ>
24             <Modulus>1/qQGYoPWaxTkL/dRmGwtrz3TTNMV3C0bhwxAPrVi00zI

```

```

24      eYJPACXQGq1uy40h+hVQ3DTpEAckQdZQTXQ3F78AWbwnimyyFC6bDa
25      tEj1aVpn3htjKQ0vVBU6TxX3eM5UIq5VWI0JTbL/A+SLwvPs+Tu4Sb
26      ItxGIfZKxBvn975yKUWTHAPN5gSPj1a9b7UC4VsXM3i9WZaiMpdh2j
27      BezOpejVO/DNnDF3eVelawXEqqHM+eFvIp90bNU6/3uL7761KXBfS
28      ORCJY6IIBaHt4QiNLc7p2FtE0R31YIKYv3ENgBtcrPuqBwZ8ecPF7r
29      orN3pKcM9U4Hr6wbtgBJlx4w4MQ==</Modulus>
30      <P>91LKnF7kybLu/YHCsjDmGht+qU61zFvU+wPFJ86mytF9AHMC5bm
31      Dx861Wl71DVNpvXY20ogHX1ekya1bGx6Mrk7ZvGoUWF1jJeA1jBHIk
32      VLIuvzTaxmoFYtqBqofMfKuriIUD0ZXRYfPU/aJEY88qeUr/oLFkujS
33      sy67P+EaGb+s=</P>
34      <Q>4Haafksos7FHAFGt6AA8Lgb8ASFuhnvkW4KU8XJTrd07947Gtdhn
35      WZ4JUSuY5Wb0Hp1KCPyxU0dGj60QCDSx71ijLmA70DS8Z5801oYznM
36      HVLaFywGEGenA4ZYbn5CMhZpFi3rSQomJLvJYEuThB3THmvg2MBhUq
37      IT7BDDrgVDVM=</Q>
38      </RSAParameters>

```

W tym samym momencie, klucz publiczny **RSA** został przypisany do lokalnych danych użytkownika. Zawartość pliku **users.json**:

```

1  [
2    {
3      "UserID": 1,
4      "FirstName": "A23781931BF0751BDE254F32084AF330D320D73854D7
5      BE897087030778D70C3FAE0DFD31F3F9BFC9B08BD7E6FD9890A93B7617
6      0DFBC14275855563CF2A2BDA5",
7      "LastName": "A89E8F336CC4906A4BB628E57A587DF8D7163FC7E6196
8      9BDB17B081C3079C469B33B140A1357BA55EE50C2D76BD9786AFE61C20
