**Podpis cyfrowy z wykorzystaniem algorytmu RSA w języku C#.**

Spis treści

[Kryptologia 3](#_Toc26951835)

[Kryptografia symetryczna 3](#_Toc26951836)

[Kryptografia asymetryczna 3](#_Toc26951837)

[Podpis cyfrowy 4](#_Toc26951838)

[Architektura 4](#_Toc26951839)

[Kryptografia 5](#_Toc26951840)

[RSA 6](#_Toc26951841)

[Opis algorytmu 6](#_Toc26951842)

[Funkcja skrótu 8](#_Toc26951843)

[Kryptograficzne funkcje skrótu 8](#_Toc26951844)

[Zastosowanie bezpiecznych funkcji skrótu 8](#_Toc26951845)

[SHA-1 10](#_Toc26951846)

[Pseudokod 11](#_Toc26951847)

[Opis programu 12](#_Toc26951848)

[Bibliografia 13](#_Toc26951849)

# Kryptologia

**Kryptologia** (z gr. κρυπτός kryptos, „ukryty”, i λόγος logos, „rozum”, „słowo”) – dziedzina wiedzy o przekazywaniu informacji w sposób zabezpieczony przed niepowołanym dostępem. Współcześnie kryptologia jest uznawana za gałąź zarówno matematyki, jak i informatyki; ponadto jest blisko związana z teorią informacji, inżynierią oraz bezpieczeństwem komputerowym.

Kryptologię dzieli się na:

* kryptografię, czyli gałąź wiedzy o utajnianiu wiadomości
* kryptoanalizę, czyli gałąź wiedzy o przełamywaniu zabezpieczeń oraz o deszyfrowaniu wiadomości przy braku klucza lub innego wymaganego elementu schematu szyfrowania (szyfru)

### Kryptografia symetryczna

W algorytmach symetrycznych klucz służy do szyfrowania i deszyfrowania wiadomości. Do obu tych czynności używa się tego samego klucza, dlatego powinien być znany tylko uczestnikom. Taki klucz jest przypisany do danej komunikacji, nie do posiadacza, dlatego zwykle do każdego połączenia jest generowany nowy klucz. Może do tego służyć np. (oparty na kryptografii asymetrycznej) protokół Diffiego-Hellmana.

### Kryptografia asymetryczna

W kryptosystemach asymetrycznych wyróżniamy klucz publiczny oraz prywatny. Ten pierwszy może być zupełnie jawny, drugi powinien znać tylko właściciel. Matematyczna konstrukcja kluczy powinna być taka, żeby wygenerowanie prywatnego na podstawie publicznego było jak najtrudniejsze obliczeniowo. Zależnie od kryptosystemu wygenerowanie klucza publicznego na podstawie prywatnego również może być trudne (RSA) lub trywialne (ElGamal).

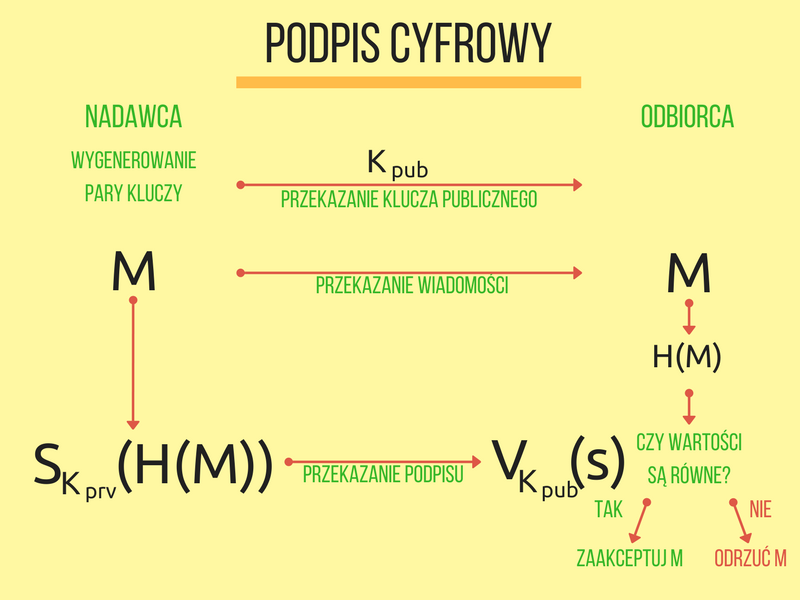
Dwie najważniejsze funkcje kryptografii asymetrycznej to:

