Szyfr RSA w języku Python

Źródło: Skitterphoto, dostępny w internecie: pixabay.com, domena publiczna.

We współczesnym świecie, gdzie dominującą formą komunikacji jest przekaz cyfrowy, niezbędne stało się opracowywanie algorytmów szyfrujących. Jedną z metod często stosowanych przez kryptografów jest szyfr RSA. W tym e-materiale przyjrzymy się implementacji algorytmu realizującego ten szyfr w języku Python.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z tej serii:

- Szyfr RSA w języku C++,
- Szyfr RSA w języku Java.

Więcej zadań? Sięgnij do: Szyfr RSA – zadania maturalne.

Twoje cele

- Przeanalizujesz funkcję w języku Python generującą klucze RSA.
- Zaimplementujesz program z wykorzystaniem biblioteki `PySimpleGUI` do szyfrowania i odszyfrowywania wiadomości.
- Przedstawisz generowanie podpisu cyfrowego za pomocą metody RSA.

Przeczytaj

Tworzenie szyfrogramu za pomocą asymetrycznego algorytmu RSA polega na wyznaczeniu pary kluczy: prywatnego i publicznego. Pozwalają one na zaszyfrowanie (klucz publiczny) i odszyfrowanie (klucz prywatny) wiadomości.

Tworzenie klucza publicznego i prywatnego

Przykład 1

Stwórzmy w języku Python funkcję generującą klucze RSA. Algorytm został szczegółowo opisany w e-materiale wstępnym dotyczącym szyfru RSA. Dobierzemy wykładnik publiczny, aby funkcja działała możliwie szybko.

W przykładzie zdefiniowane są dwie funkcje pomocnicze: pierwsza obliczająca największy wspólny dzielnik, a druga - odwrócone modulo. Zasadnicza część funkcji będzie realizowała cztery kroki wyznaczania kluczy, również opisane w e-materiale wstępnym. W pierwszym kroku wybieramy dwie liczby pierwsze i oznaczamy je jako `p` i `q`. Losujemy je z gotowego zbioru liczb pierwszych, zapisanego w liście `pierwsze`.

Następnie obliczamy wartość `n` jako iloczynu `p` i `q`. Liczba ta stanowi część klucza prywatnego i klucza publicznego. Przechodząc kolejne kroki algorytmu, wykorzystujemy funkcję Eulera, aby w następnym kroku dobrać wartość `e`, w naszym przypadku względnie małą, aby przyspieszyć obliczenia. Liczba `e` jest elementem klucza publicznego. Pozostaje tylko wyznaczenie liczby `d` za pomocą funkcji odwróconego modulo - staje się ona częścią klucza prywatnego. Na końcu kod zwraca klucz publiczny (para liczb `e` i `n`) oraz klucz prywatny (para liczb `d` i `n`).

Specyfikacja problemu:

Dane:

- `p` – liczba naturalna pierwsza
- `q` – liczba naturalna pierwsza

Wynik:

- klucz prywatny algorytmu RSA i klucz publiczny algorytmu RSA

```
def klucze_RSA():

    def nwd(a, b):
        # obliczanie NWD
```

```
        while b:
            a, b = b, a % b
        return a

    def odwr_mod(a, n):
        # obliczanie odwróconego modulo
        p0, p1, a0, n0 = 0, 1, a, n
        q, r = n0 // a0, n0 % a0
        while r:
            t = p0 - q * p1
            if t >= 0:
                t = t % n
            else:
                t = n - ((-t) % n)
                p0, p1, n0, a0 = p1, t, a0, r
                q, r  = n0 // a0, n0 % a0
        return p1

    # zasadnicza część funkcji RSA
    from random import choice
    pierwsze = [11, 13, 17, 19, 23, 29, 31] #gotowa lista liczb
    p = 0
    q = 0
    while p == q:
        p, q = choice(pierwsze), choice(pierwsze) # funkcja cho
        phi, n = (p - 1) * (q - 1), p * q   #obliczenie n i war
        e = 3   # dobieramy możliwie małą wartość dla e
        d = odwr_mod(e, phi)   # obliczamy wartość d
        while nwd(e, phi) != 1:
            e += 2
            d = odwr_mod(e, phi)

    return (e, n), (d, n)       #zwracamy klucze

# przykładowe wykonanie
print(klucze_RSA())

# przykładowy wynik:
# ((5, 551), (101, 551))
```

Musimy pamiętać, że czasami proces obliczeń może trwać długo, gdyż język Python jest interpreterem i przy dużej liczbie obliczeń bywa dość powolny.