9      A790361C6F4E3DDC1ADD25089",
10     "Username": "A23781931BF0751BDE254F32084AF330D320D73854D7B
11     E897087030778D70C3FAE0DFD31F3F9BFC9B08BD7E6FD9890A93B76170
12     DFBC14275855563CF2A2BDA5",
13     "Password": "35C500FF5BD9F15AF3EE407AD750C8EF751A384FA8FDA
14     FB872EE20108C0D6D230C899CA2E7C1E063FC47CA44D10699B89FD4BE8
15     57A7CCCA5D7976E92704FOOCF",
16     "Birthday": "86EF8C3C1A29AE13E2B323D9C5A6D0B2F2A2758ED7073
17     272DC79B7A5029486A6C057FE1C7F32F02BECEDA8DCF6F679A56C79A7C
18     275A968966615B2970DB89EF8",
19     "Question": "What is your dog's breed?",
20     "Answer": "B1A848386BB4515C5720AE4FD4BA142DA522671AE1639C8
21     01FCE8FF6AE6CA4E1B00D3D829EB10B05227FBC03CE55317B266C9D9E8
22     0B140C4EA7CAB983025B2AC",
23     "KeysQuantity": 1,
24     "PublicKeyXml": "<?xml version=\"1.0\""
                      encoding="utf-16"?>\r\n<RSAParameters
                        xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
                        xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\">\r\n
                        <Exponent>AQAB</Exponent>\r\n
                        <Modulus>1/qGKYoPWaxTkL/dRmgwtrz3TTNMV3C0bhwxAPrVi00zIeYJP

```

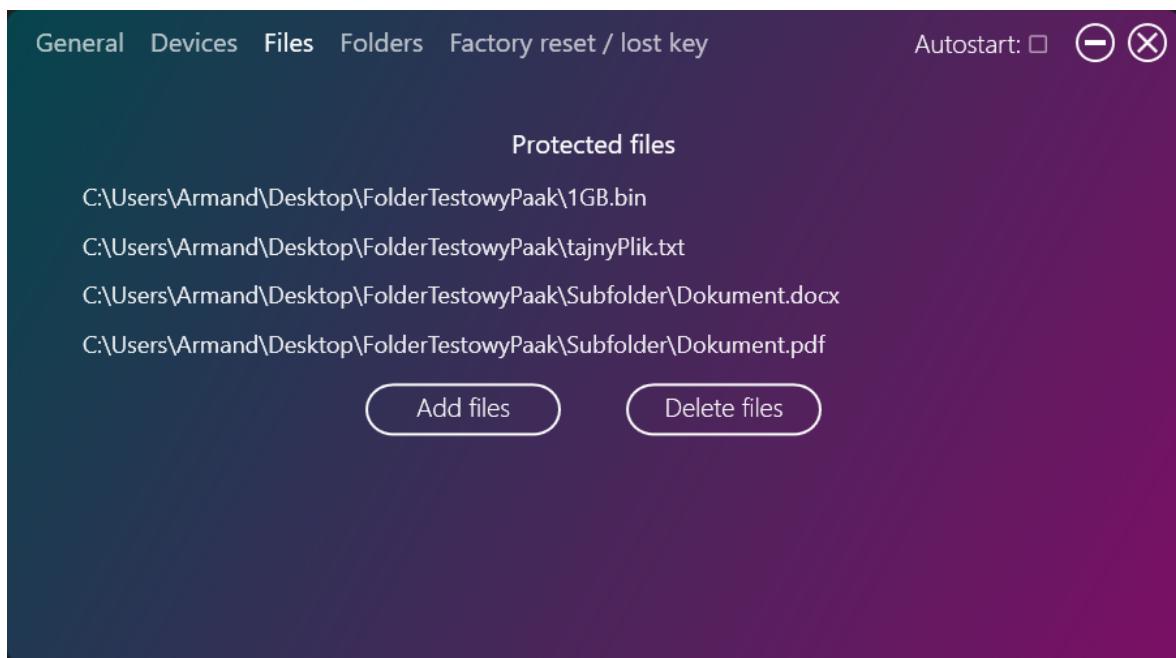
```

25      ACXQGq1uy40h+hVQ3DTpEAckQdZQTXQ3F78AWbwnimyyFC6bDatEjlaVpn
26      3htjKQ0vVBU6TxX3eM5UIq5VWI0JTbL/A+SLwvPs+Tu4SbItxGIfZKxBvn
