WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

Wydział Cybernetyki

PRACA DYPLOMOWA

Studia stacjonarne II°

Analiza możliwości sprzętowej implementacji szyfru blokowego opartego o algorytm Anubis w sposób uodparniający na ataki kanałem bocznym

Autor
sierż. pchor. Damian Krata

Promotor
kpt. dr inż. Michał Wroński

Warszawa 2019

OŚWIADCZENIE

„Wyrażam zgodę na udostępnianie mojej pracy przez Archiwum WAT”

Dnia ....
(podpis)

Pracę przyjąłem

promotor pracy
kpt. dr inż. Michał Wroński

Spis treści

Wstęp

Przedmiotem niniejszej pracy magisterskiej jest analiza możliwości sprzętowej implementacji szyfru blokowego opartego o algorytm Anubis w sposób uodparniający na ataki z kanałem bocznym.

W dzisiejszych czasach, zapewnienie podstawowych atrybutów bezpieczeństwa informacji, takich jak poufność, integralność czy dostępność, stała się zjawiskiem priorytetowym. Podstawową formą ochrony przed nieuprawnionym dostępem do danych, jest ich szyfrowanie. Wraz z upływem lat i znaczącym wzrostem efektywności i wydajności maszyn liczących, zaczęto jednak łamać dostępne szyfry. Najlepszym przykładem może być algorytm DES (ang. *Data Encryption Standard*). Przyjęty jako standard, po kilkudziesięciu latach okazał się być możliwy do złamiania, przez co musiał ustąpić nowemu standardowi AES (ang. *Advanced Encryption Standard*).

W związku z faktem, że najlepszy znany do tej pory atak z użyciem kluczy zależnych wymaga mocy obliczeniowej rzędu 2^{119}, uważa się, że algorytm AES zapewni bezpieczeństwo szyfrowanych danych jeszcze przez bardzo długi czas. Niemniej jednak pomimo dobrych właściwości matematycznych oraz dużej odporności na znane ataki, atakujący prześcigają się w sposobach konstrukcji ataków, które w przeciwieństwie do wspomnianych, nie wykorzystywałyby właściwości samego algorytmu, ale atakowałyby urządzenie na którym wykonuje się proces szyfrowania. Rozważania dotyczące czasu wykonania się jednostkowej operacji w algorytmie oraz poboru prądu danego urządzenia w zależności od przetwarzanych danych, doprowadziły do powstania nowego rodzaju ataków - ataków z kanałem bocznym. Ciągły rozwój ataków kanałem bocznym, od połowy lat 90-tych XX wieku aż do dnia dzisiejszego, doprowadził do sytuacji, w której rola twórcy modułu szyfrującego nie ogranicza się tylko do wyboru najlepszego

algorytmu szyfrowania, ale wymaga spojrzenia na problem zapewnienia bezpieczeństwa informacji z szerszej perspektywy, analizując takie aspekty jak dostęp adwersarza do urządzenia bądź realizacja potoku obliczeń i wynikające z tego podatności.

W niniejszej pracy postanowiono:

- zbadać możliwość implementacji szyfru blokowego Anubis (należącego do klasy szyfrów posiadających wspólne cechy z algorytmem AES) uodpornionego na ataki z kanałem bocznym,
- wskazać główne aspekty wymagające ochrony (pobór mocy itp.),
- wyszczególnić problemy algorytmu Anubis uniemożliwiające pełną ochronę,
- zmodyfikować algorytm w celu zaprezentowania dodatkowych środków udaremniających ataki z kanałem bocznym,
- przeprowadzić analizę wydajności i bezpieczeństwa stworzonych implementacji.

W związku z tym przyjęto następujący układ pracy:

1. **Rozdział 1** - stanowi wprowadzenie do tematyki szyfrów blokowych. Przedstawia główne sposoby ich tworzenia oraz opisuje algorytm Anubis oraz AES.

2. **Rozdział 2** - traktuje o charakterystyce ataków z kanałem bocznym. Wprowadza zasadniczy podział ataków w zależności od wymaganej ingerencji adwersarza w urządzenie oraz metody oddziaływania na urządzenie. Ponadto prezentuje ogólny sposób wykonania DPA oraz SPA. W rozdziale tym omówiono również metody przeciwdziałania (ang. *countermeasures*) atakom typu side-channel.

3. **Rozdział 3** - składa się z implementacji referencyjnej algorytmu Anubis. Ponadto opisuje przyjęte przez autora sposoby analizy wykonanych implementacji pod względem odporności na ataki oraz proponuje implementacje wstępnie zabezpieczone. Na zakończenie rozdziału ujawniana jest wada w konstrukcji algorytmu, nie pozwalająca na wykonanie dodatkowych zabezpieczeń.

4. **Rozdział 4** - wprowadza analizę warstwy nieliniowej algorytmu oraz sugeruje jej zmianę. Następnie dla zmodyfikowanego algorytmu, wykonywane są implementacje wykorzystujące kolejne metody ochrony przez atakami z kanałem bocznym. Na zakończenie, przedstawiona jest analiza porównawcza wykonanych implementacji.

Rozdział 1

Algorytmy blokowe

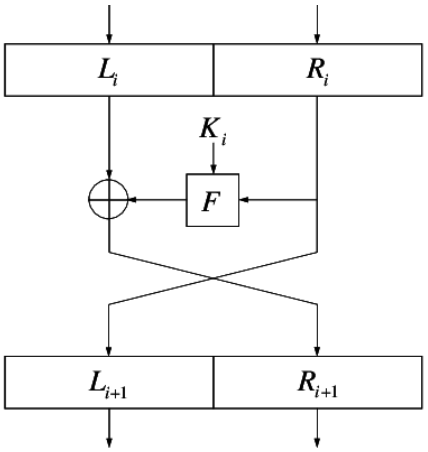
1.1 Szyfry blokowe

Szyfr blokowy jest funkcją, która odwzorowuje *n-bitowy* blok tekstu jawnego na *n-bitowy* blok szyfrogramu (n nazywamy długością bloku). Może on być rozpatrywany jako prosty szyfr podstawieniowy o dużym rozmiarze znaku. Funkcję parametryzuje k-bitowy klucz K przyjmujący wartości z podzbioru κ (przestrzeń kluczy) zbiory wszystkich k-bitowych wektorów V_k. Zazwyczaj przyjmuje się, że klucz jest wybierany losowo. Stosowanie bloków tekstu jawnego i szyfrogramu o tej samej długości umożliwia uniknięcie rozrostu danych. Aby można było jednoznacznie odszyfrować dane, funkcja szyfrująca musi być typu jeden-do-jednego (tj. odwracalna). Dla n-bitowych bloków tekstu jawnego i szyfrogramu oraz ustalonego klucza funkcja szyfrująca jest bijekcją, definiującą permutację n-bitowych wektorów. Każdy klucz potencjalnie określa inną bijekcję. Liczba kluczy wynosi $|\kappa|$, a efektywna długość klucza $lg|\kappa|$. Odpowiada to długości klucza, jeśli wszystkie k-bitowe wektory są prawidłowymi kluczami ($\kappa = V_k$). Jeśli klucze są jednakowo prawdopodobne i każdy określa inną bijekcję, entropia przestrzeni kluczy również wynosi $lg|\kappa|$.

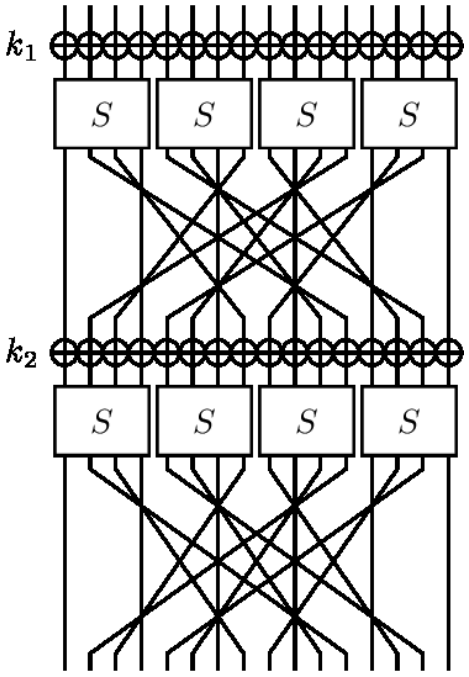
W symetrycznych algorytmach kryptograficznych do szyfrowania i deszyfrowania wykorzystywany jest ten sam klucz K. Konieczność ochrony tego klucza powoduje, ze algorytmy symetryczne określa się także jako algorytmy z *tajnym* kluczem. Współcześnie istnieją dwie główne metody konstruowania szyfrów symetrycznych,

które ze względu na rodzaj operacji, które są wykonywane w trakcie deszyfrowania i szyfrowania można podzielić na:

- algorytmy zbudowane na podstawie sieci Feistela (Lucifer, DES)
- algorytmy zbudowane na podstawie sieci podstawieniowo - przestawieniowej (ang. *substitution - permutation network* np. AES)



(a) Sieć Feistela [19].



(b) Sieć SP [18].

Rysunek 1: Porównanie dwóch sposobów budowy szyfrów blokowych.

Ważnym elementem każdego szyfru symetrycznego jest sposób rozszerzania klucza głównego K (ang. *key schedule*). Pozwala on na zwiększenie długości klucza kryptograficznego zastosowanego w algorytmie, aby każda runda posiadała swój własny klucz zwany także kluczem rundy K_r (ang. *round key*). W większości systemów kryptograficznych, schemat generacji kluczy rund wykorzystuje takie same transformacje jak procedury szyfrowania/deszyfrowania [1].

1.2 Anubis

Anubis jest 128 bitowym, iteracyjnym szyfrem blokowym z kluczami różnej długości. Pomimo tego, że nie jest on zbudowany w oparciu o sieć Feistela, jego struktura pozwala na wykonanie deszyfrowania, bez konieczności zmiany logiki modułów wykonujących przekształcenia. Jedyną różnicą pomiędzy szyfrowaniem a deszyfrowaniem, jest zmiana w kolejności wyboru klucza rundy uzgodnionego za pomocą schematu rozszerzania klucza. Powyższą właściwość osiągnięto dzięki zastosowaniu tzw. inwolucji, czyli operacji samo-odwrotnych. Zastosowanie tych operacji pozwala na zmniejszenie zajętości układów realizujących szyfrowanie i deszyfrowanie.

Szyfr Anubis został zaprojektowany zgodnie ze Strategią Szerokiej Ścieżki (ang. ***Wide Trail Strategy***). W strategii tej, runda algorytmu złożona jest z różnych odwracalnych transformacji, które posiadają swoje funkcjonalności i mają określone wymagania. Liniowe warstwy dyfuzji zapewniają, że już po kilku rundach, bity wyjściowe zależą od wszystkich bitów wejściowych. Warstwy nieliniowe z kolei zapewniają powyższej zależności złożoną naturę, uniemożliwiającą wyprowadzenie zależności, rozwiązywalnych przy pomocy dostępnych narzędzi matematycznych. Operacja dodania klucza rundy pozwala na wprowadzenie dodatkowego materiału oraz uzależnia potok obliczeń od konkretnych danych wejściowych.

Anubis jest inwolucyjnym szyfrem blokowym działającym na 128 bitowej macierzy stanu. Wykorzystuje on klucz o długości $32N$-bitów, gdzie $(4 \leqslant N \leqslant 10)$ i składa się z kilku przekształceń macierzy stanu zależnych od klucza rundy.

Operacje wykonywane są w ciele $\mathrm{GF}(2^8)$ reprezentowanym jako $\mathrm{GF}(2)[x]/p(x)$, gdzie $p(x) = x^8 + x^4 + x^3 + x^2 + 1$ jest wielomianem nierozkładalnym. Wielomian ten został tak dobrany, aby $g(x) = x$ było generatorem $\mathrm{GF}(2^8) \setminus \{0\}$.

Element $u = u_7x^7 + u_6x^6 + u_5x^5 + u_4x^4 + u_3x^3 + u_2x^2 + u_1x + u_0$ ciała $\mathrm{GF}(2^8)$ gdzie $u_i \in \mathrm{GF}(2)$ dla każdego $i = 0, \ldots, 7$, będzie oznaczony przez wartość numeryczną $u_7 \cdot 2^7 + u_6 \cdot 2^6 + u_5 \cdot 2^5 + u_4 \cdot 2^4 + u_3 \cdot 2^3 + u_2 \cdot 2^2 + u_1 \cdot 2 + u_0$, zapisaną w postaci hexadecymalnej. Dla przykładu $u = x^4 + x + 1$ odpowiada 13_{hex}. Wielomianowi redukującemu $p(x)$ odpowiada $11d_{hex}$

1.2.1 Dane wejściowe i wyjściowe

Stan szyfru jest definiowany za pomocą macierzy $\mathcal{M}_{4\times4}[\mathrm{GF}(2^8)]$, podczas gdy klucz za pomocą macierzy $\mathcal{M}_{N\times4}[\mathrm{GF}(2^8)]$. W związku z tym, 128-bitowe bloki danych oraz $32N$-bitowe bloki klucza są mapowane w formie macierzy. Działanie to wyraża się poprzez funkcję $\mu : \mathrm{GF}(2^8)^{4N} \to \mathcal{M}_{N\times4}[\mathrm{GF}(2^8)]$ i jej odwrotność:

$$\mu(a) = b \Leftrightarrow b_{ij} = a_{4i+j},\ 0 \leqslant i \leqslant N-1,\ 0 \leqslant j \leqslant 3.$$

1.2.2 Warstwa nieliniowa γ

Funkcja $\gamma : \mathcal{M}_{N\times4}[\mathrm{GF}(2^8)] \to \mathcal{M}_{N\times4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, składa się z równolegle działających skrzynek podstawieniowych $S : \mathrm{GF}(2^8) \to \mathrm{GF}(2^8)$, $x \mapsto S[x]$, które realizują podstawienie każego bajtu z osobna:

$$\gamma(a) = b \Leftrightarrow b_{ij} = S[a_{ij}],\ 0 \leqslant i \leqslant N-1,\ 0 \leqslant j \leqslant 3.$$

Sbox został wybrany pseudo - losowo tak, aby zapewnić warunek: $S[S[x]] = x$ dla każdego $x \in \mathrm{GF}(2^8)$.

1.2.3 Transpozycja τ

Mapowanie $\tau : \mathcal{M}_{4\times4}[\mathrm{GF}(2^8)] \to \mathcal{M}_{4\times4}[\mathrm{GF}(2^8)]$ transponuje elementy macierzy stanu:

$$\tau(a) = b \Leftrightarrow b = a^t \Leftrightarrow b_{ij} = a_{ji},\ 0 \leqslant i, j \leqslant 3.$$

Transpozycja jest inwolucją.

1.2.4 Warstwa dyfuzji θ

W algorytmie Anubis, rozproszenie danych realizowane jest przy pomocy przekształcenia $\theta : \mathcal{M}_{N\times4}[\mathrm{GF}(2^8)] \to \mathcal{M}_{N\times4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$. Jest to liniowe przypisanie wartości odpowiadających działaniu:

$$\theta(a) = b \Leftrightarrow b = a \cdot H,$$

gdzie $H = \mathrm{had}(\text{'01'}, \text{'02'}, \text{'04'}, \text{'06'})$, tzn.

$$H = \begin{bmatrix} \text{'01'} & \text{'02'} & \text{'04'} & \text{'06'} \\ \text{'02'} & \text{'01'} & \text{'06'} & \text{'04'} \\ \text{'04'} & \text{'06'} & \text{'01'} & \text{'02'} \\ \text{'06'} & \text{'04'} & \text{'02'} & \text{'01'} \end{bmatrix}$$

1.2.5 Dodanie klucza $\sigma[k]$

Liniowe dodanie klucza rundy $\sigma[k] : \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)] \to \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$ sprowadza się do wykonania operacji $\oplus$ na aktualnie przetwarzanej macierzy stanu i macierzy klucza $k \in \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$:

$$\sigma[k](a) = b \Leftrightarrow b_{ij} = a_{ij} \oplus k_{ij},\ 0 \leqslant i \leqslant N-1,\ 0 \leqslant j \leqslant 3.$$

1.2.6 Cykliczna permutacja π

Permutacja $\pi : \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)] \to \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, cyklicznie przesuwa każdą kolumnę oddzielnie do dołu, w ten sposób, że kolumna j jest przesuwana o j pozycji:

$$\pi(a) = b \Leftrightarrow b_{ij} = a_{(i-j) \bmod N, j},\ 0 \leqslant i \leqslant N-1,\ 0 \leqslant j \leqslant 3.$$

1.2.7 Ekstrakcja klucza ω

Przekształcenie ekstrakcji klucza jest funkcją $\omega : \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)] \to \mathcal{M}_{4\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, sprowadza się do wykonania

$$\omega(a) = b \Leftrightarrow b = V \cdot a,$$

gdzie $V = \mathrm{vdm}_N(\text{'01'}, \text{'02'}, \text{'06'}, \text{'08'})$, tzn.

$$V = \begin{bmatrix} \text{'01'} & \text{'01'} & \text{'01'} & \dots & \text{'01'} \\ \text{'01'} & \text{'02'} & \text{'02'}^2 & \dots & \text{'02'}^{N-1} \\ \text{'01'} & \text{'06'} & \text{'06'}^2 & \dots & \text{'06'}^{N-1} \\ \text{'01'} & \text{'08'} & \text{'08'}^2 & \dots & \text{'08'}^{N-1} \end{bmatrix},$$

1.2.8 Stałe rundy c^r

r-ta stała rundy ($r > 0$) jest macierzą $c^r \in \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, zdefiniowaną jako:

$$\begin{aligned} c^r_{0j} &= S[4(r-1)+j], \quad 0 \leqslant j \leqslant 3, \\ c^r_{ij} &= 0, \qquad\qquad\quad 1 \leqslant i < N, 0 \leqslant j \leqslant 3. \end{aligned}$$

1.2.9 Schemat klucza

Schemat klucza (ang. *key schedule*) rozszerza klucz $K \in \mathrm{GF}(2^8)^{4N}$, $4 \leqslant N \leqslant 10$, na klucze rund $K^0, \dots, K^R$, gdzie $K^r \in \mathcal{M}_{4\times 4}[\mathrm{GF}(2^8)]$:

$$\begin{aligned} \kappa^0 &= \mu(K), \\ \kappa^r &= (\sigma[c^r] \circ \theta \circ \pi \circ \gamma)(\kappa^{r-1}), \quad r > 0, \\ K^r &= (\tau \circ \omega \circ \gamma)(\kappa^r), \quad 0 \leqslant r \leqslant R; \end{aligned}$$

Przekształcenie $\psi[c^r] \equiv \sigma[c^r] \circ \theta \circ \pi \circ \gamma$ nazywane jest funkcją rozwoju r-tego klucza rundy podczas gdy $\phi \equiv \tau \circ \omega \circ \gamma$ nazywane jest funkcją wyboru klucza. W ogólności w fazie obliczeń, wyznaczanie klucza rundy sprowadza się do wykonania funkcji ϕ na kluczu poddawanym obliczeniom za pomocą funkcji ψ.

