**Autorstwo i patenty, ujęcia w standardach**

**SHA-3 czy Secure Hash Algorithm 3** – kryptograficzna funkcja skrótu wyłoniona w 2012 roku.

SHA-3 jest podzbiorem szerszy kryptograficznego pierwotnej rodziny Keccak, zaprojektowany przez Guido Bertoni , Joan Daemen , Michael Peeters i Gilles Van Assche. 2 października 2012 roku Keccak został zwycięzcą konkursu z tym algorytmem.

Algorytm Keccak charakteryzuje się wyższą wydajnością niż SHA-2 zarówno w implementacjach sprzętowych jak i programowych.

**Zastosowania w istniejących kryptosystemach**

NIST obecnie nie planuje wycofania SHA-2 ani usunięcia go ze zmienionego Secure Hash Standard. Celem SHA-3 jest to, że można go bezpośrednio zastąpić SHA-2 w obecnych aplikacjach, jeśli jest to konieczne, oraz aby znacznie poprawić niezawodność ogólnego zestawu narzędzi algorytmu mieszania NIST.

W 2014 roku NIST opublikował projekt FIPS 202 „Standard SHA-3: funkcje skrótu opartego na permutacji i rozszerzalnych funkcji wyjściowych”. FIPS 202 został zatwierdzony 5 sierpnia 2015 r. 5 sierpnia 2015 NIST ogłosił, że SHA-3 stał się standardem mieszającym.

Dla dokładnego SHA3-256 na x86-64 Bernstein mierzy 11,7–12,25 cpb w zależności od procesora. SHA-3 był krytykowany za powolne działanie na architekturach zestawów instrukcji (CPU), które nie mają instrukcji przeznaczonych specjalnie do obliczania funkcji Keccak - SHA2-512 jest ponad dwukrotnie szybszy niż SHA3-512, a SHA-1 jest ponad trzykrotnie szybciej na procesorze Intel Skylake o taktowaniu 3,2 GHz.

Autorzy zareagowali na tę krytykę, sugerując użycie SHAKE128 i SHAKE256 zamiast SHA3-256 i SHA3-512, kosztem zmniejszenia o połowę rezystancji przedobrazu (ale zachowując odporność na zderzenie). Dzięki temu wydajność jest porównywalna z SHA2-256 i SHA2-512. Jednak w implementacjach sprzętowych SHA-3 jest znacznie szybszy niż wszyscy inni finaliści, a także szybszy niż SHA-2 i SHA-1. Architektury ARMv8 ARM i IBM s390x już (od 2018 r.) Zawierają specjalne instrukcje, które umożliwiają szybsze działanie algorytmów Keccak.

**Schemat i algorytm obliczania funkcji**

Keccak opiera się na nowatorskim podejściu zwanym konstrukcją gąbki . Konstrukcja gąbki jest oparta na szerokiej funkcji losowej lub losowej permutacji i umożliwia wprowadzanie dowolnej ilości danych i wysyłanie dowolnej ilości danych, działając jednocześnie jako funkcja pseudolosowa wszystkie poprzednie dane wejściowe. Prowadzi to do dużej elastyczności.

W fazie absorpcji bloki wiadomości są XORowane do podzbioru stanu, który jest następnie przekształcany jako całość za pomocą funkcji permutacji . W fazie „squeeze” bloki wyjściowe są odczytywane z tego samego podzbioru stanu, na przemian z funkcją transformacji stanu . Rozmiar części stanu, który jest zapisywany i odczytywany, nazywany jest „szybkością”, a rozmiar części, na którą nie ma wpływu wejście / wyjście, nazywany jest „pojemnością”. Zdolność decyduje o bezpieczeństwie programu.

Biorąc pod uwagę wejściowy ciąg bitowy , funkcję wypełniającą , funkcję permutacji, która działa na blokach bitów o szerokości , szybkości i długości wyjściowej , mamy pojemność i konstrukcję gąbki , dającą ciąg bitów długości , działa w następujący sposób:

* wypełnij wejście **N** za pomocą funkcji **pad**, uzyskując wypełniony ciąg bitów **P** o długości podzielnej przez (taką, która jest liczbą całkowitą) **r,** ***n = len(P) / r***
* podziel **P** na **n** kolejnych **r**-bitowych kawałków **P**0, ..., **P**n-1
* zainicjuj stan **S** ciągiem **b** bitów zerowych
* absorbują wejście do stanu: dla każdego bloku **P**i :
* rozszerz **P**i na końcu ciągiem **c** bitów zerowych, uzyskując jeden o długości **b**
* **XOR** to z **S**.
* zastosuj permutację bloku **f** do wyniku, uzyskując nowy stan **S**
* zainicjuj **Z** jako pusty ciąg
* podczas gdy długość **Z** jest mniejsza niż **d** :
* dołącz pierwsze r bitów od **S** do **Z**
* jeśli **Z** jest nadal krótszy niż **d** bitów, zastosuj **f** do **S** , uzyskując nowy stan **S**.
* obetnij **Z** do d bitów

Fakt, że stan wewnętrzny **S** zawiera c dodatkowych bitów informacji oprócz tego, co jest wyprowadzane do **Z**, zapobiega atakom rozszerzającym długość, na które podatne są **SHA-2, SHA-1, MD5** i inne skróty oparte na konstrukcji Merkle – Damgard.