* **szyfrowanie** – wtedy klucz publiczny służy do szyfrowania, a prywatny do deszyfrowania
* **podpisy cyfrowe** – klucz prywatny służy do generowania podpisów, klucz publiczny do ich weryfikacji

Klucze asymetryczne są zwykle przypisane do uczestnika (osoby, programu itp.), nie do kanału komunikacji. Dwa najpopularniejsze systemy kryptografii asymetrycznej to RSA i ElGamal. Inne to m.in. DSA i ECC.

# Podpis cyfrowy

**Podpis cyfrowy** – matematyczny sposób sprawdzenia autentyczności dokumentów i wiadomości elektronicznych. Poprawny podpis oznacza, że wiadomość pochodzi od właściwego nadawcy, który nie może zaprzeczyć faktowi jej nadania oraz, że wiadomość nie została zmieniona podczas transmisji.



### Architektura

Podpis cyfrowy służy zapewnieniu między innymi następujących funkcji bezpieczeństwa:

* autentyczności pochodzenia, która daje pewność co do autorstwa dokumentu,
* niezaprzeczalności, która utrudnia wyparcie się autorstwa lub znajomości treści dokumentu,
* integralności, która pozwala wykryć nieautoryzowane modyfikacje dokumentu po jego podpisaniu.

### Kryptografia

Podpis cyfrowy jest najczęściej realizowany przy pomocy technik kryptografii asymetrycznej, w której klucz prywatny jest używany do składania podpisu, zaś klucz publiczny służy do weryfikacji.

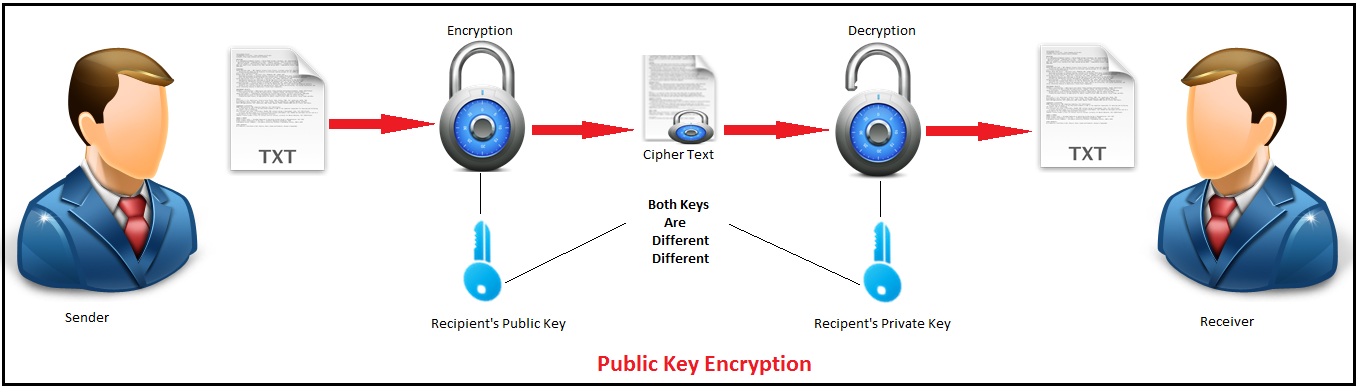
Część funkcji podpisu cyfrowego można również zrealizować przy pomocy kryptografii symetrycznej (np. ISO 13888-2).

Dwa dominujące standardy podpisu cyfrowego to RSA oraz DSA. Ten ostatni może być realizowany w oparciu o logarytmy dyskretne lub krzywe eliptyczne (ECDSA).