1.2.10 Matematyczny opis algorytmu

Dla klucza $K \in \mathrm{GF}(2^8)^{4N}$, Anubis może być zdefiniowany jako przekształcenie $\textsc{Anubis}[K] : \mathrm{GF}(2^8)^{16} \to \mathrm{GF}(2^8)^{16}$ zadane przez

$$\textsc{Anubis}[K] \equiv \mu^{-1} \circ \alpha_R[K^0, \dots, K^R] \circ \mu,$$

gdzie

$$\alpha_R[K^0, \ldots, K^R] = \sigma[K^R] \circ \tau \circ \gamma \circ \left(\bigcirc_{1}^{r=R-1} \sigma[K^r] \circ \theta \circ \tau \circ \gamma \right) \circ \sigma[K^0].$$

Standardowa liczba rund dla algorytmu R określona jest jako $R = 8 + N$ dla $32N$-bitowego klucza, $4 \leqslant N \leqslant 10$. Przekształcenie $\rho[K^r] \equiv \sigma[K^r] \circ \theta \circ \tau \circ \gamma$ nazywane jest *funkcją rundy*, natomiast $\rho'[K^R] \equiv \sigma[K^r] \circ \tau \circ \gamma$ pozbawione θ określane jest jako *funkcja ostatniej rundy.*

1.2.11 K - bezpieczeństwo

Definicja 1 ([7])**.** *Szyfr blokowy jest K - bezpieczny, jeśli wszystkie możliwe strategie ataku na niego, mają ten sam oczekiwany współczynnik wymagań pamięciowych i obliczeniowych, jak większość szyfrów blokowych z takim samym rozmiarem klucza.Powyższe odnosi się do wszystkich znanych ataków (ze znanym tekstem jawnym, szyfrogramem, atakiem adaptacyjnym itp.).*

K - bezpieczeństwo jest bardzo silnym pojęciem. System kryptograficzny nie może zostać uznany za K - bezpieczny jeśli wystąpi co najmniej jeden z następujących przypadków:

- istnieje atak szybszy niż brutalne przeszukanie,
- występuje właściwość symetrii w przekształceniach,
- istnieje klasa kluczy słabych, których istnienie nie jest pomijalne,
- możliwy jest atak z kluczami zależnymi [12].

1.3 Advanced Encryption Standard - AES

AES - zaawansowany standard szyfrowania, powstał jako nowy standard szyfrowania przyjęty przez NIST w 2001 roku. Miał za zadanie zastąpić algorytm DES (ang. *Data Encryption Standard*). Oparty został na finaliście konkursu na AES - algorytmie Rijndael. AES jest szyfrem blokowym. W przeciwieństwie do poprzedniego standardu, DES'a działającego na zasadzie sieci Feistel'a, zbudowany jest w oparciu o sieć permutacyjno - podstawieniową (ang. *SPN Network*). Na rundę algorytmu AES składają się cztery przekształcenia, które polegają na wykonywaniu działań realizowanych w ciele $GF(2^8)$:

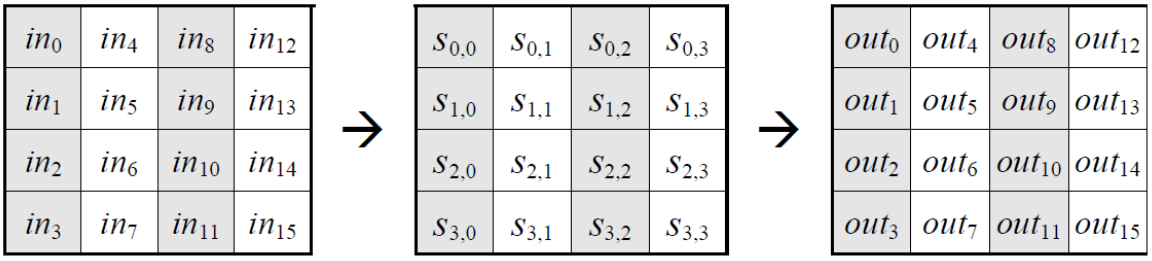
1. ***AddRoundKey*** - przekształcenie polegające na dodaniu klucza rundy w każdej iteracji algorytmu,
2. ***MixColumns*** - przekształcenie kolumn macierzy stanu,
3. ***ShiftRows*** - przesunięcie cykliczne trzech ostatnich wierszy w macierzy stanu o określoną liczbę,
4. ***SubBytes*** - zamiana bajtów - operacja nieliniowa realizowana za pomocą skrzynki podstawieniowej.

Algorytm wykonuje szyfrowanie bloków danych wielkości 128 - bitów za pomocą klucza K długości 128, 192 lub 256 bitów. W zależności od długości klucza, wersje algorytmu AES różnią się między sobą liczbą wykonywanych rund.

- 10 rund dla klucza 128 - bitowego,
- 12 rund dla klucza 192 - bitowego,
- 14 rund dla klucza 256 - bitowego,

W każdej rundzie, poza ostatnią, wykonywane są wszystkie przekształcenia przedstawione powyżej. Dla ostatniej rundy, schemat nie realizuje operacji MixColumns.

Blok danych reprezentowany jest w algorytmie w postaci macierzy, składającej się z szesnastu 8-bitowych elementów, nazywanej *Stanem S*.



Rysunek 2: Macierz stanu [4].

Wykonywanie dodawania $a(x) \oplus b(x)$ na dwóch elementach ciała $GF(2^8)$ sprowadza się do wykonywania operacji XOR dla odpowiadających współczynników $a(x)$ i $b(x)$. Sytuacja komplikuje się w przypadku obliczania iloczynu dwóch elementów. Wielomianem nierozkładalnym użytym w algorytmie AES jest:

$$p(x) = x^8 + x^4 + x^3 + x + 1$$

Zatem wszystkie operacje realizowane są modulo ten wielomian.

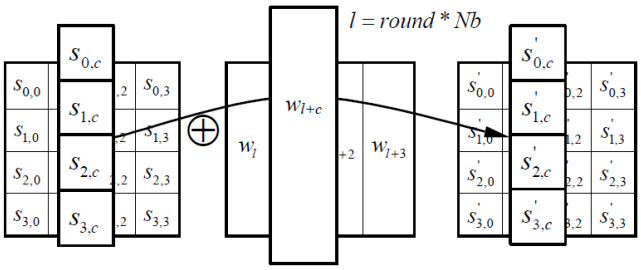
W implementacjach, najczęściej wykorzystuje się fakt, że pomnożenie elementu w ciele $GF(2^8)$ sprowadza się do jego bitowego przesunięcia w lewo i wykonaniu operacji modulo z $11b_{hex}$ w przypadku, gdy najstarszy bit (skrajny lewy) był równy zero.

1.3.1 AddRoundKey

Przekształcenie AddRoundKey pozwala na wprowadzenie do algorytmu klucza rundy. Polega ona na wykonaniu dodawania modularnego. Klucz rundy powstaje z klucza głównego za pomocą algorytmu rozszerzania klucza i przedstawiany jest w postaci macierzy 4×4. Wynikowa macierz stanu ma postać:

$$S' = [s'_{i,j}]_{4\times4} \text{ gdzie } s'_{i,j} = s_{i,j} \oplus k_{i,j}$$

Transformacja AddRoundKey przebiega w ten sam sposób w czasie szyfrowania i deszyfrowania.

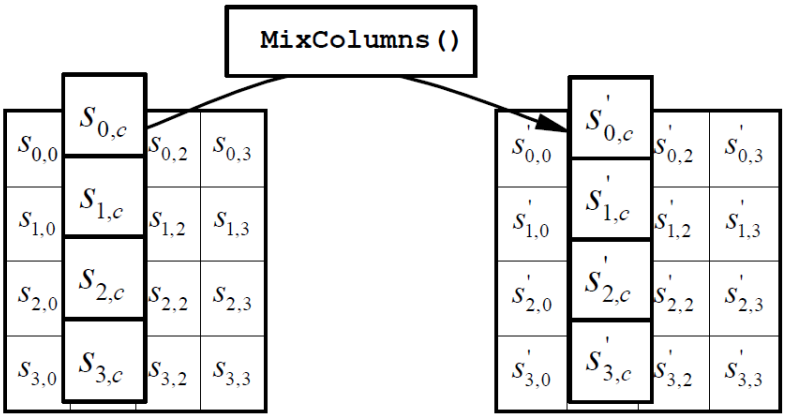


Rysunek 3: Przekształcenie AddRoundKey [4].

1.3.2 MixColumns

W przekształceniu MixColumns, cztery bajty każdej kolumny stanu są łączone za pomocą odwracalnej transformacji liniowej. Bajty te są następnie poddawane obliczeniom, które można zapisać w postaci macierzowej następująco:

$$S' = [W'_j]_{1\times 4} \text{ gdzie } W'_j = \begin{bmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{3,j} \\ s_{2,j} \\ s_{1,j} \\ s_{0,j} \end{bmatrix}$$

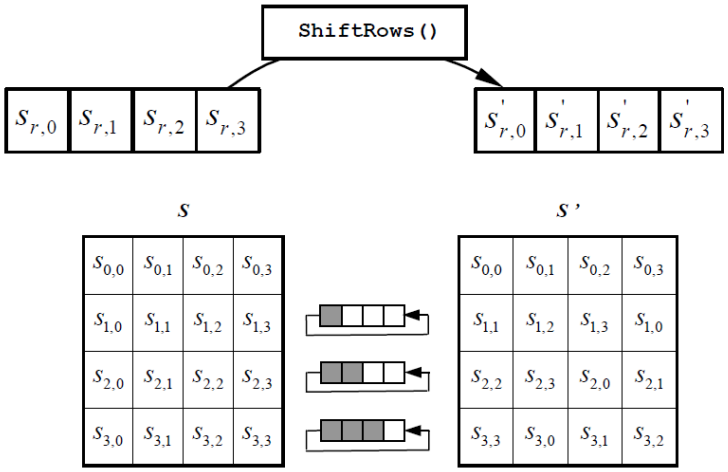


Rysunek 4: Przekształcenie MixColumns [4].

1.3.3 ShiftRows

Przekształcenie ShiftRows powoduje cykliczne przesunięcie wierszy macierzy stanu S w lewo o różną liczbę pozycji bajtowych. Dla AES rotacje są następujące:

- 1 wiersz macierzy - nie jest przesuwany,
- 2 wiersz macierzy - przesunięcie o 1 pozycję,
- 3 wiersz macierzy - przesunięcie o 2 pozycje,
- 4 wiersz macierzy - przesunięcie o 3 pozycje.



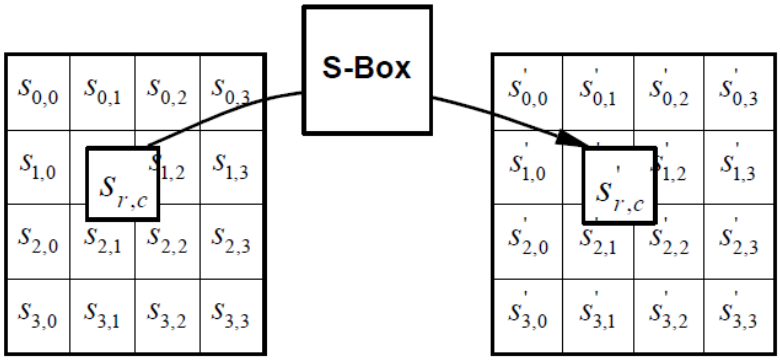
Rysunek 5: Przekształcenie ShiftRows [4].

Do deszyfracji używana jest podobna transformacja, z tą różnicą, że wiersze przesuwane są w prawo.

1.3.4 SubBytes

W przekształceniu SubBytes, każdy bajt macierzy stanu, jest zastępowany za pomocą 8-bitowej skrzynki podstawieniowej. Operacja ta jest nieliniowa, co znacząco poprawia właściwości algorytmu. Wykorzystany w algorytmie Sbox nie jest zwyczajną tablicą podstawień utworzoną empirycznie i przeanalizowaną. Wartości znajdujące się

w niej mają swoje pochodzenie w działaniu obliczania odwrotności multiplikatywnej w ciele $GF(2^8)$. W celu uniknięcia ataków opierających się na właściwościach algebraicznych, postanowiono dołożyć do obliczania odwrotności przekształcenie afiniczne. Operacja SubBytes nie jest operacją samoodwrotną, co oznacza, że w celu deszyfracji należy zamienić kolejność wykonywania się poszczególnych jej składowych.



Rysunek 6: Przekształcenie SubBytes [4].

Przekształcenie afiniczne można sprowadzić do wykonania działań:

$$s'_{i,j} = (F1 \otimes s_{i,j}^{-1} \oplus 63) mod 101 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} s_{i,j}^{-1(7)} \\ s_{i,j}^{-1(7)} \\ s_{i,j}^{-1(7)} \\ s_{i,j}^{-1(7)} \\ s_{i,j}^{-1(7)} \\ s_{i,j}^{-1(7)} \\ s_{i,j}^{-1(7)} \\ s_{i,j}^{-1(7)} \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Ze względu na dużą złożoność obliczeniową algorytmu wyznaczania odwrotności multiplikatywnej, zazwyczaj tablicuje się wartości. Niestety to rozwiązanie zwiększa wymagania pamięciowe [4].

1.4 Porównanie Anubis i AES

	AES	Anubis
Rozmiar bloku	128	128
Rozmiar klucza	128, 192, 256	128, 160, 192, 224, 256, 288, 320
Liczba rund	10, 12, 14	12, 13, 14, 15, 16, 17, 18
Schemat tworzenia kluczy	algorytm dedykowany *a priori*	funkcje rozwijania i wyboru klucza
Wielomian redukujący $GF(2^8)$	$x^8 + x^4 + x^3 + x + 1$ $(0x11B)$	$x^8 + x^4 + x^3 + x^2 + 1$ $(0x11D)$
Pochodzenie Sbox'a	odwrotność w ciele $GF(2^8)$ i przekształcenie afiniczne	losowo wybrana inwolucja
Pochodzenie stałych rundy	wielomiany x^i nad $GF(2^8)$	kolejne wejścia do Sbox'a

Tablica 1: Porównanie dwóch szyfrów blokowych. Na podstawie [20].

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0

Tablica 2: Budowa sbox w algorytmie AES w implementacji wykorzystującej *look-up tables*. Na podstawie [4].

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	a7	d3	e6	71	d0	ac	4d	79	3a	c9	91	fc	1e	47	54	bd
10	8c	a5	7a	fb	63	b8	dd	d4	e5	b3	c5	be	a9	88	0c	a2

Tablica 3: Budowa sbox w algorytmie Anubis w implementacji wykorzystującej *look-up tables*. Na podstawie [12].

Rozdział 2

Opis, analiza oraz sposoby przeciwdziałania atakom typu side-channel

2.1 Wprowadzenie

Zastosowanie narzędzi kryptograficznych ma na celu zapewnienie konkretnych właściwości danych. Właściwości te nazywają się atrybutami bezpieczeństwa informacji. Najbardziej istotne z nich, to:

- **poufność** - zapewnienie, że informacja nie jest udostępniana bądź ujawniana podmiotom nieuprawnionym,
- **integralność** - oznaczającą precyzyjność, dokładność oraz kompletność informacji,
- **autentyczność** - zapewnienie, ze tożsamość podmiotu bądź zasobu jest zgodna z deklarowaną,
- **niezaprzeczalność** - brak możliwości zanegowania swojego uczestnictwa w całości lub części wymiany danych przez jeden z podmiotów uczestniczący w tej wymianie.

W ogólności, trwa ciągły wyścig pomiędzy twórcami szyfrów a osobami próbującymi dane szyfry złamać. Historia dokładnie pokazuje, że wraz ze wzrostem mocy obliczeniowej oraz szerszym dostępem do ogólnie pojętych zasobów, przeprowadzanie konkretnych ataków staje się możliwe. Najlepszym przykładem może być standard używany do 1997 roku - DES. Zbyt krótka długość klucza danego algorytmu, była powodem, dla którego możliwe było przeprowadzenie ataku brutalnego. Od tego momentu, głównym założeniem konstrukcji algorytmów blokowych było zwiększenie długości klucza. Po drodze pojawiały się również nowe ataki. Warto w tym miejscu wspomnieć kryptoanalizę różnicową, której powstanie szacuje się na lata 90-te XX wieku. Przyczyniła się ona w znacznym stopniu do wypracowania przez twórców tzw. strategii szerokiej ścieżki (ang. *Wide Trail Strategy*), która w pewnym sensie normowała i wskazywała zalecenia do konstrukcji szyfrów blokowych odpornych na ataki kryptoanalizy liniowej i różnicowej. Aparat matematyczny użyty do konstruowania szyfrów, pozwala na oszacowanie liczby operacji potrzebnych do jego złamania. Dla współczesnych algorytmów, wymagana moc obliczeniowa jest tak duża, że nawet przyjmując prawdziwość hipotezy wypracowanej na podstawie prawa Moore'a, mówiącej że moc obliczeniowa komputerów podwaja się co 24 miesiące, nie bylibyśmy w stanie złamać algorytmu AES jeszcze przez długi czas. Powyższe stwierdzenia doprowadziły do sytuacji, w której atakujący postanowili nie atakować szyfru, traktując go jako rozwiązanie pewnego matematycznego problemu, ale analizować oddziaływanie urządzenia, wykonującego prymityw kryptograficzny, z otoczeniem. Rozważania atakujących na temat takiego oddziaływania i konsekwencje z nich płynące, dały podstawy pod ataki nowego typu - ataki typu ***side-channel***.

Prace nad atakami z kanałem bocznym rozpoczęły się już w połowie lat 90-tych XX wieku. W latach 1996-1997 prace Bihama i Shamira pokazały, że doprowadzenie do błędów w czasie wykonywania obliczeń, może prowadzić do ciekawych obserwacji. Metody prowadzenia analizy ze względu na sposób oddziaływania na urządzenie możemy podzielić na:

- pomiar charakterystyk pracy urządzenia podczas realizacji potoku obliczeniowego:

    1. zmierzenie czasu realizacji operacji kryptograficznych,

2. zmierzenie poboru mocy urządzenia,
3. zmierzenie oddziaływania elektromagnetycznego emitowanego przez urządzenie,

- oddziaływanie na urządzenie podczas wykonywania potoku obliczeń poprzez:
  1. zmiany prądu zasilania,
  2. rozstrajanie zegara taktującego urządzenie,
  3. naświetlanie układu promieniowaniem rentgenowskim lub promieniowaniem podczerwonym,
  4. oddziaływanie promieniowaniem elektromagnetycznym innego urządzenia.

Ze względu na sposób ingerencji w urządzenie wyróżniamy:

1. ataki inwazyjne (ang. *invasive attacks*), w których adwersarz ma możliwość uzyskania bezpośredniego dostępu do elementów układu znajdujących się w jego wnętrzu,
2. ataki nieinwazyjne (ang. *non-invasive attacks*) - atakujący nie ma dostępu do wewnętrznych elementów układu, może jedynie wykonywać pomiary działania, obserwacje danych wejściowych i wyjściowych układu oraz oddziaływać na urządzenie jedynie w sposób zdalny (poprzez np. zakłócanie napięcia zasilania),
3. ataki półinwazyjne (ang. *semi-invasive attacks*), w których adwersarz nie posiada bezpośredniego dostępu, niemniej jednak jest w stanie pozbawić układ wszystkich warstw ochronnych, uzyskując dostęp do układów mikroprocesorowych.

Ze względu na metodę oddziaływania na urządzenie rozróżniamy:

1. ataki pasywne (ang. *passive attacks*), w których atakujący prowadzi jedynie obserwacje działanie układu, nie ingerując w jego elementy wewnętrzne, zasoby oraz komponenty,
2. ataki aktywne (ang. *active attacks*), w których atakujący może wykonywać dodatkowe czynności (np. zakłócać napięcie zasilania, przykładać pole elektromagnetyczne) i analizować reakcje układu na te czynności [9].

2.2 Ataki pasywne

Do najczęściej wykorzystywanych metod ataku można zaliczyć pomiar poboru mocy, pomiar czasu działania oraz pomiar ulotu elektromagnetycznego. Za pomocą każdego z wyszczególnionych pomiarów, atakujący jest w stanie poznać charakterystykę działania urządzenia, co z kolei przekłada się na powodzenie ataku. W atakach pasywnych, atakujący jedynie analizuje otrzymane wyniki. Może zmieniać dane wejściowe i manipulować nimi, niemniej jednak nie jest w stanie wykonać żadnych technicznych czynności na urządzeniu.

