Funkcje skrótu - własności, bezpieczeństwo, zastosowania, przykłady

Szymon Bizoń Jan Kocierz

3 czerwca 2024

Spis treści

Wstęp

Jednym z kluczowych narzędzi zapewniających ochronę danych jest kryptografia, która znajduje zastosowanie w wielu dziedzinach, począwszy od komunikacji internetowej, przez przechowywanie danych w chmurze, aż po autoryzację dostępu do systemów informatycznych. Wśród różnorodnych technik kryptograficznych, kryptograficzne funkcje skrótu stanowią istotny element systemów zabezpieczających dane. Są one nieodłącznym narzędziem w procesach uwierzytelniania, integrowania danych, podpisów cyfrowych oraz przechowywania haseł w bazach danych. Ich uniwersalność sprawia, że są szeroko stosowane w informatyce. Jako studenci matematyki stosowanej podejmujemy temat z zakresu kryptografii, ponieważ to właśnie matematyka odgrywa w niej kluczową rolę, co pokazuje przykład funkcji skrótu. W poniższej pracy naszym celem jest, za pomocą języka matematyki, zdefiniować funkcje skrótu i ich własności, przybliżyć pojęcia związane z bezpieczeństwem owych funkcji, a następnie przedstawić jakie odzwierciedlenie ma to w ich działaniu i zastosowaniach na przykładzie funkcji MD5 oraz funkcji SHA-2. W rozdziale pierwszym więc zajmiemy się definicją matematyczną funkcji skrótu i jej najważniejszych własności. Następnie po zdefiniowaniu czym jest kolizja, sklasyfikujemy ataki na owe funkcje, przyglądając się dokładniej niektórym z nich. W kolejnych rozdziałach przybliżymy szczegółowo algorytm działania funkcji SHA-2, a także MD5. Następnie pokażemy, w jaki sposób funkcja MD5 została złamana. W ostatnim rozdziale przyjrzymy się zastosowaniom funkcji MD5 i implikacjom, jakie w tym kontekście niesie za sobą udany na nią atak.

1 Definicja i własności

Funkcją skrótu H nazywamy odwzorowanie, które dowolnemu skończonemu ciągowi bitów przyporządkowuje ciąg o stałej, ustalonej długości. Innymi słowy, zadaniem tej funkcji jest skompresowanie dużych danych w krótki kod w zapisie szesnastkowym.

Definicja 1.1. Niech $n \in \mathbb{N} \setminus \{0\}$. Przez $\{0,1\}^*$ będziemy oznaczać zbiór wszystkich ciągów bitów skończonej długości, ponadto $\{0,1\}^n$ będzie zbiorem informacji bitowych, ustalonej z góry i stałej dla funkcji, długości n. Funkcją skrótu nazywamy każde odwzorowanie H, które formalnie zapisujemy

$$H : \{0,1\}^* \mapsto \{0,1\}^n.$$

1.1 Wstępne uwagi i objaśnienia

Zanim przejdziemy do własności, musimy wprowadzić pewne oznaczenia w celu głębszego zrozumienia tego zagadnienia.

- Przez $A(x,y)$ będziemy oznaczać dowolny algorytm probabilistyczny
- Przez U_n będziemy oznaczać zmienną losową przyjmującą wartości w $\{0,1\}^n$ z rozkładem jednostajnym
- Aby uniknąć sytuacji, że trudność odwrócenia funkcji wynika jedynie z tego, że $f(x)$ jest o wiele krótsze od x, w naszych rozważaniach użyjemy współrzednej 1^n. Podpowiada ona niejako algorytmowi, jakiej długości ma być poszukiwany argument x [1].

Definicja 1.2. Jeśli *czas wykonania algorytmu $T(n)$ jest wielomianowy*, to znaczy, że istnieje stała c i takie $k \in \mathbb{N}$, że dla wszystkich wystarczająco dużych n czas wykonania algorytmu nie przekracza $c \cdot n^k$ [11].

$$\exists_{c\in\mathbb{R}\ k,m\in\mathbb{N}}\ \forall_{n>m}\left(T(n) < c \cdot n^k\right).$$

Definicja 1.3. Funkcja f jest *obliczalna przez deterministyczny algorytm wielomianowy*, jeśli istnieje taki algorytm, który dla dowolnego wejścia x jest w stanie obliczyć wartość funkcji $f(x)$ w czasie wielomianowym [11].

Definicja 1.4. *Wielomianowy algorytm probabilistyczny* to algorytm, który działa w oparciu o losowe wybory, a jego czas jest ograniczony przez wielomianową funkcję od długości danych z dużym prawdopodobieństwem. Różni się od deterministycznego wielomianowego algorytmu, który daje dokładne wyniki dla tych samych danych wejściowych tym, że może dawać różne wyniki dla tych samych danych wejściowych w różnych przebiegach [16].

Definicja 1.5. Funkcję nazywamy *zaniedbywalną*, kiedy dąży do zera szybciej niż dowolny wielomian . Formalnie: $\mu : \mathbb{N} \to \mathbb{R}$ jest zaniedbywalna, jeśli

$$\forall_{d\in\mathbb{Z}}\ \exists_{m\in\mathbb{Z}}\ \forall_{n>m}\left(\mu(n) < \frac{1}{n^d}\right)$$

[6].

1.2 Własności

1.2.1 Jednokierunkowość

Funkcja skrótu powinna być jednokierunkowa, to znaczy, że dla danego $y \in \{0,1\}^n$, trudno jest znaleźć $x \in \{0,1\}^*$ takie, że $H(x) = y$.

Funkcja $H : \{0,1\}^* \mapsto \{0,1\}^n$ jest jednokierunkowa, jeśli spełnia następujące warunki

1. H jest obliczalna przez deterministyczny algorytm wielomianowy.
2. Dla każdego probabilistycznego algorytmu wielomianowego A prawdopodobieństwo

$$\Pr\left[A(H(U_n), 1^n) \in H^{-1}(H(U_n))\right]$$

jest zaniedbywalne [1].

1.2.2 Słaba odporność na kolizje

Funkcja skrótu powinna posiadać słabą odporność na kolizje, to znaczy, że dla danego m_1 trudno jest znaleźć m_2 takie, że $m_2 \neq m_1$ i $H(m_2) = H(m_1)$

Funkcja $H : \{0,1\}^* \mapsto \{0,1\}^n$ jest słabo odporna na kolizje, jeśli dla każdego algorytmu probabilistycznego A prawdopodobieństwo, że dla danego m_1 znajdzie on $m_2 \neq m_1$, takie, że $H(m_1) = H(m_2)$ jest zaniedbywalne. Formalnie:

$$\Pr\left[A(H(U_n), 1^n) = m_2 \text{ takie że } m_1 \neq m_2 \wedge H(m_1) = H(m_2)\right]$$

jest zaniedbywalne.

1.2.3 Silna odporność na kolizje

Funkcja skrótu powinna być silnie odporna na kolizje, to znaczy, że znalezienie dwóch różnych wejść m_1, m_2 takich, że $H(m_1) = H(m_2)$ powinno być obliczeniowo niemożliwe.

Funkcja $H : \{0,1\}^* \mapsto \{0,1\}^n$ jest odporna na kolizje, jeśli dla każdego algorytmu probabilistycznego A prawdopodobieństwo, że dla dwóch wejść m_1, m_2 takich, że $m_1 \neq m_2$ znajdzie on $H(m_1) = H(m_2)$ jest zaniedbywalne. Formalnie:

$$\Pr\left[A(H(U_n), 1^n) = (m_1, m_2) \text{ takich że } m_1 \neq m_2 \wedge H(m_1) = H(m_2)\right]$$

jest zaniedbywalne [17].

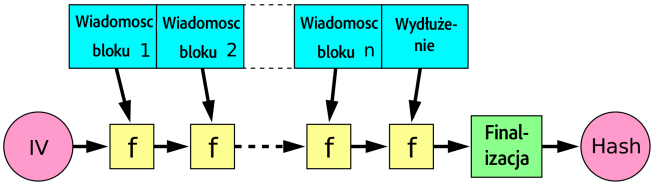
1.2.4 Implikacje

Tutaj również należy wspomnieć, że silna odporność na kolizje implikuje słabą [4]. Aby to udowodnić, zakładamy, że funkcja nie jest słabo odporna na kolizje i udaje nam się dla danego m_1 znaleźć m_2, które spełnia wszystkie warunki wymienione powyżej. W takim wypadku para (m_1, m_2) stanowi kolizje, co przeczy własności trzeciej. Niestety podobna implikacja nie istnieje jeśli chodzi o jednokierunkowość. Silna odporność w żaden sposób nie wpływa na jednokierunkowość i odwrotnie. W praktyce własność trzecia jest najtrudniejszą do spełnienia spośród trzech pozostałych i to ona jest głównym celem większości ataków.

2 Schematy budowy funkcji skrótu

2.1 Konstrukcja Merkle-Damgard

Funkcje skrótu, by mogły być użyte w poszczególnych ich zastosowaniach muszę być wydajne. Osiąga się to chociażby, poprzez zaprojektowanie ich w iteracyjnym trybie działania, gdzie funkcja przyjmująca wejście o stałej długości jest iterowana, aż do przetworzenia całkowitego wejścia



Rysunek 1: Konstrukcji Markle-Damargd
Źródło: [14]

o dowolnej długości. Taka struktura iteracyjna została niezależnie zaproponowana przez Ivana Damgårda i Ralpha Merkle'a. Nazywa się ją konstrukcją Merkle-Damgård i jest stosowana w większości funkcji skrótu - na przykład MD5 czy SHA-1, SHA-2.

W konstrukcji Merkle-Damgård funkcja skrótu na początku "wypełnia" wejściowy łańcuch bitów za pomocą ustalonego schematu , tak aby można było podzielić go na bloki o stałej długości n.(Na przykład dla n=512 bitów) Funkcja skrótu przyjmuje wejściową wiadomość i dzieli ją na n bloków o stałej długości b bitów każdy. Ostateczny blok może być wypełniony do b bitów, jeśli jest to konieczne i zawierać również wartość całkowitej długości wejścia do funkcji skrótu, co utrudnia zadanie ewentualnemu atakującemu.