27      975yKUWTHAPN5gSPj1a9b7UC4VsXM3i9WZaiMpdh2jBezOpejVO/DNnDF3
28      eVelawXEqqHM+eFvIp90bNU6/3uL7761KXBfSORCJY6IIBaHt4QiNLc7p
29      2FtE0R31YIKYv3ENgBtcrPuqBwZ8ecPF7rorN3pKcM9U4Hr6wbtgBJlx4w
30      4MQ==</Modulus>\r\n</RSAParameters>,
31      "AesKey": "Ajkc10VifmouGaG04W1FC+YU8hkuDi3cKhj24rnBV/2qxzC
32      cn1m+rm1Anix56QeXqlkYTHahxZQV/iX08KtCaqZfuAMQ7bIch1vFoPz+e
33      Uaunhcst+50I+UEUL+RUyxjD9v5v8RTGK77wPKx0HMK0IMp+cU6MPSytZ4
34      2Znv/DcMGUvoEcuC6oy9DB+vez7Y/Z/c6peJ2vvude0bNXJgj34gVLu8kY
35      lyM2W4B1HjmzCQqJtNmj6MwuGv+D39QrTFNDMV7cTtpLj92JLTsZWk0iT3
36      xzqBYen2z2TdJ3IJ23gXNyNqplhGBG30um/D/igeg7Pg9E/PtEkVfhqrcD
37      +JVdQ=="
38  }
39 ]

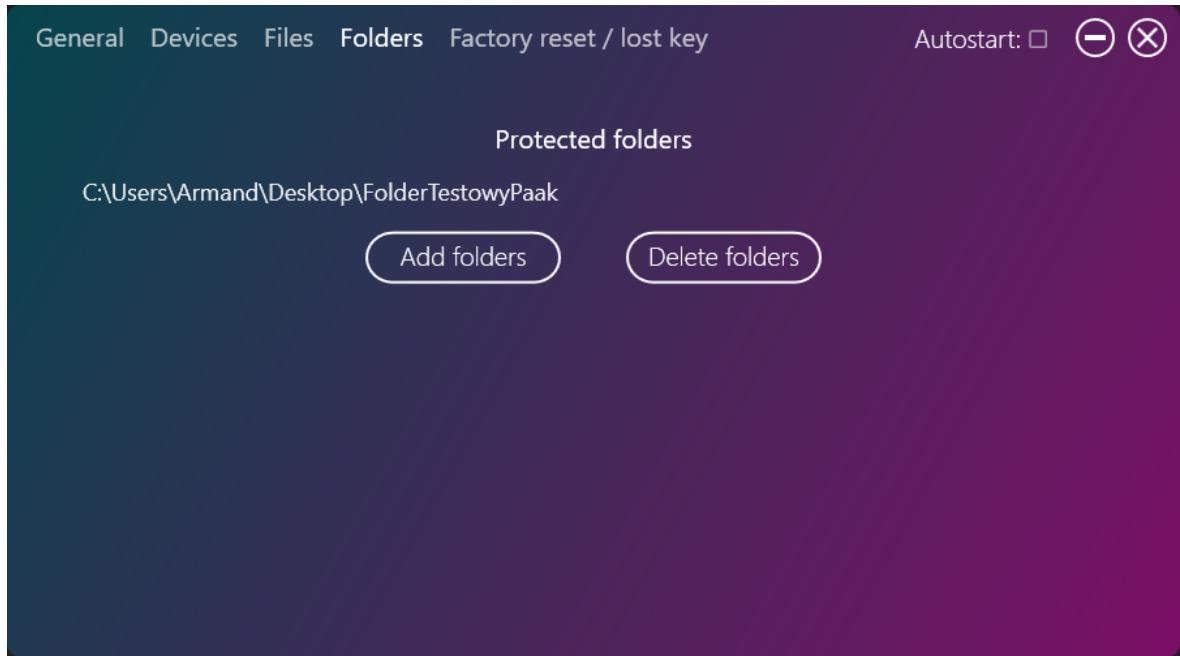
```

Wybór i szyfrowanie plików

W zakładkach **Files** oraz **Folders** użytkownik może dodawać pliki i foldery w celu zapewnienia im ochrony. Istnieje możliwość dodawania wielu plików i folderów jednocześnie. Wybranie całego folderu jest równoznaczne z wybraniem wszystkich plików znajdujących się w tym folderze, łącznie z wszystkimi podfolderami. W sytuacji, gdy użytkownik chce wybrać większość plików z wybranego folderu, może dodać cały folder w zakładce **Folders**, a następnie przejść do zakładki **Files** i usunąć pliki, które nie wymagają ochrony.

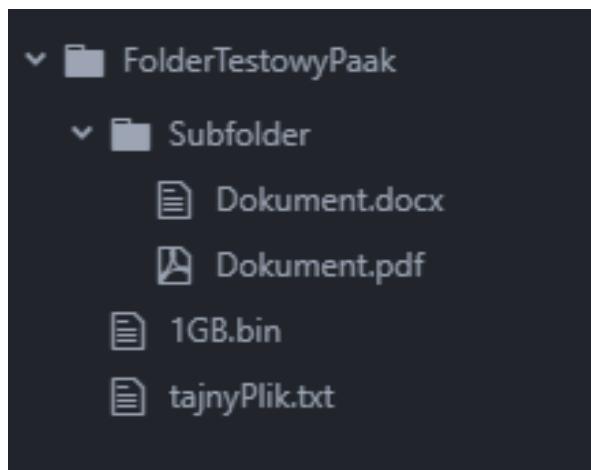


Rysunek 3.14: Wybrane przez użytkownika pliki do ochrony [Źródło własne].