Znane są również techniki wykorzystujące arytmetykę wielu zmiennych (SFLASH, Quartz) oraz operujące na kratach (NTRU).

# RSA

**Algorytm Rivesta-Shamira-Adlemana, (RSA)** – jeden z pierwszych i obecnie najpopularniejszych asymetrycznych algorytmów kryptograficznych z kluczem publicznym, zaprojektowany w 1977 przez Rona Rivesta, Adiego Shamira oraz Leonarda Adlemana. Pierwszy algorytm, który może być stosowany zarówno do szyfrowania jak i do podpisów cyfrowych. Bezpieczeństwo szyfrowania opiera się na trudności faktoryzacji dużych liczb złożonych. Jego nazwa pochodzi od pierwszych liter nazwisk jego twórców.



### Opis algorytmu

#### Generowanie kluczy

W celu wygenerowania pary kluczy (prywatnego i publicznego) należy posłużyć się algorytmem:

* Wybieramy losowo dwie duże liczby pierwsze p i q (najlepiej w taki sposób, aby obie miały zbliżoną długość w bitach, ale jednocześnie były od siebie odległe wartościami – istnieją lepsze mechanizmy faktoryzacji, jeżeli liczba ma dzielnik o wartości bliskiej )
* Obliczamy wartość n = pq
* Obliczamy wartość funkcji Eulera dla n:
* Wybieramy liczbę względnie pierwszą z
* Znajdujemy liczbę d, gdzie jej różnica z odwrotnością modularną liczby jest podzielna przez :

Ta liczba może być też prościej określona wzorem:

**Klucz publiczny** jest definiowany jako para liczb (n, e), natomiast **kluczem prywatnym** jest para (n, d).

#### Szyfrowanie i deszyfrowanie

Zanim zaszyfrujemy wiadomość, dzielimy ją na bloki o wartości liczbowej nie większej niż, a następnie każdy z bloków szyfrujemy według poniższego wzoru:

Zaszyfrowana wiadomość będzie się składać z kolejnych bloków . Tak stworzony szyfrogram przekształcamy na tekst jawny, odszyfrowując kolejne blok według wzoru:

#### Własności operacji szyfrowania i deszyfrowania

Niech będą kolejno szyfrowaniem i deszyfrowaniem kluczami i . Wtedy zachodzi:

- przemienność operacji szyfrowania

- przemienność operacji deszyfrowania

Ze względów bezpieczeństwa nie powinno się stosować więcej niż 2 zagnieżdżone szyfrowania ze względu na ataki oparte na chińskim twierdzeniu o resztach.

#### Podpisywanie i weryfikacja podpisu

Analogicznie, RSA może zostać użyte do przeprowadzenia operacji podpisu. W takim przypadku szyfruje się zazwyczaj skrót wiadomości za pomocą klucza prywatnego i propaguje taki szyfrogram wraz z oryginalną wiadomością. Odbiorca posiadający klucz publiczny deszyfruje otrzymaną z wiadomością, zaszyfrowaną wartość funkcji skrótu, następnie oblicza wartość tejże funkcji z otrzymanej wiadomości. Jeśli obie wartości się zgadzają, to przyjmuje się, że wiadomość została podpisana poprawnie.

# Funkcja skrótu

**Funkcja skrótu**, **funkcja mieszająca** lub **funkcja haszująca** – funkcja przyporządkowująca dowolnie dużej liczbie krótką, zawsze posiadającą stały rozmiar, niespecyficzną, quasi-losową wartość, tzw. skrót nieodwracalny.

W informatyce funkcje skrótu pozwalają na ustalenie krótkich i łatwych do weryfikacji sygnatur dla dowolnie dużych zbiorów danych. Sygnatury mogą chronić przed przypadkowymi lub celowo wprowadzonymi modyfikacjami danych (sumy kontrolne), a także mają zastosowania przy optymalizacji dostępu do struktur danych w programach komputerowych (tablice mieszające).