W **SHA-3** stan **S** składa się z tablicy 5 × 5 słów **w** - bitowych (przy **w** = 64), **b** = 5 × 5 × **w** = 5 × 5 × 64 = łącznie 1600 bitów. Keccak jest również zdefiniowany dla mniejszych mocy 2 słów **w** do 1 bitu (całkowity stan 25 bitów). Małe rozmiary stanów mogą być używane do testowania ataków kryptoanalitycznych, a rozmiary stanów pośrednich (od **w** = 8 , 200 bitów, do **w** = 32 , 800 bitów) mogą być używane w praktycznych, lekkich aplikacjach.

**Wyściółka**

Aby zapewnić, że wiadomość może być równo podzielona na bloki **r**- bitowe, wymagane jest wypełnienie. **SHA-3** używa wzorca 10 \* 1 w swojej funkcji dopełniania: 1 bit, po którym następuje zero lub więcej bitów 0 (maksymalnie r - 1 ) i 1 bit końcowy.

Maksymalna liczba bitów zerowych **r** - 1 występuje, gdy ostatni blok wiadomości ma długość **r** - 1 bitów. Następnie dodawany jest kolejny blok po początkowym 1 bicie, zawierający **r** - 1 zerowych bitów przed ostatnim 1 bitem.

Dwa 1 bity zostaną dodane, nawet jeśli długość wiadomości jest już podzielna przez **r** . W tym przypadku do wiadomości dodawany jest kolejny blok zawierający 1 bit, po którym następuje blok o długości **r** - 2 bity zerowe i kolejny 1 bit. Jest to konieczne, aby wiadomość o długości podzielnej przez **r** kończącą się czymś, co wygląda jak dopełnienie, nie generowała tego samego skrótu, co wiadomość z usuniętymi tymi bitami.

Wymagany jest początkowy 1 bit, więc wiadomości różniące się tylko kilkoma dodatkowymi zerowymi bitami na końcu nie generują tego samego skrótu.

Pozycja ostatniego 1 bitu wskazuje, która szybkość r została użyta (wypełnienie wieloprocentowe), co jest wymagane, aby dowód bezpieczeństwa działał dla różnych wariantów skrótu. Bez tego różne warianty skrótu tej samej krótkiej wiadomości byłyby takie same aż do obcięcia.

**Permutacja blokowa**

Transformacja blokowa **f** , czyli **Keccak-f** [1600] dla **SHA-3**, jest permutacją wykorzystującą operacje **XOR** , **AND** i **NOT** i jest zaprojektowana do łatwej implementacji zarówno w oprogramowaniu, jak i sprzęcie.

Jest zdefiniowany dla dowolnego rozmiaru słowa o potęgi dwóch , **w** = 2ℓ bitów. Główne zgłoszenie SHA-3 używa słów 64-bitowych, ℓ = 6 .