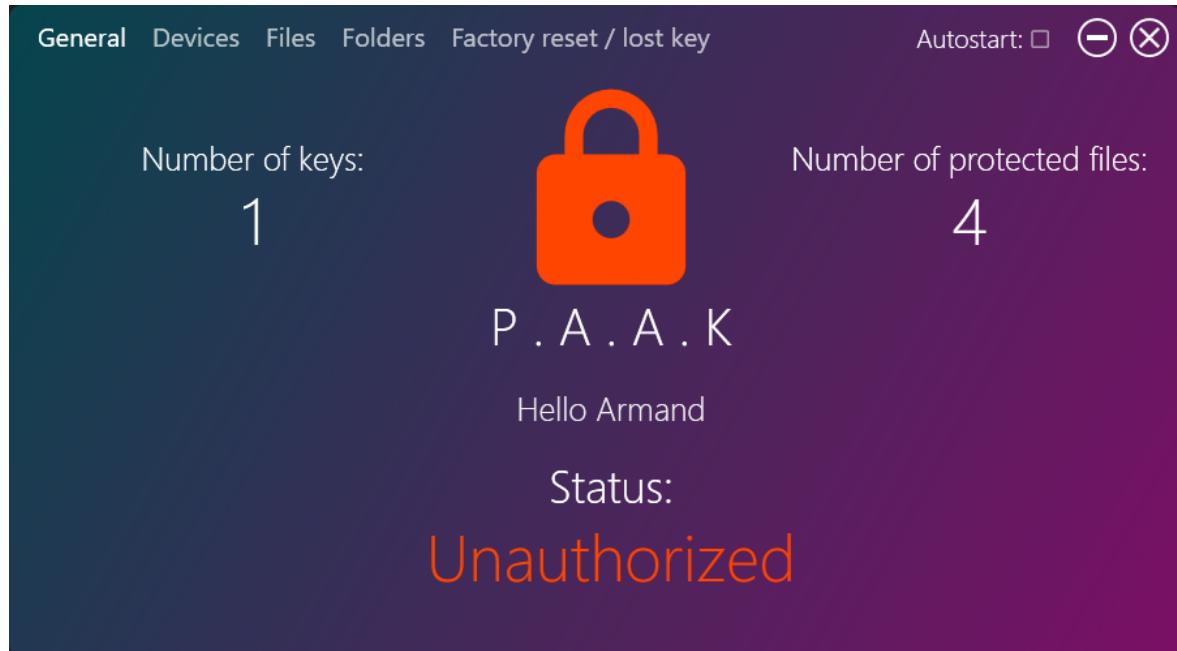
Rysunek 3.15: Wybrane przez użytkownika foldery do ochrony [Źródło własne].

Użytkownik wybrał folder zawierający 4 pliki, w tym jeden plik danych binarnych o rozmiarze 1GB. Strukturę katalogu prezentuje zrzut ekranu 3.16.



Rysunek 3.16: Struktura katalogu wybranego przez użytkownika [Źródło własne].

Po wyciągnięciu klucza sprzętowego z komputera, program natychmiast wykryje brak klucza sprzętowego i wykona szyfrowanie plików, które zostały wybrane. Użytkownik zostanie o tym poinformowany zmianą statusu na ekranie głównym aplikacji, która składa się ze zmiany napisu statusu autoryzacji oraz zamknięciem kłódki, będącej logo aplikacji. Towarzyszy temu również zmiana koloru z zielono-żółtego na czerwono-pomarańczowy.



Rysunek 3.17: Ekran główny z nieobecnym kluczem sprzętowym [Źródło własne].

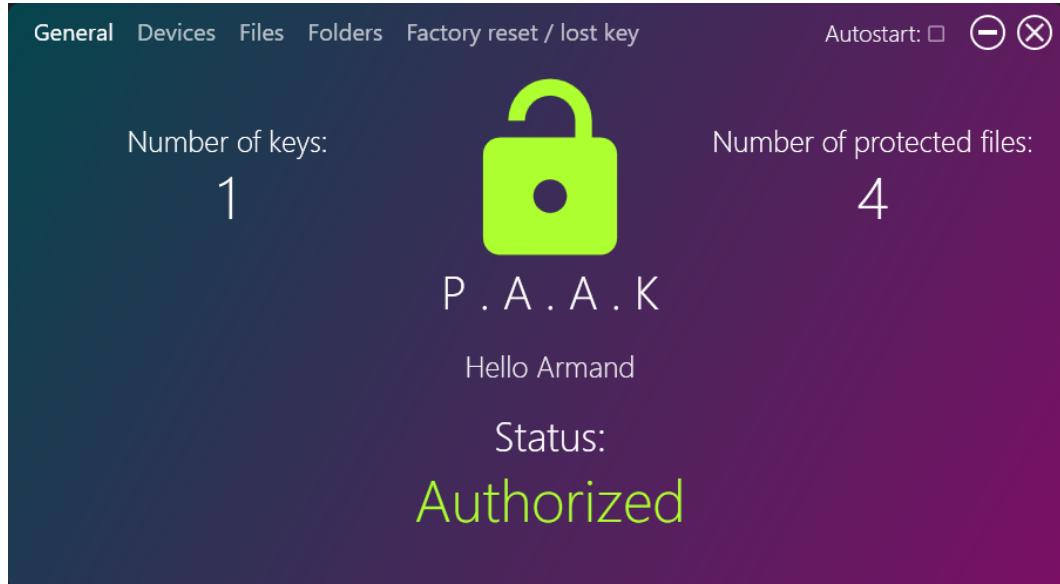
Pliki zostały zaszyfrowane w czasie poniżej 2 sekund. Oznacza to bardzo wydajne działanie algorytmu **AES** przy szyfrowaniu plików o dużym rozmiarze. Zawartość pliku tekstowego po zaszyfrowaniu prezentuje poniższy zrzut ekranu 3.18:

Rysunek 3.18: Zawartość pliku **tajnyPlik.txt.aes** po zaszyfrowaniu [Źródło własne].

