# Porównanie wydajności kryptografii symetrycznej i asymetrycznej

Kacper Ochotny, s175481
Mateusz Nieścier, s175778
Tomasz Piechocki, s175690

## 1 Wstęp teoretyczny

W komunikacji sieciowej należy chronić przesyłane dane poprzez szyfrowanie ich. Istnieje wiele algorytmów szyfrujących o różnej złożoności, podatności na złamanie szyfru bądź na propagację błędów oraz o różnych czasach kodowania i dekodowania w stosunku do wielkości danych wejściowych. Algorytmy szyfrujące można podzielić na dwa rodzaje - symetryczne i asymetryczne.

Algorytmy symetryczne wykorzystywane są najczęściej do szyfrowania wiadomości przed jej wysłaniem. Do zaszyfrowania i rozszyfrowania używany jest ten sam klucz, stąd pochodzi ich nazwa. Kodowanie symetryczne można dalej podzielić na kolejne dwa rodzaje - algorytmy blokowe i strumieniowe. Szyfry blokowe polegają na dzieleniu wiadomości na fragmenty ustalonej długości, które potem zostają zakodowane. Do szyfrowania blokowo są dostępne różne tryby pracy. Natomiast algorytmy strumieniowe kodują bit po bicie lub bajt po bajcie, zamiast dzielenia wiadomości na fragmenty.

Algorytmy asymetryczne wykorzystywane są w momencie, gdy nie ma możliwości bezpiecznej wymiany klucza między nadawcą a odbiorcą. Kodowanie asymetryczne wykorzystuje parę kluczy - jeden do zaszyfrowania, drugi do deszyfracji wiadomości. Najpierw odbiorca generuje parę kluczy - prywatny i publiczny, potem rozsyła klucz publiczny do potencjalnych nadawców. Gdy nadawca chce wysłać wiadomość do odbiorcy, szyfruje ją przy pomocy klucza publicznego otrzymanego od niego, a następnie wysyła. Odbiorca rozszyfrowuje wiadomość za pomocą klucza prywatnego, który przechowuje tylko odbiorca.

## 2 Wykorzystane narzędzia

Wszystkie skrypty napisaliśmy w Pythonie. Do porównywania wydajności wykorzystaliśmy implementacje algorytmów szyfrujących z biblioteki PyCryptodome. Stosowane były tylko podstawowe metody i tryby z pominięciem ważnych w programistycznej praktyce aspektów jak np. funkcje hashujące. Algorytmy, które wykorzystaliśmy to:

- AES oraz 3DES, jako algorytmy symetryczne blokowe
- Salsa20, jako algorytm strumieniowy
- RSA w schemacie PCKS#1_v1.5, jako algorytm asymetryczny