Szyfrowanie i odszyfrowywanie

Przykład 2

Szyfrowanie liczby `t` pozwalające uzyskać liczbę `c` polega na zastosowaniu wzoru:

$$c = t^e mod\ n$$

gdzie e to wykładnik publiczny, a n to element klucza publicznego.

Ponieważ tekst składa się z sekwencji znaków, z których każdy może być przedstawiony w postaci liczby (kodu znaku), szyfrowanie tekstu polegać będzie na szyfrowaniu kolejno każdego znaku z wykorzystaniem przedstawionego wzoru.

Specyfikacja problemu:

Dane:

- `tekst` – łańcuch znaków
- `n` – liczba naturalna
- `e` – liczba naturalna

Wynik:

- zaszyfrowany łańcuch znaków

```
1  def szyfrowanie_RSA(tekst: str, klucz_publiczny: tuple):
2      e, n = klucz_publiczny
3      szyfr = [(ord(t) ** e) % n for t in tekst]
4      return szyfr
5
6  # przykład wykonania
7  szyfrogram = szyfrowanie_RSA("Python to interpreter", (5, 551))
8  print(szyfrogram)
9
10 # efekt działania
11 # [245, 486, 203, 510, 384, 344, 185, 203, 384, 185, 421, 344,
```

Analogicznie – odszyfrowywanie liczby `c` uzyskujące liczbę `t` polega na zastosowaniu wzoru:

$$t = c^d mod\ n$$

gdzie d to wykładnik prywatny, a n to element klucza prywatnego.

Możemy zatem przygotować funkcję służącą do odszyfrowania tekstu.

Specyfikacja problemu:

Dane:

- `szyfrogram` – łańcuch znaków
- `n` – liczba naturalna
- `d` – liczba naturalna

Wynik:

- odszyfrowany łańcuch znaków

```
def odszyfrowanie_RSA(szyfrogram: list, klucz_prywatny: tuple):
    d, n = klucz_prywatny
    jawny = [chr((kod ** d) % n) for kod in szyfrogram]
    return "".join(jawny)

# przykład wykonania
szyfrogram = [245, 486, 203, 510, 384, 344, 185, 203, 384, 185,
              171, 309, 203, 309, 171]
odszyfrowany = odszyfrowanie_RSA(szyfrogram, (101, 551))
print(odszyfrowany)

# efekt działania
# Python to interpreter
```

Szyfrowanie i odszyfrowywanie w programie z graficznym interfejsem

Przykład 3

Możemy przygotować program z graficznym interfejsem dla szyfrowania lub odszyfrowywania. Użyjemy do tego celu biblioteki PySimpleGui. Bibliotekę instalujemy z poziomu wiersza poleceń za pomocą komendy:
`python -m pip install PySimpleGUI`

Wykorzystamy następujące elementy z tej biblioteki:

- `Text` – prosty napis
- `Input` – pojedyncza linijka na wpisanie tekstu
- `Radio` – przełącznik (tylko jeden z listy przełączników jest włączony)
- `MLine` – pole do wypisywania tekstu

- `Button` – prosty przycisk z tekstem w środku
- `Window` – kontener reprezentujący nasze okno

Z obiektu `Window` możemy wyciągnąć obecne wartości elementów, jakie zawiera, oraz wydarzenia, które są ich udziałem (jak np. naciśnięcie przycisku). W tym celu użyjemy metody okna `Window` – `read()`.

Niektóre z elementów przyjmują parametr `key`, określający klucz, który pozwoli nam pobierać wartość elementu za pomocą metody `read()`.

Metoda `close()` dla `Window` zamyka okno.