Odszyfrowywanie plików

Aby użytkownik odzyskał dostęp do swoich plików musi włożyć ponownie klucz do komputera. Po włożeniu go, program odczytuje jego zawartość i odszyfrowuje pliki. Proces odszyfrowania 4 plików, w tym pliku binarnego o rozmiarze 1GB był

natychmiastowy i nie zajął nawet pełnej sekundy. Proces ten jest widocznie szybszy od procesu szyfrowania. Użytkownik jest o tym poinformowany przez zmianę statusu na ekranie głównym aplikacji. Kolory napisu i logo zmieniają się na zielono-żółty, napis informuje o statusie, a kłódka, będąca logo aplikacji, otwiera się.



Rysunek 3.19: Ekran główny z obecnym kluczem sprzętowym [Źródło własne].

Pliki zostały odszyfrowane, a zawartość jednego z nich prezentuje poniższy zrzut ekranu 3.20:

```
tajnyPlik.txt
1 Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. In fermentum et
2 sollicitudin ac orci. Sapien et ligula ullamcorper malesuada proin libero nunc. Ornare suspendisse sed nisi lacus sed viverra tellus in hac. Porta
3 lorem mollis aliquam ut porttitor leo a. Lorem ipsum dolor sit amet consectetur adipiscing. Accumsan tortor posuere ac ut consequat semper. Platea
4 dictumst vestibulum rhoncus est pellentesque elit ullamcorper dignissim cras. Sit amet consectetur adipiscing elit duis tristique. Mattis nunc sed
5 blandit libero volutpat sed cras. Vitae sapien pellentesque habitant morbi tristique senectus et netus. Augue ut lectus arcu bibendum at varius vel
6 pharetra. Dam vulputate ut pharetra sit amet. In est ante in nibh mauris cursus. Leo vel fringilla est ullamcorper eget nulla facilisi etiam
7 dignissim. Amet aliquam id diam maecenas. Est pellentesque elit ullamcorper dignissim cras tincidunt lobortis feugiat. Risus quis varius quam quisque
8 id. Turpis egestas pretium aenean pharetra magna ac placerat.

2 In fermentum posuere urna nec tincidunt. Maecenas pharetra convallis posuere morbi leo urna molestie. Senectus et netus et malesuada fames ac turpis
3 egestas. At erat pellentesque adipiscing commodo elit at imperdiet duis. Viverra mauris in aliquam sem. Eu lobortis elementum nibh tellus molestie nunc
4 non. Malesuada pellentesque elit eget gravida. Morbi tincidunt ornare massa eget egestas purus. Ipsum suspendisse ultrices gravida dictum fusce ut
5 placerat orci nulla. Proin libero nunc consequat interdum. Fames ac turpis egestas integer eget aliquet nibh praesent.

4 Arcu cursus vitae congue mauris rhoncus aenean vel elit. Nisi scelerisque eu ultrices vitae auctor. Non enim praesent elementum facilisis leo. Nibh
5 cras pulvinar mattis nunc sed blandit libero volutpat sed. Adipiscing elit duis tristique sollicitudin. Nulla facilisi etiam dignissim diam quis enim
6 lobortis. Netus et malesuada fames ac. Egestas purus viverra accumsan in nisl nisi scelerisque eu ultrices. Eu consequat ac felis donec et odio
7 pellentesque diam. Leo in vitae turpis massa sed elementum. Ipsum dolor sit amet consectetur. Consequat id porta nibh venenatis. In cursus turpis
8 massa tincidunt duis ut ornare. Ac placerat vestibulum lectus mauris. Dictum non consectetur a erat nam at lectus urna. Convallis tellus id interdum
9 velit laoreet id donec ultrices tincidunt. Non arcu risus quis varius quam quisque. Adipiscing tristique risus nec feugiat in.