Kolejną cechą, jaką wyróżnia się ta konstrukcja, jest inicjalizacja stanu. Na początku, inicjalizowany jest stan wewnętrzny algorytmu skrótu. Ten stan początkowy jest zwykle stałego rozmiaru i może być zdefiniowany w specyfikacji algorytmu skrótu. Konstrukcje Merkle-Damgard wyróżnia także kompresja bloków. Każdy blok danych wejściowych jest kompresowany w celu zmniejszenia jego rozmiaru i aktualizacji stanu wewnętrznego algorytmu skrótu. Proces kompresji zwykle obejmuje wykorzystanie operacji logicznych i przekształceń bitowych. Na przytoczonym przez nas schemacie f oznacza funkcję, która odpowiada za kompresję. Proces kompresji jest powtarzany dla każdego bloku danych wejściowych. Wynik kompresji poprzedniego bloku jest również używany jako część stanu wewnętrznego dla kompresji następnego bloku - mamy tu więc kolejną cechę konstrukcji Merkle-Damgard, jaką jest iteracja. Po skompresowaniu wszystkich bloków danych wejściowych, stan wewnętrzny algorytmu jest poddawany dodatkowym operacjom, aby wygenerować ostateczny skrót.

W swoich odpowiednich pracach Damgård i Merkle przedstawiają twierdzenia pokazujące, że jeśli istnieje funkcja kompresji odporna na kolizje o stałej długości wejścia $f : \{0,1\}^b \mapsto \{0,1\}^t$, to można zaprojektować funkcję skrótu o zmiennym długości wejścia $H : \{0,1\}^* \mapsto \{0,1\}^t$ przez iterację tej funkcji kompresji. Innymi słowy, jeśli funkcja kompresji jest odporna na kolizje, to również iterowana funkcja skrótu jest. Zatem jeśli funkcja kompresji jest podatna na jakiekolwiek ataki, to także iterowana funkcja skrótu, ale odwrotność tego wyniku nie jest generalnie prawdziwa.

Konstrukcja Merkle-Damgård ma wiele zalet, w tym prostotę implementacji, efektywność obliczeniową i możliwość generowania skrótów o stałej długości niezależnie od długości danych wejściowych. Jednak istnieją pewne wady i potencjalne podatności, takie jak kolizje w funkcji skrótu (czyli sytuacja, gdy dwa różne zestawy danych prowadzą do tego samego skrótu), które mogą być wykorzystane w celu ataków kryptoanalizy. O tym jednak w dalszej części naszej pracy.

3 Ataki na funkcje skrótu

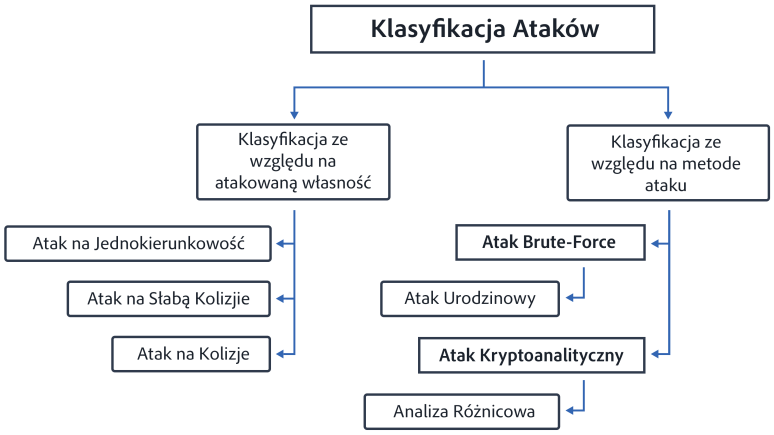
3.1 Wprowadzenie - czym są kolizje?

Kolizja odnosi się do sytuacji, w której dwa różne dane wejściowe generują ten sam hash. Funkcje skrótu są używane w kryptografii do odwzorowania danych o dowolnej długości na dane o stałej długości, które są trudne do odwrócenia. Jednakże, z powodu ograniczonej długości skrótu, istnieje możliwość, że dwa różne zestawy danych wejściowych wygenerują ten sam skrót, co jest nazywane kolizją.

Formalnie kolizja to oczywiście para wejść m_1, m_2 taka, że $m_1 \neq m_2$ i $H(m_1) = H(m_2)$. Istnienie kolizji wynika z zasady szufladkowej Dirichleta. W kontekście funkcji skrótu możemy powiedzieć, że polega ona na tym, że jeśli istnieje więcej możliwych wejść niż możliwych hash'ów, to jest to równoznaczne, z tym że niektóre wejścia dają ten sam hash. Wynika z tego że zdefiniowana przez nas funckja skrótu nie może być iniektywna.

3.2 Klasyfikacja

Pomyślny atak na funkcję skrótu oznacza, że jedna z jej własności została "złamana". Ataki na funkcje skrótów możemy klasyfikować właśnie według klasyfikacji opartej na atak na daną własność lub opartej na metodologię ataku. Poniżej przedstawimy oba rodzaje klasyfikacji, a także przyjrzymy się bardziej szczegółowo niektórym rodzajom ataków.



Rysunek 2: Schemat podziału ataków na funkcje skrótu
Źródło: Opracowanie własne

3.2.1 Klasyfikacja ze względu na atakowaną własność

- Atak na jednokierunkowość - atak, w którym atakujący potrafi znaleźć wejście znając hash - wartość wyjściową.
- Atak na słabą odporność na kolizję (2^{nd} *preimage attack*) - atakujący dla danego wejścia m_1 potrafi znaleźć inne wejście m_2 które jako wyjście daje ten sam hash.
- Atak kolizyjny - polegający na znalezieniu pary wejść m_1, m_2 takiej, że $m_1 \neq m_2$ i $H(m_1) = H(m_2)$.

3.2.2 Klasyfikacja ze względu na metodę ataku

1. **Atak Brute-force**

   Jest najprostszym, ale często najbardziej czasochłonnym sposobem złamania zabezpieczeń, ponieważ wymaga przetestowania ogromnej liczby możliwości. Atak ten polega na próbie wszystkich możliwych kombinacji danych wejściowych w celu złamania funkcji skrótu. Ta metoda ma zastosowanie do wszystkich funkcji skrótu, niezależenie jak dobrze spełnione są jej właściwości. Atakujący próbuje wszystkie możliwe kombinacje, aż znajdzie poprawną wartość wejściową.

   W kontekście ataku brute force, "wysiłek" odnosi się do ilości pracy lub zasobów potrzebnych do przeprowadzenia ataku. Jest to ogólna miara trudności ataku, która może być określona na przykład poprzez liczbę prób, czas potrzebny do przetestowania wszystkich możliwych kombinacji, lub ilość zasobów komputerowych potrzebnych do wykonania ataku. Można też rozumieć wysiłek jako stopień trudności, jaki atakujący musi pokonać, aby osiągnąć cel ataku, na przykład znalezienie kolizji w funkcji skrótu czy odwrócenie funkcji jednokierunkowej.

   Jedną z metod pomiaru trudności takich ataków jest liczenie czasu trwania ataku w jednostce liczby wywołań funkcji skrótu podczas weryfikacji wylosowanej kombinacji wejścia. Wysiłek potrzebny do przeprowadzenia pomyślnego ataku Brute-force na funkcje skrótu zależy głównie od długości hashu.

   - Złożoność ataku na jednokierunkowość - wysiłek potrzebny do znalezienia wejścia m, takiego, że $H(m) = h$ szacowany jest na czas 2^n operacji, które gwarantują nam pewny sukces, ponieważ dla danego n-bitowego skrótu h funkcji H, atakujący sprawdza wartość H dla każdego prawdopodobnego wejścia m, dopóki nie uzyska oczekiwanej wartości wyjściowego skrótu h. Realnie jednak szacuje się, że proces wymaga średnio 2^{n-1} operacji.

- Złożoność ataku 2^{nd} preimage - wysiłek potrzebny do znalezienia wejścia m_2 różnego od danego wejścia m_1 jest taki sam jak w przypadku ataku na jednokierunkowość, wymaga 2^n operacji. Analogicznie średni czas to 2^{n-1}.
- Złożoność ataku kolizyjnego - trudność znalezienia pary wejść m_1, m_2 takiej, że $m_1 \neq m_2$ i $H(m_1) = H(m_2)$ wynosi $1.2 \cdot 2^{\frac{n}{2}}$. W tym miejscu na chwilę się zatrzymamy, aby przyjrzeć się dokładnie skąd, bierze się magiczna liczba $1.2 \cdot 2^{\frac{2}{n}}$. Wyjaśnimy to w następnym akapicie.

2. **Atak urodzinowy**

   Paradoks urodzinowy (*Birthday Paradox*) odnosi się do pozornie zaskakującej obserwacji, że w grupie osób, nawet stosunkowo niewielkiej, istnieje duża szansa na to, że dwie z nich będą miały urodziny w tym samym dniu. Załóżmy, że zebraliśmy 23 losowo wybrane osoby [18]. Prawdopodobieństwo, że wśród nich znajdziemy dwie osoby o tej samej dacie urodzin wynosi blisko 51%. Chociaż może się to wydawać nieintuicyjne, matematyczne obliczenia pokazują jasno, że szanse na takie zdarzenie są duże.

   Atak urodzinowy (*Birthday Attack*) jest techniką ataku kryptograficznego opartą na paradoksie urodzinowym. W przypadku funkcji skrótu atak ten polega na próbie znalezienia dwóch różnych argumentów, które generują ten sam hasz. Tutaj warto zauważyć, że w wyżej wspomnianym doświadczeniu próba znalezienia w grupie 23-osobowej kogoś z datą urodzin taką samą jak twoja jest radykalnie mniejsza. Dlatego ataki na jednokierunkowość są znacznie bardziej skomplikowane od ataku na kolizje.

   Wróćmy znowu do naszego urodzinowego doświadczenia i zastanówmy się, skąd bierze się liczba 23 i jakie to ma znaczenia dla naszych funkcji haszujących [5]. Chcemy uniknąć zderzenia by w grupie losowo wybranych osób, znalazła się para mająca tę samą datę urodzin. Przeanalizujemy więc zdarzenie odwrotne.

