

Martyna Wróbel¹

Politechnika Opolska

dr inż. Wiktor Sędkowski²

Politechnika Opolska; Nokia

PRZEGLĄD TECHNIK HASZOWANIA W ZABEZPIECZANIU MODELI AI I ROZPOZNAWANIU ATAKÓW TYPU DATA POISONING PRZY UŻYCIU DETEKCJI KOLIZJI HASZY

OVERVIEW OF HASHING TECHNIQUES FOR SECURING AI MODELS AND RECOGNIZING DATA POISONING ATTACKS USING HASH COLLISION DETECTION

Streszczenie

Praca koncentruje się na bezpieczeństwie modeli sztucznej inteligencji (AI) oraz wykorzystaniu funkcji skrótu do ochrony integralności danych treningowych przed atakami typu *data poisoning*. Modele AI, szczególnie w zastosowaniach krytycznych, są narażone na manipulacje danymi, które mogą prowadzić do błędnych decyzji systemu. Funkcje skrótu, takie jak SHA-256 czy BLAKE2, pozwalają wykrywać nawet

¹ <https://orcid.org/0009-0003-5382-314X>, e-mail: ma.wrobel@student.po.edu.pl

² <https://orcid.org/0000-0002-4543-0499>, e-mail: w.sedkowski@po.edu.pl

Wróbel, M., Sędkowski, W. (2026). Przegląd technik haszowania w zabezpieczaniu modeli AI i rozpoznawaniu ataków typu data poisoning przy użyciu detekcji kolizji haszy. W: A. Siedlecka, M. Pyra (red.), *AI 360°: innowacje i zastosowania sztucznej inteligencji w gospodarce, społeczeństwie i nauce* (s. 161-180). Biała Podlaska: Akademia Bialska im. Jana Pawła II. https://doi.org/10.29316/9788368103366_9

minimalne zmiany w danych, dzięki czemu stanowią ważny element systemów zabezpieczeń. W badaniach porównano wydajność algorytmów MD5, SHA-1, SHA-256, BLAKE2 i bcrypt w językach Python i Rust, analizując zależność między szybkością a poziomem bezpieczeństwa. Wyniki pokazały, że lekkie algorytmy działają błyskawicznie, lecz są mniej odporne, natomiast bcrypt – mimo większych opóźnień – oferuje najwyższy poziom ochrony haseł. Wnioski podkreślają konieczność stosowania silnych funkcji skrótu do weryfikacji integralności danych oraz adaptacyjnych algorytmów (np. bcrypt) w systemach przechowujących dane wrażliwe.

Słowa kluczowe: cyberbezpieczeństwo, kryptografia, sztuczna inteligencja (AI), modele AI

Abstract

The study focuses on AI model security and the use of hash functions to protect the integrity of training data against *data poisoning* attacks. AI models, especially in critical applications, are exposed to data manipulation that can lead to incorrect or harmful system behavior. Hash functions such as SHA-256 and BLAKE2 can detect even minimal data alterations, making them key elements of security systems. The research compared the performance of MD5, SHA-1, SHA-256, BLAKE2, and bcrypt implemented in Python and Rust, analyzing the relationship between speed and security. Results showed that lightweight algorithms are extremely fast but less secure, while bcrypt – though slower – provides strong password protection. The conclusions emphasize the importance of using robust hash functions for data integrity and adaptive algorithms (like bcrypt) for safeguarding sensitive information.

Keywords: cybersecurity, cryptography, artificial intelligence (AI), AI models

Wstęp

Zabezpieczanie modeli sztucznej inteligencji (AI) stanowi jedno z ważniejszych wyzwań w obszarze systemów autonomicznych i analitycznych, zwłaszcza wobec faktu, że ich wykorzystanie ma coraz większe znaczenie w sektorach takich jak medycyna, transport, finanse czy infrastruktura krytyczna (Biggio, Roli, 2018). W przypadku uczenia maszynowego, bezpieczeństwo nie ogranicza się jedynie do ochrony kodu czy fizycznej infrastruktury, ale obejmuje przede wszystkim zapewnienie integralności wykorzystywanych danych oraz odporności modeli na celowe manipulacje, czego wyrazem są liczne podatności ujawnione w ostatnich latach.

Modele AI mogą być podatne na rozmaite typy ataków, z których jednym z najbardziej rozpoznawalnych jest *data poisoning* – zjawisko polegające na celowym wprowadzeniu złośliwych próbek do zbioru treningowego w taki sposób, by wytrenowany model działał niezgodnie z oczekiwaniami w określonym scenariuszu. Typowym przykładem są ataki skierowane na systemy rozpoznawania obrazów, gdzie niewielka liczba zmanipulowanych przykładów w zbiorze uczącym może całkowicie zmienić kategoryzację istotnych obiektów – np. w przypadku systemów automatycznego rozpoznawania znaków drogowych udało się wykazać, że wystarczy zmienić kilka pikseli w obrazie, by model błędnie rozpoznawał znak „Stop” jako „Ograniczenie prędkości” (Gu i in., 2017).

Inny rodzaj zagrożeń stanowią ataki typu *adversarial*, polegające na niewielkich, często niedostrzegalnych modyfikacjach sygnału wejściowego, które powodują błędne działanie modelu, nawet jeśli został wytrenowany na poprawnych danych. W literaturze naukowej wykazano, że głębokie sieci neuronowe mogą być nie tylko podatne na ataki ukierunkowane, ale też odtwarzalne w różnych środowiskach i warstwach sprzętowych (Kurakin i in., 2017). W sferze NLP (przetwarzania języka naturalnego) problem stanowią tzw. *backdoored models*, czyli modele z wszczepionym „tylnym wejściem”, działające poprawnie w normalnych warunkach, lecz po wykryciu określonego wzorca w danych wejściowych generujące złośliwe lub błędne wyjścia.