Wartości elementów możemy aktualizować za pomocą metody `update()`.

```
def gui_szyfrowanie_RSA():
    import PySimpleGUI as sg
    uklad = [[sg.Text("Podaj tekst jawny lub zaszyfrowany jako li
              [sg.Input(key="dane")],
              [sg.Text("Wpisz klucz prywatny (jako tuplę) do odszy
              [sg.Input(key="klucz_prywatny")],
              [sg.Text("Wpisz klucz publiczny (jako tuplę) do szyf
              [sg.Input(key="klucz_publiczny")],
              [sg.Radio("Szyfruj (kluczem publicznym)",1 , key="s"
              [sg.Radio("Odszyfruj (kluczem prywatnym)",1 , key="o
              [sg.Text("Zszyfrowana lub jawna wiadomość:", size=(5
              [sg.MLine(size=(40,8), key='-MSG-')],
              [sg.Button('Wykonaj'), sg.Exit()]]
    okno = sg.Window("Szyfrowanie RSA", uklad)

    while True:
        event, values = okno.read()

        if event == 'Exit' or event is None:
            break
        if event == 'Wykonaj':
            if values['s']:
                k_pub = values["klucz_publiczny"].split(",")
                k_pub = (int(k_pub[0]), int(k_pub[1]))
                wiadomosc = values["dane"]
                szyfrogram = szyfrowanie_RSA(wiadomosc, k_pub)
                okno["-MSG-"].update(szyfrogram)
            if values['o']:
```

```
                k_priv = values["klucz_prywatny"].split(",")
                k_priv = (int(k_priv[0]), int(k_priv[1]))
                szyfr = values["dane"].split(",")
                szyfr = [int(e) for e in szyfr]
                wynik = odszyfrowanie_RSA(szyfr, k_priv)
                okno["-MSG-"].update(wynik)
    okno.close()
```

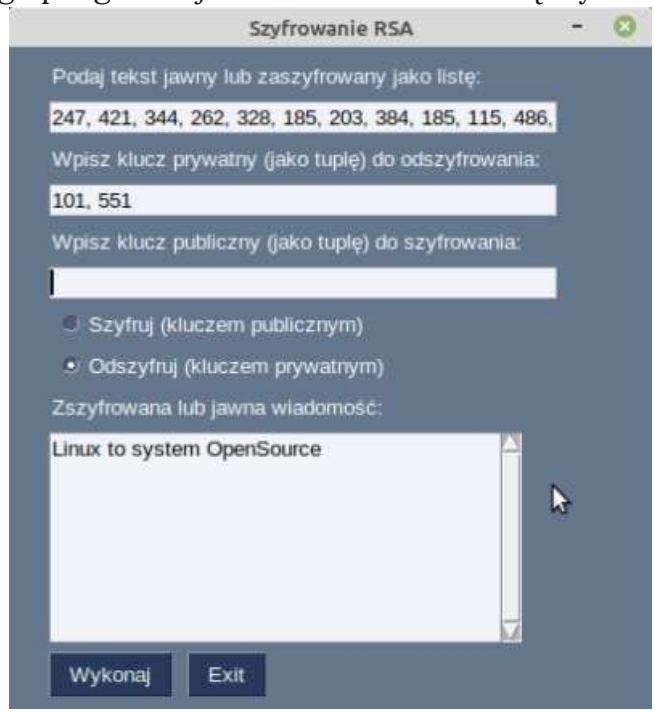
Oto pełny kod programu, łączący przedstawione funkcje:

```
def klucze_RSA():

    def nwd(a, b):
        # obliczanie NWD
        while b:
            a, b = b, a % b
        return a

    def odwr_mod(a, n):
        # obliczanie odwróconego modulo
        p0, p1, a0, n0 = 0, 1, a, n
        q, r = n0 // a0, n0 % a0
        while r:
            t = p0 - q * p1
            if t >= 0:
                t = t % n
            else:
                t = n - ((-t) % n)
                p0, p1, n0, a0 = p1, t, a0, r
                q, r  = n0 // a0, n0 % a0
        return p1

    # zasadnicza część funkcji RSA
    from random import choice
    pierwsze = [11, 13, 17, 19, 23, 29, 31] #gotowa lista liczb p
    p = 0
    q = 0
    while p == q:
        p, q = choice(pierwsze), choice(pierwsze) # funkcja choic
        phi, n = (p - 1) * (q - 1), p * q   #obliczenie n i warto
        e = 3   # dobieramy możliwie małą wartość dla e
```

```
        d = odwr_mod(e, phi)   # obliczamy wartość d
        while nwd(e, phi) != 1:
            e += 2
            d = odwr_mod(e, phi)

    return (e, n), (d, n)       #zwracamy klucze


def szyfrowanie_RSA(tekst: str, klucz_publiczny: tuple):
    e, n = klucz_publiczny
    szyfr = [(ord(t) ** e) % n for t in tekst]
    return szyfr


def odszyfrowanie_RSA(szyfrogram: list, klucz_prywatny: tuple):
    d, n = klucz_prywatny
    jawny = [chr((kod ** d) % n) for kod in szyfrogram]
    return "".join(jawny)


def gui_szyfrowanie_RSA():
    import PySimpleGUI as sg
    uklad = [[sg.Text("Podaj tekst jawny lub zaszyfrowany jako li
             [sg.Input(key="dane")],
             [sg.Text("Wpisz klucz prywatny (jako tuplę) do odszy
             [sg.Input(key="klucz_prywatny")],
             [sg.Text("Wpisz klucz publiczny (jako tuplę) do szyf
             [sg.Input(key="klucz_publiczny")],
             [sg.Radio("Szyfruj (kluczem publicznym)",1 , key="s"
             [sg.Radio("Odszyfruj (kluczem prywatnym)",1 , key="o
             [sg.Text("Zszyfrowana lub jawna wiadomość:", size=(5
             [sg.MLine(size=(40,8), key='-MSG-')],
             [sg.Button('Wykonaj'), sg.Exit()]]
    okno = sg.Window("Szyfrowanie RSA", uklad)

    while True:
        event, values = okno.read()

        if event == 'Exit' or event is None:
            break
        if event == 'Wykonaj':
            if values['s']:
```

```
                k_pub = values["klucz_publiczny"].split(",")
                k_pub = (int(k_pub[0]), int(k_pub[1]))
                wiadomosc = values["dane"]
                szyfrogram = szyfrowanie_RSA(wiadomosc, k_pub)
                okno["-MSG-"].update(szyfrogram)
            if values['o']:
                k_priv = values["klucz_prywatny"].split(",")
                k_priv = (int(k_priv[0]), int(k_priv[1]))
                szyfr = values["dane"].split(",")
                szyfr = [int(e) for e in szyfr]
                wynik = odszyfrowanie_RSA(szyfr, k_priv)
                okno["-MSG-"].update(wynik)
    okno.close()


gui_szyfrowanie_RSA()
```