   Losowo wybieramy M osób tak by $M < 365$. Prawdopodobieństwo, że każda z wybranych osób ma różną datę urodzin będzie równe

   $$\frac{\binom{365}{M} \cdot M!}{365^M}.$$

   Chcemy by było one niższe niż 50% więc mamy

   $$\frac{\binom{365}{M} \cdot M!}{365^M} < \frac{1}{2}.$$

   Po dalszych obliczaniach dojdziemy do wniosku, że wymagana liczba prób, aby prawdopodobieństwo kolizji było wyższe od 50% dana jest wzorem

   $$M \approx 1.2\sqrt{n} \tag{2.1}$$

   gdzie n to liczba kombinacji, jakie może przyjąć nasza zmienna losowa. Podstawmy za n 365 wówczas

   $$M = 1.2\sqrt{365} \approx 22.92 \ .$$

   Nasza funkcja skrótu generuje n-bitowe hasze co oznacza, że istnieje 2^n różnych skrótów [18]. Stąd wiemy z (2.1), że wymagana ilość operacji, aby prawdopodobieństwo sukcesu ataku na kolizje było większe niż 50% (wartość minimalna, by uznać funkcje za mało bezpieczną), wynosi $1.2 \cdot 2^{\frac{n}{2}}$.

   Zauważmy, że jeśli A jest zdarzeniem, w którym po wygenerowaniu M skrótów otrzymaliśmy kolizję, a A' oznacza brak kolizji w identycznym eksperymencie, to dla dowolnego $d \in \mathbb{Z}$ chcemy, aby była spełniona następująca nierówność:

   $$1 - P(A') < \frac{1}{n^d}$$
   $$n^d \cdot (1 - P(A')) < 1$$

   Da się jednak udowodnić, że dla ustalonego $M < 2^n$ i dowolnego $d \in \mathbb{Z}$

   $$\lim_{n \to \infty} \left(\left(1 - \frac{(2^n)!}{(2^n - M)! \cdot 2^{Mn}} \right) n^d \right) = 0.$$

3. **Ataki Kryptoanalityczne**

   Ataki kryptoanalityczne nastawione są na identyfikację słabych punktów w algorytmach szyfrujących i funkcjach skrótu poprzez badanie argumentów (haseł), zaszyfrowanych wiadomości oraz samych mechanizmów działania tych funkcji. Celem jest odszukanie technik umożliwiających tworzenie kolizji. Bardziej szczegółowo opiszemy sposób działania tych ataków w dalszej części pracy [9].

4 Przykłady funkcji haszujących

4.1 Wstępne uwagi i objaśnienia

Aby uniknąć nieścisłości w dalszej części pracy, objaśnimy oznaczenia, które będziemy używać do opisania algorytmów.

Definicja 4.1. Operacja przesunięcia $ROTR^y$, S^y i SHR^y

Operacja $ROTR^y(x)$ jest odwzorowaniem, które trudno jest sformalizować matematycznie, ale łatwo wyjaśnić na przykładzie [19]. $ROTR^y(x)$ mówi nam, że musimy wykonać y okrężnych, logicznych przesunięć bitowych w prawo na wartości x.

Jako przykład weźmy 8-bitowy zapis binarny 1101 0010. Załóżmy, że nasze $y = 3$ wtedy kolejne zera i jedynki przesuną się do prawej strony o 3 miejsca, z tym że ostanie wartości muszą wrócić na początek naszej wiadomości. Wykonamy kolejne przesunięcia aż do $y = 3$.

pierwsze przesunięcie - 0110 1001,
drugie przesunięcie - 1011 0100,
trzecie przesunięcie - 0101 1010.

Operacja $S^y(x)$ działa analogicznie jak $ROTR^y(x)$ ale przesunięcie następuje w lewo.

Operacja $SHR^y(x)$ oznacza logiczne przesunięcie bitowe w prawo. Polega ono na przesunięciu wszystkich bitów informacji o y pozycji w prawo, z tą różnicą, że na puste miejsca po lewej stronie wstawiane są zera. Wykonamy tę operację w kolejnych krokach dla $y = 4$ i $x = 1101\ 0101$.

pierwsze przesunięcie - **0**110 1010,
drugie przesunięcie - **00**11 0101,
trzecie przesunięcie - **000**1 1010,
czwarte przesunięcie - **0000** 1101.

Definicja 4.2. Działanie XOR $\oplus$

Operacja XOR, jest logiczną operacją bitową, która działa na dwóch argumentach i zwraca wynik 1 tylko wtedy, gdy dokładnie jeden z nich ma wartość 1. W przeciwnym razie wynik jest równy 0. Inaczej mówiąc to działanie można zdefiniować jako dodawanie w $\mathbb{Z}_2$. $(1+1) \mod 2$ daje nam wartość zero. $(1+0) \mod 2$ zwraca nam wartość 1. Z tą uwagą, że będziemy interpretować działanie XOR nie jako operację na całej liczbie w zapisie binarnym, ale dodawanie kolejnych bitów. Przedstawimy przykład w tabelce poniżej

Tabela 1: Tabela dla operacji XOR

p	01101010
q	11001101
$p \oplus q$	10100111

Definicja 4.3. Działanie AND $\wedge$

AND podobnie jak XOR jest bitową operacją logiczną, która zwraca wartość 1 wtedy i tylko wtedy gdy obie wartości są równe 1.

Definicja 4.4. Działanie NOT $\neg$

NOT odwraca wartości naszego zapisu binarnego z 0 na 1 i odwrotnie.

Tabela 2: Tabela dla operacji AND

p	01101010
q	11001101
$p \wedge q$	01001000

Tabela 3: Tabela dla operacji NOT

p	01101010
¬p	10010101

4.1.1 Dodawanie modularne w pierścieniu $\mathbb{Z}_{2^{32}}$

Zanim przejdziemy do operacji dodawania w $\mathbb{Z}_{2^{32}}$, niezbędne jest wprowadzenie szesnastkowego systemu liczbowego. System binarny, znany również jako system dwójkowy, wykorzystuje jedynie dwie cyfry: 0 i 1 do reprezentacji informacji. Jest on powszechnie stosowany w komputerach i innych urządzeniach cyfrowych ze względu na prostotę i łatwość implementacji w układach elektronicznych.

Natomiast system dziesiętny, z którym mamy do czynienia na co dzień, opiera się na dziesięciu cyfrach: 0-9. System ten oferuje 10 różnych wartości, co pozwala na reprezentowanie większych ilości danych przed koniecznością łączenia cyfr. Na przykład, łącząc cyfry 0 i 1, otrzymujemy liczbę 10.

System szesnastkowy, jak sama nazwa wskazuje, wykorzystuje 16 symboli do reprezentacji informacji. Obejmuje on cyfry 0-9 oraz litery a-f. System ten jest często stosowany w informatyce ze względu na zwartość zapisu. Przykładowo, liczba 10 w systemie szesnastkowym zapisana jest jako "a", a liczba 11 jako "b". Tak jak w dziesiętnym systemie cyfry na drugim miejscu informują nas o ilości dziesiątek do potęgi pierwszej w liczbie tak w szesnastkowym przykładowo liczba $b0$ mówi nam, że dana liczba jest równa $11 \cdot 16^1 + 0 \cdot 16^0$.

Mając już podstawową znajomość systemu szesnastkowego, możemy przejść do omówienia dodawania modularnego liczb w $\mathbb{Z}_{2^{32}}$. Wyjaśnimy to na prostym przykładzie. Załóżmy, że mamy do dodania następujące liczby

- $01101011110001011000001001100001 = 6BC58261$,
- $01010000011100100111101001111001 = 50727A79$,
- $01100111011011110111001001111001 = 676F7279$,
- $01100001001000000110000101101100 = 6120616C$.

Teraz sumujemy

$$6BC58261 + 50727A79 + 676F7279 + 6120616C = 184C7D0BF.$$

Po dodaniu wszystkiego otrzymaliśmy wartość $184C7D0BF$. Pamiętamy, jednak że to nie jest ostateczny wynik należy jeszcze policzyć

$$184C7D0BF \mod 2^{32}.$$

Szczęśliwą własnością tego zapisu jest to, że przy wyznaczaniu reszty z dzielenia przez 2^{32} dla liczb długości 9 znaków należy tylko skreślić pierwszą cyfrę, co wynika z faktu, że $8^{16} = 2^{32}$. Ostatecznie wyjdzie nam, że

$$184C7D0BF \mod 2^{32} = 84C7D0BF.$$

W skrócie możemy zdefiniować dodawanie modularne następująco.

Definicja 4.5. Dodawanie modulo $\mathbb{Z}_{2^{32}}$
Niech $a, b \in \{0, 1, \ldots, 2^{32} - 1\}$, wówczas ich sumą modularną będzie

$$a \oplus_{2^{32}} b = (a + b) \mod 2^{32},$$

gdzie $\mod$ to reszta z dzielenia $a + b$ przez 2^{32}. Innymi słowy, jeśli $q, r \in \mathbb{Z}$ i $(a + b) \cdot q + r = 2^{32}$ to dla $r \in \{0, 1 \ldots 2^{32} - 1\}$

$$(a + b) \mod 2^{32} = r.$$

5 Funkcja SHA-2

Funkcja haszująca SHA-2 generuje skrót o stałej długości 64 znaków w zapisie szesnastkowym dla dowolnej wiadomości wejściowej, niezależnie od jej długości. Nawet dla pojedynczego bitu danych algorytm musi wygenerować odpowiedni skrót. Oznacza to, że funkcja ta musi posiadać zarówno zdolność kompresji danych wejściowych, jak i ich rozszerzania, zachowując przy tym wszystkie kluczowe cechy funkcji skrótu.

Należy podkreślić, że każda informacja wprowadzana do komputera podlega przetwarzaniu w postaci binarnej. Algorytm działa analogicznie – na każdym etapie operuje wyłącznie na danych binarnych, do których konwertowana jest wiadomość na samym początku procesu.

Omawiany algorytm SHA-2 operuje na blokach 512-bitowych, co oznacza, że nasza wiadomość, oznaczana jako M, jest dzielona na l części o długości 512 bitów, tak by spełniała równanie $l \cdot 512 = M$. Przed dokonaniem tego podziału, wiadomość wejściowa jest rozszerzana zgodnie z określonymi regułami dla SHA-2. Proces rozszerzania różni się w zależności od reszty z dzielenia długości wiadomości przez 512. W naszym schemacie poniżej użyjemy symbolu plus, który będzie oznaczał zwykłe dołączenie jednej wiadomości do drugiej.

$$+ : \{0,1\}^n \times \{0,1\}^k \ni (x, y) \longmapsto xy \in \{0,1\}^{n+k}$$

Natomiast przez M_r będziemy oznaczać resztę z dzielenia M na 512-bitowe bloki.