Wspomniane zagrożenia występują zarówno w modelach klasyfikacji obrazów, jak i przetwarzania tekstu czy systemach rekomendacyjnych, co podkreśla uniwersalność problemu bezpieczeństwa AI. W odpowiedzi na te zagrożenia opracowywane są zaawansowane procedury kontroli integralności danych, w których to funkcje skrótu odgrywają istotną rolę – umożliwiają szybkie wykrycie nieautoryzowanych zmian w zbiorach danych lub parametrach modeli, stanowiąc podstawowy element procesu hardeningu infrastruktury uczenia maszynowego.

Podstawy teoretyczne funkcji skrótu

Funkcja skrótu (ang. *hash function*) to specjalny rodzaj funkcji matematycznej przekształcającej dane wejściowe dowolnej długości w ciąg o ustalonej, zwykle krótkiej długości, zwany skrótem lub haszem. Kluczową cechą

funkcji skrótu jest to, że dowolne, nawet bardzo długie dane źródłowe, takie jak plik czy ciąg znaków, są mapowane na stałej długości ciąg bitów, co umożliwia efektywne porównywanie i weryfikację integralności informacji (Stallings, 2017).

Podstawowe właściwości funkcji skrótu, które warunkują jej użyteczność w zastosowaniach kryptograficznych i bezpieczeństwa, to: jednokierunkowość, odporność oraz występowanie efektu lawinowego. Jednokierunkowość oznacza, że dla znanego skrótu jest praktycznie niemożliwe wyliczenie oryginalnych danych wejściowych. Ta cecha chroni przed odtwarzaniem lub ujawnianiem oryginalnej informacji na podstawie samego hasza. Odporność na kolizje zapewnia, że jest niepraktyczne znalezienie dwóch różnych danych wejściowych, które dają ten sam skrót. Dzięki temu można mieć pewność, że dany hasz jednoznacznie identyfikuje konkretny zestaw danych, co jest fundamentalne dla detekcji manipulacji. Innymi słowy dla dwóch różnych wiadomości $m_1 \neq m_2$ powinno być obliczeniowo niemożliwe znalezienie skrótu takiego, że $H(m_1)=H(m_2)$. Efekt lawinowy polega na tym, że nawet minimalna zmiana w danych wejściowych (np. zmiana pojedynczego bitu) powoduje zupełnie inną wartość skrótu, zwykle zmieniając większość bitów w wyniku. Ta właściwość zwiększa zdolność do wykrywania nawet niewielkich modyfikacji danych. Funkcje skrótu, które posiadają wszystkie trzy powyższe cechy, stanowią podstawy wielu protokołów kryptograficznych, systemów weryfikacji integralności danych, podpisów cyfrowych czy mechanizmów uwierzytelniania (Katz, Lindell, 2014). W kontekście zabezpieczania modeli AI szczególnie ważna jest odporność na kolizje, ponieważ umożliwia wykrywanie prób fałszowania lub wstrzykiwania zmanipulowanych danych treningowych (Tabela 1).

Tabela 1. Wykaz użytych algorytmów haszujących

Algorytm	Jak działa w skrócie	Długość hasz'a	Bezpieczeństwo	Główne zastosowanie
MD5	Bardzo szybki, ale dziś łatwo złamać (można znaleźć dwie różne dane dające ten sam hasz).	32 znaki (128 bitów)	Słaby	Sprawdzanie integralności plików (nie do haseł!)
SHA-1	Ulepszony MD5, ale też już przestarzały – można znaleźć kolizje.	40 znaków (160 bitów)	Średni	Stare systemy, podpisy cyfrowe (już niezalecany)

Algorytm	Jak działa w skrócie	Długość hasz'a	Bezpieczeń- stwo	Główne zastosowanie
SHA-256	Mocniejszy i nowszy z rodziny SHA-2; trudny do złamania, popularny.	64 znaki (256 bitów)	Bardzo dobry	Bezpieczeństwo danych, podpisy cyfrowe, certyfikaty
BLAKE2	Szybszy niż SHA-256, a równie bezpieczny; nowoczesny algorytm.	64 znaki (domyślnie)	Bardzo dobry	Nowe aplikacje, kryptografia, blockchain
bcrypt	Inny typ – wolny z premedytacją, co utrudnia łamanie haseł siłowo. Dodatkowo używa soli (losowych danych) przy każdym hasle.	ok. 60 znaków	Najlepszy do haseł	Przechowywanie haseł użytkowników

Źródło: Aumasson, 2024; Aumasson i in., 2017; Rietta, 2016.

Metodologia badań eksperymentalnych

Opis środowiska testowego (sprzęt, oprogramowanie, wersje bibliotek)

Testy przeprowadzono na sprzęcie klasy biurowej, aby możliwie wiernie odzwierciedlić warunki pracy typowego użytkownika końcowego. Do badań wykorzystano trzy komputery przenośne o zróżnicowanej konfiguracji sprzętowej, co umożliwiło analizę wpływu parametrów technicznych na wyniki eksperymentów.

Pierwszy z nich to laptop MSI, wyposażony w procesor AMD Ryzen 5 7535HS, zintegrowany układ graficzny Radeon oraz dedykowaną kartę graficzną NVIDIA GeForce RTX 4060 z 8 GB pamięci VRAM. Urządzenie posiada 16 GB pamięci operacyjnej RAM.

Drugi laptop, marki ACER, wykorzystuje procesor Intel Core i5 jedenastej generacji oraz dedykowaną kartę graficzną NVIDIA GeForce RTX 3050 z 4 GB pamięci VRAM. Wyposażony jest również w 16 GB pamięci RAM, co zapewnia stabilną wydajność w zastosowaniach wielozadaniowych.

Trzeci z opisywanych komputerów to Apple MacBook Air z procesorem Apple M1, zintegrowanym układem graficznym oraz 8 GB zintegrowanej pamięci RAM. Charakteryzuje się wysoką efektywnością energetyczną i optymalizacją pod kątem środowiska macOS.