Efektem działania takiego programu jest okno z możliwością wyboru operacji.



Dla zainteresowanych

Wśród wielu innych bibliotek umożliwiających operacje kryptograficzne możemy wyróżnić cryptography. Dysponuje ona gotowymi metodami dla różnych sposobów szyfrowania, m.in.:

- RSA,
- kodowanie Huffmana,

- skróty wiadomości.

Słownik

cryptography

biblioteka umożliwiająca szyfrowanie; jej twórcy mieli na celu stworzenie standardowej biblioteki kryptograficznej, niezależnej od systemu operacyjnego; nie jest dostępna w standardowej instalacji języka Python – należy ją zainstalować, korzystając z mechanizmu `pip`

graficzny interfejs

nakładka graficzna na program, pozwalająca użytkownikom na korzystanie z niego w sposób interaktywny

klucz prywatny

służy do odszyfrowania informacji; jest w wyłącznym posiadaniu adresata informacji, więc tylko on może ją odczytać

klucz publiczny

służy do zaszyfrowania informacji

PySimpleGUI

biblioteka do wyświetlania prostych okien dialogowych, niezależna od systemu operacyjnego; nie jest dostępna w standardowej instalacji języka Python – należy ją zainstalować, korzystając z mechanizmu `pip`

szyfrogram

(inaczej kryptogram) wiadomość, która została zaszyfrowana

Film samouczek

Polecenie 1
Przeanalizuj krok po kroku tworzenie kluczy RSA oraz szyfrowanie i deszyfrowanie tekstu za pomocą tego algorytmu.

Problem 1
Napisz program, który za pomocą algorytmu RSA wygeneruje podpis i zweryfikuje jego poprawność.

Program przetestuj dla łańcucha znaków: `wiadomosc = "Magda ma kota"`.

Specyfikacja problemu:

Dane:

- `wiadomosc` - łańcuch znaków; wiadomość do wygenerowania podpisu

Wynik:

Program generuje podpis dla zadanej wiadomości, a następnie sprawdza jego poprawność.

Polecenie 2
Porównaj swoje rozwiązanie z zaprezentowanym w filmie.