- Jeżeli długość wiadomości $M \mod 512 \leqslant 447$, ostatni blok jest rozszerzany poprzez dopisanie jedynki oraz zer do momentu osiągnięcia długości 448 bitów. Proces ten można przedstawić następująco:

  $$\overbrace{M_r + 1 + 00\ldots0}^{448 \text{ bitów}}.$$

  Wciąż brakuje nam 64 bitów. Tutaj następuje dopisanie długości wiadomości M w zapisie binarnym i poprzedzenie jej zerami tak by całość była wymaganej długości 64, czyli

  $$\overbrace{00\ldots0 + k}^{64 \text{ bity}}.$$

  gdzie

  k - długość wiadomości M w zapisie binarnym.

  Należy zauważyć że jeśli wiadomość będzie wystarczająco długa, to dopisywanie zer do jej długości nie będzie wymagane. Natomiast jeśli k będzię dłuższe od 64, to wybierane są tylko jej pierwsze 64 bity. Całość będzie wyglądała wtedy następująco

  $$\overbrace{M_r + 1 + 00\ldots0 + k}^{512 \text{ bitów}}.$$

- Jeśli $M \mod 512 > 447$ to ostani blok będzie rozszerzony na dwa bloki. W pierwszym standardowo dopisujemy jedynkę i zera, tak by dopełnić do wymaganej długości 512. Następny blok będzie składał się z 448 zer, do których zostanie dopisane odpowiednio rozszerzone k tak jak zostało wspomniane powyżej.

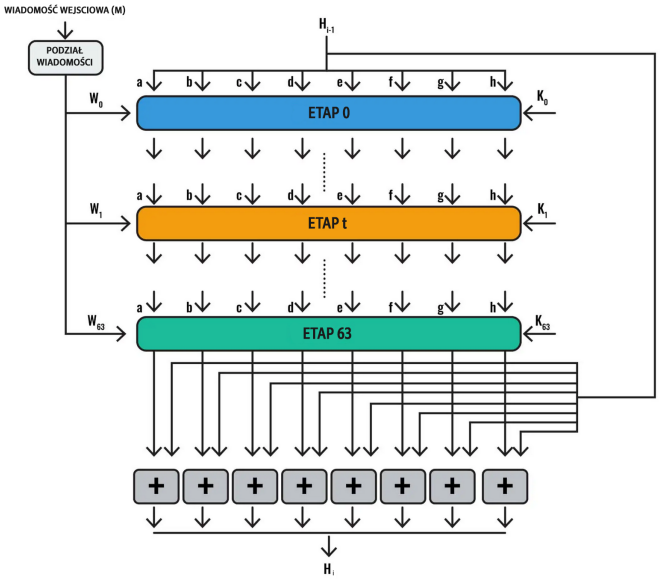
  $$\overbrace{M_r + 1 + 00\ldots0}^{512 \text{ bitów}}, \qquad \overbrace{00\ldots0 + k}^{512 \text{ bitów}}.$$

5.1 Dane wejściowe

Teraz kiedy mamy już naszą wiadomość podzieloną na bloki możemy przejść do schematu działania SHA-2. Algorytm dla każdego bloku będzie działał tak samo, więc skupimy się tylko na pierwszym. Poniżej widzimy graficzne przedstawienie poszczególnych etapów algorytmu [19].

Schemat algorytmu podzielony jest na 64 etapy, które będziemy numerować od zera. Widzimy, że dane wyjściowe z kroku poprzedniego stają się danymi wejściowymi do kroku następnego. To się będzie powtarzało za każdym razem.

Zaczniemy od etapu zerowego. Widzimy na ilustracji, że potrzebne jest do niego 10 danych wejściowych W_0, K_0 i dane $a - h$.



Rysunek 3: Schemat algorytmu i etapów funkcji SHA-2. Źródło: [19]

5.1.1 Wartości wejściowe W_t

Aby wyjaśnić, jak otrzymujemy wartości W_{0-63} posłużymy się przykładem. Niech naszą wiadomością M będzie zdanie „ Przykład dla algorytmu SHA-2 ”. Konwertujemy litery, cyfry i symbole na język binarny, korzystając z amerykańskiego standardowego kodu wymiany informacji ASCII. Każdy znak zapisywany kodzie ASCII składa się z 8-bitowego kodu. Przykładowo duża litera P w tym kodzie to 01010000. Po przekonwertowaniu całego zdania otrzymujemy

01010000 01110010 01111010 01111001 01101011 11000101 10000010 01100001 01100100
00100000 01100100 01101100 01100001 00100000 01100001 01101100 01100111 01101111
01110010 01111001 01110100 01101101 01110101 00100000 01010011 01001000 01000001
00101101 00110010.

Jak zostało wspomniane wcześniej, jeden blok musi być długości dokładnie 512-bitowej. Według schematu rozszerzającego naszą wiadomość w zapisie binarnym otrzymamy

01010000 01110010 01111010 01111001 01101011 11000101 10000010 01100001 01100100
00100000 01100100 01101100 01100001 00100000 01100001 01101100 01100111 01101111
01110010 01111001 01110100 01101101 01110101 00100000 01010011 01001000 01000001
00101101 00110010 10000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
11101010.

Gdzie ostatni fragment naszej wiadomości 11101010 to zapis długości zdania „ Przykład dla algorytmu SHA-2 ” w kodzie ASCII, czyli wcześniej oznaczone k. W_i to po prosto kolejne fragmenty naszego bloku, który dzielimy odpowiednio na 16 części.

W_i	**Reprezentacja binarna**	**Reprezentacja szesnastkowa**
W_0	01010000 01110010 01111010 01111001	50727A79
W_1	01101011 11000101 10000010 01100001	6BC58261
W_2	01100100 00100000 01100100 01101100	6420646C
W_3	01100001 00100000 01100001 01101100	6120616C
W_4	01100111 01101111 01110010 01111001	676F7279
W_5	01110100 01101101 01110101 00100000	746D7520
W_6	01010011 01001000 01000001 00101101	5348412D
W_7	00110010 10000000 00000000 00000000	32800000

Tabela 4: Reprezentacja binarna i szesnastkowa W_i 0-8

W_i	**Reprezentacja binarna**	**Reprezentacja szesnastkowa**
W_8	00000000 00000000 00000000 00000000	00000000
W_9	00000000 00000000 00000000 00000000	00000000
W_{10}	00000000 00000000 00000000 00000000	00000000
W_{11}	00000000 00000000 00000000 00000000	00000000
W_{12}	00000000 00000000 00000000 00000000	00000000
W_{13}	00000000 00000000 00000000 00000000	00000000
W_{14}	00000000 00000000 00000000 00000000	00000000
W_{15}	00000000 00000000 00000000 11101010	000000EA

Tabela 5: Reprezentacja binarna i szesnastkowa W_i 8-15

Analizując diagram, dostrzegamy, że wciąż brakuje nam 48 wartości W. Po podzieleniu całego bloku konieczne jest wyznaczenie pozostałych wartości, które ze względów bezpieczeństwa muszą różnić się od wcześniej obliczonych. W tym celu dla $t \geqslant 16$ wartości W obliczane są na podstawie wzoru

$$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}, \tag{4.1}$$

gdzie

- $\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$
- $\sigma_0(x) = ROTR^{7}(x) \oplus ROTR^{18}(x) \oplus SHR^{3}(x)$
- $\oplus : \mathbb{Z}_2 \times \mathbb{Z}_2 \ni (x, y) \longmapsto x + y \in \mathbb{Z}_2$
- $+ : \mathbb{Z}_{2^{32}} \times \mathbb{Z}_{2^{32}} \ni (x, y) \longmapsto x + y \in \mathbb{Z}_{2^{32}}$

Wzór może początkowo wydawać się skomplikowany, dlatego jego działanie wyjaśnimy na przykładzie. Załóżmy, że mamy do dodania następujące składowe

- $\sigma_1(x) = 01101011110001011000001001100001$,
- $\sigma_0(y) = 01010000011100100111101001111001$,
- $W_z = 01100111011011110111001001111001$,
- $W_v = 01100001001000000110000101101100$.

Teraz zamienimy wartości na zapis szesnastkowy.

- $\sigma_1(x) = 6BC58261$,
- $\sigma_0(y) = 50727A79$,
- $W_z = 676F7279$,
- $W_v = 6120616C$.

Podstawiamy do wzoru (4.1) i otrzymujemy

$$W = 6BC58261 + 50727A79 + 676F7279 + 6120616C.$$

Po dodaniu wszystkiego wychodzi nam, że $W = 184C7D0BF$. Pamiętamy, jednak że to nie jest ostateczny wynik należy jeszcze policzyć

$$184C7D0BF \mod 2^{32} = 84C7D0BF.$$

5.1.2 Stałe K_t

Jeśli spojrzymy na diagram po lewej stronie, zauważymy, że potrzebujemy na wejściu 64 wartości stałej K. Istnieją 64 oddzielne 32-bitowe wartości K na każdy z etapów. Wartość K powstają z pierwiastków sześciennych pierwszych 64 liczb pierwszych.

$$\sqrt[3]{2} \approx 1.25992104989487316476721 0.$$

W zapisie szesnastkowym.

$$\sqrt[3]{2} \approx 1.428A2F98D728AE2.$$

Więc stała $K_0 = 428A2F98$.

5.1.3 Wartości $a - h$

Naszym ostatnim kamieniem milowym są wartości $a - h$ które będziemy oznaczać jako $H_{0a} - H_{0h}$. Taki oznaczenie ma sugerować, że H_i będzie zmienne. Indeks i oznacza numer bloku, który przechodzi przez algorytm. Aby uniknąć zamieszania w dalszej części pracy, wyjaśnimy oznaczenia. Nasz algorytm będzie korzystał ze zmiennych $H_{ix}^{(y)}$ gdzie

- i - indeks oznaczający, na jakim bloku aktualnie pracuje algorytm. Z tego wynika, że nasze $i \in \{0, 1, \ldots, l\}$. Tutaj l to ilość bloków, na jakie została podzielona wiadomość M.
- x - indeks oznaczający, jaka to jest część danego H_i^y które dzielimy na osiem części od a do h, tak jak pokazano na diagramie. Z tego wynika, że $x \in \{a, b, \ldots, h\}$.
- y - mówi nam, w jakim etapie zmienna jest używane. Tutaj $y \in \{0, 1, \ldots, 63\}$. Z tym że dla etapu (0) indeks y będziemy pomijać.