2.2.1 Analiza poboru mocy

Ataki z pomiarem poboru mocy wykorzystują fakt, że w czasie swojego działania, urządzenia elektryczne czy elektroniczne pobierają określoną ilość mocy. W zależności od fazy wykonywania się obliczeń w urządzeniu, bądź różnic występujących w przetwarzanych danych, można zbudować pewne charakterystyki, których analiza pozwala na określenie z pewną dokładnością takich informacji jak:

- wartość przetwarzanych danych,
- liczba cykli obliczeń,
- liczba użytych układów,
- określenie, który moduł układu jest w danym momencie czasu aktywny.

W układach scalonych itp. w ogólności wykorzystuje się bardzo popularną technologię CMOS. Jej głównymi zaletami jest zredukowany pobór mocy, który praktycznie nie występuje w czasie, gdy układ pozostaje w stanie statycznym, natomiast jest używany tylko w momencie przełączania się tranzystorów. Ponadto, przy zmianie stanu tranzystorów zauważono, że ilość pobranej mocy w celu zmiany wartości układu z 0 na 1 jest wyższa niż zmiana z 1 na 0.

Powyższe obserwacje zostały użyte w celu rozwinięcia dwóch komplementarnych sposobów analizy poboru mocy:

1. bezpośredniej analizy poboru mocy SPA (ang. *Simple Power Analysis*)

2. różnicowej analizy poboru mocy DPA (ang. *Differential Power Analysis*).

W związku z tym, że przeprowadzenie badań zależności poboru mocy urządzenia w zależności od jego stanu jest zadaniem trudnym, rozważania przedstawione w dalszej części pracy będą miały dodatkowe założenia.

- atakujący, w każdej chwili może manipulować tekstami jawnymi użytymi do szyfrowania przy pomocy tajnego klucza, który chce odzyskać,

- atakujący jest w stanie wykonywać bardzo wiele powtórzeń szyfrowania dla każdego użytego tekstu jawnego,

- podczas analizy, na urządzeniu działa tylko algorytm docelowy, żadne inne algorytmy nie są dopuszczalne.

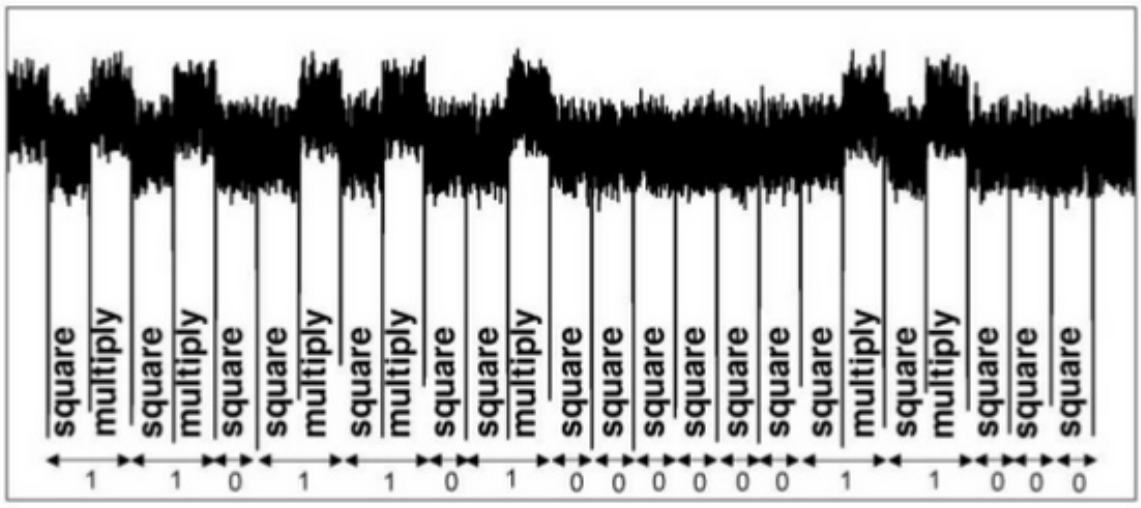
Założenia te wynikają bezpośrednio z nieograniczonego dostępu atakującego do urządzenia szyfrującego.

W celu przeprowadzenia analizy danych otrzymanych po wykonaniu obserwacji, należy przyjąć referencyjny model sposobu interpretowania wyników. W przypadku SPA i DPA, najbardziej popularnym modelem poboru mocy jest waga Hamminga (ang. *Hamming weight*) oraz odległość Hamminga (ang. *Hamming distance*). Waga Hamminga, to zmienna określająca liczbę jedynek w układzie, natomiast odległość Hamminga to miara zliczająca liczbę przejść ze stanu niskiego do wysokiego jak również w drugą stronę. W przypadku tak uproszczonego modelu poboru mocy, przyjęto dodatkowe założenie, że w ustalonym interwale czasowym, przejście z 0 na 1, pobiera tyle samo energii co przejście z 1 na 0. Ponadto założono, że brak zmiany wartości rejestru/tranzystora nie pobiera mocy. Model HD najczęściej wykorzystywany jest w celu oceny dwóch pomiarów następujących po sobie z określonym interwałem czasowym. Zależnośc pomiędzy wagą Hamminga a odległością Hamminga dla zestawu danych V_1 oraz V_2 można określić następująco:

$$HD(V_1, V_2) = HW(V_1 \oplus V_2)$$

Bezpośrednia analiza poboru mocy

Paul Kocher w swoim artykule *Differential Power Analysis* zaproponował wstępną wersję ataku SPA (ang. Simple Power Analysis). Wersja ta zakładała wykonanie bardo dużej liczby pomiarów poboru mocy oraz uzależnienie zmian poboru od operacji realizowanych przez układ. Warto w tym miejscu wspomnieć, że zanim zaczęto analizować algorytmy blokowe, sztandarowym przykładem wykorzystania SPA była próba złamania algorytmu RSA. Dobrze widoczne fazy potęgowania i mnożenia pozwalają na skompromitowanie klucza bez konieczności faktoryzacji dużych liczb.



Rysunek 7: SPA dla RSA [17].

Na rysunku nr 7 widzimy, że fazy podnoszenia do kwadratu i mnożenia są bardzo dobrze widoczne. Na podstawie analizy wykresu, atakujący jest w stanie podać bity wykładnika.

Atak Bezpośredniej analizy poboru mocy jest skuteczny w przypadku algorytmów, w których wykonanie się dalszego potoku obliczeń zależy od przetwarzanych danych i może być różne dla różnych danych wejściowych. W szczególności operacjami wrażliwymi są:

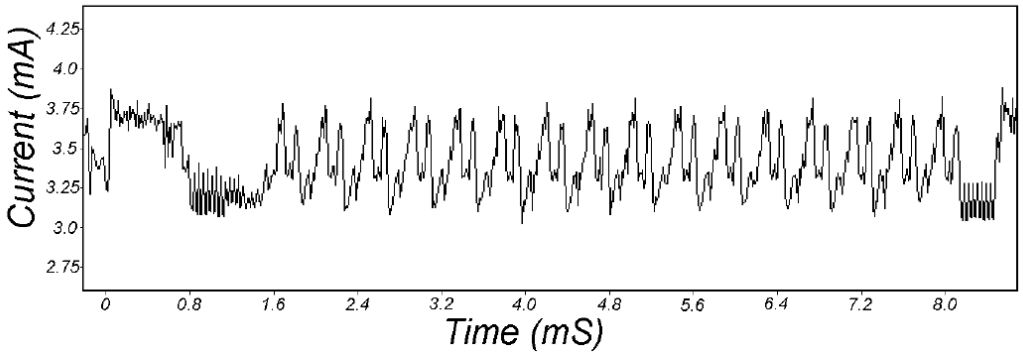
1. przesunięcia (rotacje) i permutacje,

2. instrukcje warunkowe, których argument wpływa na wybór ścieżki obliczeń. W przypadku przedstawionym powyżej, dla pozycji wykładnika równej 1, realizowane było mnożenie, natomiast dla 0 podnoszenie do kwadratu,

3. mnożenia, dla których pobór prądu zależy od specyficznych właściwości operandów,

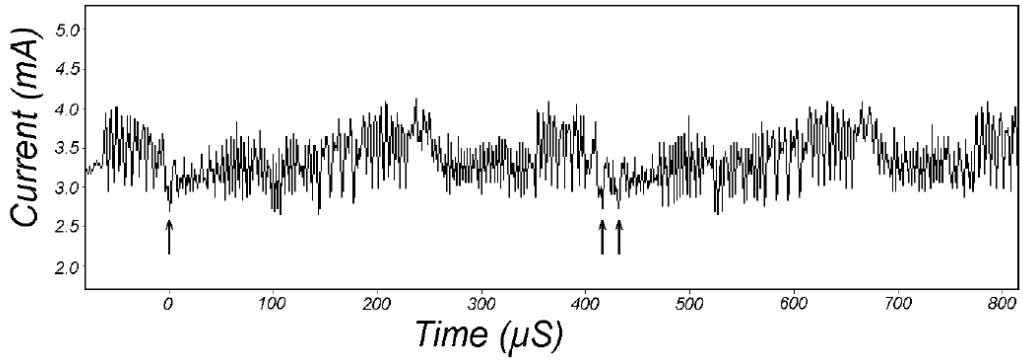
4. potęgowania wykonywane przy pomocy algorytmów *square and multiply*, np. dla RSA.

Niestety otrzymane z analizy algorytmów wyniki nie zawsze dostarczają tylu informacji, ile przykład zaprezentowany dla rysunku nr 7. Im bardziej skomplikowany układ oraz wykonywanych jest więcej operacji, tym trudniej prawidłowo ocenić, które fragmenty wykresu odnoszą się do konkretnych obliczeń bądź danych.



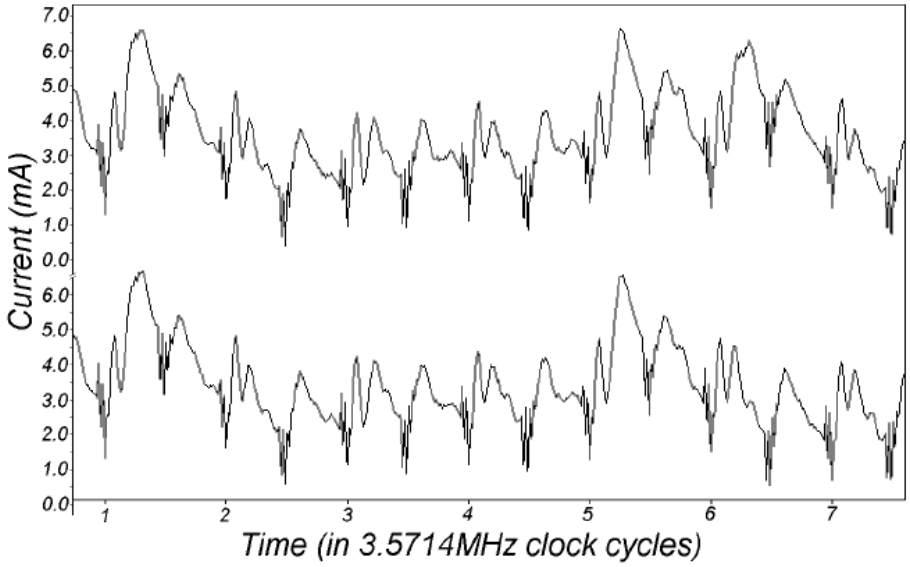
Rysunek 8: Ślad SPA obrazujący wykonanie się algorytmu DES [11].

Na powyższym rysunku przedstawiona została zależność poboru mocy od czasu. Dla próbkowania z częstotliwością 5MHz i czasu 1 milisekundy otrzymujemy 5000 punktów wykresu. Jest to wynik pozwalający na zauważenie wykonania się 16 rund algorytmu DES.



Rysunek 9: Ślad SPA obrazujący wykonanie się dwóch rund algorytmu DES [11].

Rysunek nr 9 obrazuje wykonanie się dwóch rund algorytmu. Zauważmy, że dla miejsc zaznaczonych strzałkami, widać w jednym przypadku pojedyncze, natomiast w drugim dwukrotne wykonanie się pewnej operacji. Analiza budowy algorytmu DES pozwala na stwierdzenie, że jest to przesunięcie 28-bitowego rejestru klucza o jedną lub dwie pozycje. Informacja ta pozwala na zorientowanie się atakującego, w której fazie algorytmu się znajduje.

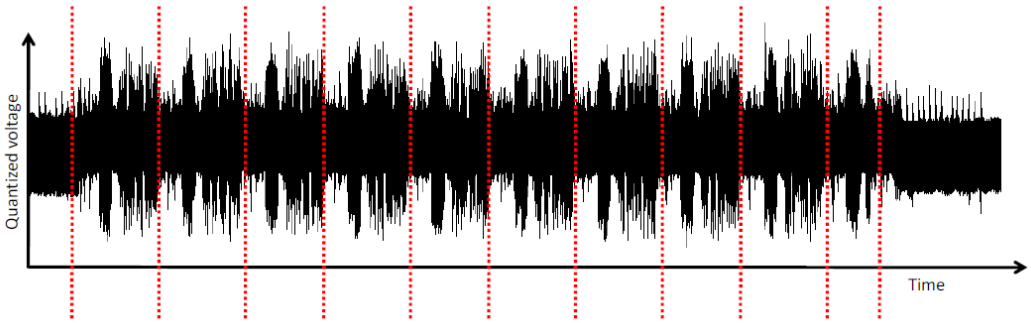


Rysunek 10: Ślad SPA obrazujący pojedyncze cykle zegara algorytmu DES [11].

Zastosowanie pętli warunkowych wiąże się z ryzykiem kompromitacji danych. Na rysunku nr 10 przedstawiono dwa przebiegi pewnej fazy algorytmu DES z

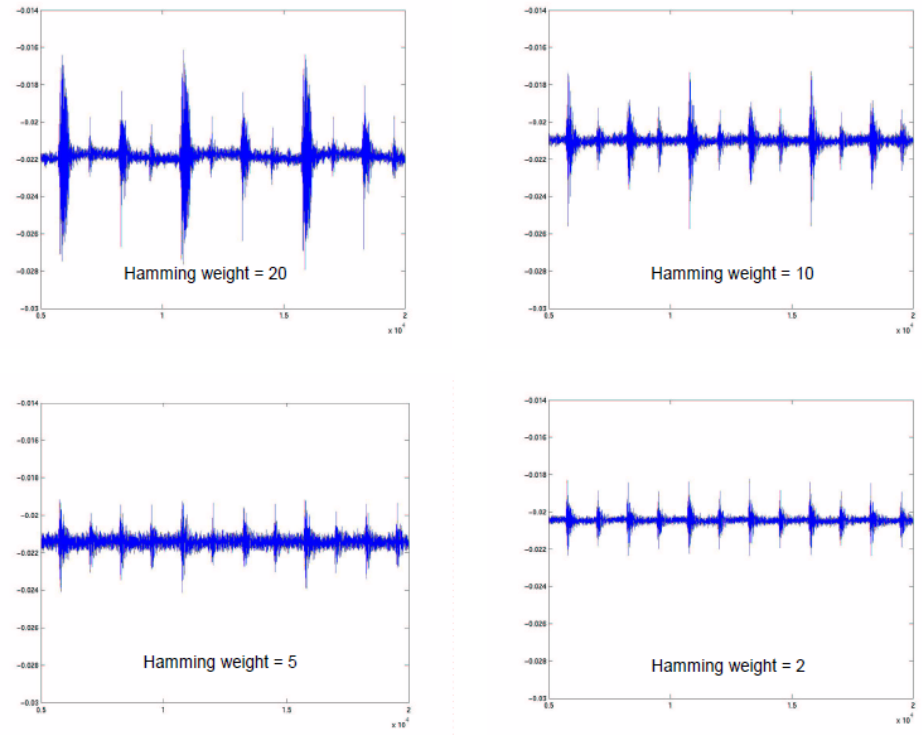
dokładnością do pojedynczego cyklu zegara. Zauważmy, że dla argumentu równego 6, występuje zmiana w przebiegu wykresu. Spowodowane jest to wystąpieniem warunku sprawdzającego przesuwany bit w schemacie rozszerzenia klucza. Dokładna analiza algorytmu, oraz wykonanie dokładnych pomiarów, pozwala na otrzymanie danych ułatwiających kryptoanalizę.

Pomimo tego, że w przedstawionych powyżej przykładach, nie byliśmy w stanie stwierdzić dokładnie jakie dane są używane w układzie kryptograficznym, otrzymaliśmy bardzo wiele informacji. Na wykresach bardzo dokładnie można zobaczyć liczbę rund algorytmu, czy wręcz oszacować długość klucza. Przykładem potwierdzającym, że dzieje się tak nie tylko dla algorytmu DES, jest rysunek nr 11, na którym widoczne jest wykonanie się 10 rund algorytmu AES (ostatnia runda jest krótsza ponieważ nie jest realizowana MixColumns).



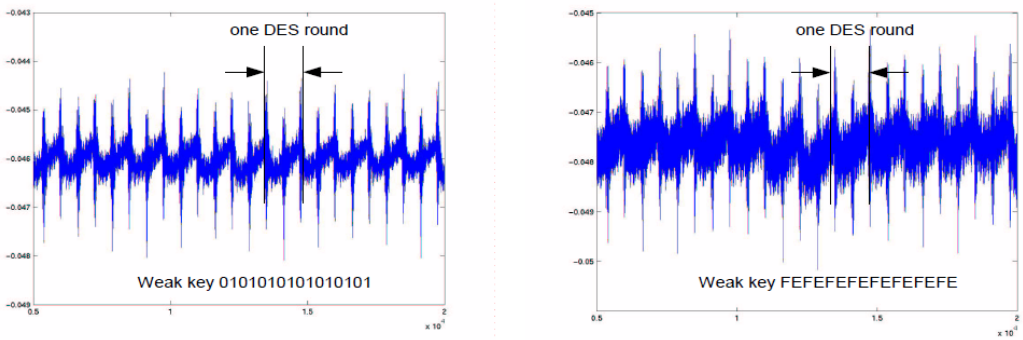
Rysunek 11: Ślad SPA obrazujący wykonanie się AES [3].

Ataki wykorzystujące SPA mogą być także wykonywane na strukturach programowalnych - FPGA (ang. *Field Programmable Gate Array*). W artykule [5], autorzy, na podstawie przeprowadzonych testów, doszli do wniosku, że waga Hamminga ciągu bitów wpisanych do przerzutników układu programowalnego wpływa znacząco na pobór prądu zasilania.



Rysunek 12: Pobór prądu w zależności od wagi Hamminga bitów [5].

Powyższy rysunek wskazuje wykresy, które uzyskali dla wag Hamminga równych odpowiednio 20, 10, 5, 2.



Rysunek 13: Zależność poboru mocy od liczby jedynek przetwarzanych w rejestrach [5].

Przedstawione na rysunkach nr 12 i 13 zależności są użyteczne do dalszych rozważań na temat ataków z kanałem bocznym. Pozwalają one na tworzenie kolejnych

ataków, które są aplikowane do różnych struktur, algorytmów i systemów. Kolejnym przykładem ataku, opartego na zależności poboru mocy od liczby przetwarzanych jedynek w układzie może być atak zaproponowany w [16], który opiera się na wyznaczaniu kolejnych bitów nieznanego ciągu bitowego na podstawie wartości wagi Hamminga różnicy symetrycznej (XOR) tego nieznanego ciągu i odpowiednio dobieranych masek.

Dodatkowym warunkiem, jaki należy spełnić podczas przeprowadzania tego ataku, jest uzyskanie dostępu do układu FPGA w ten sposób, że możliwe jest odczytanie liczby zapisanych jedynek w przerzutnikach struktury programowalnej w każdej chwili czasu działania urządzenia.