Film dostępny pod adresem /preview/resource/RQOj5MB851t8G

Film przedstawia etapy pisania programu w języku Python, który przy pomocy algorytmu RSA wygeneruje podpis i zweryfikuje poprawność.

Kod programu zaprezentowanego w filmie:

Plik o rozmiarze 1.54 KB w języku polskim

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Wykorzystaj moduł primePy (pobierz go z witryny Python Package Index (PyPI), a następnie zainstaluj za pomocą narzędzia `pip`). Przeanalizuj kod, uruchamiając go lokalnie w środowisku IDLE, a następnie wykonaj polecenie. Zastosowana w kodzie funkcja `primes.between(x, y)` (z modułu primePy) zwraca wszystkie liczby pierwsze z zakresu od `x` do `y`. Funkcja `fabs(a, b)` zwraca bezwzględną różnicę między `a` i `b` jako typ `float`. Funkcja `choice(zbiór)` zwraca losowy element danego zbioru (funkcja `seed` inicjalizuje generator liczb losowych).

```
from primePy import primes
from math import fabs
from random import choice, seed

seed(23487523)

def sprawdz_dlugosc(num1, num2, max=2):
    return True if fabs(num1.bit_length() - num2.bit_length()

liczby_pierwsze = primes.between(20, 20000)

while True:
    n1 = choice(liczby_pierwsze)
    n2 = choice(liczby_pierwsze)
    if sprawdz_dlugosc(n1, n2): break

print("Znalezione liczby pierwsze:",n1,"i",n2)
```

Ćwiczenie 2

Przeanalizuj kod, uruchamiając go lokalnie w środowisku IDLE lub w dowolnym innym IDE, a następnie wykonaj polecenie. Funkcja `gcd(a, b)` zwraca największy wspólny dzielnik liczb a i b.

```
from math import gcd

def mod_odwr(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

def testuj_mod(a, b):
    y = mod_odwr(a, b)
    if (y is not None and y != a):
        return True
    else:
        return False

phi = (233 - 1) * (137 - 1)
print("Dla phi =",phi)

l = [x for x in range(2, phi) if gcd(phi, x) == 1 and testuj_r

print("Znaleziono elementów:",len(l),".")
```

Ćwiczenie 3

Zdefiniuj funkcję, która wykona następujące operacje:

- wygeneruje listę liczb względnie pierwszych (czyli takich, których największym wspólnym dzielnikiem jest 1), która zostanie użyta jako `tablica_kodowania`,
- zwróci zaszyfrowane dane w postaci listy bajtów.

Szyfrowanie polega na pobraniu kolejnego bajtu łańcucha wejściowego i wykonaniu na nim operacji `xor` z kolejnym bajtem `tablica_kodowania`. Jeśli tekst jawny będzie dłuższy niż tablica klucza, należy użyć funkcji `cycle()` z biblioteki `intertools`, która zapętla podaną tablicę.

Program przetestuj dla danych:

```
tekst_jawny = "Python to jezyk programowania"
krotka_nq = (43, 79)
```

oraz

```
tekst_jawny = "Linux - system operacyjny"
krotka_nq = (31, 89)
```

Specyfikacja problemu:

Dane:

- `tekst_jawny` – łańcuch znaków; tekst wejściowy
- `krotka_nq` – krotka zawierająca dwie liczby pierwsze

Wynik:

Program wypisuje zaszyfrowany tekst.

Przykładowe wywołanie:

```
wynik = szyfruj("Linux - system operacyjny", (37, 67))
print(wynik)
# [73, 110, 99, 100, 107, 55, 52, 61, 108, 90, 86, 93, 78, 66,
```

Ćwiczenie 4

W standardowej tabeli ASCII jest 95 znaków, których używamy do pisania – w obrębie jednej linii nie ma możliwości formatowania (użycia tabulatorów itp.). Do zbioru znaków należą litery, cyfry oraz inne znaki specjalne (w tym tylko jeden znak biały – spacja). Kody ASCII, o których mowa, to znaki od kodu 32 (znak spacji `' '`) do kodu 126 (znak tyldy `'~'`). Liczba 95 jest iloczynem dwóch liczb pierwszych: 5 i 19.

Napisz program, który będzie szyfrował i wypisywał podane na wejściu wiadomości metodą RSA, na podstawie klucza publicznego, składającego się z wykładnika publicznego `wykladnik` i liczby `n`.