Na ilustracji możemy zauważyć, że dane H przechodzą przez każdy etap, zmieniając się z danych wejściowych na wyjściowe. Na początku jednak te stałe pochodzą od pierwiastków kwadratowych pierwszych 8 liczb pierwszych.

$$\sqrt{2} \approx 1.41421356237309504880168872.$$

W zapisie szesnastkowym.

$$\sqrt{2} \approx 1.6A09E667F3BCC9.$$

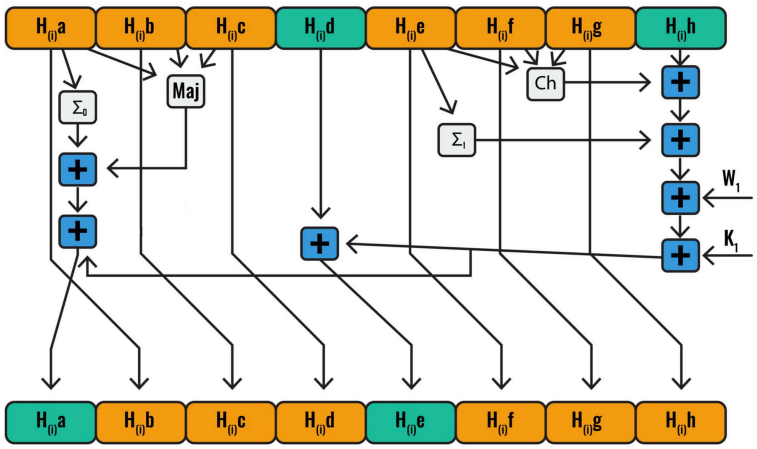
Więc nasze $H_{0a} = 6A09E667$.

$$\sqrt{3} \approx 1.73205080756887729352744634.$$

W zapisie szesnastkowym.

$$\sqrt{3} \approx 1.BB67AE8584CAA73.$$

Więc nasze $H_{0b} = BB67AE85$.



Rysunek 4: Schemat algorytmu i etapu zerowego funkcji SHA-2 z działaniami wewnętrznymi. Źródło: [19]

5.2 Etap (0)

Posiadamy już wszystko, co jest potrzebne, by wykonać kolejne kroki algorytmu. Zaczniemy od głębszego rozpatrzenia pierwszego etapu (0). W tej sekcji nie będziemy dużo o nim pisać, tylko po krótko objaśnimy diagram [23].

Na początku możemy zauważyć w lewym górnym rogu operacje $Maj(x)$, do którego wskazują nasze H_{0a}, H_{0b}, H_{0c} w skrócie będziemy je zapisywać jako a, b, c. Funkcja $Maj(x)$ w naszym algorytmie dana jest wzorem

$$Maj(a, b, c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c).$$

Następnie po lewej stronie widać grot z a który trafia w funkcje $\Sigma_0(x)$ w SHA-2 danej wzorem

$$\Sigma_0(a) = ROTR^2(a) \oplus ROTR^{13}(a) \oplus ROTR^{22}(a)$$

W dalszej części możemy zauważyć symbol plus, do którego są skierowane funkcje $Maj(x)$ i $\Sigma_0(a)$. Plus symbolizuje dodawanie wartości tych funkcji, w już wcześniej wspomnianym $\mathbb{Z}_{2^{32}}$. Następnie widzimy kolejny symbol dodawania, w którym modyfikowane są skumulowane dane z prawej strony diagramu. W tym miejscu się zatrzymamy i przejdziemy do lewej strony schematu.

Przechodzimy teraz do kwadratu oznaczonego jako $Ch(x)$ czyli funkcji warunkowej. Jak widać argumentami tej funkcji będą nasze e, f, g, które będą modyfikowane na wartość wejściową następnych operacji. Funkcja $Ch(x)$ dana jest wzorem

$$Ch(e, f, g) = (e \wedge f) \oplus (\neg e \wedge g).$$

Na lewo od funkcji $Ch(x)$ widzimy blok Σ_1 do której strzela grot z e.

$$\Sigma_1(e) = ROTR^6(e) \oplus ROTR^{11}(e) \oplus ROTR^{25}(e).$$

Po przekonwertowaniu e, wartość funkcji $\Sigma_1(e)$ trafia do dodawania modularnego z dodanymi już wcześniej wartościami h i $Ch(x)$. Wynik z poprzedniego działania jest sumowany z W_0 i K_0. Następnie dochodzimy do kolejnego przetworzenia naszych danych. Zsumowane modularnie wartości Σ_1, W_0, K_0 (oznaczmy je przez X) razem z d przetwarzane są w środkowym bloku z plusem, a następnie stają się wartością $e = H_{0e}^{(1)}$ dla kolejnego etapu (1).

Tutaj wracamy do lewej strony naszej ilustracji gdzie dodaliśmy modularnie $Maj(x)$ i $\Sigma_1(e)$, które oznaczymy przez Y. Finalnie dochodzimy do ostatniej operacji sumujemy X i Y, których suma staje się wartością $a = H_{0a}^{(1)}$ następnego etapu. Pozostałe dane wejściowe zmiennej $H_0^{(1)}$ wyznaczane są następująco

- $H_{0b}^{(1)} = H_{0a}$,
- $H_{0c}^{(1)} = H_{0b}$,
- $H_{0d}^{(1)} = H_{0c}$,
- $H_{0f}^{(1)} = H_{0e}$,
- $H_{0g}^{(1)} = H_{0f}$,
- $H_{0h}^{(1)} = H_{0g}$.

5.3 Zakończenie

Finalnie po wykonaniu 64 etapów otrzymujemy wartości $H_{0a}^{(63)} - H_{0h}^{(63)}$. Tutaj wykonuje się ostatnia operacja przed przejściem do następnego bloku. Niech L_i będzie dane wzorem

$$L_i = H_{0i}^{(63)} + H_{0i} \qquad \text{dla } i \in \{a, b, \ldots, h\} = G$$

gdzie + oznacza dodawanie modularne. Wtedy wartość H_1 otrzymujemy ze wzoru

$$H_1 = \sum_{i \in G} L_i.$$

gdzie dodawanie to zwyczajne dopisanie pierwszej wiadomości w zapisie binarnym do drugiej.

$$+ : \{0,1\}^n \times \{0,1\}^k \ni (x, y) \longmapsto xy \in \{0,1\}^{n+k}$$

Parametr wejściowy H_1 przekazujemy do algorytmu kolejnego bloku gdzie jest on dzielony z powrotem na 8 części oznaczone $H_1a - H_1h$. Proces ten powtarzamy dla kolejnych bloków, aż do uzyskania kluczowej wartości skrótu H_l. Warto zwrócić uwagę, że H_l jest 256-bitową wartością skrótu, która powstaje z 8 liczb 32-bitowych. Skrót ten zapisywany jest w 64 znakach w systemie szesnastkowym, jak wspomniano wcześniej. Nasza wiadomość „ Przykład dla algorytmu SHA-2 ” po przejściu przez funkcje skrótu zostanie zamieniona na

$$881E383553BDD10E5B9084AF3E0F3277A93BAB61D5CFEF60D8585BDE.$$

6 Funkcja MD5

6.1 Wprowadzenie

MD5 (z ang. *Message-Digest algorithm 5*) to kryptograficzna funkcja skrótu opracowana w 1991 roku przez Rona Rivesta jako bezpieczny następca funkcji MD4, która wykazała w bezpieczeństwie pewne braki. Funkcja MD5 z podanego jej na wejściu ciągu danych generuje 128-bitowy skrót.

6.2 Działanie algorytmu MD5 na przykładzie

Poniżej przedstawimy na przykładzie prostej wiadomości (łańcucha znaków) działanie algorytmu MD5.

Łańcuch wejściowy: *Kryptografia i funkcje*

Pierwszym krokiem jaki musimy podjąć przed działaniem właściwego algorytmu jest przekonwertowanie naszego łańcucha znaków na wersję binarną. Robimy to oczywiście w oparciu o standard ASCII.

Kryptografia i funkcje = 01001011 01110010 01111001 01110000 01110100 01101111 01100111 01110010 01100001 01100110 01101001 01100001 00100000 01101001 00100000 01100110 01110101 01101110 01101011 01100011 01101010 01100101

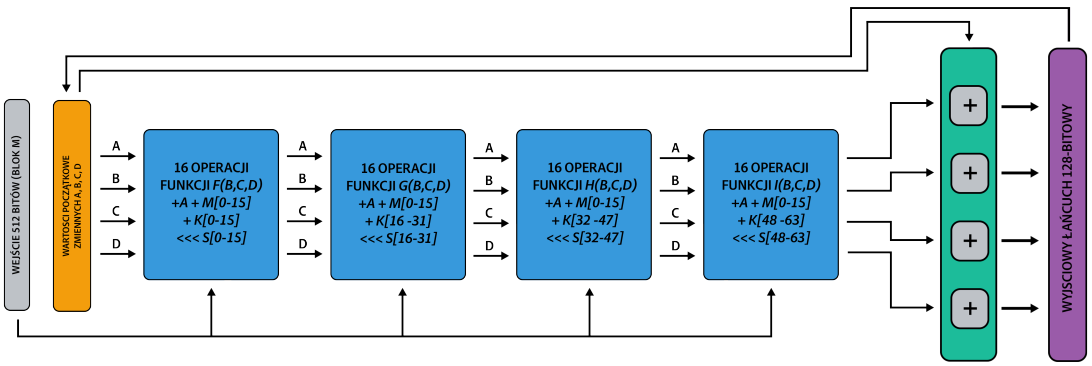
6.2.1 Wypełnianie łańucha i objaśnienia potrzebnych stałych i zmiennych

Na początku przechodzimy do tak zwanego wypełnienia naszego łańcucha. Funkcja MD5 działa na wejściach dzielonych na bloki, każdy z nich ma 512 bitów. Łańcuch wejściowy rozpatrywany przez nas w przykładzie ma 22 znaki (licząc oczywiście ze spacjami), każdy z nich liczy 8 bitów czyli mamy 176 bitów. Pozostaje nam 336 bitów do zagospodarowania. Funkcja MD5 robi to w taki sam sposób jak SHA-2. Tutaj pokażemy to na konkretnej wiadomości.