Programy wykorzystane w procesie eksperymentalnym: Visual Studio Code, PowerShell, PyCharm.

Do realizacji eksperymentów zastosowano dwa języki programowania – Python oraz Rust. Wybór tych języków wynikał z chęci porównania ich wydajności, sposobu implementacji oraz efektywności funkcji skrótu (haszy) w językach wysokiego poziomu o odmiennych paradygmatach programistycznych i charakterystyce działania.

Implementacja skryptu badawczego

W celu przeprowadzenia pomiarów wydajności różnych funkcji skrótu przygotowano dwa niezależne skrypty badawcze – jeden w języku Python, a drugi w Rust. Oba programy realizują tę samą logikę pomiarową, różniąc się jedynie składnią i sposobem zarządzania czasem wykonania. Takie podejście umożliwia porównanie nie tylko samych algorytmów haszujących, lecz również potencjalnych różnic wynikających z implementacji w odmiennych językach programowania.

Generowanie danych wejściowych

W obu programach zastosowano pseudolosowe generowanie ciągów znaków o zadanej długości. Ciągi te tworzone są z zestawu znaków alfanumerycznych (A-Z, a-z, 0-9). Wartość generowanego ciągu nie ma znaczenia dla wyników eksperymentu – istotna jest jedynie jego długość, wpływająca na czas obliczenia funkcji skrótu. Wersja Pythona korzysta z modułu `random`, natomiast wersja w języku Rust wykorzystuje bibliotekę `rand::Rng`.

Zakres badanych algorytmów

Do testów wydajności wybrano pięć popularnych funkcji skrótu.

- Implementacja w języku Python

Skrypt w Pythonie został zbudowany w oparciu o biblioteki: `hashlib`, `bcrypt`, `random`, `string`, `time`, `pandas` oraz `numpy`. Program umożliwia użytkownikowi określenie liczby testowanych ciągów oraz zakresu ich długości. Dla

każdego algorytmu i każdej długości ciągu wykonywana jest wielokrotna seria pomiarów czasu. Czas mierzony jest przy użyciu funkcji `time.perf_counter()`, która oferuje wysoką precyzję pomiaru w sekundach z ułamkami milisekund. Dane z pomiarów są następnie zapisywane do obiektu `DataFrame`, a na ich podstawie obliczane są średnie wartości czasu haszowania dla poszczególnych długości i algorytmów. Wyniki prezentowane są w formie tabelarycznej (`pivot table`) w konsoli, z dokładnością do sześciu miejsc po przecinku.

– Implementacja w języku Rust

Analogiczny skrypt w języku Rust wykorzystuje biblioteki `rand::std`, `md5`, `sha1`, `sha256`, `blake2` oraz `bcrypt`. Pomiar czasu realizowany jest z wykorzystaniem struktury `Instant` z modułu `std::time`, która pozwala na dokładne określenie czasu trwania operacji w sekundach.

Program testuje trzy różne długości wejściowych ciągów znaków (np. 10, 30 i 60 znaków), a dla każdego algorytmu wykonuje kilka powtórzeń w celu uśrednienia wyników. Średni czas obliczeń dla każdej funkcji haszującej obliczany jest jako średnia arytmetyczna z uzyskanych czasów (`as_secs_f64()`), po czym prezentowany w czytelnej formie tabelarycznej w konsoli.

Sposób przeprowadzenia pomiarów

Każdy eksperyment przebiega według następującej sekwencji kroków:

1. Generacja losowego ciągu znaków o określonej długości.
2. Rozpoczęcie pomiaru czasu (`Instant::now()` / `time.perf_counter()`).
3. Obliczenie skrótu za pomocą wybranego algorytmu.
4. Zatrzymanie pomiaru i zapis czasu wykonania.
5. Powtórzenie operacji dla wszystkich algorytmów i długości.

Tak uzyskane czasy dla wszystkich algorytmów są następnie uśredniane, co pozwala zminimalizować wpływ jednorazowych fluktuacji środowiska uruchomieniowego (np. obciążenia procesora). Wyniki operacji przedstawiono na Rysunkach 1-6.

Algorithm	BLAKE2	MD5	SHA-1	SHA-256	bcrypt
String Length					
10	0.00000983	0.00001777	0.00000732	0.00000713	0.45758470
30	0.00000208	0.00000158	0.00000143	0.00000144	0.45690823
60	0.00000202	0.00000174	0.00000157	0.00000153	0.44477743

Rysunek 1. Wyniki pomiarów na komputerze MSI w środowisku Python

Źródło: opracowanie własne.

```
PS C:\Users\zyx\Desktop\pracanaukowa\hashrust> cargo run main.rs
Compiling hash v0.1.0 (C:\Users\zyx\Desktop\pracanaukowa\hashrust)
Finished `dev` profile [unoptimized + debuginfo] target(s) in 3.71s
Running `target\debug\hash.exe main.rs`
```

Length	MD5	SHA1	SHA256	BLAKE2	BCRYPT
10	0.000011	0.000005	0.000013	0.000045	0.625615
30	0.000004	0.000003	0.000009	0.000025	0.624331
60	0.000005	0.000006	0.000012	0.000028	0.621467

Rysunek 2. Wyniki pomiarów na komputerze MSI w środowisku Rust

Źródło: opracowanie własne.

Algorithm	BLAKE2	MD5	SHA-1	SHA-256	bcrypt
String Length					
10	0.00036406	0.00000234	0.00000120	0.00000114	0.24879453
30	0.00000079	0.00000094	0.00000088	0.00000089	0.23504114
60	0.00000080	0.00000103	0.00000092	0.00000095	0.21499794

Rysunek 3. Wyniki pomiarów na komputerze ACER w środowisku Python

Źródło: opracowanie własne.

```
Finished `dev` profile [unoptimized + debuginfo] target(s) in 1.38s
Running `target\debug\hash_benchmark.exe`
```

Length	MD5	SHA1	SHA256	BLAKE2	BCRYPT
10	0.000008	0.000004	0.000012	0.000037	0.587911
30	0.000004	0.000003	0.000010	0.000026	0.583712
60	0.000006	0.000006	0.000011	0.000024	0.597863

Rysunek 4. Wyniki pomiarów na komputerze ACER w środowisku Rust

Źródło: opracowanie własne.