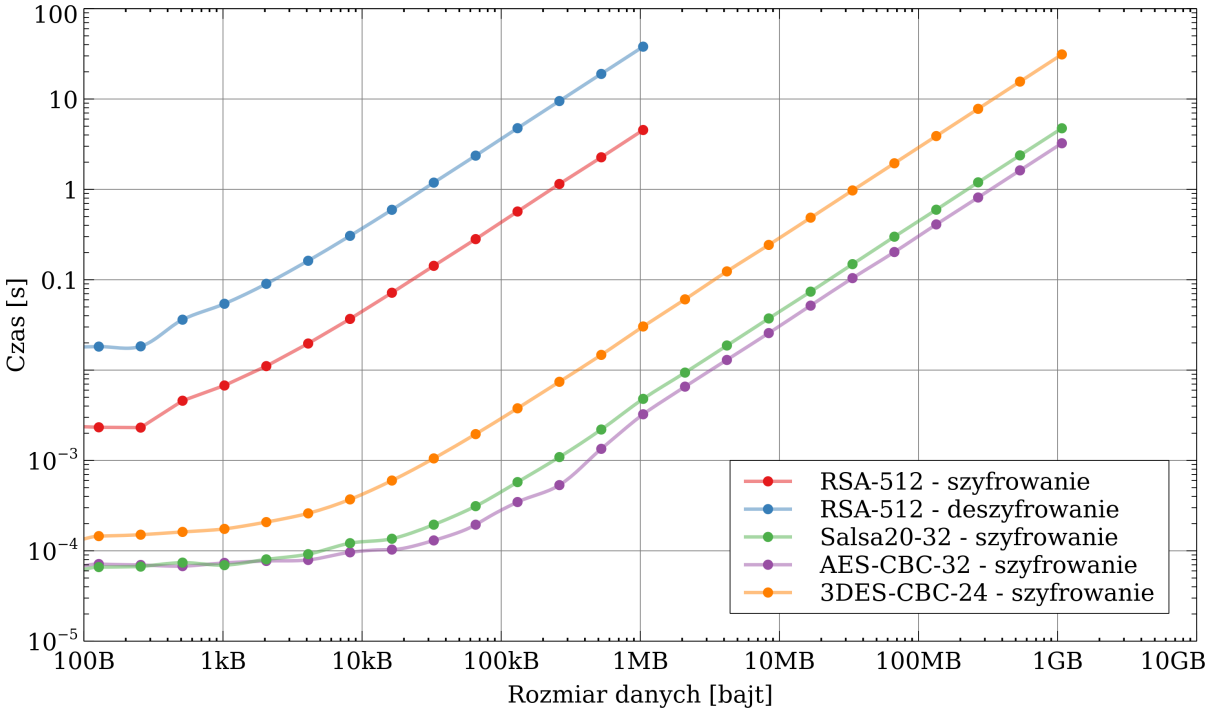
## 3 Sposób naszego działania

Zaczęliśmy od stworzenia makr do łatwego wywołania poszczególnych algorytmów. Znajdują się one w pliku ciphers.py. Następnie wygenerowaliśmy ciągi losowych znaków jednobajtowych o różnych długościach, od 32 B do 1 GiB. Dla każdego z algorytmów wykonaliśmy serię szyfrowań i rozszyfrowań dla tych ciągów znaków, jednocześnie mierząc czas każdego szyfrowania. Następnie zapisywaliśmy rezultaty do plików *.csv, które znajdują się w folderze /csv. Skrypt wykonujący to dla wszystkich wybranych przez

nas algorytmów to times.py[1]. Warto tutaj dodać, że dla niektórych algorytmów nie wykonaliśmy testów dla najdłuższych ciągów, bo trwałoby to zbyt długo. Dodatkowo, algorytm RSA może jednocześnie zaszyfrować tylko pewną liczbę znaków, więc musieliśmy własnoręcznie podzielić dłuższe wiadomości przed kodowaniem.

# 4 Wyniki

W folderze /plots znajdują się porównania czasów wygenerowanych wcześniej w różnych kombinacjach. Poniżej znajduje się wykres porównujący wszystkie algorytmy ze sobą. Postanowiliśmy większość wykresów umieścić na osiach logarytmicznych, żeby były widoczne rezultaty dla różnych rzędów danych wejściowych, gdyż wydajność algorytmu asymetrycznego jest zdecydowanie gorsza.