Docelowo MD5 dopełnia łańcuch bitami w taki sposób by jej długość była o 64 bity krótsza od wielokrotności 512 bitów (bity są dodawane nawet kiedy warunek jest spełniony). Do naszego łańcucha jest „doklejany" niejako jeden bit o wartości 1. W naszym prostym przypadku mamy więc $336 - 64 - 1 = 271$ bitów do zagospodarowania. Miejsca te są wypełniane bitami o wartości 0. Pozostało nam dokleić długość naszej wiadomości w zapisie binarnym ($176 = 10110000$), resztę miejsc (czyli 56 pozostałych bitów) wypełniamy zerami.

A więc naszym wejściowym łańcuchem obecnie jest:

01001011 01110010 01111001 01110000 01110100 01101111 01100111 01110010 01100001
01100110 01101001 01100001 00100000 01101001 00100000 01100110 01110101 01101110
01101011 01100011 01101010 01100101 10000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 10110000



Rysunek 5: Schemat działania algorytmu MD5
Źródło: Opracowanie własne na podstawie [19]

Następnym krokiem będzie podzielenie wejściowego łańcucha na szesnaście 32-bitowych łańcuchów, a następnie zamiana ich na system szesnastkowy. Oznacza się je literą M z indeksami od 0 do 15. W naszym przypadku będzie to:

M_i	**Reprezentacja binarna**	**Reprezentacja szesnastkowa**
M_0	01001011011100100111100101110000	4B727970
M_1	01110100011011110110011101110010	746F6772
M_2	01100001011001100110100101100001	61666961
M_3	00100000011010010010000001100110	20692066
M_4	01110101011011100110101101100011	756E6B63
M_5	01101010011001011000000000000000	6A658000
M_6	00000000000000000000000000000000	00000000
M_7	00000000000000000000000000000000	00000000
M_8	00000000000000000000000000000000	00000000
M_9	00000000000000000000000000000000	00000000
M_{10}	00000000000000000000000000000000	00000000
M_{11}	00000000000000000000000000000000	00000000
M_{12}	00000000000000000000000000000000	00000000
M_{13}	00000000000000000000000000000000	00000000
M_{14}	00000000000000000000000000000000	00000000
M_{15}	00000000000000000000000010110000	000000B0

Tabela 6: Reprezentacja binarna i szesnastkowa M_i

Każdy z tych bloków będzie służył jako wejście do prowadzonych przez nas operacji w określonym porządku. Dla ustalenia uwagi w dalszej części pracy zobrazowane na schemacie niebieskie bloki będziemy nazywać cyklami, każdy z cykli będzie dzielił się na 16 części, które dla odróżnienia będziemy nazywać operacjami. Mamy więc 4 cykle po 16 operacji, czyli 64 rundy. W każdej cyklu algorytmu MD5 każdy z M_j inputów jest używany, jednak dodawane są one w różnej kolejności. Kolejność ta zależy od rundy i cyklu w którm używana jest M_j. Niech $i \in \{0, \ldots, 63\}$ oznacza numer rundy przy numerowaniu od zera wówczas

- W pierwszym cyklu: $j = i$, czyli użyte zostaną M_0, M_1, M_2, ..., M_{15}.
- W drugim cyklu: $j = (5i+1) \mod 16$ dla $i \in \{16, \ldots, 31\}$. Użyte zostaną $M_1, M_6, \ldots, M_{12}$.
- W trzecim cyklu: $j = (3i + 5) \mod 16$ dla $i \in \{32, \ldots, 47\}$.
- W czwartym cyklu: $j = 7i \mod 16$ dla $i \in \{48, \ldots, 63\}$.

W każdej operacji algorytm działa na zmiennych A, B, C, D, których początkowe wartości są ustalone w RFC 1321, a są to:

- $A = 01234567$
- $B = 89ABCDEF$
- $C = FEDCBA98$
- $D = 76543210$

Jak widać na schemacie w poszczególnych cyklach na zmiennych B, C, D działają funkcje F, G, H, I, są to funkcje boolowskie o następujących wzorach:

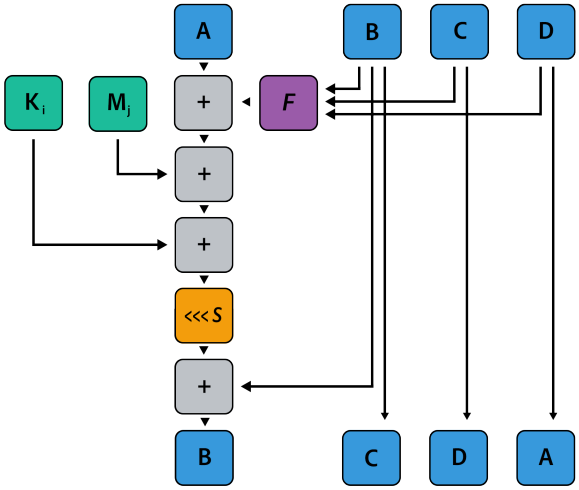
- $F(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$
- $G(B, C, D) = (B \wedge D) \vee (C \wedge \neg D)$
- $H(B, C, D) = B \oplus C \oplus D$
- $I(B, C, D) = C \oplus (B \wedge \neg D)$

W każdej rundzie będzie nam potrzebna także stała K_i, gdzie $i \in \{0, ..., 63\}$. Jest ona wyliczana ze wzoru $|\sin(i + 1) \cdot 2^{32}|$.

K_i są kolejno używane w każdej z 64 rundach przypadających na 512-bitowy łańcuch. $K_0, K_1, \ldots, K_{15}$ w pierwszym cyklu, $K_{16}, K_{17}, \ldots, K_{31}$ w drugim i tak dalej.

6.2.2 Przykład

Schemat prezentujący działanie cyklu pierwszego.



Rysunek 6: Schemat cyklu pierwszego algorytmu MD5 z działaniami wewnętrznymi
Źródło: Opracowanie własne na podstawie [19]

Poniżej przyjrzymy się pojedynczej operacji, aby lepiej zrozumieć działanie algorytmu MD5. W pierwszym cyklu kluczową rolę odgrywa funkcja F. W pierwszej rundzie F przyjmie jako argumenty startowe wartości zmiennych B, C i D. Przypomnijmy, że są to $B = 89ABCDEF$, $C = FEDCBA98$, $D = 76543210$.

$$F(89ABCDEF, FEDCBA98, 76543210) = 89ABCDEF \wedge FEDCBA98) \vee (\neg 89ABCDEF \wedge 76543210)$$

$$= FEDCBA98$$

Następnym krokiem będzie dodanie obliczonej wartości funkcji F do obecnej (startowej) wartości zmiennej A. Jest to oczywiście dodawanie modulo 2^{32}. Mamy więc $(01234567 + FEDCBA98) \mod 100000000 = FFFFFFFF$. Jak widać na schemacie w następnym kroku, czeka nas kolejne dodawanie modulo. Tym razem jego składnikiem obok wyniku poprzedniej operacji będzie łańcuch bitów M_i (czyli najpierw $M_0 = 4B727970$). Otrzymujemy $(4B727970 + FFFFFFFF) \mod 100000000 = 4B72796F$. W następnym kroku dodajemy do otrzymanej wartości stałą K_i ($K_1 = D76AA478$). Mamy więc $(4B72796F + D76AA478) \mod 100000000 = 22DD1DE7$ Nadszedł czas na przesunięcie bitowe otrzymanej wartości, w tym celu zamieniamy ją na system dwójkowy.

$$22DD1DE7 = 00100010110111010001110111100111$$

Blok $<<< S$ oznacza przesunięcie bitowe wartości o określoną stałą S_i, jest ona różna w zależności od numeru rundy. Wartość stałej S_i zależy od dwóch czynników: reszty z dzielenia i przez 4 oraz całkowitego ilorazu dzielenia i przez 16 (i div 16).

Tabela 7: Tabela Wartości S_i

		i mod 4			
		0	**1**	**2**	**3**
i div 16	**0**	7	12	17	22
	1	5	9	4	20
	2	4	11	16	23
	3	6	10	15	21

Przykładowo dla cyklu drugiego wszystkie rundy których numer daje reszte 1 przy dzieleniu przez 4: $S_{17} = S_{21} = S_{25} = S_{29} = 9$. Natomiast dla cyklu pierwszego wszystkie rundy, których numer jest podzielnych przez 4: $S_0 = S_4 = S_8 = S_{12} = 7$. W tej rundzie następuje więc przesunięcie o 7 bitów w lewo.

$$00100010110111010001110111100111 = 00100010110111010001110111XXXXXXX$$
$$= 00100010110111010001110111\mathbf{0010001} = 22DD1D91$$

Po przesunięciu bitowym następuje kolejne dodawanie modulo tym razem drugim składnikiem będzie poprzednia wartość zmiennej B (w naszym przykładzie nazwana startową)

$(22DD1DE7 + 89ABCDEF) \mod 100000000 = AC88EB80$.

Tak jak obrazuje to schemat otrzymana wartość w następnej rundzie stanie się początkową wartością zmiennej B, zmienna A otrzymuje wartość zmiennej D,C zmiennej B, a D zmiennej C. Będą to startowe wartości zmiennych A, B, C, D w następnej rundzie.

W drugiej rundzie postępujemy tak samo jak w pierwszej, różni się ona jedynie wejściami. Po szesnastej operacji algorytm przechodzi do następnego niebieskiego „bloku" ze schematu nazywanego przez nas cyklem. Działanie każdego z nich jest bardzo podobne, różni się dodawanymi modularnie stałymi K_i i ilościami przesunięć bitowych S_i i co najważniejsze funkcją która działa na zmienne B, C, D. Po ostatniej rundzie w bloku w którym na zmiennych działa funkcja I obecne wartości zmiennych A, B, C, D są dodawane modularnie do ich wartości z samego początku działania algorytmu. Na końcu zmienne są łączone w jeden łańcuch $ABCD$ i otrzymujemy wyjście – 128 bitowy hash, który w naszym przykładzie wygląda następująco:

$$H(\textit{Kryptografia i funkcje}) = 3491DFA59AD78C8BAE9D00960F9B9258$$

6.3 Bezpieczeństwo MD5

Obecnie MD5 nie jest uważana za bezpieczną kryptograficzną funkcję skrótu. W 2004 roku przeprowadzono udany atak, dowiedli tego Xiaoyun Wang and Hongbo Yu w swojej pracy *How to Break MD5 and Other Hash Functions* [10].