Algorithm	BLAKE2	MD5	SHA-1	SHA-256	bcrypt
String Length					
10	0.00000046	0.00000080	0.00000044	0.00000043	0.23301162
30	0.00000038	0.00000046	0.00000042	0.00000040	0.23308694
60	0.00000038	0.00000057	0.00000042	0.00000041	0.23274149
Process finished with exit code 0					

Rysunek 5. Wyniki pomiarów na komputerze MacBook w środowisku Python

Źródło: opracowanie własne.

Finished	`dev` profile [unoptimized + debuginfo] target(s) in 0.08s				
Running	`/Users/waleria/Desktop/hashrust/target/debug/hash`				
Length	MD5	SHA1	SHA256	BLAKE2	BCRYPT
10	0.000016	0.000006	0.000036	0.000051	1.487426
30	0.000008	0.000006	0.000018	0.000037	1.476239
60	0.000015	0.000013	0.000021	0.000031	1.475577

Rysunek 6. Wyniki pomiarów na komputerze MacBook w środowisku Rust

Źródło: opracowanie własne.

Prezentacja wyników

Wyniki pomiarów wyświetlane są w formie tabelarycznej, gdzie w wierszach znajdują się długości testowanych ciągów, a w kolumnach – czasy wykonania dla poszczególnych funkcji skrótu. Dzięki temu możliwe jest wizualne porównanie wydajności algorytmów. Uzyskane dane w dalszej części pracy zostały wykorzystane do analizy statystycznej i opracowania wykresów zależności czasu haszowania od długości ciągu wejściowego.

Wyniki pomiarów

Zestawienie czasów obliczeń dla różnych algorytmów (Tabele 2-3, Rysunek 7).

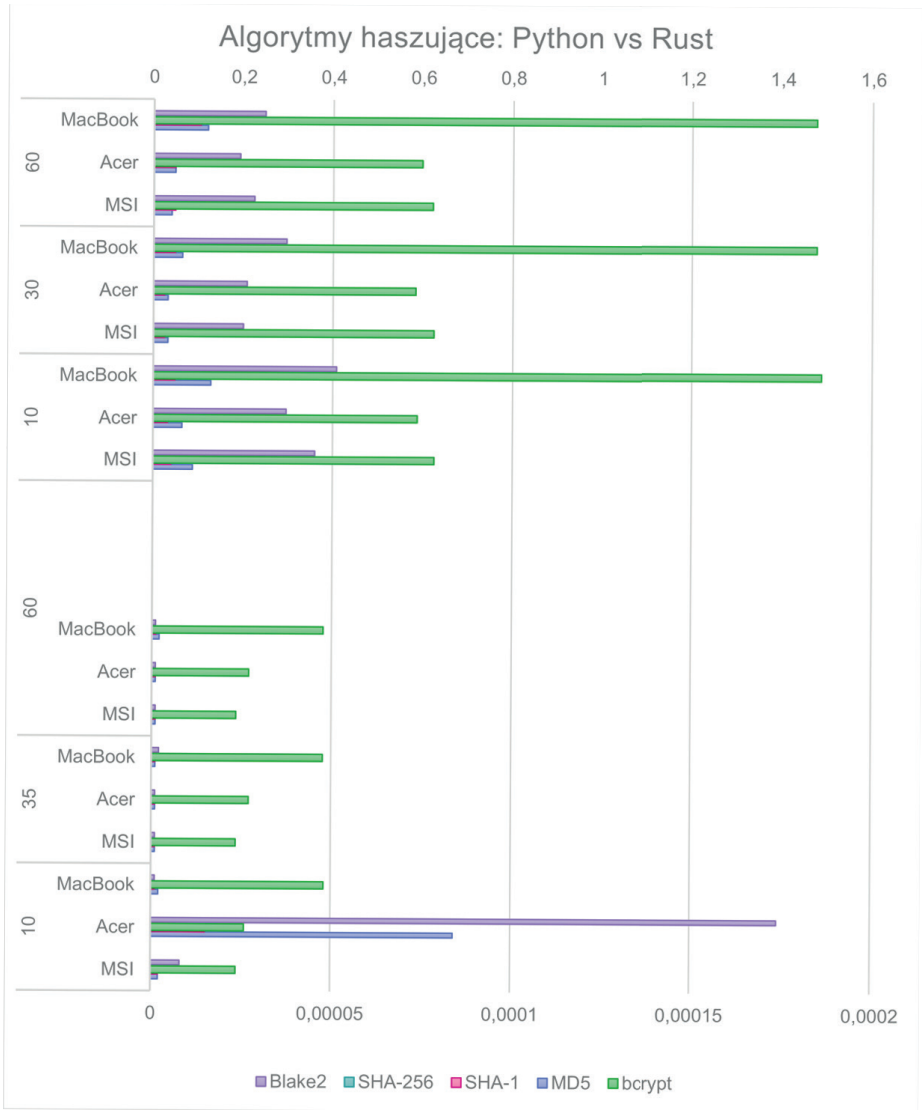
Tabela 2. Wyniki pomiarów dla środowiska Python

Długość ciągu	Platforma	Czas wykonania funkcji skrótu				
		MD5	SHA-1	SHA-256	Blake2	bcrypt
10	MSI	0.00001777	0.00000732	0.00000713	0.00000983	0.45758470
	ACER	0.00000234	0.00000120	0.00000114	0.00036406	0.24879453
	MacBook	0.00000080	0.00000044	0.00000043	0.00000046	0.23301162
30	MSI	0.00000158	0.00000143	0.00000144	0.00000208	0.45690823
	ACER	0.00000094	0.00000088	0.00000089	0.00000079	0.23504114
	MacBook	0.00000046	0.00000042	0.00000040	0.00000038	0.23308694
60	MSI	0.00000174	0.00000157	0.00000153	0.00000202	0.44477743
	ACER	0.00000103	0.00000092	0.00000095	0.00000080	0.21499794
	MacBook	0.00000057	0.00000042	0.00000041	0.00000038	0.23274149