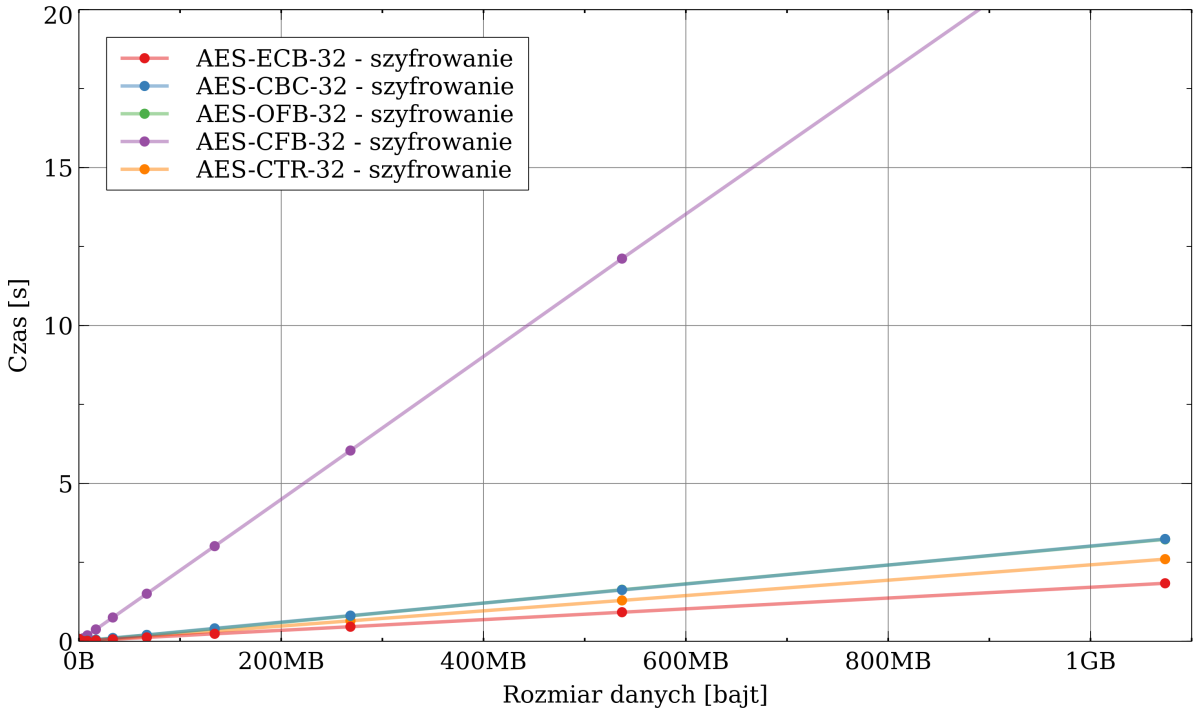

## 4.1 Wnioski

- Algorytm asymetryczny RSA szybko przestaje się nadawać do szyfrowania i rozszyfrowywania dłuższych ciągów znaków, jest on wolniejszy o około 4-5 rzędów od algorytmów symetrycznych.
- 3DES jest wolniejszy niż AES i Salsa20, o około 1 rząd, i także szybciej traci efektywność czasową, natomiast jest on starszym algorytmem.
- AES oraz Salsa20 mają podobną wydajność, gdzie algorytm blokowy jest niewiele szybszy.
- Różnica w czasach szyfrowania i deszyfrowania dla algorytmu RSA jest w okolicy jednego rzędu. Dla pozostałych algorytmów czas szyfrowania i rozszyfrowania jest bardzo podobny.
- Dla badanych algorytmów symetrycznych długość klucza nie wpływała znacząco na prędkość szyfrowania, jednakże w przypadku RSA wraz ze wzrostem długości klucza, rósł także czas wykonywania algorytmu.

[1]W tym pliku znajduje się wiele praktycznie identycznych fragmentów kodu, gdyż zmienia się tylko algorytm

# 5 Tryby szyfrów blokowych

Ostatnim elementem naszego tematu było zbadanie trybów dla algorytmów blokowych. Zbadaliśmy podstawowe tryby, czyli: ECB - Electronic CodeBook; CBC - Cipher Block Chaining; CFB - Cipher Feedback; OFB - Output Feedback oraz CTR - Counter. Do zbadania wydajności wykorzystaliśmy oczywiście implementacje z biblioteki. Rezultaty znajdują się na poniższym wykresie (CBC i OFB się praktycznie pokrywają).




## 5.1 Wnioski

- ECB jest najszybszym trybem dla obu algorytmów, jest on też najprostszym trybem.
- Dla AES najdłużej wykonywał się CBC (około 10 razy wolniejszy niż inne tryby), natomiast dla 3DES - CFB (nawet 20-krotnie wolniejszy niż ECB, i około 10-krotnie od pozostałych).
- W przypadku AES tryby CFB, OFB i CTR wykonują się w względnie podobnym czasie. Dla 3DES takimi trybami są CBC, OFB oraz CTR.

## 5.2 Sposób działania i cechy trybów, propagacja błędów

Jeszcze nie jesteśmy w stanie przewidzieć, w jakim stopniu uda nam się przedstawić szczegóły trybów algorytmów blokowych w danym nam czasie na prezentację. By pokazać jak one wyglądają stworzyliśmy klasę CustomModes na podstawie najprostszego trybu ECB. Oczywiście okazało się, że nasza wysokopoziomowa implementacja jest skrajnie powolna, jednak wydaje nam się, że może posłużyć do pokazania podczas prezentacji co trzeba wykonać w każdym z trybów. Dodatkowo chcieliśmy ukazać tam wielowątkowość w miejscach, w których jest to możliwe, choć niestety z powodów ograniczeń Pythona (Global Interpreter Lock) implementacje wielowątkowe okazały się wolniejsze.