

# Kryptografia prezentacja

## Cel projektu:

Ogólnym celem projektu jest zrozumienie i demonstracja mechanizmów bezpieczeństwa algorytmu SHA-3 (Keccak). Chodzi o to, aby nie tylko opisać teorię, ale udowodnić w praktyce, dlaczego ten standard jest bezpieczny. Realizujemy to poprzez własnoręczne zaprogramowanie algorytmu, wizualizację tego, jak skutecznie "miesza" on dane (efekt lawinowy), oraz próby jego przełamania w warunkach testowych.

## Zrealizowane:

- architektura stanu: poprawnie zaimplementowana macierz 5x5xW
- zaimplementowane operacje liniowe:
  - Theta - mieszanie kolumn
  - Rho i Pi - permutacje
- zaimplementowane operacje nieliniowe:
  - Chi - kluczowa nieliniowość
- zaimplementowane stałe rundy:
  - Iota oraz generator stałych rc
- szkielet klasy Keccak z metodami wchłaniania i wyciskania
- implementacja funkcji padding\_i\_wchlanianie do obsługi specyficznego paddingu SHA-3 (10\*1)
- eksperyment lawinowy: funkcja zmieniająca 1 bit wiadomości i mierząca różnice na wyjściu
- wizualizacja: aplikacja gui generuje wykres (liczba rund vs odległość Hamminga) przy użyciu **matplotlib**
- analiza parametrów: gui pozwala na zmianę parametrów w - wymagane do analizy bezpieczeństwa/wydajności

## Napotkane problemy

### Reprezentacja stanu

- Keccak operuje na stanie 1600-bitowym, w dokumentacji opisanej jako macierz 5x5xW
- opcja A: pomiędzy trzymaniem stanu jako tablicy 25 liczb 64-bitowych (lanes) - bardzo szybkie operacje bitowe ale trudne do debugowania
- opcja B: trzymanie stanu jako listy bitów - wybrane

### Padding

- SHA-3 wymaga specyficznego paddingu 10\*1
- jak obsłużyć sytuację gdy brakuje tylko 1 bajtu?
- rozróżniamy ten jako przypadek jako if needed == 1 i dodając bajt 0x86 który łączy bit początku i końca paddingu w jednym bajcie

## Dylemat wizualizacji

- celem projektu jest pokazanie dyfuzji i efektu lawinowego
- stanem jest trójwymiarowa kostka - jak pokazać żeby było czytelne dla użytkownika?
- zamiast pokazywać mapę cieplną całej macierzy 5x5, zdecydowaliśmy się na wykres liniowy Odległości Hamminga (ile bitów się różni w funkcji rund)
- bardziej czytelne - użytkownik od razu widzi czy krzywa dąży do 50% (idealna dyfuzja)

## Do zrealizowania:

### Implementacja ataku kolizji

- w obecnym kodzie brakuje skryptu próbującego znaleźć kolizję
- napiszemy skrypt który będzie szukał kolizji dla zredukowanej wersji (np.: 2/3 rundy) i małego stanu (w=8 lub w=16)
- metoda: algorytm urodzinowy - generujemy losowe wejścia, zapisujemy hashe w słowniku, sprawdzamy czy nowy hash już tam istnieje

## Oficjalne testy

- w obecnym kodzie brakuje mechanizmów do udowodnienia poprawności
- cel: pobranie plików testowych NIST, napiszemy skrypt który parsuje ten plik, przepuszcza Msg przez naszą klasę KeccakSponge i porównuje wynik z MD z pliku

## Optymalizacja kodu

- w prawdziwym świecie Keccak operuje na 64-bitowych liczbach całkowitych (gdy w=64)
- zamiast pętli po z w funkcji theta czy chi, można użyć operacji bitowych (^, &, ~)
- uwaga: musimy uważać na kolejność bajtów

## Wnioski z GUI

- jak zmiana 'w' wpływa na szybkość dyfuzji - analiza na większej ilości przypadków

## Skrypt KAT

- do walidacji zgodności z wektorami NIST

## **Atak Birthday Attack**

- do znajdowania kolizji dla zredukowanych rund/małego stanu
- opisanie różnic SHA-2 vs SHA-3