Tabela 3. Wyniki pomiarów dla środowiska Rust

Długość ciągu	Platforma	Czas wykonania funkcji skrótu				
		MD5	SHA-1	SHA-256	Blake2	bcrypt
10	MSI	0.000011	0.000005	0.000013	0.000045	0.625615
	ACER	0.000008	0.000004	0.000012	0.000037	0.587911
	MacBook	0.000016	0.000006	0.000036	0.000051	1.487426
30	MSI	0.000004	0.000003	0.000009	0.000025	0.624331
	ACER	0.000004	0.000003	0.00001	0.000026	0.583712
	MacBook	0.000008	0.000006	0.000018	0.000037	1.476239
60	MSI	0.000005	0.000006	0.000012	0.000028	0.621467
	ACER	0.000006	0.000006	0.000011	0.000024	0.597863
	MacBook	0.000015	0.000013	0.000021	0.000031	1.475577

Źródło: opracowanie własne.



Rysunek 7. Porównanie czasu działania algorytmów haszujących wykonywane w środowiskach Python oraz Rust

Źródło: opracowanie własne.

Na podstawie precyzyjnych pomiarów czasu wykonania identycznego zestawu algorytmów haszujących (MD5, SHA-1, SHA-256, BLAKE2, bcrypt) na trzech platformach sprzętowych w dwóch środowiskach programistycznych, uzyskano następujące rezultaty.

Python

- Wydajność dla bcrypt jest stosunkowo stała przy rosnącej długości tekstu (10 → 60 znaków), co wskazuje na znaczący wpływ samej złożoności algorytmu, a nie długości danych.
- Dla większości lekkich algorytmów (MD5, SHA-1, SHA-256, Blake2) MacBook i ACER osiągają niemal identyczne czasy (~0.0000004-0.000002s), natomiast MSI w przypadku krótkich tekstów (10 znaków) wykazuje wyraźnie większe opóźnienia (np. MD5: 0.00001777 s).
- Najwolniejszym algorytmem we wszystkich konfiguracjach jest bcrypt, gdzie najkrótsze czasy uzyskuje MacBook (~0.233 s), a najdłuższe MSI (~0.457 s).

Rust

- Najszybsze wykonania lekkich algorytmów (MD5, SHA-1, SHA-256, Blake2) osiąga ACER i MSI, w granicach ~0.0000004-0.000011s. MacBook jest nieco wolniejszy, szczególnie przy SHA-256 (0.0000036 s dla krótkiego tekstu).
- Algorytm bcrypt pozostaje najbardziej wymagający: najkrótsze czasy uzyskuje ACER (~0.584 s), a najdłuższe MacBook (~1.487s).

Mimo że Rust jest językiem kompilowanym, charakteryzującym się wysoką wydajnością przy ogólnych obliczeniach, w tym konkretnym scenariuszu czas haszów dla krótkich stringów jest relatywnie krótki (rzędu mikrosekund). W takich przypadkach czas wykonywania kodu nie zależy głównie od języka, lecz od:

1. Przełączeń kontekstu i narzutów środowiskowych – inicjalizacja funkcji haszujących, wywołania bibliotek kryptograficznych, zarządzanie pamięcią mogą stanowić większą część czasu niż same obliczenia.
2. Optymalizacji niskopoziomowych w bibliotekach Pythona – np. algorytmy haszujące w Pythonie korzystają z dobrze zoptymalizowanych implementacji C, które działają szybciej niż ręcznie zaimplementowany Rust w tym samym teście.
3. Krótki czas operacji – przy tak mikroskopijnych czasach pojedynczych operacji (mikro- lub nanosekundy) nawet minimalne różnice

w wywołaniu funkcji, zarządzaniu pamięcią czy synchronizacji mogą sprawić, że Rust wydaje się wolniejszy od Pythona, mimo że w dużych i długotrwałych obliczeniach przewaga Rusta byłaby znacząca.

Zależność między szybkością a bezpieczeństwem

1. Szybkie algorytmy (MD5, SHA-1, SHA-256, Blake2):
 - charakteryzują się bardzo krótkim czasem obliczeń (~mikrosekundy dla pojedynczego stringa);
 - z punktu widzenia systemu szyfrowania haseł lub generowania tokenów, ich szybkość jest korzystna dla wydajności, ale jednocześnie ułatwia atak brute-force. Atakujący z odpowiednim sprzętem (GPU lub FPGA) może wykonywać miliony prób na sekundę;
 - dlatego algorytmy te są bezpieczne dla integralności danych, ale nie nadają się do bezpiecznego przechowywania haseł bez dodatkowych mechanizmów spowalniających (salting, key stretching, iteracje).
2. Wolniejsze, algorytmy (bcrypt, scrypt, Argon2):
 - wymagają kilkudziesięciu do kilkuset milisekund na pojedyncze haszowanie, co znacząco ogranicza liczbę prób możliwych do wykonania przez atakującego w jednostce czasu;
 - przykładowo, w przeprowadzonych pomiarach bcrypt wymaga ~0,187 s (MSI, Python) do ~1,48 s (MacBook, Rust) na jedną próbę, co redukuje potencjalną prędkość ataku brute-force o kilka rzędów wielkości w porównaniu z MD5 czy SHA-256;
 - takie algorytmy są projektowane specjalnie pod kątem bezpieczeństwa haseł, gdzie celem jest maksymalne ograniczenie liczby prób w określonym czasie przy zachowaniu akceptowalnej wydajności dla użytkownika.