### Kryptograficzne funkcje skrótu

Szczególną podgrupą funkcji skrótu są funkcje uznawane za bezpieczne do zastosowań kryptologicznych (jak np. SHA-3). Kryptograficzna funkcja skrótu powinna spełniać kombinację następujących kryteriów, **w zależności od zastosowania:**

1. Odporność na kolizje (*collision resistance*) – brak praktycznej możliwości wygenerowanie dwóch **dowolnych** wiadomości o takim samym skrócie
2. Odporność na kolizje konkretnych wiadomości (*target collision-resistance, preimage resistance*) pierwszego i drugiego rzędu – brak praktycznej możliwości wygenerowania wiadomości o takim samym skrócie jak **wskazana** wiadomość
3. Jednokierunkowość (*one-wayness*) – brak możliwości wnioskowania o wiadomości wejściowej na podstawie wartości skrótu. Zmiana dowolnego pojedynczego bitu wiadomości powinna zmieniać średnio połowę bitów skrótu w sposób, który nie jest istotnie podatny na kryptoanalizę różnicową.

### Zastosowanie bezpiecznych funkcji skrótu

#### Weryfikacja integralności plików bądź wiadomości

Istotnym zastosowaniem funkcji skrótu jest weryfikacja spójności danych. Porównanie skrótów dwóch plików umożliwia stwierdzenie, czy w pliku zostały dokonane jakiekolwiek zmiany.

Z tego powodu, większość algorytmów podpisu cyfrowego działa na zasadzie wygenerowania skrótu wiadomości, dzięki czemu można w dowolnym momencie sprawdzić, czy wiadomość jest autentyczna.

Funkcje skrótu używane są również do weryfikacji haseł. W celu zwiększenia bezpieczeństwa, w bazie danych przechowuje się skrót hasła, zamiast tekstu jawnego. Hasło wprowadzone przez użytkownika jest haszowane i dopiero wtedy porównywane ze skrótem w bazie danych. Taki sposób przechowywania haseł utrudnia ich odzyskanie, ze względu na jednokierunkowość funkcji haszujących.

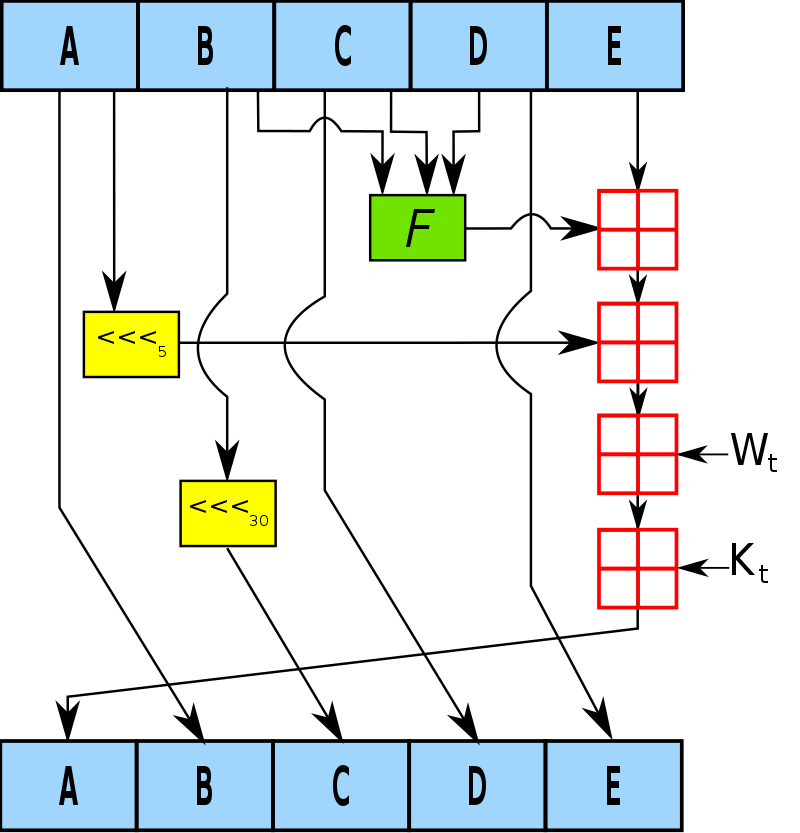
#### Identyfikacja plików lub danych

Skrót może również służyć do identyfikacji plików; Systemy kontroli wersji, takie jak Git lub Mercurial używają funkcji skrótu SHA do identyfikacji różnego rodzaju zawartości (zawartość plików, drzewa katalogów).