6.3.1 Atak kryptoanalityczny na MD5

Ataki kryptoanalityczne opierają się na badaniu haszy, wiadomości szyfrowanych i schematów działania funkcji skrótu, w celu znajdowania technik tworzenia kolizji. Zadaniem ataku jest wychwycenie słabości w schematach szyfrujących. Kryptoanaliza funkcji skrótu opiera się również głównie na algorytmie funkcji i sposobach kompresji danych. Istotą tego zagadnienia jest zazwyczaj analiza zmian bitowych z rundy na rundę, co pozwala wychwycić schematy różnicowe [12].

Jedną z najbardziej skutecznych metod jest atak różnicowy. Sposobem działania tego ataku jest znalezienie dwóch wiadomości dających ten sam skrót. Różnica jest zazwyczaj definiowana jako funkcja logiczna XOR, a filozofia ataku wykorzystuje fakt, iż poprzez zmianę kilku bitów w wiadomości prawdopodobne jest zniwelowanie różnicy wewnątrz funkcji kompresującej po kilku rundach. Ten atak możemy podzielić na dwa etapy [12].

- W pierwszym etapie analizuje się uproszczone modele funkcji i szuka ścieżki różnicowej o dużym prawdopodobieństwie tworzenia kolizji.
- W drugim etapie określa się zbiór warunków, które zapewniają propagację różnicy według ścieżki i szuka rozwiązania.

Postaramy się przybliżyć architekturę działania udanego ataku przeprowadzonego przez Wanga na funkcje MD5 wspominanego przez nas wcześniej [14]. Metoda złamania funkcji polega na znalezieniu pary 1024-bitowych wiadomości, które dadzą tę samą wartość skrótu przez MD5. Oznaczymy je przez $M = (M_0, M_1)$ i $M' = (M'_0, M'_1)$, gdzie każde M_i i M'_i jest blokiem 512-bitowym. Atak na MD5 Wanga jest atakiem różnicowym, który wykorzystuje nie tylko funkcję logiczną XOR ale również innowacyjne odejmowanie modularne.

Rozważmy parę $x = 01010010$ i $x' = 01000010$ oraz pare $y = 10100011$ i $y' = 10110011$. Wówczas

$$x - x' = y - y' = 2^4$$

Co oznacza, że te pary są podobne jeśli chodzi o różnicę modularną z dokładnością do 2^4. Teraz znajdziemy różnice XOR.

$$y \oplus y' = 10100011 \oplus 10110011 = 00010000$$

Różny od zera wynik pokazuje nam, że różnica w ciągach bitów znajduje się na trzecim miejscu, licząc od początku. Dla ułatwienia oznaczymy przez $y = (y_0, y_1, \ldots, y_7)$ i $y' = (y'_0, y'_1, \ldots, y'_7)$. Oznacza to, że $y_4 - y'_4 \neq 0$. Co warto zauważyć nie zależnie czy $y_4 = 0 \wedge y'_4 = 1$, czy $y_4 = 1 \wedge y'_4 = 0$, wynik XOR byłby identyczny. Atak wymaga więc więcej informacji niż to. Niech symbol ∇y

oznacza różnice XOR, $\nabla y = y \oplus y' = 00010000$ z tym że potrzebujemy jeszcze informacji o kolejności zer i jedynek więc funkcje ∇ będziemy definiować następująco

Definicja 6.1. Niech $y, y^{'} \in \{0,1\}^k$ wówczas ∇y to różnica XOR przykładowo $\nabla y = y \oplus y' = (... - ....)$, gdzie " . " oznacza 0 w działaniu XOR, " - " oznacza, że pierwszy w kolejności pojawia się zero a druga jedynka. Na przykładzie $0 \oplus 1 = (-)$ i analogicznie $1 \oplus 0 = (+)$.

Porównajmy tym razem nasz x i x'.

$$\nabla x = x \oplus x' = 01010010 \oplus 01100010 = (.. - +....).$$

Definicja 6.2. Δy będzie naszą różnicą modularną, czyli dla x $\Delta x = x - x' = 2^4$.

W taki właśnie sposób, analizowane są różnice bitowe dla iteracji w kolejnych krokach algorytmu w atakach różnicowych.

6.3.2 Ścieżka różnicowa Wanga

Nasz wektor początkowy, zmiennych początkowych, wcześniej wspomnianych A, B, C, D oznaczymy przez $IV = IV_0 = (A, B, C, D)$. Wektor zmiennych wejściowych dla następnego bloku (w naszym przypadku M_1) będzie zapisywany jako $IV_1 = (A_1, B_1, C_1, D_1)$, gdzie A_1, B_1, C_1, D_1 to wartości wyjściowe po 4 cyklu dla bloku M_0. Wówczas $H(M) = H(M_0, M_1) = h$. Tak samo będziemy zapisywać wartość h' i IV_1' dla M'.

Teraz kiedy mamy wszystkie wstępne oznaczenia, możemy zdefiniować ścieżkę różnicową opracowaną przez zespół Wanga, która jest jak mapa do znalezienia kolizji. W poszukiwaniu kolizji nie są dla nas istotne realne wartości M i M'. Interesuje nas tylko ΔM [13]. Szukamy ciągu różnic Δi takiego, że $\exists_i (\Delta i = 0)$ to znaczy, że istnieje liczba iteracji lub kompresji, dla których różnica haszy zanika [15]. Dzięki pracy grupy znana nam jest taka ścieżka różnicowa i wyglądaja ona następująco

$$\Delta M_0 = M_0' - M_0 = (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, 2^{15}, 0, 0, 2^{31}, 0),$$

$$\Delta M_1 = M_1' - M_1 = (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, -2^{15}, 0, 0, 2^{31}, 0).$$

Taki ciąg różnicowy narzuca nam dalsze wartości dla IV_1, IV_1'. Oznacza to, że

$$\Delta IV_1 = IV_1' - IV_1 = (2^{31}, 2^{25} + 2^{31}, 2^{25} + 2^{31}, 2^{25} + 2^{31}).$$

W taki sposób dobrane wartości M, M' gwarantują nam, że $\Delta h = (0, 0, 0, 0)$ czyli $H(M) = H(M')$. Powstaje pytanie, czy znane nam są inne ścieżki i jeśli nie to jak ich sukcesywnie szukać [13]. Okazuje się, że jest to niezwykle trudne zadanie, które przez wiele lat było nierozwiązywalne. Jeśli jednak jest to na tyle trudne to, w jaki sposób udało się znaleźć taką ścieżkę i czy da się to powtórzyć? Niestety grupa odpowiedzialna za tę pracę nie daje kluczowych informacji, a sukces uzasadnia intuicją.

Co ważne różnice modularne dla wartości wejściowych M_0 i M_1 nie wystarczą do udanego ataku, kluczowa jest również znajomość różnić wewnątrz funkcji, czyli naszych wartości tymczasowych podczas obliczania skrótu (z uwagi na dużą ilość tych wytycznych pominiemy ich opis).

6.3.3 Idea ataku

Znalezienie ścieżki kolizji, chociaż jest ogromną trudności, nie sprawia, że atak uznany jest za udany. Należy jeszcze znaleźć odpowiedni argument M. Szukanie odpowiedniego argumentu i dalszą część ataku opiszemy w podpunktach [15].

- Generujemy losowy ciąg 512-bitowy (nasze M_0).
- Modyfikujemy nasz Blok M_0 tak by dało się go wpasować w ścieżkę różnicową. Jeśli jest to nie możliwe, zaczynamy od nowa. Warunki, które musi spełniać wygenerowana wiadomość, pominiemy w naszej pracy z uwagi na zbyt dużą złożoność (wiążą się one ściśle ze ścieżką różnicową).
- Jeśli wszystkie warunki są spełnione, to udało nam się znaleźć M_0.
- Używamy M_0 i funkcji H by znaleźć wektor IV_1 czyli naszą wartość początkową dla M_1.
- Generujemy losowy M_1 spełniające wyżej wspomniane warunki.

- Jeśli wszystkie warunki są spełnione, to udało nam się znaleźć M_1.
- Używamy naszej ścieżki, by znaleźć M'_0 i M'_1.

$$M'_0 = \Delta M_0 + M_0$$

$$M'_1 = \Delta M_1 + M_1$$

Takim sposobem otrzymujemy parę (M, M') takie, że $H(M) = H(M')$.

W latach 2004 r. kolizja Wanga była jedyną znaną na świecie [12]. Do dziś dzień udało się znaleźć zaawansowane i efektywne algorytmy szukające nowych ścieżek. Na ten moment znajdywanie nowej kolizji dla MD5 to kwestia około minuty.

7 Zastosowania MD5

Przed wykazaniem słabości MD5, miała ona wiele, typowych dla funkcji skrótu zastosowań jak podpisy cyfrowe, weryfikacja integralności danych, szczególnie tych pobranych z Internetu, czy uwierzytelnianie haseł.

7.1 Hashowanie haseł

MD5 było używane do przechowywania skrótów haseł użytkowników w bazach danych. Umożliwiało to sprawdzanie poprawności hasła podczas procesu logowania bez konieczności przechowywania hasła w formie jawnej [2]. Z perspektywy zastosowania funkcji skrótu, jakim jest hashowanie haseł najważniejszą jej właściwością jest tzw. słaba odporność na kolizje. Wynika to z tego, że atakujący może manipulować tylko z jednym wejść podczas próby znalezienia pasującego hashu hasła. MD5 nie została jednoznacznie "złamana" pod tym względem, więc można pomyśleć, że jest ona nadal dobrym rozwiązaniem, jeśli chodzi o hashowanie haseł. Istnieją jednak czynniki, które sprawiają, że MD5 się do tego nie nadaje. Jednym z nich jest szybkość algorytmu. Krótkie i proste hasła, hashowane za pomocą MD5 są podatne na tzw. ataki Rainbow Table czy po prostu ataki Brute Force. Jeśli atakujący posiada bazę haseł zahashowanych przez MD5, szybkość tego algorytmu pozwoli mu szybko odgadnąć hasła pasujące do hashy w bazie danych [2]. Można tu podać przykład wycieku danych z Ashley Madison, gdzie grupa CynoSure Prime była w stanie w zaledwie kilka godzin uzyskać miliony haseł użytkowników, ponieważ były one hashowane przez MD5 [21]. Ze względu na te problemy National Institute of Standards and Technology nie rekomenduje już MD5 do zastosowania, jakim jest hashowanie haseł [22] .

Mimo braków w bezpieczeństwie i odporności na kolizje funkcja MD5 nadal znajduje zastosowanie we współczesnej kryptografii stosowanej.