Wpływ mocy obliczeniowej atakującego

- Liczba prób, jaką atakujący może wykonać w jednostce czasu, rośnie proporcjonalnie do dostępnej mocy obliczeniowej (CPU, GPU, FPGA).

- Dla algorytmów szybkich, wzrost mocy obliczeniowej powoduje niemal liniowy wzrost liczby prób, co znacząco obniża bezpieczeństwo.
- Dla algorytmów spowolnionych, nawet duży wzrost mocy obliczeniowej daje ograniczoną poprawę liczby prób na sekundę, ponieważ projekt algorytmu wprowadza narzut czasowy, który jest trudny do obejścia.

Analiza bezpieczeństwa

Jak wspomniano wcześniej, najważniejszą właściwością kryptograficzną funkcji skrótu jest odporność na kolizje – dla dwóch różnych wiadomości $m_1 \neq m_2$ powinno być obliczeniowo niemożliwe znalezienie takich danych wejściowych, dla których zachodzi równość $H(m_1)=H(m_2)$. Powiązanym z tą właściwością parametrem wpływającym na bezpieczeństwo jest długość skrótu, która determinuje przestrzeń możliwych wyników i tym samym odporność na ataki typu brute-force. Przykładowo, długość skrótu 128 bitów zapewnia 2^{128} możliwych wyników, co jednak w przypadku współczesnych mocy obliczeniowych nie gwarantuje już wystarczającej odporności kryptograficznej.

MD5, opracowany przez R. Rivesta w 1991 r., generuje skróty o długości 128 bitów. Już w 2004 roku wykazano możliwość generowania kolizji w czasie rzeczywistym, co spowodowało jego całkowitą dyskwalifikację jako funkcji kryptograficznej. „Wang przedstawia przepis na znajdowanie kolizji dla dwublokowych wiadomości. Znalezienie kolizji na komputerze klasy PC zajmuje około godziny” (Rodwald, 2013). Dziś MD5 uznawany jest wyłącznie za narzędzie kontrolne (np. do wykrywania błędów transmisji), ale nie do celów bezpieczeństwa.

SHA-1, opracowany przez NSA w 1995 r., generuje skróty 160-bitowe. W 2017 roku zespół z Google i CWI Amsterdam opublikował pierwszą praktyczną kolizję (projekt SHattered), co ostatecznie potwierdziło niewystarczający poziom bezpieczeństwa SHA-1 w zastosowaniach kryptograficznych. Obecnie funkcja ta jest wycofywana z użycia w certyfikatach SSL/TLS i systemach podpisów cyfrowych.

SHA-256, stanowiący część rodziny SHA-2, jest obecnie jednym z najczęściej stosowanych algorytmów funkcji skrótu w praktyce. Zapewnia długość skrótu 256 bitów i nie posiada znanych praktycznych ataków skutkujących

kolizją. Pomimo kilku teoretycznych analiz redukujących złożoność obliczeniową ataków (Nahiyen, 2024), poziom bezpieczeństwa SHA-256 pozostaje wysoki.

BLAKE2, opracowany w 2013 roku jako następca finalisty konkursu SHA-3 (BLAKE), łączy bardzo wysoki poziom bezpieczeństwa z efektywnością energetyczną i wydajnością implementacyjną. Wykazuje odporność porównywalną do SHA-3, przy czym działa znacznie szybciej w oprogramowaniu. BLAKE2 jest rekomendowany w wielu współczesnych projektach open source i protokołach kryptograficznych.

Bcrypt to funkcja bazująca na szyfrze blokowym Blowfish, zaprojektowana w 1999 r. specjalnie dla bezpiecznego przechowywania haseł. Jej głównym mechanizmem ochronnym jest możliwość konfigurowalnego spowolnienia obliczeń (tzw. *cost factor*), dzięki czemu ataki typu brute-force i słownikowe stają się nieopłacalne obliczeniowo. Bcrypt jest odporna na technologię GPU i ASIC, co czyni ją szczególnie skuteczną w ochronie haseł użytkowników.

Ewolucja funkcji skrótu od MD5 i SHA-1 do współczesnych konstrukcji, takich jak SHA-256 czy BLAKE2, obrazuje rosnące wymagania bezpieczeństwa w kryptografii. W praktyce, preferowanym wyborem są dziś funkcje SHA-256, SHA-3 lub BLAKE2.

Ataki typu *data poisoning* w kontekście AI

Ataki typu *data poisoning* stanowią jedne z najbardziej niebezpiecznych zagrożeń dla modeli sztucznej inteligencji, które opierają się na uczeniu maszynowym. Polegają one na celowym zanieczyszczeniu danych wykorzystywanych podczas trenowania modelu poprzez wprowadzanie do zbiorów treningowych próbek, które zostały zmanipulowane w taki sposób, aby model nauczył się błędnych lub szkodliwych zależności. Dzięki temu po wytrenowaniu model może wykazywać nieoczekiwane i często bardzo trudne do wykrycia błędy wywierające istotny wpływ na jego działanie i bezpieczeństwo.

Manipulacje danymi wejściowymi, mogą przyjmować różnorodne formy. Jednym z przypadków jest wstrzykiwanie złośliwych przykładów do zbioru treningowego, które mimo iż wyglądają na integralne, zawierają ukryte cechy mające na celu wywołanie nieprawidłowej klasyfikacji lub błędów decyzyjnych w modelu. Innym sposobem jest zmiana etykiet przypisanych do danych, tzw.

label flipping, gdzie odpowiedzi w zbiorze treningowym są celowo błędne, co prowadzi do utrwalenia niepoprawnych wzorców w modelu. Ponadto, istnieją subtelne metody manipulacji, które wprowadzają drobne i często niemożliwe do zauważenia przez człowieka zakłócenia.