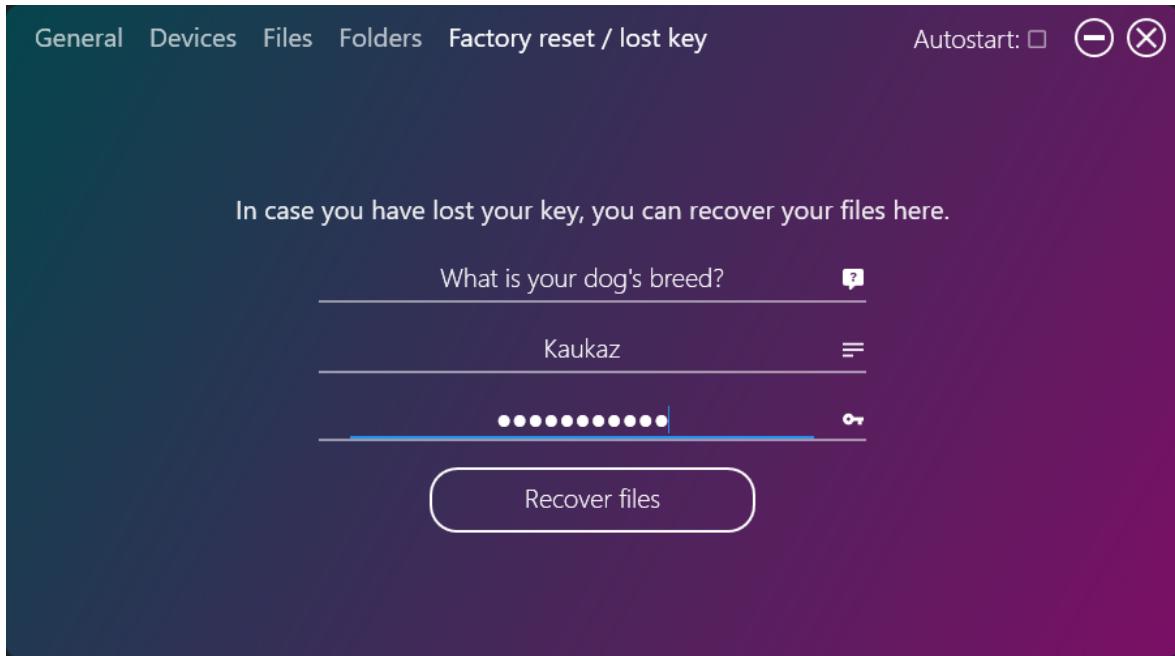
6 Condimentum mattis pellentesque id nibh tortor id. Vel fringilla est ullamcorper eget. Convallis aenean et tortor at risus viverra adipiscing at in.
7 Id velit ut tortor pretium viverra. Venenatis a condimentum vitae sapien pellentesque habitant morbi tristique senectus. Commodo nulla facilisi nullam
8 vehicula ipsum a arcu cursus vitae. Viverra aliquet eget sit amet tellus cras. Elit pellentesque habitant morbi tristique senectus et. Enim neque
9 volutpat ac tincidunt. Eget gravida cum sociis natque penatibus et. Magna fringilla urna porttitor rhoncus dolor purus non enim. Arcu dictum varius
10 duis at consectetur lorem donec massa. Cras sed felis eget velit aliquet sagittis. Viverra vitae congue eu consequat ac felis. Ut faucibus pulvinar
11 elementum integer. Nisl condimentum id venenatis a condimentum. Eu scelerisque felis imperdiet proin fermentum leo vel orci. Quisque id diam vel quam.
12 Diam volutpat commodo sed egestas egestas fringilla phasellus faucibus scelerisque. Amet nisl suscipit adipiscing bibendum est ultricies.
```

Rysunek 3.20: Zawartość pliku **tajnyPlik.txt** po odszyfrowaniu [Źródło własne].

Moduł odzyskiwania danych

Aplikacja **PAAK** posiada moduł odzyskiwania danych. W sytuacji, gdy użytkownik zgubi lub uszkodzi swój klucz sprzętowy, będzie on mógł odszyfrować zabezpieczone

pliki. W celu odzyskania danych należy odpowiedzieć na pytanie bezpieczeństwa, które wybierało się podczas procesu rejestracji oraz odpowiedzieć na wybrane pytanie, udzielając tej samej odpowiedzi jak w momencie rejestracji konta. Użycie tego modułu spowoduje zresetowanie ustawień programu do ustawień fabrycznych. Usunięte zostaną listy z chronionymi plikami oraz wszystkie dane dotyczące zgubionego lub zniszczonego klucza sprzętowego.



Rysunek 3.21: Moduł odzyskiwania danych [Źródło własne].

Moduł odzyskiwania danych wykorzystuje generator klucza **AES** użytkownika, który tworzony jest na podstawie danych użytkownika. Utworzony łańcuch znaków jest następnie haszowany funkcją skrótu **SHA512**.

```
1 public static string SHA512(string input)
2 {
3     var bytes = Encoding.UTF8.GetBytes(input);
4     using (var hash =
5         System.Security.Cryptography.SHA512.Create())
6     {
7         var hashedInputBytes = hash.ComputeHash(bytes);
8         var hashString =
9             BitConverter.ToString(hashedInputBytes).Replace("-", "");
10    return hashString;
11 }
```

Otrzymanym haszem klucza odszyfrowywane są wszystkie zabezpieczone dane użytkownika.