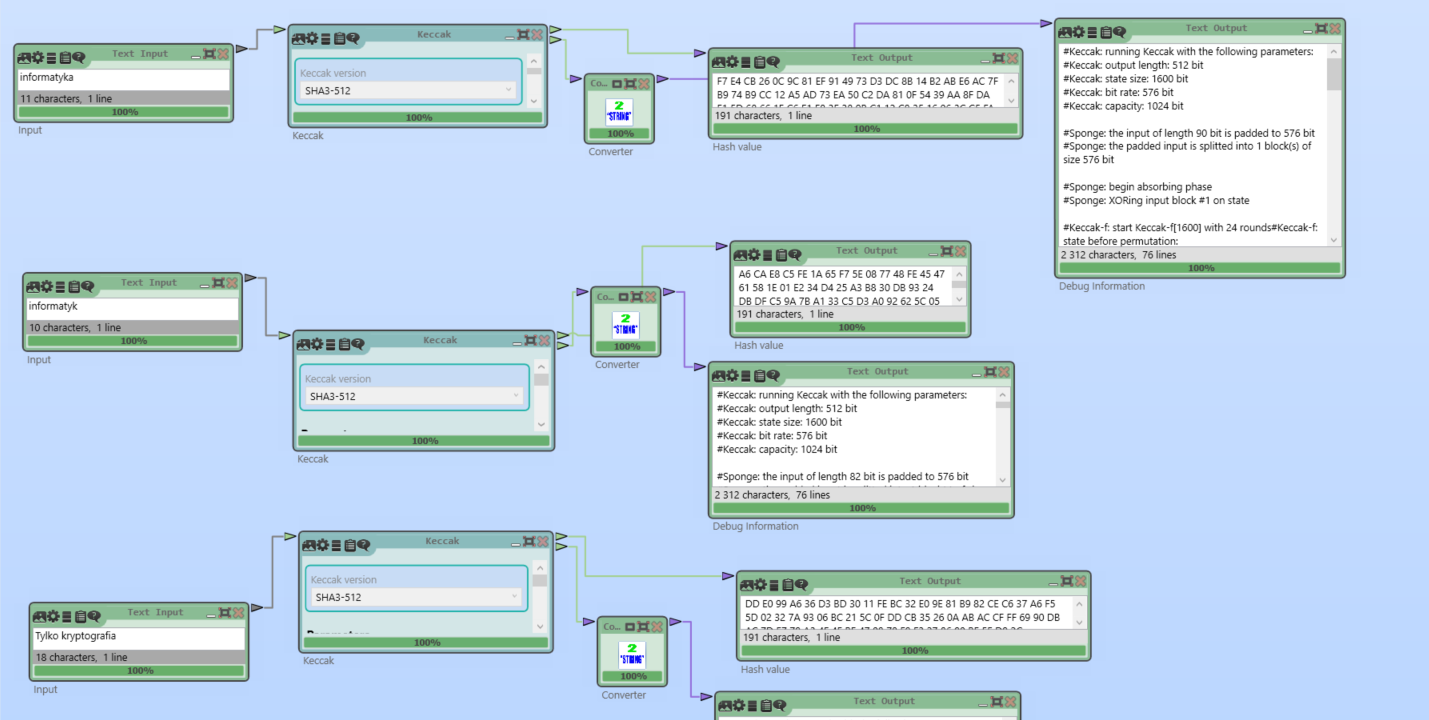
Stan można uznać za tablicę bitów 5 × 5 × **w** . Niech ***o*** **[ i ] [  j ] [ k ]** będzie kawałek (5 i + j ) x w + k danych wejściowych przy użyciu ostrokońcej konwencja numeracji bitowy i rząd major indeksowania. Czyli **ja** wybiera wiersz, **j** kolumnie, a **k** bit.

Arytmetyka indeksów jest wykonywana modulo 5 dla pierwszych dwóch wymiarów i modulo w dla trzeciego.

**Lista bibliotek kryptograficznych obsługujących SHA-3:**

* Rusta sha3
* Botan
* Dmuchany zamek
* Crypto ++
* Libgcrypt
* Pokrzywa
* OpenSSL
* LibreSSL
* wolfSSL
* MIRACL Cryptographic SDK
* Golang's x / crypto /

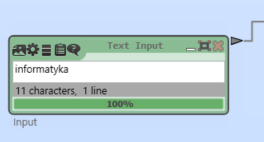
**Demonstacja dzialania algorytmu**

****

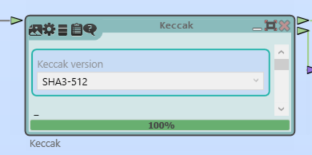
Na zdjęciu widać trzy różnych teksty do szyfrowania: ‘informatyka’, ‘informatyk’ i ‘tylko kryptografia’. Na danych przykładach użyłem instancję SHA3-512 z rozmiarem wyjściowym **d** **512**, stawką **r 576,** pojemnością **c 1024** Definicja:Keccak [1024] ( M || 01, 512).

Zauważmy, że ‘informatyk’ i ‘Informatyka’ są całkowicie różne, chociaż i mają różnicę w jeden symbol w formacie tekstu.

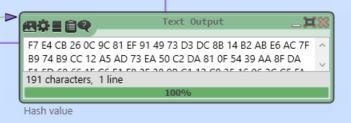
Na wejściu mamy tekst do szyfrowania



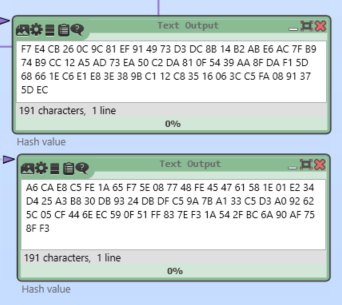
Podłączamy go do algorytmu Keccak z definicją Keccak [1024] ( M || 01, 512)



I otrzymujemy na wyjściu taki szyfrogram:



Porównanie dwóch podobnych tekstów na wejściu: (górny – informatyka, dolny – informatyk)



Jak widać są one kompletnie różne, przez to, że etap szyfrowania nazywany Padding (**Wyściółka**), która była opisana na początku dokumentu.