Pierwszym etapem ataku jest określenie liczby bitów o wartości '1' w ciągu. W przypadku wybrania maski zerowej, waga Hamminga tego ciągu odpowiada liczbie jedynej w nim się znajdujących. W kolejnych krokach maski dobiera się w ten sposób aby '1' znajdowała się na dokładnie jednej pozycji. Na podstawie uzyskanych wartości wag Hamminga dla kolejnych realizacji można wywnioskować, że:

- zwiększenie się wagi Hamminga zamaskowanego ciągu oznacza, że bit na maskowanej pozycji miał wartość '0',
- zmniejszenie się wagi Hamminga zamaskowanego ciągu oznacza, że bit '1' z maski musiał wyzerować '1' z nieznanego ciągu, co oznacza, że maskowana pozycja miała wartość '1',
- nie może wystąpić sytuacja w której waga Hamminga maskowanego ciągu się nie zmienia.

Przykład: Niech dany będzie nieznany ciąg k mający 4 bity i równy $k = 0110$. Początkowo wyznaczana jest waga Hamminga dla maski zerowej $hwt(k \oplus 0000) = 2$. Następnie:

- $hwt(k \oplus 1000) = 3 > hwt(k \oplus 0000)$, czyli bit o numerze 3 ma wartość $k(3) = 0$,
- $hwt(k \oplus 0100) = 1 < hwt(k \oplus 0000)$, czyli bit o numerze 2 ma wartość $k(3) = 1$,
- $hwt(k \oplus 0010) = 1 < hwt(k \oplus 0000)$, czyli bit o numerze 1 ma wartość $k(3) = 1$,

- $hwt(k \oplus 0001) = 3 > hwt(k \oplus 0000)$, czyli bit o numerze 0 ma wartość $k(3) = 0$.[1]

Dla przedstawionego ataku, wagę Hamminga ciągu można określić na podstawie poboru mocy [16]. Dany atak można zastosować do kart mikroprocesorowych, na których wykonywane są obliczenia.

Różnicowa analiza poboru mocy

Wraz ze wzrostem świadomości nie tylko twórców oprogramowania, ale także zwykłych użytkowników, zaczęto stosować podstawowe środki ochrony przed bezpośrednią analizą poboru mocy. Okazało się, że ujednolicenie działań w algorytmie, bądź dodawanie operacji nie mających żadnego znaczenia, zaczęło wytwarzać szumy, które były znaczącym utrudnieniem dla atakujących.

Postanowiono więc opracować metodę, która będzie bardziej uniwersalna. Tak powstała różnicowa analiza poboru mocy - DPA (ang. *Differential Power Analysis*). Metoda ta opierała się na kilku zasadach wywodzących się z metodyki dziel i zwyciężaj:

- szyfr blokowy jako całość, jest silny. Składa się on natomiast z wykonania wielu kroków, których bezpieczeństwo można podważyć,
- wyniki pośrednie w trakcie działania algorytmu zależą od niewielkiej liczby bitów.

W literaturze [11] zaprezentowano atak DPA na algorytm DES. W każdej rundzie tego algorytmu, następuje podstawienie za pomocą warstwy nieliniowej złożonej z 8 Sbox'ów. Każdy z nich przyjmuje 6 bitów, natomiast zwraca 4, które są permutowane i dodawane modulo 2 do bloku L, przez co zamieniają sią w blok R. Atak zakłada wykorzystanie funkcji wyboru DPA (ang. *DPA selection function*) $D(C, b, K_s)$, która posiada następujące składowe:

1. szyfrogram C,
2. indeks szukanego bitu danych b,
3. możliwa wartość 6 bitów klucza rundy K_r, odpowiadających tablicy Sbox, której wyjście zostało dodane do bitu L_{16}^b.

[1] Przyjęto numerację bitów w n-bitowym ciągu postaci $b_{n-1}, ..., b_1, b_0$

Funkcja zwraca dwie wartości:

- rzeczywistą wartość szukanego bitu, gdy klucz K_r jest poprawny,
- 0 w przeciwnym przypadku.

Zauważmy, że prawdopodobieństwo zwrócenia rzeczywistej wartości bitu jest równe $\frac{1}{2}$, gdy klucz K_r został źle znaleziony. Wykonanie się ataku, polega na wykonaniu m operacji szyfrowania oraz zapisaniu śladów poboru mocy w postaci macierzy $T_{1..m}[1..k]$, złożonej z k próbek dla każdego m. Analiza DPA pozwala na stwierdzenie, czy przy otrzymanej charakterystyce poboru mocy, odgadywany blok klucza jest prawidłowy czy nie. Dla każdego bitu, atakujący znajduje charakterystykę różnicową wszystkich charakterystyk T_i:

$$\Delta_D[j] = \frac{\sum_{i=1}^{m} D(C_i, b, K_s)T_i[j]}{\sum_{i=1}^{m} D(C_i, b, K_s)} - \frac{\sum_{i=1}^{m}(1 - D(C_i, b, K_s))T_i[j]}{\sum_{i=1}^{m}(1 - D(C_i, b, K_s))}, \tag{2.1}$$

co po oszacowaniu daje wartość:

$$\Delta_D[j] \approx 2(\frac{\sum_{i=1}^{m} D(C_i, b, K_s)T_i[j]}{\sum_{i=1}^{m} D(C_i, b, K_s)} - \frac{\sum_{i=1}^{m} T_i[j]}{m}). \tag{2.2}$$

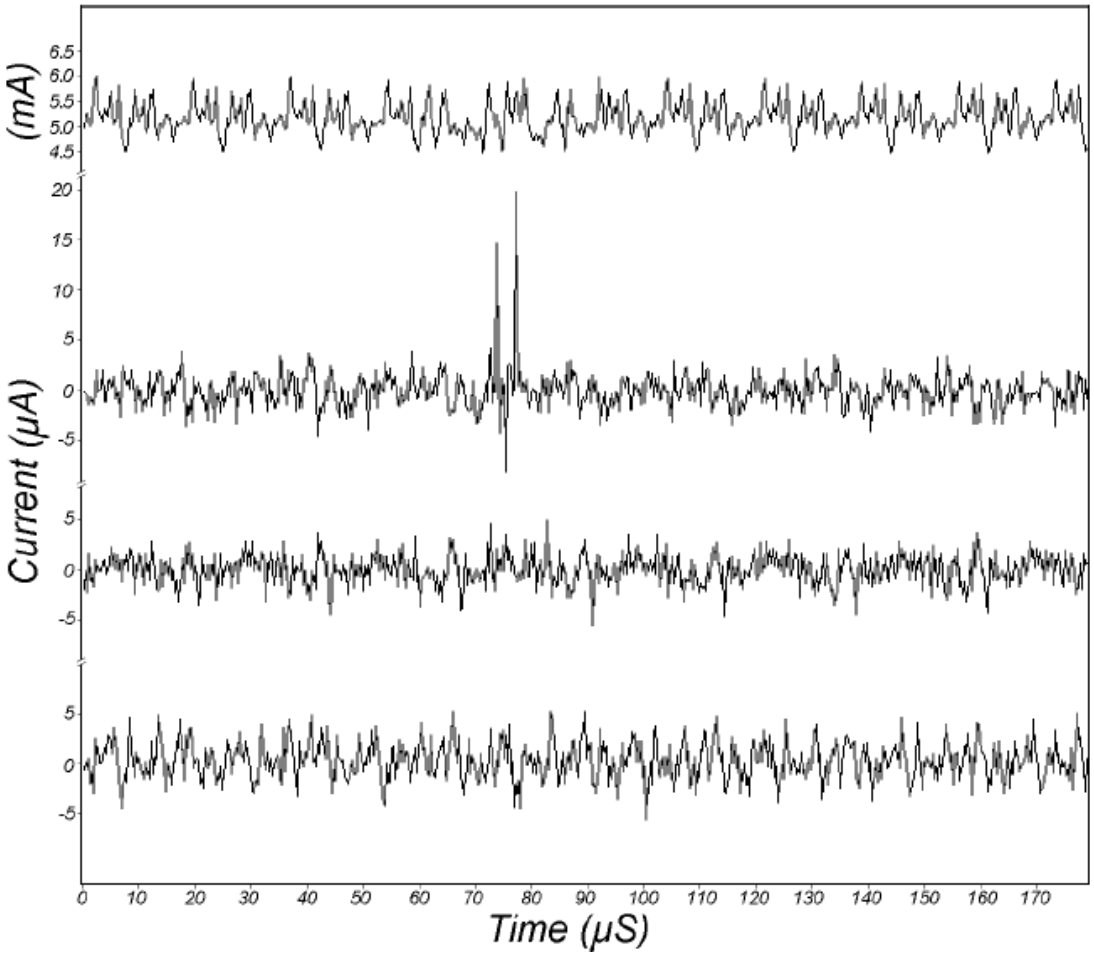
Definicja 2. *Jeżeli podział zbioru odbywa się na podstawie funkcji losowej, to różnica średnich jego podzbiorów dąży do zera.*

Dla tak zbudowanych charakterystyk, funkcja wyboru $D(C_i, b, K_s)$ jest funkcją losową dla niepoprawnego klucza.

Z powyższego i definicji 2 wynika, że jeśli K_s jest błędny, to:

$$\lim_{m\to\infty} \Delta_D[j] \approx 0. \tag{2.3}$$

Dla poprawnego klucza K_s, wartości funkcji wyboru DPA mają poprawną wartość, co oznacza, że charakterystyka przy rosnącej liczbie pomiarów m będzie się stabilizować.



Rysunek 14: Ślad DPA obrazujący wykonanie się trzech przebiegów [11].

Na rysunku nr 14 przedstawiono 4 wykresy. Pierwszy od góry, wskazuje pobór mocy w zależności od czasu, dla tekstu jawnego, zaszyfrowanego danym kluczem K. Poniższe trzy wykresy obrazują różnicę w poborze mocy pomiędzy wartością referencyjną (pierwszy wykres) a wartością otrzymaną dla szyfrowania innym kluczem. Zauważmy, że dla drugiego wykresu od góry, zauważono, pomimo występowania pików, mniejsze odchylenia od wartości średniej, niż dla pozostałych. Oznacza to, że dwa dolne wykresy reprezentują dane zaszyfrowane złymi kluczami, podczas gdy drugi od góry, poprawnym.

W pracy [3] zaproponowano również atak DPA na algorytm AES, przebiegał on w kilku krokach.
Krok 1 - analiza:

- kryptoanalitycy - wyszukują miejsc w algorytmie, w których przetwarzane dane są najbardziej narażone na atak wyczerpujący (np. wejścia do skrzynek podstawieniowych, dla których pełny przegląd sprowadza się do przeszukania małej przestrzeni danych),
- inżynierowie/statystycy - zapewniają dostęp do urządzenia i opracowują narzędzia statystyczne do analizy danych uzyskanych za pomocą specjalistycznych urządzeń.

Krok 2 - przeprowadzenie ataku:

- zebranie śladów/próbek dla różnych danych wejściowych,
- wybranie wrażliwych danych pośrednich określonych w kroku pierwszym,
- dla każdej hipotezy klucza:
    1. obliczenie hipotetycznych wartości pośrednich oraz posortowanie wyników w podzbiory,
    2. obliczenie różnic pomiędzy zbiorami.
- podjęcie decyzji:
    1. zły wybór klucza, wynikający z losowości podzbiorów,
    2. prawidłowy wybór klucza, wynikający z występujących widocznych różnic pomiędzy zbiorami.

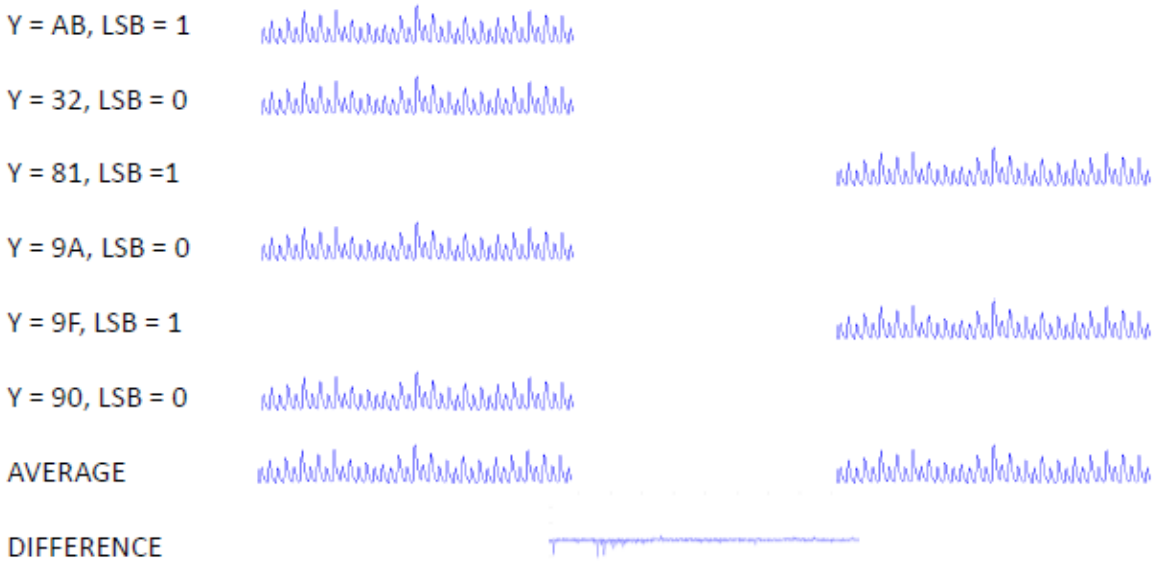
Przykład klasycznego, 1-bitowego ataku DPA na 128-bitowy algorytm AES:

1. Wybranie funkcji wrażliwej na atak:
    - każdy bajt stanu zależy od bajtu tekstu jawnego i bajtu klucza do operacji MixColumns,

- określenie celu - skrzynki podstawieniowe - analiza bajt po bajcie,
- określenie najbardziej wrażliwej zmiennej - $LSB(Y)$ (najmniej znaczący bit wyjścia),

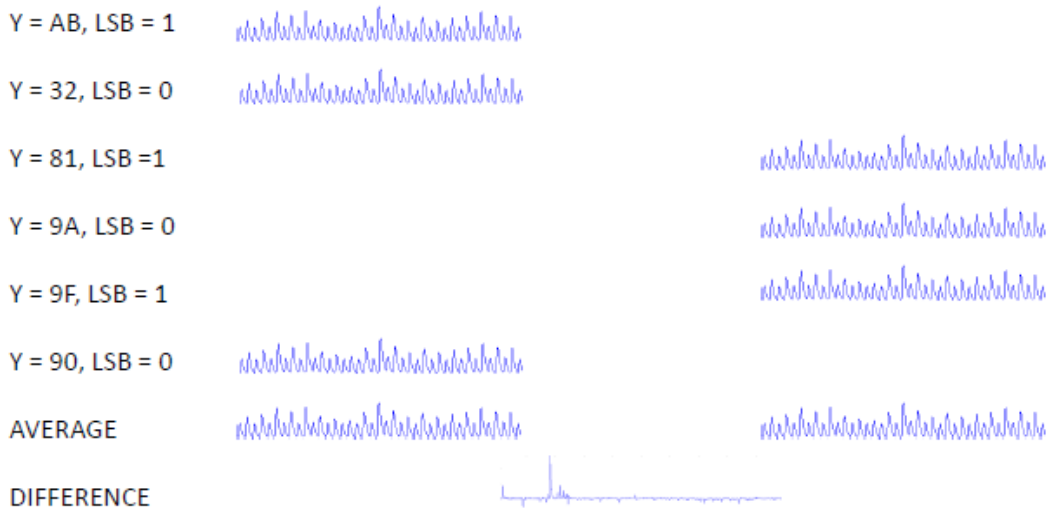
2. Dla każdej możliwej wartości klucza $K \in 0...255$:
   - obliczenie wyjścia i sprawdzenie czy $LSB(Y)$ jest równy 0 czy 1,
   - pogrupowanie otrzymanych krzywych na dwa zbiory,
   - obliczenie średnich dla każdego zbioru i ich różnic,
3. Przeanalizowanie krzywych różnicowych:
   - dla poprawnej wartości K, krzywe różnicowe osiągają piki dla chwil czasu, w których wykonywane są operacje na danym bicie.

Atak na pierwszy bajt klucza:
Przypadek dla $K = 0x00$:

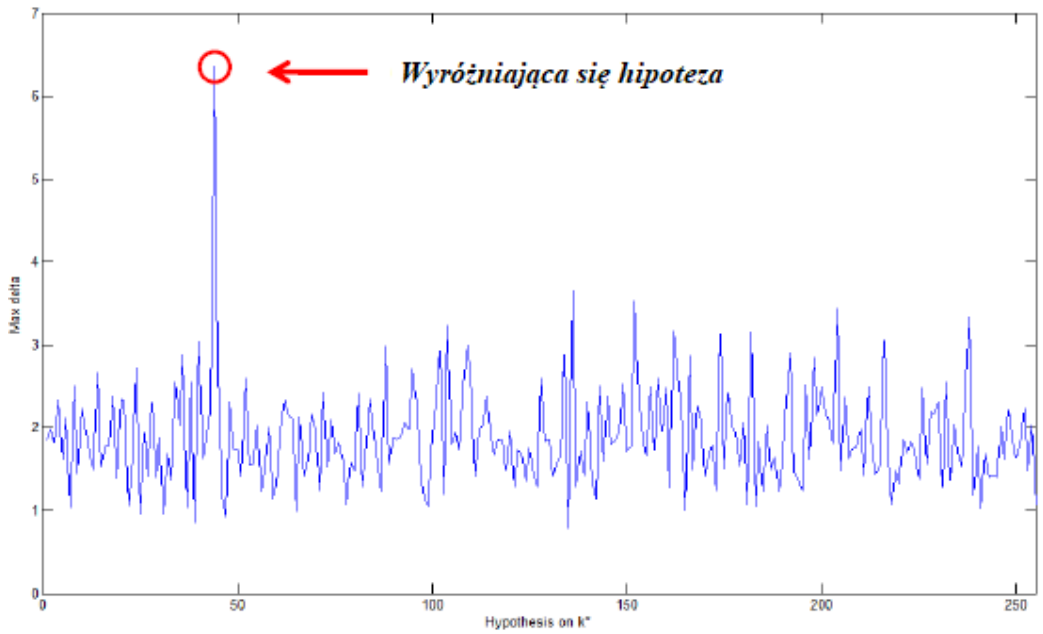


Rysunek 15: Ślad DPA dla klucza równego $0x00$ i kolejnych przebiegów [3].

Przypadek dla $K = 0x2B$:



Rysunek 16: Ślad DPA dla klucza równego $0x2B$ i kolejnych przebiegów [3].



Rysunek 17: Zestawienie średnich dla wszystkich kluczy [3].

Po przeanalizowany wszystkich możliwych kluczy, na rysunku nr 17 możemy zauważyć, że dla jednej hipotezy, krzywa osiąga swoje maksimum. Oznacza to,

w związku z poprzednimi obserwacjami, że charakterystyka różnicowa wszystkich charakterystyk, nie dążyła do zera, co oznacza, że testowany klucz (argument, dla którego wykres osiągnął pik), jest kluczem prawidłowym.

2.2.2 Ataki z pomiarem czasu działania

Ataki z pomiarem czasu działania (ang. *timing attacks*), wykorzystują różnice czasu wykonania się poszczególnych operacji. Różnice te mogą wynikać nie tylko z budowy algorytmu (rodzaj wykonywanych działań, np. mnożenie lub podnoszenie do kwadratu) ale także z właściwości użytych urządzeń. Czas dostępu do pamięci cache, różnice w przepustowości magistral, to wszystko prowadzi do słabości układu. Atakujący, aby przeprowadzić atak z pomiarem czasu działania musi zatem zebrać bardzo wiele informacji o urządzeniu, algorytmie oraz znać szereg technologii, które mają wpływ na otrzymane charakterystyki (np. zmiana listy rozkazów (ang. *instruction set*) dla różnych architektur procesorów może prowadzić do zróżnicowanych wyników).