Innym zastosowaniem jest używanie skrótów w tablicach z haszowaniem w celu szybkiego wyszukiwania danych.

# SHA-1

**SHA (Secure Hash Algorithm)** – rodzina powiązanych ze sobą kryptograficznych funkcji skrótu zaprojektowanych przez NSA (National Security Agency) i publikowanych przez *National Institute of Standards and Technology*.



Rysunek Funkcja kompresji algorytmu SHA-1

Pierwszy z nich opublikowany w 1993 oficjalnie nazwany SHA (nieoficjalnie, żeby nie pomylić z następcami określany jako **SHA-0**).

**SHA-1** opublikowany został w 1995 i całkowicie zastąpił wycofany (ze względu na nieujawnione oficjalnie wady) z użytku SHA-0. SHA-0 i SHA-1 tworzą 160-bitowy skrót z wiadomości o maksymalnym rozmiarze 264 bitów i jest oparty na podobnych zasadach co MD5. Algorytm SHA-1 nie powinien być używany w nowych aplikacjach.

W 2001 powstały cztery następne warianty określane jako SHA-2 (**SHA-224, SHA-256, SHA-384, SHA-512**).

Podstawowym celem publikacji SHA był amerykański Standard Podpisu Cyfrowego (Digital Signature Standard). SHA jest podstawą szyfru blokowego SHACAL.

W 2009 NIST przeprowadził publiczny konkurs na następcę dotychczasowych funkcji skrótu w duchu podobnym do procesu wyłaniania algorytmu szyfrującego AES. Zostało zgłoszonych kilkadziesiąt kandydatur, które zostały rygorystycznie ocenione. Zwycięski algorytm wyłoniony w 2012 nosi miano SHA-3.

### Pseudokod

Wartości początkowe:

h0 := 0x67452301

h1 := 0xEFCDAB89

h2 := 0x98BADCFE

h3 := 0x10325476

h4 := 0xC3D2E1F0

Przetwarzanie wstępne:

dopisz '1' do wiadomości;

dopisz k '0', gdzie 0 ≤ k < 512 jest liczbą taką, że wynikowa długość wiadomości jest kongruentna do 448 modulo 512;

dopisz długość wiadomości w bitach (przed wypełnieniem) jako 64-bitową liczbę całkowitą zakodowaną big endian.

Przetwarzaj wiadomość 512-bitowymi porcjami:

podziel wiadomość na 512-bitowe porcje

for (każda porcja)

podziel porcję na 16 32-bitowych słów kodowanych big-endian w(i), 0 ≤ i ≤ 15

Rozszerz 16 32-bitowych słów w 80 32-bitowych słów:

for i from 16 to 79

w(i) := (w(i-3) **xor** w(i-8) **xor** w(i-14) **xor** w(i-16)) <<< 1

Zainicjuj zmienne dla tej porcji:

a := h0

b := h1

c := h2

d := h3

e := h4

Główna pętla:

for i from 0 to 79

if 0 ≤ i ≤ 19 then

f := (b **and** c) or ((not b) **and** d)

k := 0x5A827999

else if 20 ≤ i ≤ 39

f := b **xor** c **xor** d

k := 0x6ED9EBA1

else if 40 ≤ i ≤ 59

f := (b **and** c) or (b **and** d) or (c **and** d)

k := 0x8F1BBCDC

else if 60 ≤ i ≤ 79

f := b **xor** c **xor** d

k := 0xCA62C1D6

temp := (a **<<<** 5) + f + e + k + w(i)