7.2 Sprawdzanie integralności danych

Zanim omówimy to zastosowanie funkcji MD5 ważne jest wyróżnienie typów weryfikacji danych. A więc wyróżniamy tutaj:

- Weryfikacje integralności polegającą na sprawdzeniu, czy dane nie zostały uszkodzone przypadkowo. Myślimy tu o sytuacji, w której zostały wprowadzone w danych niezamierzone zmiany powodujące utratę pewnych informacji czy utrudnienia w dostępie do nich. Zmiany takie mogą mieć miejsce, chociażby w wyniku błędów w transmisji danych, bugów w oprogramowaniu czy wszelkiego rodzaju problemów z nośnikiem danych.
- Weryfikacje danych pod kątem ich autentyczności to znaczy weryfikację tego czy dane nie zostały potajemnie zmienione podczas przechowywania, czy przesyłania w celu wprowadzenia w nie niepożądanych rzeczy takich jak złośliwe oprogramowanie [2].

Kryptograficzne funkcje skrótu mogą być używane pod kątem weryfikacji integralności jak i autentyczności danych, jednak MD5 nadaje się jedynie do pierwszej z nich [2].

Weryfikacja integralności pliku za pomocą funkcji skrótu MD5 odbywa się przez liczenie przez nią sumy kontrolnej dla danego pliku. Jako przykład można tu podać stosowanie polecenia md5sum w systemach operacyjnych z rodziny Unix/Linux. Polecenie to oblicza sumę kontrolną MD5 danego pliku na podstawie jego zawartości [8]. Kluczową rolę odgrywa tutaj własność funkcji skrótu, jaką jest determinizm. Przypomnijmy, że własność ta mówi, że wynik dla danego wejścia jest zawsze taki sam, a nawet niewielkie zmiany wejścia powodują powstanie całkiem innego skrótu. Mimo

pomyślnego ataku na funkcję MD5 w kontekście kolizji nie stanowi on dużego problemu, gdy używamy algorytmu jako sumy kontrolnej do weryfikacji integralności danych. Istnieje możliwość, że dwa osobne pliki będą miały ten sam wartość skrótu, ale prawdopodobieństwo, że plik ulegnie uszkodzeniu w taki specyficzny sposób, że nadal będzie produkował ten sam skrót, jest tak małe, że je pomijamy. Jak zostało jednak wspomniane wcześniej, z powodu braku odporności na kolizje nie powinno się używać MD5 do weryfikacji autentyczności danych. Z podobnych powodów MD5 nie nadaje się do innych zastosowań, w których wymagane jest uwierzytelnianie, takich jak w cyfrowych podpisach i certyfikatach SSL [2].

8 Zastosowania SHA-2

SHA-2, a w szczególności jego wariant SHA-256, ma kluczowe znaczenie w świecie kryptowalut, a zwłaszcza w Bitcoinie. SHA-256 jest wykorzystywana do zapewnienia integralności i bezpieczeństwa transakcji oraz w procesie wydobywania nowych bloków w sieci blockchain.

8.1 Hashowanie bloków w Bitcoinie

SHA-256 pełni centralną rolę w systemie blockchain Bitcoina. Każdy blok w blockchainie jest identyfikowany przez swój hash, który jest obliczany przy użyciu SHA-256. Zastosowanie tego algorytmu zapewnia, że jakakolwiek modyfikacja w bloku (np. zmiana transakcji) skutkuje kompletnie innym wynikiem skrótu, co uniemożliwia nieautoryzowane zmiany w łańcuchu bloków. SHA-256 jest również używany do tworzenia adresów portfeli Bitcoin. Adresy te są generowane poprzez haszowanie publicznych kluczy użytkowników za pomocą SHA-256, co dodatkowo zwiększa bezpieczeństwo transakcji i chroni prywatność użytkowników. W ten sposób nawet znając adres portfela, nie można odtworzyć oryginalnego klucza publicznego ani prywatnego. Nowe bloki w blockchainie są odkrywane poprzez skomplikowane obliczenia komputerowe, które wymagają dużych zasobów obliczeniowych bazujących na SHA-256. Maszyny, które skutecznie rozwiązują te zadania, są wynagradzane nowo wydobytymi jednostkami waluty, dzięki czemu ustalana jest realną wartość Bitcoina. Mechanizm ten zapewnia stabilność wartości, co czyni Bitcoin atrakcyjnym aktywem inwestycyjnym i środkiem wymiany w cyfrowym świecie [24].

8.2 Weryfikacja integralności danych

Podobnie jak MD5, SHA-256 może być używany do weryfikacji integralności danych. Algorytm ten zapewnia znacznie wyższy poziom bezpieczeństwa, zwłaszcza w porównaniu do MD5, dzięki swojej odporności na kolizje i szybkiemu obliczaniu skrótu. W kryptowalutach haszowanie transakcji jest niezbędne, aby zapobiec ich modyfikacji, a zastosowanie SHA-256 w tej roli gwarantuje wysoką odporność na ataki kryptograficzne [24].

Podsumowanie

Podsumowując, niniejsza praca miała na celu zbadanie funkcji skrótu w kontekście kryptografii oraz analizę ich praktycznego zastosowania. Głównym celem było zdefiniowanie funkcji skrótu, omówienie ich własności, analiza zagrożeń oraz prezentacja konkretnych przykładów algorytmów, takich jak MD5 i SHA-2. W trakcie pracy udało się zdefiniować funkcje skrótu oraz przedstawić ich kluczowe właściwości. Zidentyfikowaliśmy również zagrożenia, jakim mogą być podatne, oraz omówiliśmy praktyczne implikacje złamania funkcji MD5, co stanowiło ważny wniosek z punktu widzenia bezpieczeństwa danych. Należy jednak zauważyć, że istnieją pewne ograniczenia przeprowadzonych badań, takie jak ograniczona analiza konkretnych algorytmów funkcji skrótu oraz brak uwzględnienia najnowszych metod ataków. Dlatego sugerujemy kontynuację badań w tym obszarze, ze szczególnym uwzględnieniem rozwijania bardziej odpornych na ataki funkcji skrótu takich jak, chociażby rekomendowana obecnie funkcja SHA-3 oraz analizy nowych zagrożeń.

Bibliografia

[1] Oded Goldreich *Foundations of Cryptography Basic Tools*, Press Syndicate Of The University of Cambridge, 2004 , str. 33.

[2] Josh Lake *comparitech.com* https://www.comparitech.com/blog/information-security/md5-algorithm-with-examples/ Dostęp: 19 czerwca 2025.

[3] Josh Lake *comparitech.com* https://www.comparitech.com/blog/information-security/what-is-md5/ Dostęp: 19 czerwca 2025.

[4] Odporność na kolizje https://pl.wikipedia.org/wiki/Second_preimage_resistance dostęp: 19 czerwca 2025

[5] Damian Niwiński *Funkcje skrótu* https://www.mimuw.edu.pl/~niwinski/Crypto/wyklad3.pdf dostęp: 19 czerwca 2025

[6] Funkcja zaniedbywalna https://pl.wikipedia.org/wiki/Funkcja_zaniedbywalna_(kryptografia) dostęp: 19 czerwca 2025

[7] Dokumentacja RFC1321 https://datatracker.ietf.org/doc/html/rfc1321 - dostęp 19 czerwca 2025

[8] Dokumentacja md5sum https://man.archlinux.org/man/md5sum.1.pl dostęp 19 czerwca 2025

[9] Wahome Macharia *Cryptographic Hash Function* https://www.researchgate.net/profile/Wahome_Macharia/publication/351837904_Cryptographic_Hash_Functions/links/60aca8aa45851522bc156067/Cryptographic-Hash-Functions.pdf dostęp 19 czerwca 2025

[10] Xiaoyun Wang, Hongbo Yu *How to Break MD5 and Other Hash Functions* https://web.archive.org/web/20090521024709/http://merlot.usc.edu/csac-f06/papers/Wang05a.pdf dostęp 19 czerwca 2025

[11] Thomas H. Cormen Charles E.Leiserson Ronald L.Rivest *Wprowadzenie do algorytmów*, Wydawnictwa Naukowo-Techniczne Warszawa, wydanie 4, 1997,2001, str. 1025-1026

[12] Krystian Matusiewicz *Ataki na kryptograficzne funkcje skrótu* http://www2.mat.dtu.dk/people/oldusers/K.Matusiewicz/talks/enigma2008-slides-matusiewicz.pdf dostęp: 19 czerwca 2025

[13] Antoine Delignat-Lavaud *Collisions in MD5* https://antoine.delignat-lavaud.fr/doc/slides_md5.pdf dostęp: 19 czerwca 2025

[14] Narayana D. Kashyap *A Meaningful MD5 Hash Collision Attack* https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1020&context=etd_projects dostep: 19 czerwca 2025

[15] Idea ataku https://www.cs.sjsu.edu/~stamp/crypto/PowerPoint_PDF/16_MD5.pdf dostęp: 19 czerwca 2025

[16] Wikipedia.com, Algorytm probabilistyczny, https://pl.wikipedia.org/wiki/Algorytm_probabilistyczny

[17] Oded Goldreich *Foundations of Cryptography II Basic Applications*, Press Syndicate Of The University of Cambridge, 2004, str. 513.

[18] Ilya Mironov *Hash functions: Theory, attacks, and applications* http://crypto.stanford.edu/~mironov/papers/hash_survey.pdf dostep: 19 czerwca 2025

[19] Josh Lake *comparitech.com* https://www.comparitech.com/blog/information-security/what-is-sha-2-algorithm/ dostep: 19 czerwca 2025

[20] Josh Lake *comparitech.com* https://www.comparitech.com/blog/information-security/what-is-a-collision-attack/ dostep: 19 czerwca 2025

[21] CynoSure Prime, How we cracked millions of Ashley Madison bcrypt hashes efficiently, https://blog.cynosureprime.com/2015/09/how-we-cracked-millions-of-ashley.html

[22] Recommendation for Password-Based Key Derivation Part 1: Storage Applications, Meltem Sönmez Turan, Elaine Barker, William Burr, and Lily Chen https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf

[23] Operacja na blokach https://en.wikipedia.org/wiki/SHA-2 dostęp: 19 czerwca 2025

[24] Zastosowania SHA-2 https://www.comparitech.com/blog/information-security/what-is-sha-2-how-does-it-work/ dostęp: 19 czerwca 2025