W przypadku urządzeń kryptograficznych, otrzymane przez adwersarza charakterystyki będą miały ścisły związek z użytymi kluczami oraz danymi wejściowymi. Prace opublikowane przez Paula Kochera w [10] pokazują, że charakterystyki czasowe dla operacji mnożenia i redukcji modularnej w RSA mogę prowadzić do odzyskania całego klucza.

Przykładem pomyślnie przeprowadzonego ataku może być także atak na OpenSSL'owe implementacje wykonany przez Boneh oraz Brumley. Atak ten opierał się na procedurze przewidywania skoków podczas realizacji algorytmu przez procesor.

Kolejnym sukcesem timing attacks było zaatakowanie przez Bernsteina w 2005 roku, OpenSSL'owej implementacji algorytmu AES. W tym przypadku, analizowane były czasy dostępu do pamięci CACHE. Atak składał się z kilku faz:

1. faza profilowania i ataku - podczas tej fazy zbierano czasy wykonywania się pojedynczego szyfrowania dla wielu powtórzeń (około miliona),

2. faza korelacji - szukano odpowiednich wzorców dla danych zebranych w fazie pierwszej oraz prezentowano potencjalną listę kandydatów do klucza,

3. faza przeszukania klucza - brutalny atak na kandydatów wyłonionych w fazie drugiej.

Wykonanie się tego ataku było możliwe ze względu na zastosowaną implementację warstwy nieliniowej algorytmu AES. Użyte zostały cztery T-tabele (wyniki dla wykonania się skrzynek podstawieniowych i MixColumns dla pojedynczych bajtów). Czas dostępu do różnych fragmentów tabel się różnił, co wpłynęło na uzyskanie dokładnych charakterystyk.

2.3 Ataki aktywne

W przeciwieństwie do ataków pasywnych, gdzie atakujący miał możliwość jedynie biernej analizy pewnych charakterystyk, ataki aktywne zakładają pewną ingerencję w urządzenia. Ataki aktywne są pewnym dopełnieniem ataków pasywnych, ponieważ niejednokrotnie ich wykonanie, musi być poprzedzone wstępną analizą, w celu zdjęcia zabezpieczeń będących w układzie. Z powodu wprowadzania błędnych wartości i zmian potoku obliczeń układu, niejednokrotnie w literaturze, ataki aktywne, określa się mianem ataków z uszkodzeniami (ang. *fault analysis*). Rozróżniamy następujące rodzaje:

- ataki proste - błędne wyniki analizowane są indywidualnie,
- ataki różnicowe - dla których następuje wstępne grupowanie a dopiero później przeprowadzana jest analiza.

Podział ataków z uszkodzeniami opiera się także na metodzie wprowadzania uszkodzeń:

1. Promieniowanie rentgenowskie - mikroprocesory niejednokrotnie zbudowane są z pierwiastków promieniotwórczych, które oddziałują na przetwarzane dane. Wzmocnienie promieniowania może powodować stałe określenie żądanych wartości, co w konsekwencji prowadzi do wykonania się błędnych obliczeń.
2. Promieniowanie podczerwone - układy są przewidziane do pracy w pewnym zakresie temperatur. Podczas dłuższej ekspozycji na promieniowanie

podczerwone, temperatura układu wzrasta, przez co w związku z prawami fizyki może dochodzi do opóźnień przełączania się przerzutników itp. co skutkuje wprowadzeniem uszkodzeń.

3. Zmienne napięcie zasilania - wytwarzanie zakłóceń w zasilaniu skutkuje brakiem możliwości poprawnego naładowania / rozładowania podstawowych elementów elektronicznych takich jak kondensatory i tranzystory.

4. Zakłócenie zegara taktującego dane urządzenie - prowadzi do zwiększenia / zmniejszenia częstotliwości działania układu, przez co można doprowadzić do sytuacji, w której niektóre instrukcje się nie wykonają.

5. Pole elektromagnetyczne - precyzyjne oddziaływanie na konkretne tranzystory może prowadzić do ustalenia konkretnych bitów układu.

Podczas analizy ataków z uszkodzeniami, szczególną uwagę zwraca się na skutki wykonania ataku. Wiedza np. o wyjściu z Sbox'a dla wejścia 8 bitowego, gdzie 2 bity klucza są ustalone poprzez oddziaływanie z urządzeniem, pozwala na dokładne ustalenie wartości użytych w algorytmie [9].

Każda implementacja algorytmu blokowego pociąga za sobą ryzyko wystąpienia wycieku informacji. Wraz z rozwojem ataków z kanałem bocznym, zaczęto zwracać większą uwagę na ochronę układów wykonujących obliczenia. Przeciwdziałanie atakom typu side - channel, to zbiór środków, które należy przyjąć w celu zminimalizowania ryzyka ataku, lub w przypadku incydentu naruszenia poufności danych, ograniczenia jego skutków. Niestety, pomiędzy twórcami ataków a osobami wykonującymi zabezpieczenia zawsze istnieje zależność, że najpierw pojawia się atak a dopiero wtedy opracowywane są metody przeciwdziałania.

2.4 Przeciwdziałanie atakom z pomiarem czasu

Wykonanie się każdej operacji w algorytmie zabiera czas. Czas wykonania się operacji zależy od jej rodzaju. W celu utrudnienia analizy atakującemu, należy doprowadzić do sytuacji, w której potok obliczeń realizuje się zawsze tak samo,

niezależnie od przyjętych danych oraz trwa określoną jednostkę czasu. Pierwszym sposobem na uzyskanie takiego rezultatu, może być tworzenie wysoce specjalizowanych implementacji, przygotowanych na konkretne docelowe urządzenia. Każde urządzenie posiada swoją architekturę oraz listę rozkazów (tzw. *instruction set*). Stworzenie wieloplatformowego rozwiązania, może okazać się nieoptymalne dla wielu urządzeń, przez co zaczną pojawiać się wycieki danych. Twórca przede wszystkim powinien wiedzieć, jakie zależności występują pomiędzy podzespołami bądź modułami, przez co efektywniej może sterować potokiem obliczeń. Idealną sytuacją byłoby, gdyby atakujący nie mógł na podstawie zdobytych informacji stwierdzić, kiedy dana faza algorytmu się zaczyna a kiedy kończy.

Jednym ze sposobów osiągnięcia takiego stanu rzeczy, jest rozbicie algorytmu na czynniki pierwsze i analiza wszystkich składowych. Występujące w algorytmie instrukcje warunkowe można zaimplementować w ten sposób, żeby zawsze wykonywały się te same operacje, mimo że, niektóre nie będą uwzględniane w całościowym procesie. Kolejnym sposobem jest pewna rezygnacja z pamięci RAM. Czas dostępu do różnych fragmentów pamięci jest różny, przez co uniezależniając się od niej, można ograniczyć ryzyko ataku z pomiarem czasu. Powyższe może być osiągnięte poprzez zastąpienie sparametryzowanych wartości przechowywanych w rejestrach przez dodatkowe potoki obliczeniowe, które realizują się w sposób jednolity niezależnie od danych wejściowych.

Ponadto zaleca się aby:

1. nie poddawać wartości tajnych porównywaniu fragmentarycznemu,
2. unikać instrukcji warunkowych i pętli zależnych od danych sekretnych,
3. stosować operacje matematyczne zamiast wartości tablicowych,
4. dopasować potrzeby algorytmu do architektury urządzenia i jego charakterystyk.

2.5 Przeciwdziałanie atakom poboru mocy

Ataki z pomiarem czasu są bardzo silnie powiązane z atakami poboru mocy. Czas wykonania się operacji wpływa na pobór prądu przez urządzenie. W dodatku

do podstawowych środków ochrony przed atakami z pomiarem czasu (ujednolicenie operacji liczbowych), należy podjąć kroki w celu zabezpieczenia implementacji przed atakami z bezpośrednią bądź różnicową analizą poboru mocy. Sposobami na ochronę przed SPA może być zaimplementowanie algorytmu w ten sposób, aby instrukcje warunkowe, pętle, nie zależały od danych wrażliwych. Atakujący wtedy nie jest w stanie prześledzić wejścia do konkretnej operacji przez co atak zostanie udaremnione. Sprawa okazuje się bardziej skomplikowana w przypadku DPA. Przedstawione wcześniej metody nie działają, ponieważ atak polega na zastosowaniu zaawansowanych metod statystycznych. Dlatego też, używa się dwóch technik, które pomagają tworzyć wielowarstwową ochronę:

- ukrywanie,
- maskowanie.

2.5.1 Ukrywanie

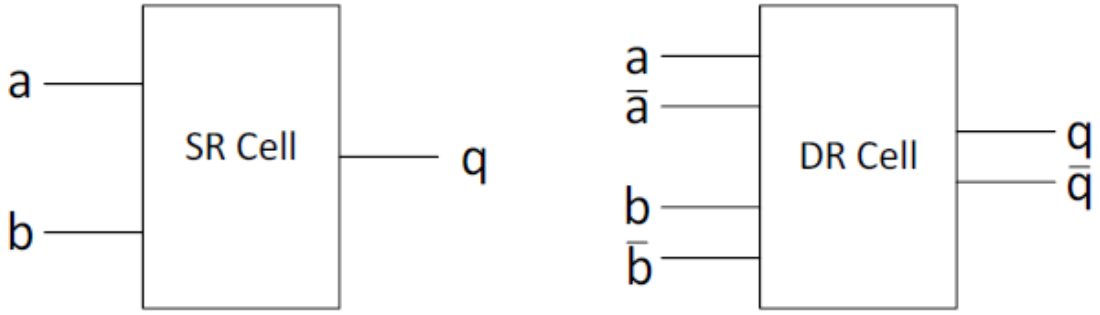
Ukrywanie danych ma na celu zmylenie atakującego, kiedy dana zmienna jest używana w algorytmie. Nie dokonywana jest zmiana jej wartości, ale poprzez specjalne techniki nie można powiedzieć, czy jest ona wykorzystywana, czy nie. Wyobraźmy sobie atakującego dokonującego analizy charakterystyk poboru mocy. Istnieją dwie sytuacje, w których nie będzie on mógł wyciągnąć z wykresów żadnych wniosków:

- wykres przyjmuje wartości losowe,
- wykres jest stały.

Wymaganie pierwsze można osiągnąć poprzez dodawanie niezależnych operacji, które będą wykonywane z losową częstotliwością na losowych danych, których wynik jest zbędny w realizacji całości algorytmu. Ponadto dobrym sposobem, o ile to możliwe, jest zamiana kolejności funkcji użytych w algorytmie.

W przypadku drugiego wymagania, przyjmuje się, że najlepiej jest używać takich operacji, które wykorzystują instrukcje procesora, wykonujące się w podobnej liczbie cykli. Dodatkowym środkiem zabezpieczającym jeśli chodzi o stronę sprzętową, jest

użycie technologii *dual-rail pre-charge*, polegającej na bilansowaniu poboru prądu poprzez ciągłe doładowywanie drugiej nóżki komórki, w ten sposób, aby całkowita moc pobrana przez sygnał, w każdym cyklu zegara była identyczna.



Rysunek 18: Komórki single i dual rail cell [13].

2.5.2 Maskowanie

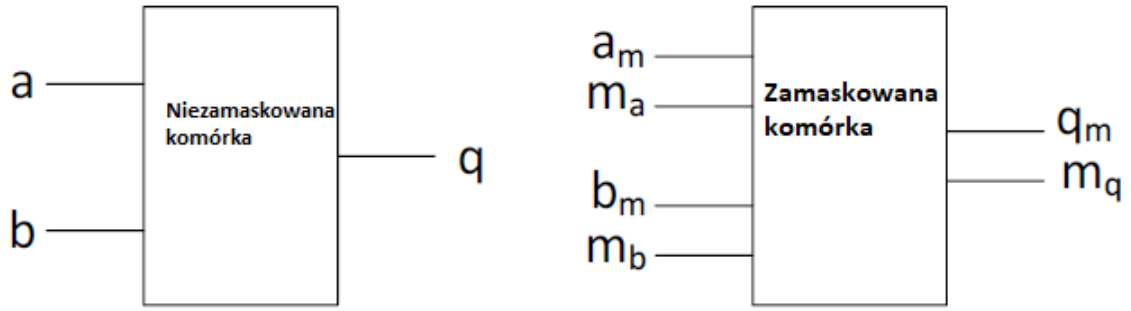
W przeciwieństwie do ukrywania, maskowanie zmienia wartości danych przetwarzanych przez algorytm. Najczęściej realizowane jest przy pomocy schematu dzielenia sekretu, w którym daną tajemnicę dzieli się na tzw. udziały. W przypadku obliczeń wykonywanych na urządzeniach, dzielenie sekretu polega na wykonywaniu się kilku równoległych potoków obliczeniowych, dla których dane nie są skorelowane z danymi tajnymi. Pozwala to na uniknięcie sytuacji, w której atakujący, który posiada dostęp do urządzenia, jest w stanie wyciągnąć wnioski na podstawie analizy potoku obliczeniowego jednego udziału. W końcowej fazie obliczeń, udziały są poddawane operacji dodawania modularnego co pozwala na otrzymanie wartości wyjściowej z algorytmu.

W przypadku operacji liniowych takich jak np. ShiftRows dla algorytmu AES bądź transpozycja τ dla Anubisa, korzysta się z zależności:

$$f(x \oplus m) = f(x) \oplus f(m). \tag{2.4}$$

Niestety powyższa własność jest prawdziwa tylko dla operacji liniowych. W przypadku warstw nieliniowych używa się innych metod przedstawionych na przykładzie w dalszych rozdziałach.

W podobieństwie do komórek *dual-rail pre-charged* używanych dla ukrywania danych, powstały komórki maskujące dane, które nie wykorzystują matematycznych narzędzi maskowania takich jak maskowanie oparte na algebrze Boole'a bądź maskowanie oparte na operacjach arytmetycznych. Zakładają one wykorzystanie dodatkowych linii przenoszących bity maski związane z wejściem i wyjściem z komórki.



Rysunek 19: Zamaskowana i niezamaskowana komórka [13].

2.6 Ochrona przed atakami z uszkodzeniami

Dotychczas przedstawione sposoby ochrony algorytmów przed atakami typu side-channel miały za zadanie utrudnić atakującemu analizę danych zebranych podczas pracy urządzenia kryptograficznego. W przypadku ochrony przed atakami z uszkodzeniami, przeciwdziałanie polega na udaremnieniu atakującemu wprowadzania zmian w algorytmie. Metody ochrony dzielimy na:

- udaremnienie wprowadzenia błędów,
- wykrywanie wprowadzonych błędów,
- poprawianie wprowadzonych błędów.

Podczas realizacji danych metod, najczęściej twórcy implementacji wykorzystują operacje odwrotne do operacji użytych w algorytmie, w celu sprawdzenia poprawności danych wejściowych. Ponadto korzystają oni z dodatkowych układów realizujących kontrolę parzystości, bądź wykorzystują cykliczną sumę kontrolną CRC. Dla przykładu, w przekształceniu AddRoundKey algorytmu AES, rezultat obliczany jest na podstawie dodawania modulo 2 odpowiednich bitów klucza i stanu. Operacja ta zachowuje parzystość zatem dla każdego bajtu powinny zachodzić równości:

$$p'_{i,j} = \bigoplus_{t=0}^{7} s'^{(t)}_{i,j} = \bigoplus_{t=0}^{7} (s^{(t)}_{i,j} \oplus k^{t}_{i,j}) = \bigoplus_{t=0}^{7} s^{(t)}_{i,j} \oplus \bigoplus_{t=0}^{7} k^{(t)}_{i,j} = p_{i,j} \oplus \bigoplus_{t=0}^{7} k^{(t)}_{i,j}. \tag{2.5}$$

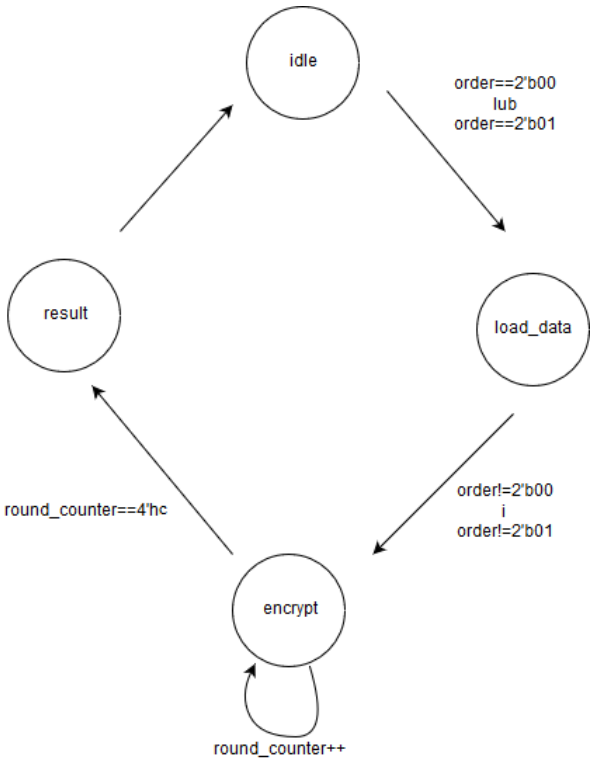
Lewa strona w zapisie powyżej oznacza parzystość bajtu stanu po przekształceniu. Widzimy, że parzystość kolejnego stanu zależy od parzystości poprzedniego stanu i parzystości bajtu klucza [9].

Rozdział 3

Sprzętowe implementacje algorytmu Anubis, w sposób uodparniający na ataki kanałem bocznym, analiza bezpieczeństwa i wydajności wykonanych implementacji

Procedura tworzenia implementacji sprzętowej szyfru blokowego odpornego na ataki z kanałem bocznym wymaga realizacji bardzo wielu kroków. Nie istnieje jeden konkretny schemat postępowania, który pozwoliłby na wypracowanie idealnego układu. Za każdym razem, twórca określa wymagania do swojej implementacji, a następnie przeprowadza procedurę, analizując każdy z komponentów układu. Niejednokrotnie, nie można sprostać wymaganiom stawianym przez zamawiającego. W niniejszym rozdziale pokażemy proces tworzenia implementacji szyfru Anubis oraz zaprezentujemy komponenty układu, które uległy zmianie w celu zachowania określonego poziomu bezpieczeństwa. Wykorzystana została wersja algorytmu z długością klucza równą 128 bitów. Wprowadzanie kolejnych ulepszeń będzie krok po kroku opisywane w kolejnych podrozdziałach. Każdy z podrozdziałów będzie zawierał także analizę wydajnościową zaproponowanych implementacji.

Ogólną cechą wspólną wszystkich przedstawionych poniżej implementacji jest zasada działania oparta o skończoną maszynę stanów (ang. *Finite State Machine*), która działa według schematu:



Rysunek 20: Maszyna stanów dla zaproponowanych implementacji.

Początkowym stanem dla układu jest stan *idle*. W tym stanie nie wykonują się żadne operacje algorytmu, natomiast układ czeka na sygnał, kiedy może rozpocząć działanie. Wraz z nadejściem sygnału *order* o wartości 00_b, stan zmienia się na *load_data* i następuje wczytanie 128 bitów klucza. Następnie (dalej w stanie *load_data*) dla wartości rozkazu *order*=01_b, 128 bitów tekstu jawnego jest dostarczanych na wejście, przez co wczytywanie danych zostaje ukończone. Zmiana wartości rozkazu na wartość inną od podanych skutkuje przejściem do stanu *encrypt*, gdzie wykonywanych jest 12 rund algorytmu. Każda runda wykonuje się w oddzielnym takcie zegara (na zboczu narastającym). Po ustawieniu się wartości licznika *round_counter* (inkrementowanej z każdym wykonaniem się rundy) na wartość 12,

stan układu zmienia się na *result*, co skutkuje wypisaniem wartości szyfrogramu na wyjściu układu.