Podsumowanie

Celem praktycznym niniejszej pracy było utworzenie darmowej aplikacji do ochrony plików użytkownika przed nieautoryzowanym dostępem. Aplikacja **PAAK** pozwala na szyfrowanie wybranych przez użytkownika plików. Metoda dostępu do chronionych plików została oparta o klucz sprzętowy, którym może zostać dowolne urządzenie pamięci masowej. Takie rozwiązanie nie wymaga wkładu finansowego użytkownika w zakup prefabrykowanego klucza sprzętowego.

W trakcie tworzenia aplikacji **PAAK**, prototypowanie rozwiązania było kluczowym etapem, który pozwolił na skrupulatne zaplanowanie procesu implementacji rozwiązania. Podczas tego etapu, wybrano język programowania **C#**, który dzięki rozbudowanym bibliotekom, zapewnił bezpieczną implementację kluczowych algorytmów kryptograficznych oraz umożliwił na wykorzystanie operacji wielowątkowych.

Cel praktyczny niniejszej pracy został osiągnięty poprzez implementację oprogramowania w języku **C# WPF**. Aplikacja okienkowa została wykonana przy użyciu środowiska **Visual Studio 2019** wraz z bibliotekami przestrzeni nazw **System.Security.Cryptography**.

Temat pracy pozwolił na pogłębienie wiedzy autora w tematach, takich jak: bezpieczeństwo informacji, procesy uwierzytelniania i autoryzacji w kontrolowanym dostępie do zasobów informacji oraz kryptografii pod kątem historycznym i praktycznym. Dobór algorytmów kryptograficznych oraz ich praktyczne wykorzystanie w aplikacji ukazuje wysokopoziomowe zrozumienie działania użytych algorytmów przez autora, jak i systemu szyfrowania, który stworzył.

Autor zamierza dalej szukać rozwiązań nieobsługiwanego wyjątku, który został opisany w dokumentacji technicznej 3.5 i rozwijać stworzoną aplikację **PAAK** o dodatkowe funkcje takie, jak:

- logowanie za pomocą istniejącego klucza sprzętowego,
- używanie wielu kluczy sprzętowych przez jednego użytkownika,
- dodatkową i opcjonalną funkcję zwiększącą bezpieczeństwo zaszyfrowanych danych, która uniemożliwi ich odzyskanie,
- przenoszenie zaszyfrowanych plików za pomocą klucza sprzętowego,

- używanie tego samego klucza sprzętowego na wielu komputerach, na których uruchomiona jest aplikacja **PAAK**.

Na podstawie efektów końcowych pracy można stwierdzić, iż autorowi udało się zrealizować wszystkie postawione we wstępie cele.

Literatura

[Brabonet(2020)] Brabonet. Strona producenta, 2020. <https://brabonet.com/?q=keylock>, Ostatni dostęp: grudzień 2020.

[Buchanan(2018)] Bill Buchanan. How guessable is your password?, 2018. <https://medium.com/a-security-site-when-bob-met-alice/how-guessable-is-your-password-f0deeb69f985>, Ostatni dostęp: styczeń 2021.

[Dictionary(2021a)] Cambridge Dictionary. Cryptanalysis, 2021a. <https://dictionary.cambridge.org/dictionary/english/cryptanalysis>, Ostatni dostęp: styczeń 2021.

[Dictionary(2021b)] Cambridge Dictionary. Cryptography, 2021b. <https://dictionary.cambridge.org/dictionary/english/cryptography?topic=codes-and-decoding>, Ostatni dostęp: styczeń 2021.

[Harbowski(2014)] Marcin Harbowski. *Podstawy kryptografii Wydanie III*. Helion, 2014.

[Housley(2009)] R. Housley. Cryptographic message syntax, 2009. <https://tools.ietf.org/html/rfc5652>, Ostatni dostęp: styczeń 2021.

[Joan Daemen(1999)] Vincent Rijmen Joan Daemen. Rijndael, 1999. <https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Standards-and-Guidelines/documents/aes-development/Rijndael-ammended.pdf>, Ostatni dostęp: styczeń 2021.

[Kowalczyk(2020a)] Krzysztof Kowalczyk. Aes, 2020a. <http://www.crypto-it.net/pl/symetryczne/aes.html>, Ostatni dostęp: styczeń 2021.

[Kowalczyk(2020b)] Krzysztof Kowalczyk. Tryby działania szyfrów blokowych, 2020b. <http://www.crypto-it.net/pl/teoria/tryby-szyfrow-blokowych.html>, Ostatni dostęp: styczeń 2021.