Wpływ takich zmanipulowanych danych na proces trenowania i końcową jakość modelu jest znaczący. Nawet niewielka liczba skażonych danych może zafałszować całą przestrzeń uczącą, skutkując obniżeniem dokładności oraz ryzykiem powtarzalnych błędów przy określonych warunkach wejściowych. W skrajnych przypadkach model może być w pełni kontrolowany przez atakującego, który wykorzystuje zatrucie danych do uzyskania celowo błędnych wyników, na przykład w systemach bezpieczeństwa, gdzie wprowadzenie fałszywych próbek może pozwolić na obejście mechanizmów kontrolnych.

W obliczu tych zagrożeń funkcje skrótu zyskują na dużym znaczeniu jako podstawowe narzędzie ochrony integralności danych. Dzięki swojej właściwości jednokierunkowości, funkcje te umożliwiają szybkie i pewne wyliczenie unikatowych sygnatur (haszy) dla danych treningowych, które w razie jakiegokolwiek zmiany w zawartości ulegają radykalnej i nieprzewidywalnej zmianie. Odporność na kolizje gwarantuje, że praktycznie niemożliwe jest znalezienie dwóch różnych danych generujących ten sam hasz, co czyni funkcje haszujące skutecznym filtrem wykrywającym nieautoryzowaną ingerencję w dane. Monitoring i porównywanie haszy zbiorów treningowych pozwala na wczesne wykrywanie prób manipulacji i wprowadzenia potencjalnie niebezpiecznych próbek, co znacznie podnosi poziom bezpieczeństwa procesu trenowania modelu.

Powiązanie kolizji haszy z bezpieczeństwem modeli AI

Kolizje haszy są jednym z zagrożeń dla bezpieczeństwa zarówno tradycyjnych systemów kryptograficznych, jak i nowoczesnych modeli sztucznej inteligencji. W kontekście uczenia maszynowego naruszenie odporności funkcji skrótu może prowadzić do sytuacji, w której złośliwy podmiot wprowadza skażone dane do zbiorów treningowych, zachowując jednocześnie ich pozorną autentyczność. W wyniku kolizji – czyli sytuacji, w której dwie różne próbki danych generują ten sam hasz – atakujący jest w stanie ominąć mechanizmy weryfikacji integralności danych, które opierają się na unikalnych wartościach skrótów.

Jednakże, aby w pełni umotywić to zagrożenie w kontekście *data poisoning*, należy podkreślić, że choć dla kryptograficznie bezpiecznych funkcji skrótu kolizje są skrajnie mało prawdopodobne, to realne ryzyko pojawia się właśnie przy stosowaniu algorytmów uznanych za złamane w procesach weryfikacji integralności danych treningowych. Algorytmy MD5 i SHA1 celowo dobrano do weryfikacji integralności plików, ponieważ udowodniono możliwość generowania sprawnej kolizji – atakujący może w czasie rzeczywistym skonstruować złośliwy plik o identycznym skrócie, co czyni ochronę przed atakami *data poisoning* iluzoryczną. Analogicznie, SHA-1 został praktycznie złamany w 2017 roku przez atak SHAttered, który zademonstrował generację kolizji przy złożoności obliczeniowej znacznie poniżej teoretycznego progu bezpieczeństwa. Haszowanie każdego pliku z osobna przy użyciu kryptograficznie złamanych funkcji skrótu nie zapewnia wystarczającej gwarancji integralności w środowiskach narażonych na celową manipulację danymi. W związku z tym „detekcja kolizji haszy” nie jest proponowana jako samodzielny mechanizm wykrywania skażenia, lecz jako ilustracja luki, która powstaje przy niewłaściwym doborze algorytmu. Lukę tę można skutecznie zamknąć poprzez łączenie haszowania z innymi mechanizmami, takimi jak podpisy cyfrowe, drzewa Merkle’a czy systemy provenance.

W efekcie modele uczone na takich danych mogą zostać celowo zdeformowane, co prowadzi do powstania skażonych modeli zachowujących się poprawnie w typowych scenariuszach, ale ujawniających niepożądane zachowania po spełnieniu określonych warunków (Malinowski, 2024).

Znaczenie doboru algorytmu haszującego w tym kontekście bardzo istotne gdyż nie wszystkie funkcje skrótu zapewniają ten sam poziom odporności na ataki kolizyjne i tzw. ataki drugiego obrazu (*second preimage attacks*). Starsze algorytmy, takie jak MD5 czy SHA-1, zostały uznane za niebezpieczne, ponieważ możliwe jest dla nich generowanie celowych kolizji w rozsądnym czasie obliczeniowym. Użycie takich algorytmów w procesie walidacji zbiorów danych lub podpisów kryptograficznych modeli AI może skutkować umożliwieniem niezauważonego wprowadzenia zmodyfikowanych próbek lub całych klas danych o identycznych wartościach haszy. W środowisku, gdzie modele sztucznej inteligencji opierają się na integracji ogromnych zbiorów danych pochodzących z wielu źródeł, istotna jest więc nie tylko szybkość algorytmu ale też fakt, że zastosowanie niewłaściwego algorytmu haszującego może

prowadzić do skażenia całego procesu uczenia. Algorytmy takie jak SHA-256, czy Blake2 oferują znacznie wyższą odporność na kolizje dzięki większej przestrzeni mieszania i udoskonalonej strukturze funkcji kompresującej. Wybór funkcji skrótu powinien być więc uzależniony od charakteru danych, wymaganego poziomu bezpieczeństwa i ryzyka kryptoanalizy różnicowej (Rodwald 2013). W projektowaniu bezpiecznych systemów AI odpornych na złośliwe manipulacje w warstwie danych dobór takiego algorytmu jest istotny.