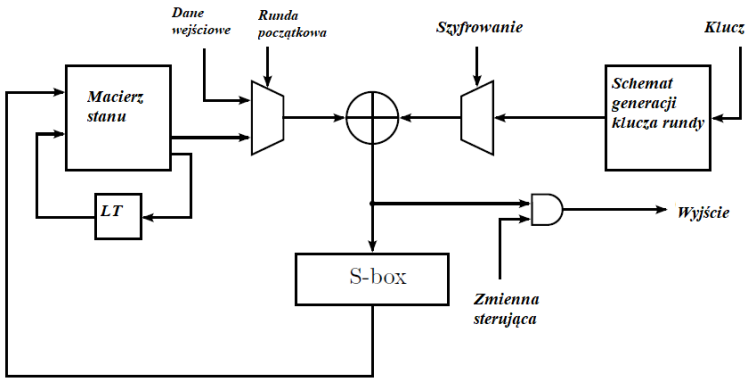
Wszystkie implementacje zostały wykonane w języku opisu sprzętu ***Verilog***, natomiast oprogramowaniem użytym do syntezy układu był Quartus Prime w wersji 16.1. Do sprawdzenia poprawności działania układu wykorzystano pakiet ***ModelSim*** dostępny w Quartus Prime.

Układy zostały zsyntezowane na płytce FPGA Altera Cyclone® V SE 5CSEMA6U23C6N posiadającej około 42 tysięcy elementów logicznych ALM (ang. *Adaptive Logic Module*).

3.1 Implementacja algorytmu Anubis

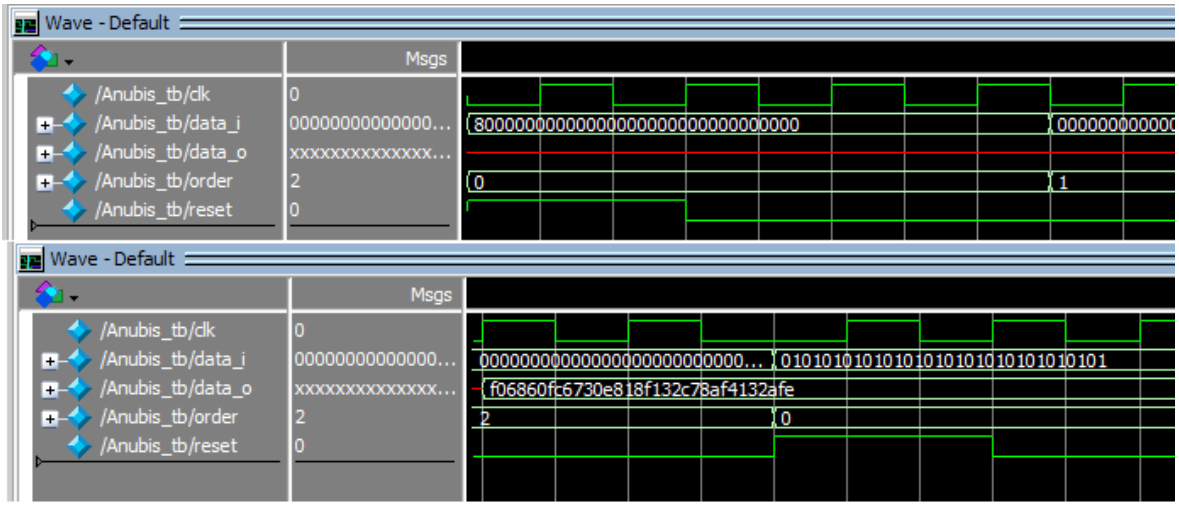
3.1.1 Implementacja nr 1

Pierwsza implementacja zakładała wykonanie się algorytmu Anubis zgodnie z dokumentacją dostarczoną przez twórców. Diagram przepływu danych został zaprezentowany poniżej:



Rysunek 21: Schemat algorytmu Anubis.

Jak widzimy na rysunku nr 21, zachowanie się wszystkich komponentów układu wynika wprost z dokumentacji. ***LT*** oznacza przekształcenia liniowe, które wykonywane są w algorytmie natomiast ***S-box*** oznacza warstwę nieliniową.



Rysunek 22: Wynik działania pierwszej implementacji.

Na rysunku nr 22 na wyjściu układu (*data_o* otrzymaliśmy ciąg zgodny z wektorami testowymi udostępnionymi przez twórców. Złożoność układu określa raport wygenerowany przez *Quartus Prime*.

Revision Name	Anubis
Top-level Entity Name	Anubis_2
Family	Cyclone V
Device	5CSEMA6U23C6
Timing Models	Final
Logic utilization (in ALMs)	2,652 / 41,910 (6 %)
Total registers	524
Total pins	4 / 314 (1 %)
Total virtual pins	256

Rysunek 23: Wykaz użytych zasobów dla pierwszej implementacji.

3.2 Narzędzia użyte do analizy bezpieczeństwa i analiza implementacji referencyjnej

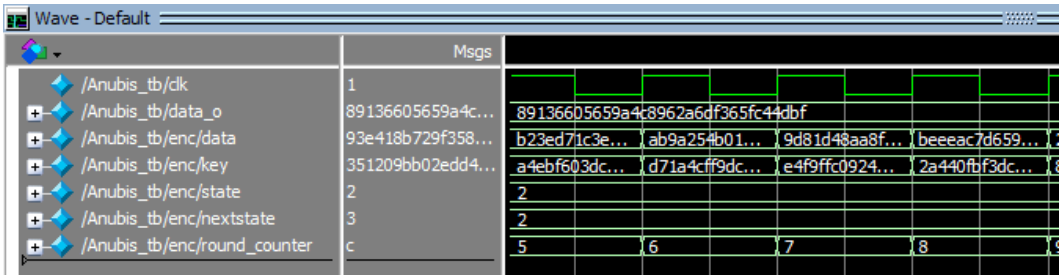
W celu sprawdzenia układu pod kątem odporności na ataki z kanałem bocznym postanowiono określić charakterystyki poboru mocy. Dla uproszczonego modelu, założono, że układ pobiera prąd w chwili przełączania przerzutników układu z wartości 0 na 1 bądź odwrotnie. W języku *Verilog*, zmiennymi odpowiedzialnymi za przechowywanie wartości do czasu jej zmiany są rejestry (ang. *registers*). W programie ***Quartus Prime***, istnieje możliwość sprawdzenia stanów rejestrów, w zależności od czasu działania urządzenia.

W celu zbadania działania algorytmu pod względem poboru mocy, ustalony był następujący schemat postępowania:

1. wykonanie implementacji,
2. wykonanie pliku testowego (*testbench'a*),
3. wydzielenie wartości rejestrów,
4. dokonanie symulacji.

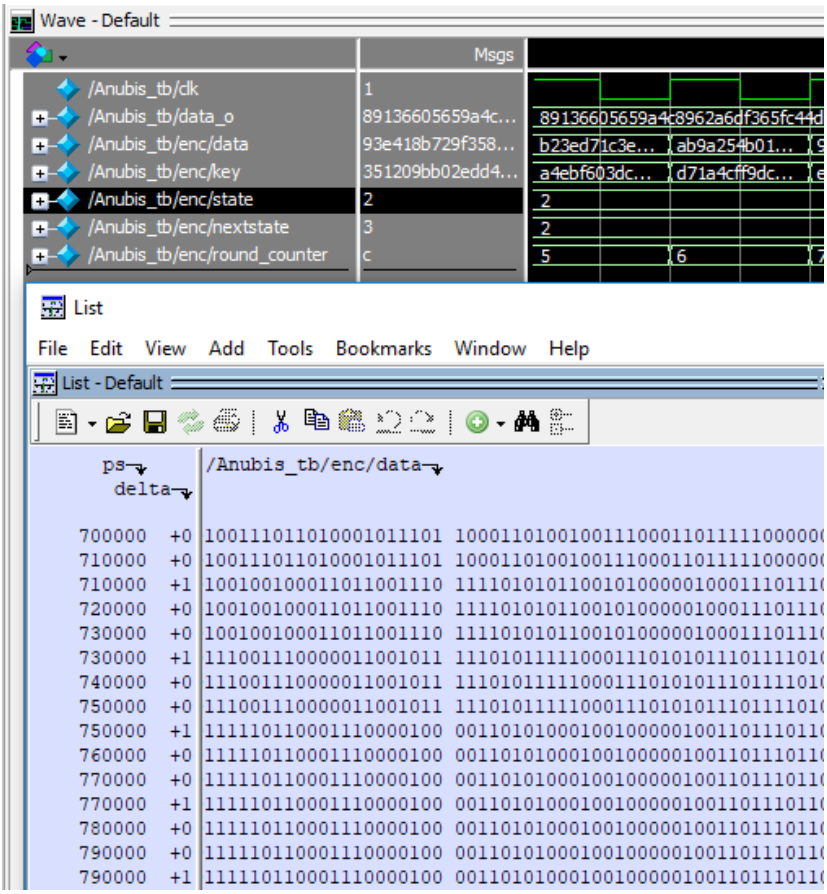
Poniżej przedstawiony zostanie przykładowy proces zbierania próbek do wykonania badań.

Po wykonaniu implementacji, utworzony został plik testowy, który miał za zadanie pokazanie wartości osiąganych przez układ. W celu zwiększenia losowości danych danych wejściowych a także ich liczby, powstał skrypt w języku *python*, którego zadaniem było utworzenie części głównej *testbench'a*. Następnie dla każdej implementacji określono główne rejestry odpowiedzialne za przechowywanie danych oraz przedstawiono ich wykres w środowisku *ModelSim Altera*. Na wykresie zaznaczono także zmienną odpowiadającą zegarowi (*clk*).



Rysunek 24: Symulacja za pomocą ModelSim Altera dla wartości z rejestrów.

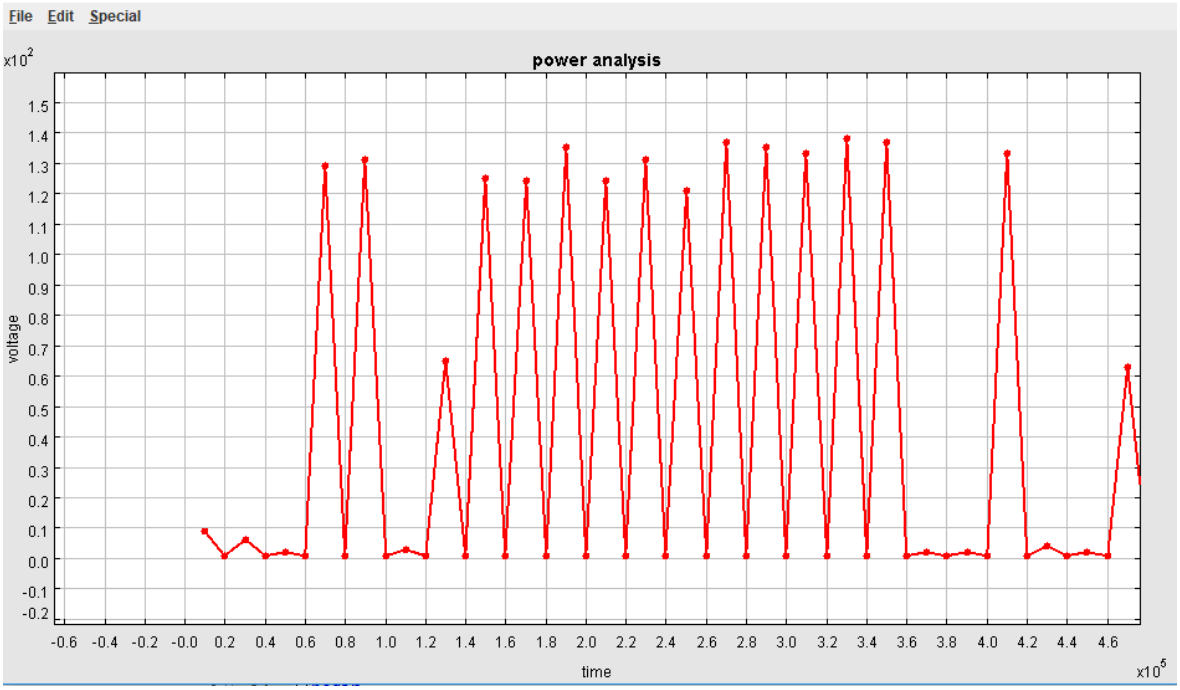
Tak uporządkowane dane przeniesiono do innego widoku dostępnego w symulacji - listy:



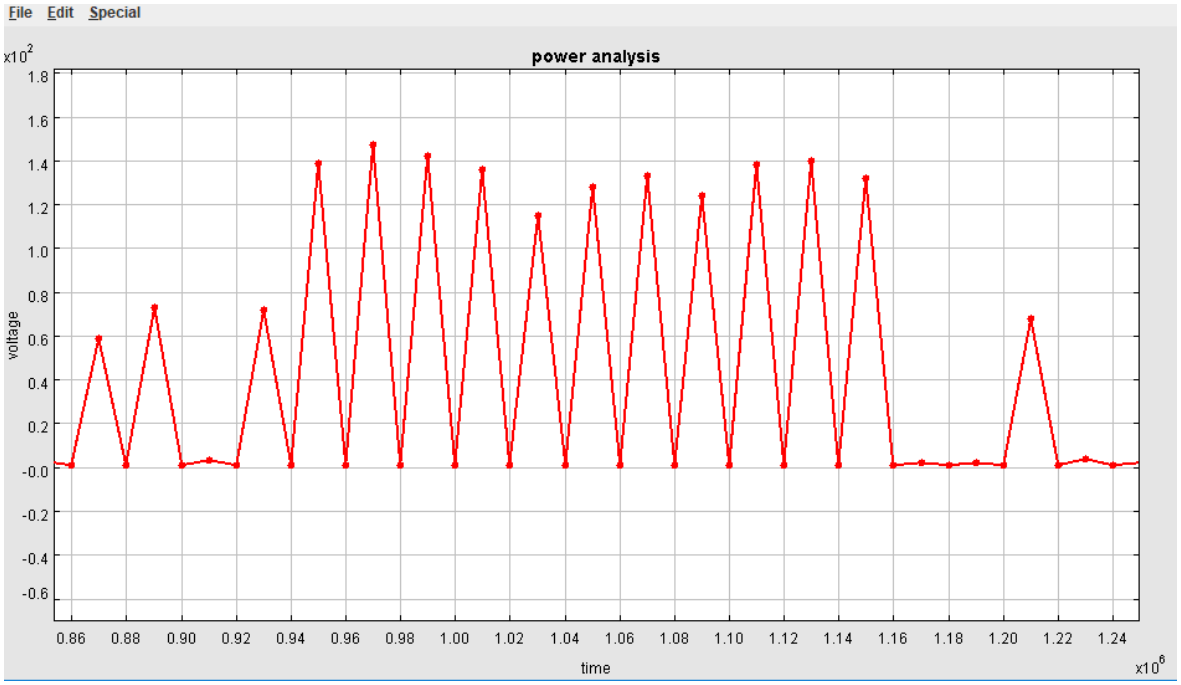
Rysunek 25: Przeniesienie danych do widoku w postaci listy.

Dane z tego widoku zostały następnie przekazane do specjalnie napisanego na te potrzeby programu w języku *Java*, obliczającego odległość Hamminga bitów rejestrów pomiędzy dwoma momentami czasu. Program ten działał na zasadzie porównywania dwóch ciągów znaków, gdzie poszczególne pozycje w ciągu odpowiadały konkretnym bitom rejestrów. Następnie za pomocą biblioteki ***PtPlot*** utworzona została wizualizacja danych w formie wykresu poboru mocy (liczby zmienionych bitów rejestru), od czasu. Drugim programem w języku *Java*, który powstał na potrzeby analizy implementacji jest program obliczający wagę Hamminga bitów rejestru w zależności od czasu. Informacje zawarte na wykresach otrzymanych za pomocą pierwszego i drugiego programu są w stanie znacząco ułatwić atakującemu analizę, dlatego ważne jest, aby w implementacjach zabezpieczonych doprowadzić do zbilansowania wartości osiąganych na wykresie.

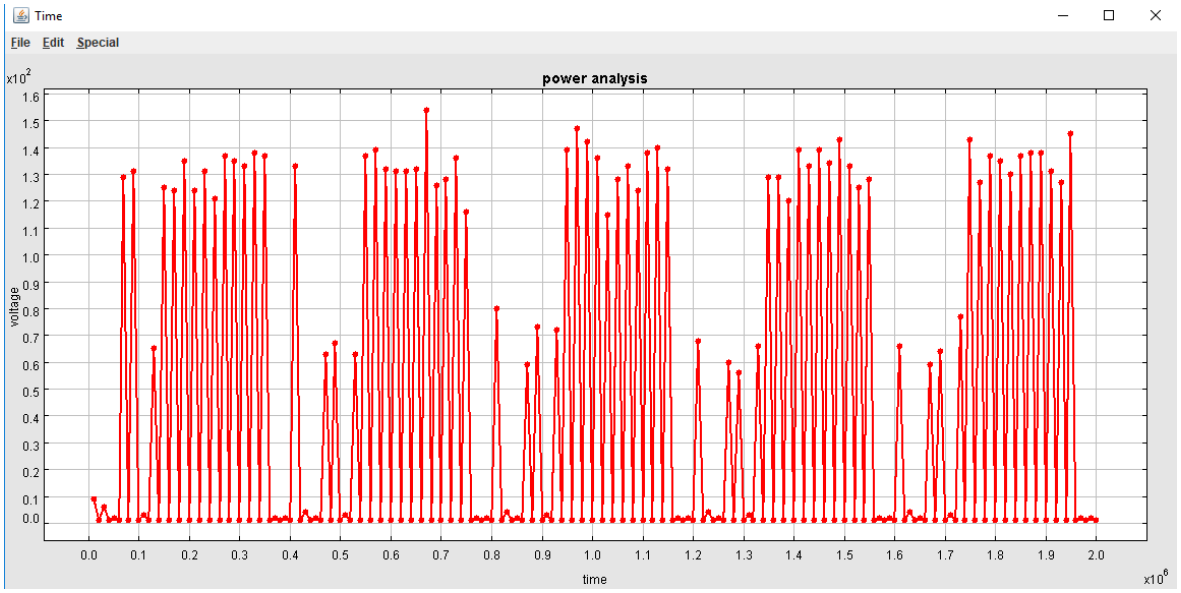
Wyniki otrzymane dla pierwszej implementacji przedstawiają się następująco:



Rysunek 26: Pierwsze uruchomienie symulacji.



Rysunek 27: Kolejne uruchomienie symulacji.



Rysunek 28: Kilka przebiegów symulacji niezabezpieczonej.