Program przetestuj dla następujących danych:

- `wykladnik = 11`
- `n = 95`
- `tekst_jawny = "Lex retro non agit"`

Specyfikacja problemu:

Dane:

- `wykladnik` – wykładnik publiczny, liczba naturalna
- `n` - iloczyn dwóch liczb pierwszych, liczba naturalna
- `tekst_jawny` – ciąg znaków do zaszyfrowania

Wynik:

Program na standardowe wyjście wypisuje zaszyfrowaną wiadomość.

Ćwiczenie 5

Dany jest szyfrogram zaszyfrowany za pomocą pewnego klucza publicznego. Masz dostęp do klucza prywatnego `klucz_prywatny`. Napisz program, który odszyfruje wiadomość i wypisze ją na wyjście standardowe. Przetestuj działanie programu dla następujących danych:

- `klucz_prywatny = (65, 95)`
- `szyfrogram = "ZRst- d*.z^ -7t Ltde m7 de^Rd*-."`

Specyfikacja problemu:

Dane:

- `klucz_prywatny` – krotka liczb naturalnych
- `szyfrogram` – zaszyfrowana wiadomość, ciąg znaków

Wynik:

Program na standardowe wyjście wypisuje odszyfrowaną wiadomość.

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Szyfr RSA w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

V. Przestrzeganie prawa i zasad bezpieczeństwa. Respektowanie prywatności informacji i ochrony danych, praw własności intelektualnej, etykiety w komunikacji i norm współżycia społecznego, ocena zagrożeń związanych z technologią i ich uwzględnienie dla bezpieczeństwa swojego i innych.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres podstawowy. Uczeń:

2) do realizacji rozwiązań problemów prawidłowo dobiera środowiska informatyczne, aplikacje oraz zasoby, wykorzystuje również elementy robotyki;

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

f) metodę szyfrowania z kluczem publicznym i jej zastosowanie w podpisie elektronicznym,

V. Przestrzeganie prawa i zasad bezpieczeństwa.

Zakres podstawowy. Uczeń:

3) stosuje dobre praktyki w zakresie ochrony informacji wrażliwych (np. hasła, pin), danych i bezpieczeństwa systemu operacyjnego, objaśnia rolę szyfrowania informacji;

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) objaśnia rolę technik uwierzytelniania, kryptografii i podpisu elektronicznego w ochronie i dostępie do informacji;

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Przeanalizujesz funkcję w języku Python generującą klucze RSA.
- Zaimplementujesz program z wykorzystaniem biblioteki `PySimpleGUI` do szyfrowania i odszyfrowywania wiadomości.
- Przedstawisz generowanie podpisu cyfrowego za pomocą metody RSA.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Szyfr RSA w języku Python". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

Faza wstępna:

1. Prowadzący wyświetla na tablicy interaktywnej zawartość sekcji „Wprowadzenie" i omawia cele do osiągnięcia w trakcie lekcji.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytania dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu, np.
   – czym charakteryzuje się szyfr RSA?
   – co można powiedzieć o nim w kontekście innych szyfrów?
   Chętni lub wybrani uczniowie udzielają na nie odpowiedzi.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające, prosi wybraną osobę o przedstawienie najważniejszych informacji z sekcji „Przeczytaj".
   Na forum klasy uczniowie analizują przedstawione w sekcji rozwiązania przykładów 1, 2 i 3. Następnie samodzielnie piszą i testują omawiane algorytmy na swoich komputerach.

2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie wspólnie analizują treść prezentacji multimedialnej, a następnie w parach rozwiązują problem 1. W kolejnym kroku porównują swoje rozwiązania z przedstawionym w filmie.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenia nr 1 i 2 z sekcji „Sprawdź się", a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.
4. Liga zadaniowa – uczniowie pracując w parach, wykonują ćwiczenia nr 3 i 4 i z sekcji „Sprawdź się", a następnie dzielą się swoimi wynikami przez porównywanie napisanego kodu z inną grupą, która również zakończyła zadanie.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.
2. Na koniec zajęć nauczyciel prosi uczniów o rozwinięcie zdania: „Na dzisiejszych zajęciach nauczyłam/łem się jak…".

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 5 z sekcji „Sprawdź się".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.

Wskazówki metodyczne:

- Treści w sekcji „Przeczytaj" można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.