e := d

d := c

c := b **<<<** 30

b := a

a := temp

Dodaj skrót tej porcji do dotychczasowego wyniku:

h0 := h0 + a

h1 := h1 + b

h2 := h2 + c

h3 := h3 + d

h4 := h4 + e

Wytwórz ostateczną wartość skrótu (zakodowaną big-endian):

skrót = h0 **dopisz** h1 **dopisz** h2 **dopisz** h3 **dopisz** h4

Jedyna różnica pomiędzy algorytmami SHA-0 i SHA-1 występuje w drugim kroku algorytmu i polega na nieobecności w SHA-0 1-bitowego przesunięcia (obrotu), tzn.:

SHA-1:

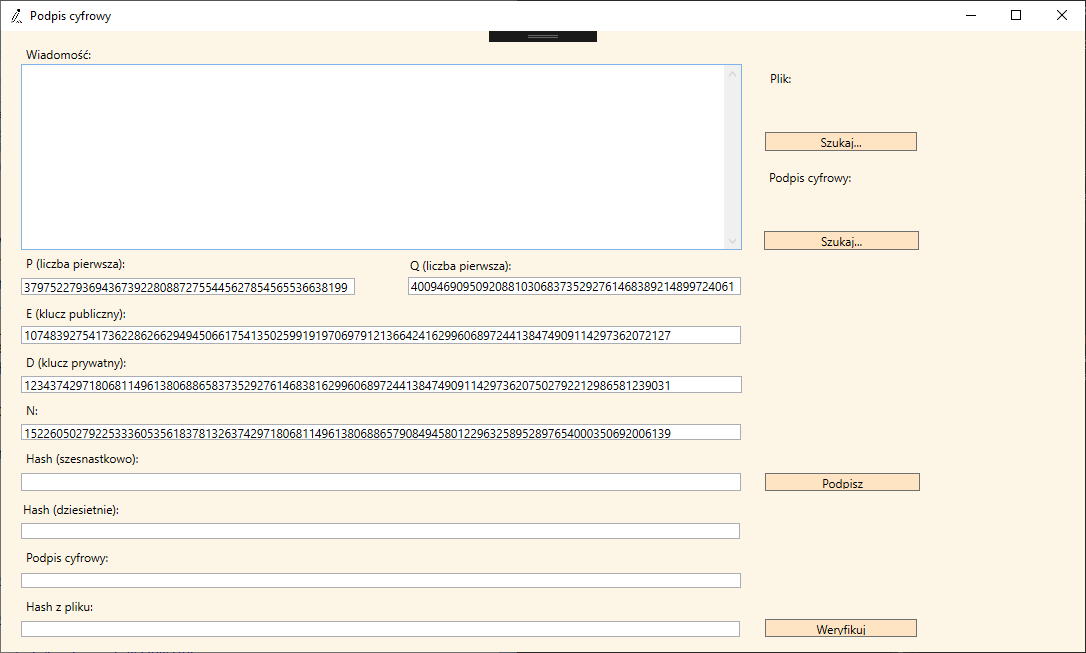
w(i) := (w(i-3) **xor** w(i-8) **xor** w(i-14) **xor** w(i-16)) <<< 1

SHA-0:

w(i) := (w(i-3) **xor** w(i-8) **xor** w(i-14) **xor** w(i-16))