Jak widzimy na wykresie nr 26, początkowo następuje faza wczytania danych, świadczą o tym dwa piki o wysokości około 128 bitów. Zauważmy, że dla kolejnego uruchomienia, początkowe piki są niższe. Związane jest to z tym, że podczas pierwszego uruchomienia wartości rejestrów nie są określone, przez co w pierwszym uruchomieniu zapisaniu podlegają wszystkie bity, natomiast w kolejnych statystycznie połowa. Zauważmy również, że wartość piku dla pierwszej rundy (trzeciego piku od lewej strony) zawsze jest zaniżona w stosunku do kolejnych rund. Spowodowane jest to schematem generacji klucza rundy, który nie zmienia wartości klucza przechowywanej w rejestrze (klucza używanego w funkcji rozwoju klucza), natomiast dla rundy zerowej i pierwszej różni się tylko funkcją wyboru. Wszystkie pozostałe rundy w algorytmie można wskazać bardzo dokładnie. W końcowej fazie na wyjście przekazywany jest szyfrogram. Na rysunku nr 28 widać, że charakter zachowania się algorytmu jest stały, bez zależności od danych przechowywanych w algorytmie. Warto również zwrócić uwagę na fakt, że zbliżony poziom osiąganych pików wynika z dobrych właściwości dyfuzji i konfuzji szyfru Anubis. Statystycznie połowa bitów w każdej rundzie jest przetwarzana i nie występują anomalie.

3.3 Uodpornienie standardowej implementacji

W początkowej fazie uodparniania algorytmu Anubis na ataki typu side-channel postanowiono poprawić implementację z poprzedniego rozdziału w ten sposób, aby ujednolicić pobór mocy. Zastosowano dwie metody, z których jedna polegała na dołączeniu modułu liczącego wykonującego się na zboczu opadającym natomiast druga zakładała użycie rejestrów przechowujących dane zanegowane. Wprowadzanie podanych metod odbywało się stopniowo.

3.3.1 Implementacja nr 2

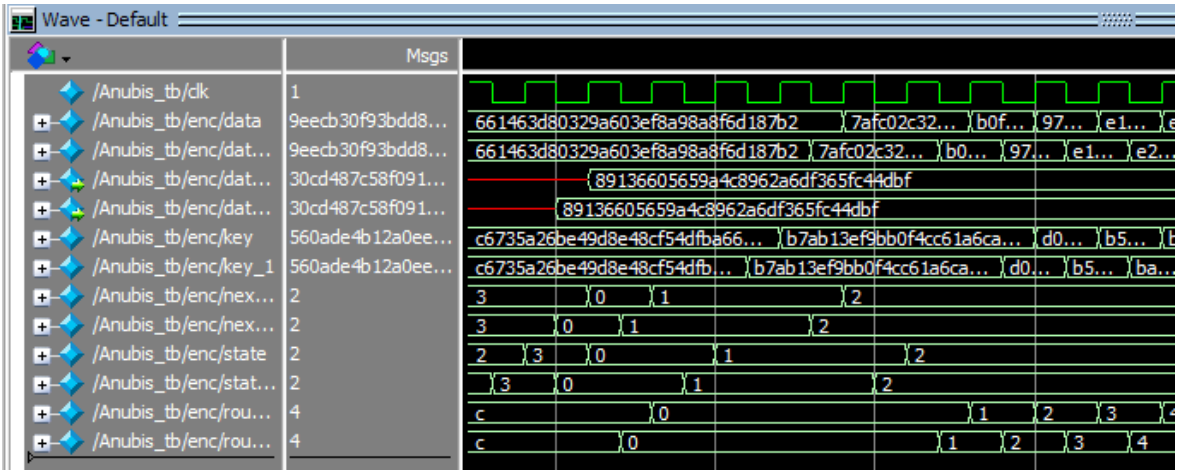
Początkowo testowano, jakie obliczenia mają się wykonywać dla opadającej wartości zbocza zegara. W pierwszym wariancie założono, ze wystarczy powielić obliczenia. Niestety rozwiązanie to nie okazało się dobre, ponieważ dla zbocza narastającego i opadającego na wykresie poboru mocy otrzymywaliśmy piki na tym samym poziomie.

Oznacza to, że atakujący, który zauważyłby podaną zależność, mógłby pominąć jeden zbiór punktów, który jest przesunięty w fazie w stosunku do drugiego. Postanowiono zatem wykonywać obliczenia dla danych wejściowych zanegowanych, co dało następujące rezultaty:

Revision Name	Anubis
Top-level Entity Name	Anubis_2
Family	Cyclone V
Device	5CSEMA6U23C6
Timing Models	Final
Logic utilization (in ALMs)	5,127 / 41,910 (12 %)
Total registers	1048
Total pins	132 / 314 (42 %)
Total virtual pins	256
Total block memory bits	0 / 5,662,720 (0 %)

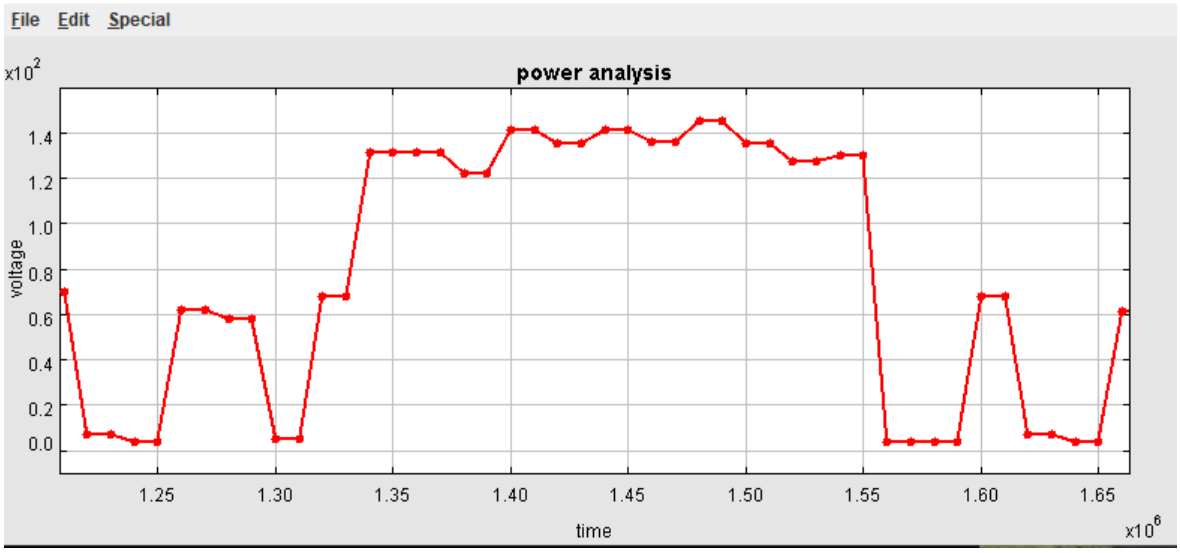
Rysunek 29: Zasoby potrzebne dla implementacji nr 2.

Jak widzimy, liczba zasobów potrzebnych do obsłużenia implementacji wzrosła prawie dwukrotnie.

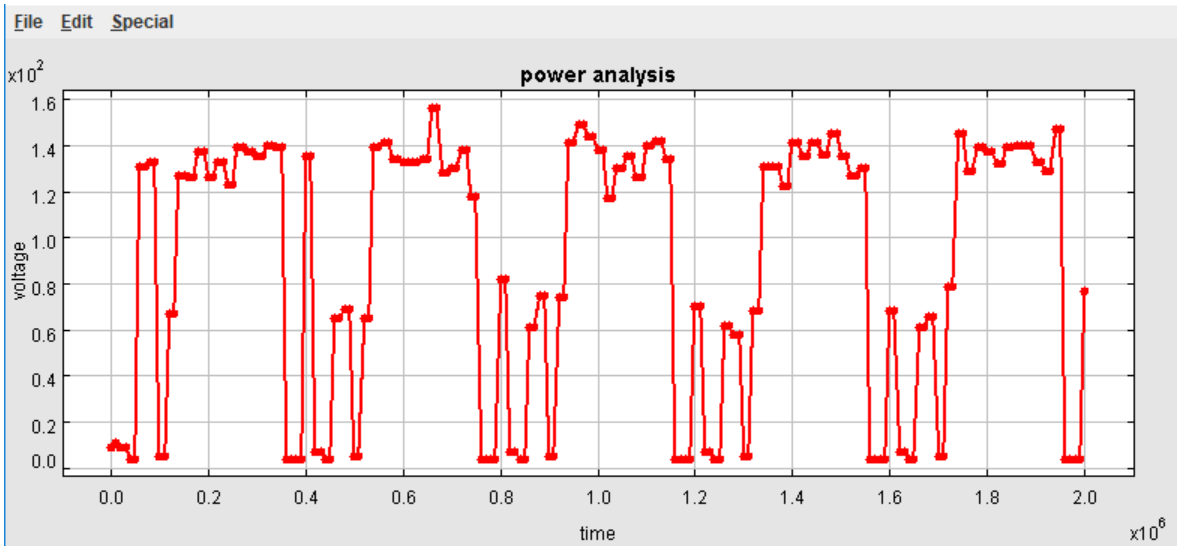


Rysunek 30: Koncepcja użycia zbocza opadającego dla danych wejściowych takich samych jak dla zbocza narastającego.

Dla takich samych danych wejściowych otrzymujemy wartości rejestrów przesunięte w fazie, co można także zaobserwować na poniższym rysunku:

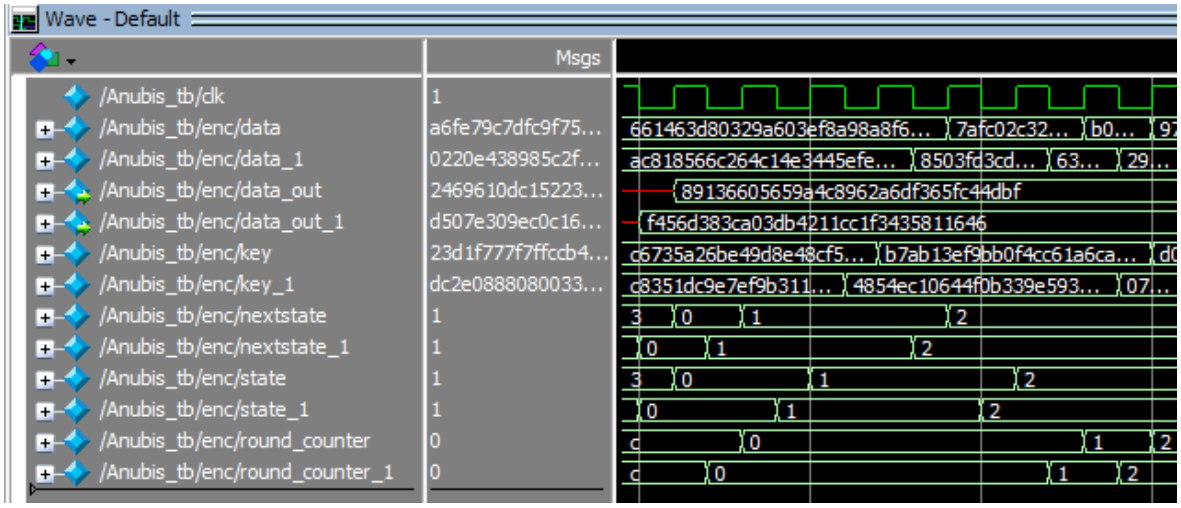


Rysunek 31: Przykładowy przebieg dla takich samych danych wejściowych na zboczu opadającym i narastającym.

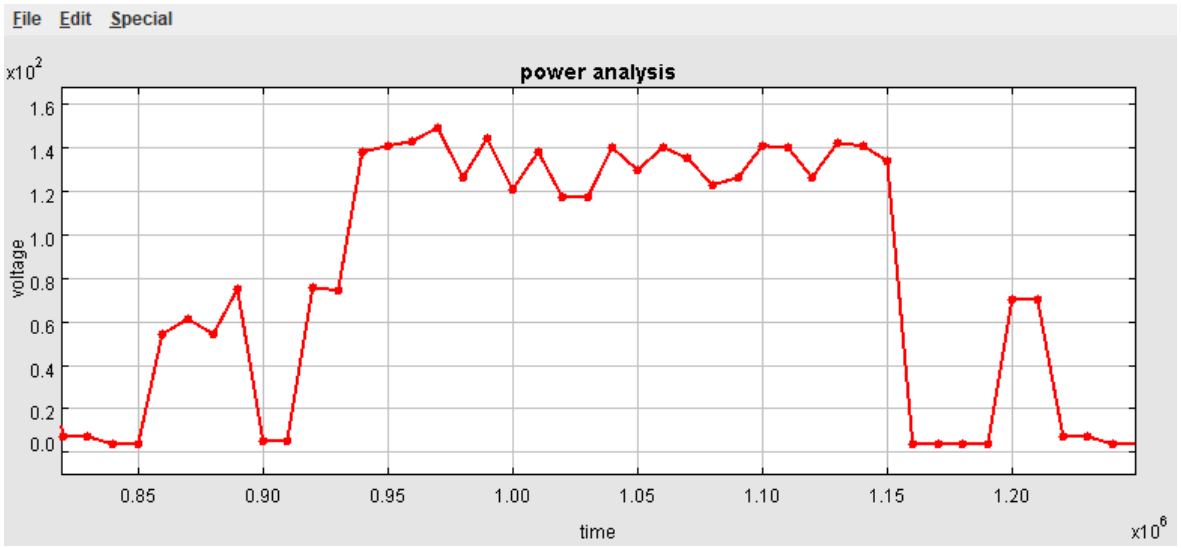


Rysunek 32: Kilka przebiegów.

W celu wprowadzenia pewnej losowości, na zboczu opadającym podano dane zanegowane. Otrzymano następujące wyniki:



Rysunek 33: Różne dane przetwarzane w rejestrach.



Rysunek 34: Przykładowy przebieg dla różnych danych wejściowych dla zbocza narastającego i opadającego.

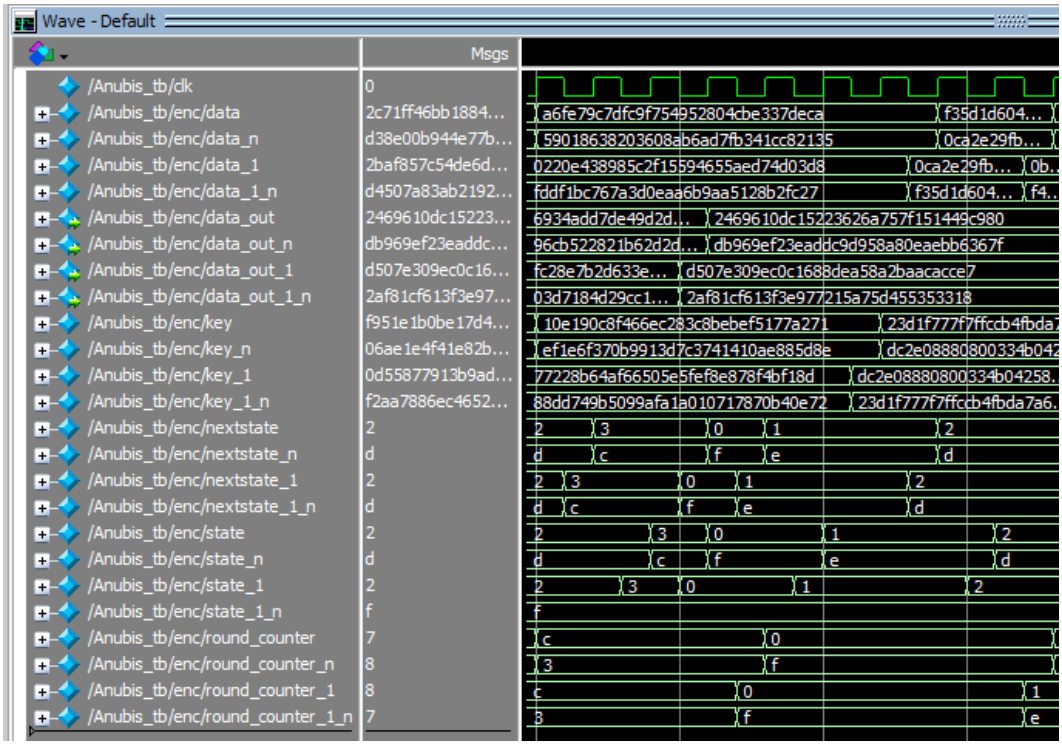
Wprowadzenie pewnej losowości, wymusza na atakującym wykonanie dodatkowej pracy w celu chociażby poprawnego stwierdzenia, które piki odnoszą się do układu realizującego obliczenia na danych, które atakujący faktycznie chce poznać.

3.3.2 Implementacja nr 3

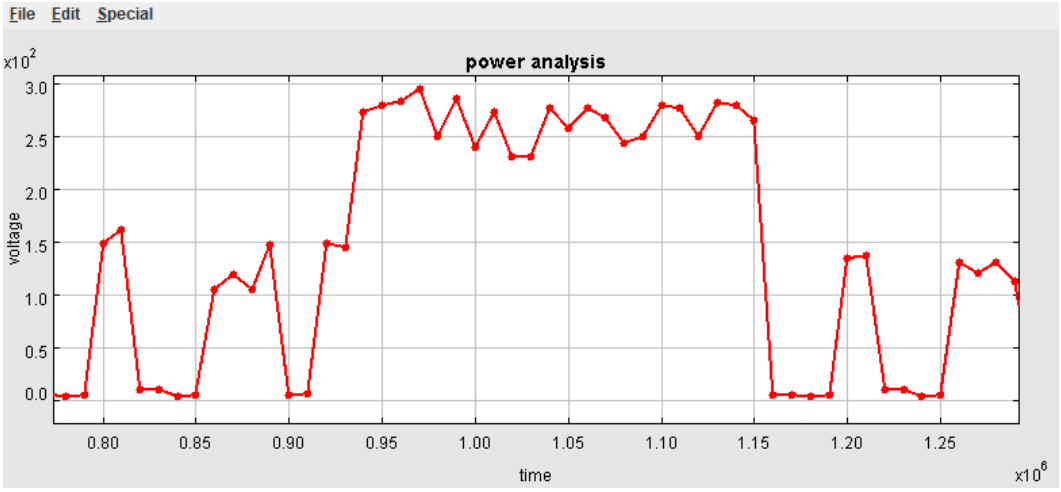
Po dodaniu obliczeń wykonujących się w przeciwfazie, należy jeszcze zadbać o zbilansowanie drugiej charakterystyki poboru mocy, a mianowicie wagi Hamminga bitów. Zbilansowanie to postanowiono uzyskać poprzez dołożenie rejestrów, które przechowują wartości zanegowane do tych w rejestrach oryginalnych. Zwiększa to oczywiście liczbę użytych rejestrów, niemniej jednak pozwala na znaczące utrudnienie analizy atakującemu.Na rysunku nr 35 widzimy, że liczba bitów rejestrów wzrosła do 1300. Mimo tego, w porównaniu do implementacji nr 2, liczba wymaganych komórek logicznych (ALM) wzrosła tylko o około 300 jednostek, co związane jest z obsłużeniem tych rejestrów zanegowanych.

Revision Name	Anubis
Top-level Entity Name	Anubis_2
Family	Cyclone V
Device	5CSEMA6U23C6
Timing Models	Final
Logic utilization (in ALMs)	5,472 / 41,910 (13 %)
Total registers	1300
Total pins	4 / 314 (1 %)
Total virtual pins	640
Total block memory bits	0 / 5,662,720 (0 %)

Rysunek 35: Zużycie zasobów dla implementacji nr 3.



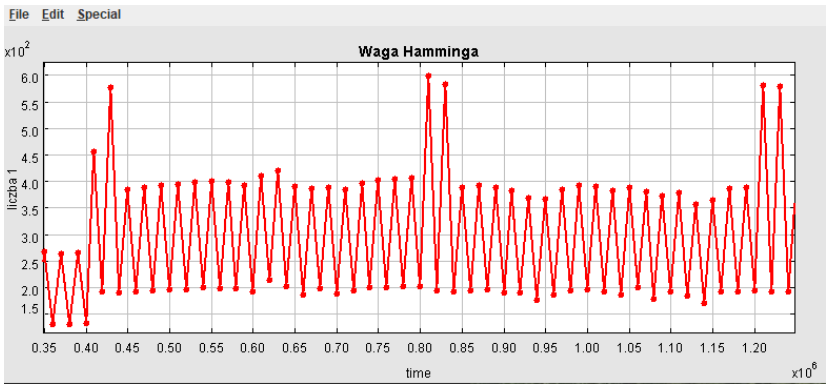
Rysunek 36: Symulacja implementacji nr 3 przedstawiająca wartości rejestrów.