[Kowalczyk(2020c)] Krzysztof Kowalczyk. Dopełnianie bloków, 2020c. <http://www.crypto-it.net/pl/teoria/dopelnienie.html>, Ostatni dostęp: styczeń 2021.

- [Kowalczyk(2020d)] Krzysztof Kowalczyk. Protokół diffiego-hellmana, 2020d. <http://www.crypto-it.net/pl/asymetryczne/diffiego-hellmana.html>, Ostatni dostęp: styczeń 2021.
- [Kowalczyk(2020e)] Krzysztof Kowalczyk. Szyfrujące maszyny rotorowe, 2020e. <http://www.crypto-it.net/pl/proste/maszyny-rotorowe.html>, Ostatni dostęp: styczeń 2021.
- [Kowalczyk(2020f)] Krzysztof Kowalczyk. Rsa, 2020f. <http://www.crypto-it.net/pl/asymetryczne/rsa.html>, Ostatni dostęp: styczeń 2021.
- [Marino(2020)] Michael Marino. 20 najczęściej hakowanych haseł na świecie, 2020. <https://pl.safetydetectives.com/blog/the-most-hacked-passwords-in-the-world-pl/>, Ostatni dostęp: styczeń 2021.
- [Microsoft(2020)] Microsoft. Dokumentacja programu visual studio, 2020. <https://docs.microsoft.com/pl-pl/visualstudio/designers/getting-started-with-wpf?view=vs-2019>, Ostatni dostęp: listopad 2020.
- [NIST(2017)] NIST. Digital identity guidelines, 2017. <https://pages.nist.gov/800-63-3/sp800-63b.html>, Ostatni dostęp: styczeń 2021.
- [Piwowarczyk(2012)] Krystian Piwowarczyk. Ilustrowny opis standardu aes, 2012. <https://krystianpiwowarczyk.pl/content/article/show/ilustrowny-opis-standardu-aes>, Ostatni dostęp: styczeń 2021.
- [PKN(2002)] Polski Komitet Normalizacyjny PKN. Technika informatyczna. zabezpieczenia w systemach informatycznych. terminologia, 2002. <http://www.wsb.edu.pl/download.php?id=1180&source=pr>, Ostatni dostęp: styczeń 2021.
- [Point(2020)] Tutorials Point. Block cipher, 2020. https://www.tutorialspoint.com/cryptography/block_cipher.htm, Ostatni dostęp: styczeń 2021.
- [Predator(2020)] Predator. Strona producenta, 2020. <https://www.predator-usb.com/predator/en/index.php>, Ostatni dostęp: grudzień 2020.
- [PWN(1999)] PWN. Słownik języka polskiego pod red. w. doroszewskiego, 1999. <https://sjp.pwn.pl/slowniki/standard.html>, Ostatni dostęp: styczeń 2021.
- [Rohos(2020)] Rohos. Strona producenta, 2020. <https://www.rohos.com/products/rohos-logon-free/>, Ostatni dostęp: grudzień 2020.
- [Simmons(1999)] Gustavus J. Simmons. Cryptology, 1999. <https://www.britannica.com/topic/cryptology>, Ostatni dostęp: styczeń 2021.
- [Singh(2020)] Jas Singh. What is aaa in cyber security?, 2020. <https://cybersecuritykings.com/2020/06/07/what-is-aaa-in-cyber-security-must-know-info/>, Ostatni dostęp: styczeń 2021.

- [Singh(2001)] Simon Singh. *Księga szyfrów*. Albatros, 2001.
- [Staśkiewicz(2021)] Jakub Staśkiewicz. Szyfrowanie, kodowanie, hashowanie, 2021. <https://opensecurity.pl/szyfrowanie-kodowanie-hashowanie/>, Ostatni dostęp: styczeń 2021.
- [W3C(2018)] W3C. Web content accessibility guidelines (wcag) 2.1, 2018. <https://www.w3.org/TR/WCAG21/>, Ostatni dostęp: styczeń 2021.
- [Walkowski(2020)] Debbie Walkowski. What is the cia triad?, 2020. <https://www.yubico.com/works-with-yubikey/catalog/>, Ostatni dostęp: styczeń 2021.
- [Wałaszek(2020)] Jerzy Wałaszek. Algorytmy i struktury danych, 2020. https://eduinf.waw.pl/inf/alg/001_search/0067.php, Ostatni dostęp: styczeń 2021.
- [Yubico(2021a)] Yubico. Sklep yubico, 2021a. <https://www.yubico.com/pl/product/yubikey-5-nfc/>, Ostatni dostęp: styczeń 2021.
- [Yubico(2021b)] Yubico. Yubico, 2021b. <https://www.yubico.com/products/services-software/download/computer-login-tools/>, Ostatni dostęp: styczeń 2021.