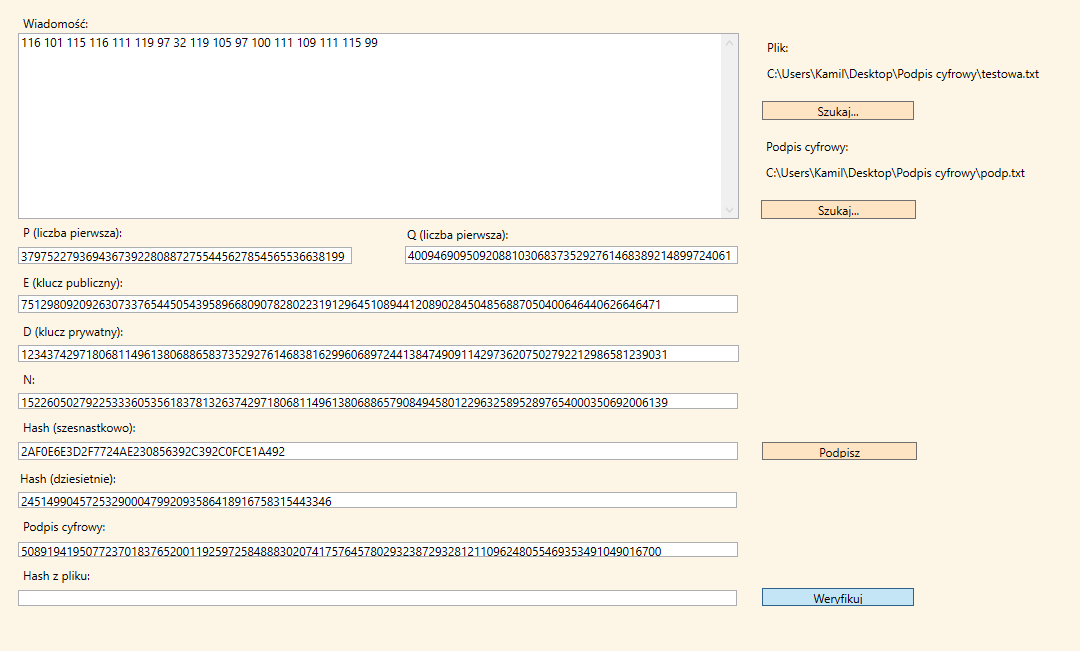
# Opis programu

Przygotowana przez mnie aplikacja ma służyć do podpisywania cyfrowego plików. Opiera się ona głównie na algorytmie RSA.



Na podstawie dwóch losowo wybranych liczb pierwszych P i Q dobieram odpowiednie liczby E i D, oraz obliczam N, które jest iloczynem liczb P i Q. Klucz publiczny jest definiowany jako para liczb (N,E), natomiast kluczem prywatnym jest para (N,D). Wszystkie wymienione tutaj zmienne są statycznie, ponieważ esencją mojej aplikacji nie jest generacja kluczy, a weryfikacja podpisu cyfrowego.

Korzystanie z aplikacji polega na wczytaniu pliku, który chcemy podpisać. Gdy zostanie on już załadowany, mamy możliwość podpisania go. W tym celu wybieramy przycisk „Podpisz”, a nasz aplikacja w tym czasie wypisuje nam hash w systemie szesnastkowym i dziesiętnym oraz generuje podpis cyfrowy. Dodatkowo w tej samej lokalizacji w której znajduje się wcześniej przez nas wybrany plik, tworzony jest nowy, o tej samej nazwie, z dopiskiem „\_Podpis.txt”. Jego zawartością jest skrót wiadomości w systemie dziesiętnym, który później posłuży nam do weryfikacji. Ważne jest to, że nasza aplikacja nie pozwala na weryfikację podpisu, jeśli wcześniej nie podpiszemy wybranego przez nas pliku. Weryfikacja polega na tym, że odszukujemy plik tekstowy z dopiskiem „\_Podpis.txt” i wczytujemy go do programu. Weryfikacja jest dostępna pod odpowiednim przyciskiem. Jeśli sygnatury się zgadzają, zostajemy poinformowani o autentyczności podpisu. W przeciwnym wypadku otrzymamy komunikat, że podpis się nie zgadza.



# Bibliografia

<https://pl.wikipedia.org/wiki/Kryptologia>

<https://pl.wikipedia.org/wiki/Klucz_(kryptografia)>

<https://pl.wikipedia.org/wiki/Podpis_cyfrowy>

<https://pl.wikipedia.org/wiki/RSA_(kryptografia)>

<https://pl.wikipedia.org/wiki/Funkcja_skr%C3%B3tu>

<https://pl.wikipedia.org/wiki/SHA-1>