Wnioski

Badane algorytmy haszujące wykazują różne właściwości w kontekście zabezpieczania modeli AI przed atakami typu *data poisoning*. Algorytmy takie jak MD5 i SHA-1 są obecnie niewystarczająco bezpieczne ze względu na możliwość generowania kolizji, co czyni je nieodpowiednimi do ochrony integralności danych treningowych. Z kolei SHA-256 i BLAKE2 oferują solidną odporność na kolizje, przez co są rekomendowane do zapewnienia integralności danych w procesie trenowania modeli. Algorytmy takie jak bcrypt, choć mają większe czasy obliczeń, są bardzo skuteczne w ochronie haseł dzięki mechanizmom spowalniającym ataki *brute-force*, lecz ich zastosowanie w klasycznym zabezpieczaniu modeli AI jest ograniczone.

Projektanci systemów uczenia maszynowego powinni wybierać algorytmy haszujące o wysokiej odporności na kolizje (np. SHA-256, BLAKE2) do monitorowania integralności danych treningowych oraz parametrów modeli. Przy przechowywaniu haseł i danych wrażliwych rekomenduje się stosowanie algorytmów adaptacyjnych, takich jak bcrypt, aby minimalizować ryzyko złamania haseł. Należy również uwzględniać wydajność oraz kontekst wdrożenia – np. w środowiskach o ograniczonych zasobach może preferować szybsze funkcje.

Wskazane jest pogłębienie integracji technik haszowania z nowoczesnymi mechanizmami uczenia federacyjnego, co mogłoby zapewnić lepszą ochronę integralności rozproszonych modeli i danych bez konieczności centralnego udostępniania danych. Kolejnym kierunkiem jest rozwijanie algorytmów detekcji kolizji haszy dostosowanych do specyfiki zagrożeń w AI, a także badania nad nowymi konstrukcjami funkcji skrótu odpornych na ataki specyficzne dla uczenia maszynowego, takie jak ataki *adversarialne* i *data poisoning*.

Równocześnie powinno się analizować kompromisy między bezpieczeństwem a wydajnością, aby umożliwić szerokie wdrożenie efektywnych technik w systemach produkcyjnych.

Literatura

1. Albertini, A., Stevens, M. (b.d.). *Collisions: Hash collisions and exploitations*. GitHub repository. Pobrane z: <https://github.com/corkami/collisions> (dostęp: 16.03.2026).
2. Al-Odat, Z., Khan, S. (2019). The Sponge Structure Modulation Application to overcome the security breaches for the MD5 and SHA-1 hash functions. W: *IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA* (s. 811-816). IEEE. <https://doi.org/10.1109/COMPSAC.2019.00119>
3. Aumasson, J.-P. (2024). *Serious cryptography: a practical introduction to modern encryption*. 2nd edition. No Starch Press.
4. Aumasson, J.-P., Neves, S., Wilcox-O’Hearn, Z., Winnerlein, C. (2017). *BLAKE2 – fast secure hashing*. Pobrane z: <https://www.blake2.net> (dostęp: 12.12.2025).
5. Biggio, B., Roli, F. (2018). Wild patterns: ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84, 317-331. <https://doi.org/10.1016/j.patcog.2018.07.023>
6. Cao, D., Yang, B. (2010). Design and implementation for MD5-based data integrity checking system. W: *2nd IEEE International Conference on Information Management and Engineering, Chengdu, China, 2010* (s. 608-611). IEEE. <https://doi.org/10.1109/ICIME.2010.5477912>
7. Dash, S. P., Khandeparkar, K. V., Gajjar, G. (2025). A GAN-based data integrity attack on wide-area damping control systems in the smart grid. W: *5th International Conference on AI-ML-Systems (AIMLSystems), Bangalore, India* (s. 56-60). IEEE. <https://doi.org/10.1109/AIMLSystems67835.2025.11330765>
8. Eastlake, D., Jones, P. (2001). US secure hash algorithm 1 (SHA1). *RFC 3174*. <https://doi.org/10.17487/RFC3174>
9. Gu, T., Dolan-Gavitt, B., Garg, S. (2017). BadNets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv*, 1708.06733. <https://doi.org/10.48550/arXiv.1708.06733>

10. Katz, J., Lindell, Y. (2014). *Introduction to modern cryptography* (2nd ed.). Chapman and Hall/CRC. <https://doi.org/10.1201/b17668>
11. Klabnik, S., Nichols, C. (2021). *The Rust Programming Language*, 2nd edition. No Starch Press.
12. Kurakin, A., Goodfellow, I., Bengio, S. (2017). Adversarial examples in the physical world. *arXiv*, 1607.02533. <https://doi.org/10.1201/9781351251389-8>
13. Malinowski, M. (2024). *Adwersarialne ataki na sztuczną inteligencję*. Pobrane z: <https://www.drmalinowski.edu.pl/posts/2824-adwersarialne-ataki-na-sztuczna-inteligencje> (dostęp: 12.12.2025).
14. Nahiyani, A., Nejati, S., Bright, C. (2024). SHA-256 Collision Attack with Programmatic SAT. *arXiv*, 2406.20072v1. <https://doi.org/10.48550/arXiv.2406.20072>
15. Rietta, F. (2016). *What is the difference between bcrypt and SHA256?*. Pobrane z: <https://rietta.com/blog/bcrypt-not-sha-for-passwords/> (dostęp: 12.12.2025).
16. Rivest, R. (1992). The MD5 message-digest algorithm. *RFC 1321*. <https://doi.org/10.17487/RFC1321>
17. Rodwald, P. (2013). Kryptograficzne funkcje skrótu. *Zeszyty Naukowe Akademii Marynarki Wojennej*, 193.
18. Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice*. Pearson.
19. Turner, S., Chen, L. (2011). Updated security considerations for the MD5 message-digest and the HMAC-MD5 algorithms. *RFC 6151*. <https://doi.org/10.17487/RFC6151>
20. Younis, R., AlKasassbeh, M. (2025). Evaluating deep learning for detecting data integrity attacks in energy smart grids. W: *International Conference on New Trends in Computing Sciences (ICTCS)*, Amman, Jordan (s. 368-372). IEEE. <https://doi.org/10.1109/ICTCS65341.2025.10989460>