Rysunek 37: Wykres pobory mocy dla implementacji nr 3.

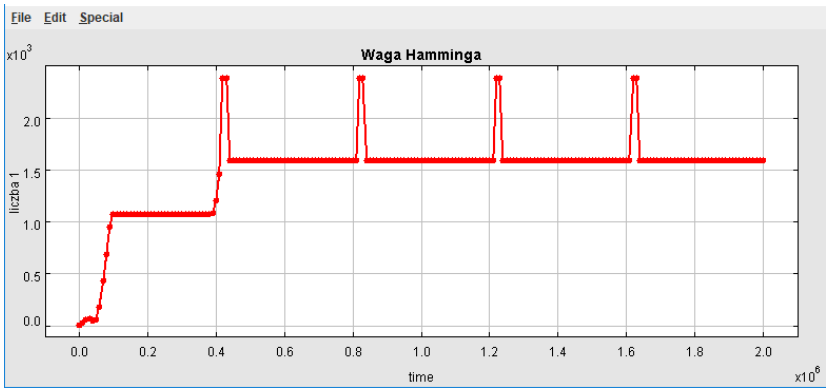
Dla nowo powstałej implementacji, wykres poboru mocy nie zmienił się znacząco w stosunku do poprzedniej implementacji. Zwiększyły się tylko różnice pomiędzy pikami, co spowodowane jest dołożeniem liczby rejestrów.

W przypadku charakterystyki odnoszącej się do wagi Hamminga, nastąpiło znaczące wyrównanie się liczby jedynek w rejestrach. Dla pierwszej implementacji otrzymaliśmy wykres:

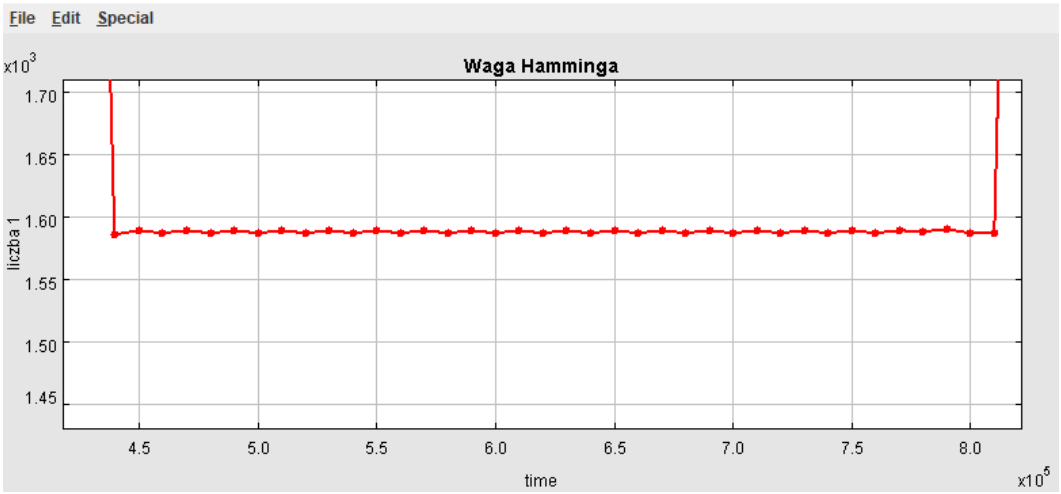


Rysunek 38: Waga Hamminga dla implementacji nr 1.

Podczas gdy dla trzeciej:



Rysunek 39: Waga Hamminga dla implementacji nr 3.



Rysunek 40: Zbliżenie dla powyższego wykresu.

Zauważmy, że wykres jest prawie równoległy do osi *OX*. Zmiany wartości o jeden bit wynikają ze zmiennej *clk* odnoszącej się do zegara i nie mającej wpływu na pozostałe rejestry. Piki na wykresach o znaczącej wartości należy rozumieć jako momenty czasu, dla których w rejestrach po raz pierwszy zapisywane są dane.

3.4 Wady w konstrukcji

Wszystkie wprowadzone do tej pory ulepszenia wynikały z chęci ujednolicenia poboru mocy oraz liczby jedynek w rejestrach. Zastanówmy się teraz, co się stanie, gdy atakujący poza nieograniczonym dostępem do urządzenia szyfrującego, będzie miał także dostęp do każdego z komponentów. Każdy komponent we wcześniejszych implementacjach przetwarzał dane w postaci takiej, jak wynika to z dokumentacji. Oznacza to, że dla atakującego, możliwym jest wykonanie ataków DPA i SPA na konkretne moduły, przez co zastosowane dotychczas metody ochrony wydają się być niewystarczające. Na ratunek przychodzi jednak zaprezentowane w rozdziale 2.5.2 maskowanie. Najczęściej wykonuje się je za pomocą schematu podziału sekretu, niemniej jednak nie wszystkie szyfry nadają się do zastosowania tego narzędzia. Algorytmem takim jest niestety Anubis.

Warstwa nieliniowa algorytmu Anubis zbudowana jest z 16 Sbox'ów, w których każdy realizuje podstawienie zgodnie z wyznaczoną przez twórców tabelą *look-up table*. Baretto i Rijmen nie przewidywali wykonywania implementacji sprzętowych swojego algorytmu (świadczy o tym brak referencyjnej implementacji na hardware), przez co, pomimo swoich dobrych właściwości, skrzynka okazuje się być najsłabszą stroną implementacji sprzętowej. Przypomnijmy, że dla opisywanego algorytmu, skrzynka powstała poprzez losowanie wartości tak, aby zachować pewne warunki:

- skrzynka S musi być przekształceniem inwolucyjnym, tzn. $S[S[x]] = x$ dla każdego $x \in GF(2^8)$,
- wartość parametru δ nie może przekraczać $8x2^{-8}$,
- wartość parametru λ nie może przekraczać $16x2^{-6}$,
- rząd nieliniowości ν musi być maksymalny (7),
- skrzynka nie może mieć punktów stałych, tzn. $S[x] \neq x$, dla każdego $x \in GF(2^8)$.

W niniejszej pracy postanowiono dokonać modyfikacji skrzynki podstawieniowej algorytmu Anubis, na tę odpowiednią dla algorytmu AES, w celu pokazania schematu dzielenia sekretu oraz analizy wniosków płynących z tej zamiany. Powstało kolejne kilka implementacji, z których każda wprowadza dodatkowe elementy.

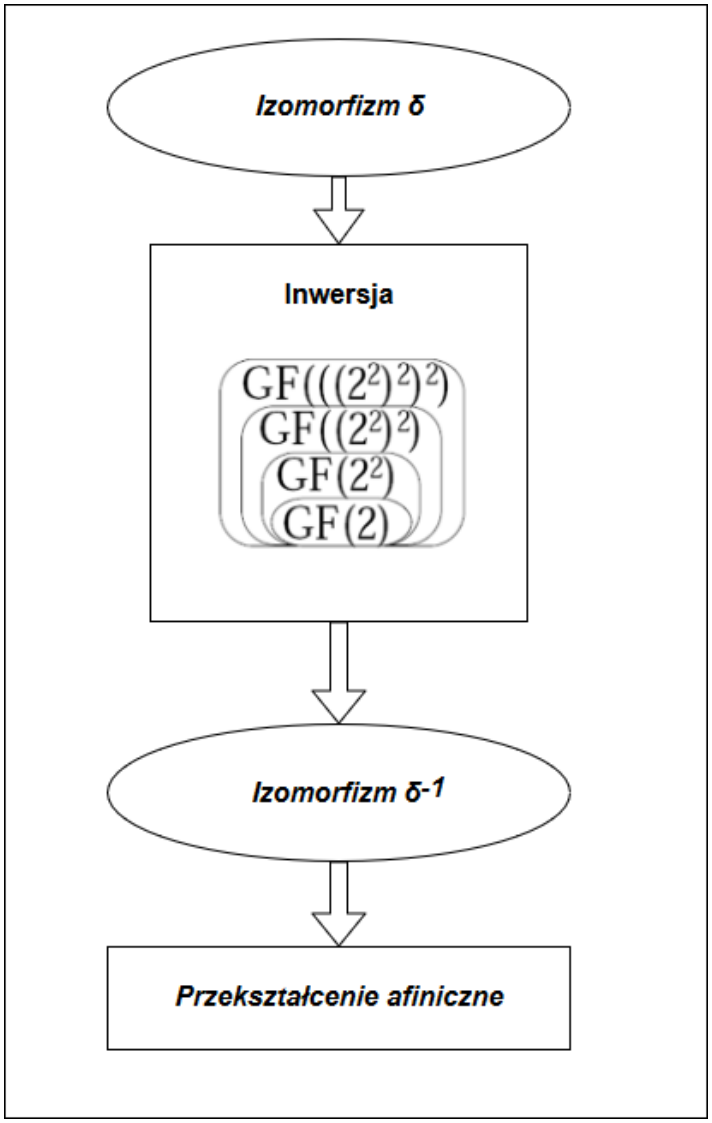
Rozdział 4

Modyfikacje algorytmu Anubis pozwalające na dalsze uodparnianie algorytmu na ataki kanałem bocznym, analiza bezpieczeństwa i wydajności wykonanych implementacji

4.1 Analiza warstwy nieliniowej zmodyfikowanego algorytmu

Skrzynka podstawieniowa algorytmu AES dana jest za pomocą tabeli. W przeciwieństwie jednak do algorytmu Anubis, wartości nie wynikają z losowego przypisania, ale są wynikami wykonywania działań w ciele $GF(2^8)$. Początkowo, dla wejścia liczona jest odwrotność modularna w ciele, natomiast otrzymany wynik, poddawany jest przekształceniu afinicznemu (z tego względu np. $0x00_{hex}$ przechodzi na $0x63_{hex}$). Wraz z rozpowszechnieniem się algorytmu AES, zaczęto zastanawiać się nad efektywnym wyznaczaniem wartości podstawień nie poprzez odczytanie wartości

elementu z pamięci układu ale poprzez realizację szeregu obliczeń. Opracowano metodę implementacji skrzynki w sprzęcie za pomocą następujących przekształceń:



Rysunek 41: Potok obliczeń dla skrzynki podstawieniowej.

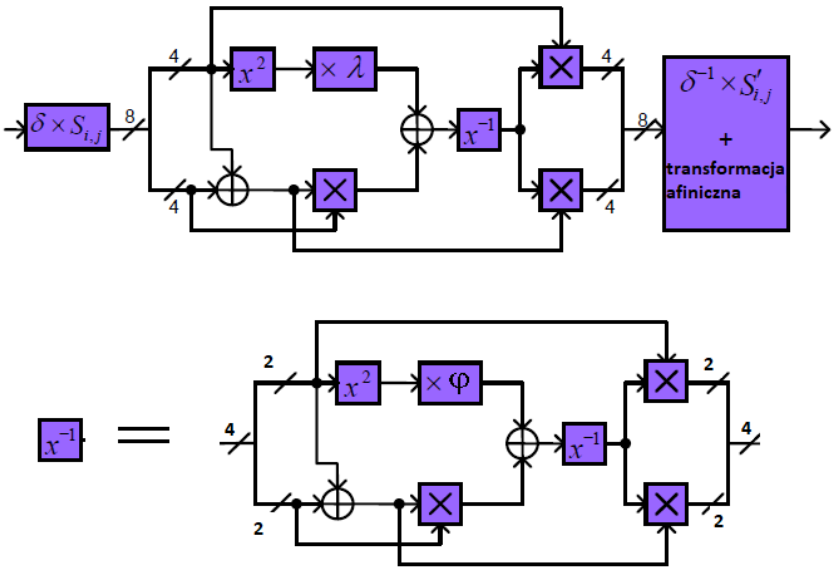
Dla powyższego schematu, izomorfizm δ przyjmuje postać:

$$GF(2^8) \Rightarrow GF(((2^2)^2)^2) : f(q) = \delta \times q$$
$$GF(((2^2)^2)^2)) \Rightarrow GF(2^8) : f^{-1}(q) = \delta^{-1} \times q,$$

gdzie δ oraz δ^{-1} dane są w postaci macierzowej jako:

$$\delta = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \delta^{-1} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Następnym etapem jest dekompozycja ciała $GF(2^8)$, w którym ciężko oblicza się odwrotność, do mniejszego ciała za pomocą następującej konwersji:

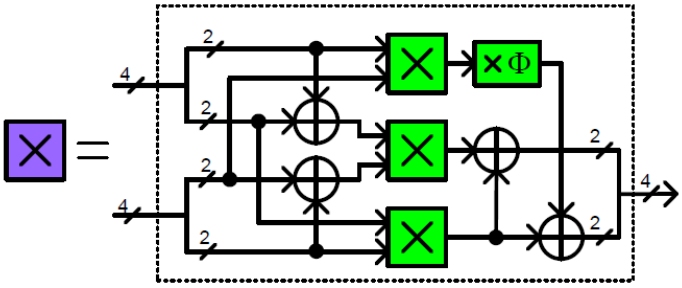


Rysunek 42: Dekompozycja skrzynki podstawieniowej w implementacji sprzętowej [15].

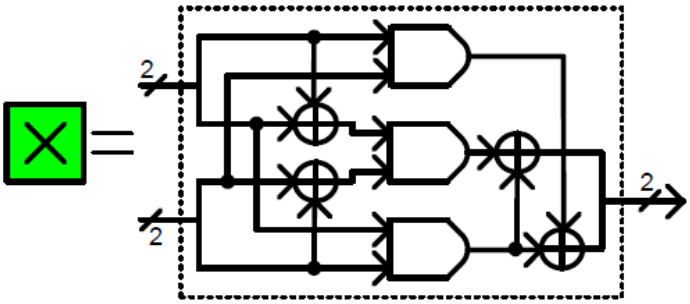
Do implementacji powyższego układu, potrzebujemy takie moduły jak:

- podnoszenie elementu ciała do kwadratu,

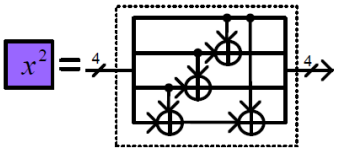
- mnożenie w ciele,
- dodawanie w ciele,
- liczenie odwrotności w mniejszym ciele.



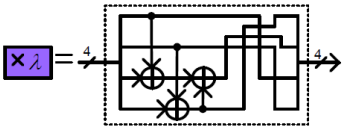
(a) Mnożenie w $GF(2^4)$.



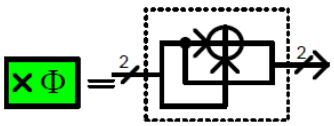
(b) Mnożenie w $GF(2^2)$.



(c) Podnoszenie do kwadratu w $GF(2^4)$.



(d) Mnożenie przez λ.



(e) Mnożenie przez ϕ.

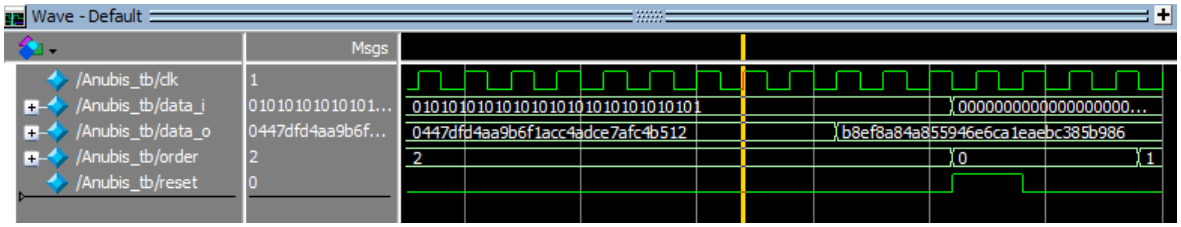
Rysunek 43: Komponenty Sbox [15].

4.2 Implementacja algorytmu Anubis ze zmienioną skrzynką podstawieniową

W związku z brakiem wektorów testowych dla szyfru Anubis o podmienionej skrzynce, postanowiono dokonać dwóch implementacji.

4.2.1 Implementacja nr 4 i 5

Zmiana w implementacji nr 4, w stosunku do implementacji nr 1, polegała jedynie na podstawieniu w miejsce Sbox'a wartości odpowiednich nie dla algorytmu Anubis, ale dla algorytmu AES. Dzięki zastosowaniu takiego rozwiązania, powstały wektory testowe do sprawdzenia implementacji, w której skrzynka podstawieniowa realizuje potok obliczeń opisany w 4.1. Implementacja ta nie zwiększyła wymagań pod względem zasobów w stosunku do pierwszej a jej realizacja przedstawia się następująco:



Rysunek 44: Symulacja implementacji nr 4.

Dla implementacji nr 5 (ze skrzynką podstawieniową działającą na zasadzie wykonywania obliczeń) otrzymujemy identyczne wartości co na rysunku nr 44, co oznacza, że implementacja wykonana jest poprawnie. Pomimo zgodności działania układów, występują duże rozbieżności pod względem liczby wymaganych zasobów. Obrazuje to rysunek nr 45:

Revision Name	Anubis
Top-level Entity Name	Anubis_2
Family	Cyclone V
Device	5CSEMA6U23C6
Timing Models	Final
Logic utilization (in ALMs)	2,624 / 41,910 (6 %)
Total registers	524
Total pins	4 / 314 (1 %)
Total virtual pins	256

(a) Implementacja nr 4.

Revision Name	Anubis
Top-level Entity Name	Anubis_2
Family	Cyclone V
Device	5CSEMA6U23C6
Timing Models	Final
Logic utilization (in ALMs)	4,786 / 41,910 (11 %)
Total registers	524
Total pins	4 / 314 (1 %)
Total virtual pins	256

(b) Implementacja nr 5.

Rysunek 45: Porównanie użytych zasobów w implementacji nr 4 i 5.

4.2.2 Schemat dzielenia sekretu

W implementacji wykorzystano schemat dzielenia sekretu, opisany w [2]. Zasada jego działania, polega na zamaskowaniu zmiennych użytych w programie. Początkowo dane wartości wejściowe rozdziela się tak, aby ich różnica symetryczna dawała początkową wartość. Następnie dla tak utworzonych zmiennych zapisuje się równania, które tworzą zmienne wyjściowe, które po wykonaniu na nich operacji modulo 2, zwracają taki wynik, jak dla danych wejściowych. Przewagą tego rozwiązania jest praca układu na wartościach innych niż wejściowe. Niestety zajętość układu znacząco wzrasta w związku z wykorzystaniem układ opisujących większą liczbę zmiennych. Przykładowe operacje i równania dla jednomianów drugiego i trzeciego stopnia oraz różnej liczby udziałów wejściowych i wyjściowych zaprezentowano poniżej:

$F = XY,$
$F = F_1 \oplus F_2 \oplus F_3,$
$X = X_1 \oplus X_2 \oplus X_3 \oplus X_4,$
$Y = Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4,$
$F_1 = (X_2 \oplus X_3 \oplus X_4)(Y_2 \oplus Y_3) \oplus Y_4,$
$F_2 = ((X_1 \oplus X_3)(Y_1 \oplus Y_4)) \oplus X_1Y_3 \oplus X_4,$
$F_3 = ((X_2 \oplus X_4)(Y_1 \oplus Y_4)) \oplus X_1Y_2 \oplus X_4 \oplus Y_4.$

$F = XYZ \oplus XY \oplus Z,$
$F = F_1 \oplus F_2 \oplus F_3 \oplus F_4 \oplus F_5,$
$X = X_1 \oplus X_2 \oplus X_3 \oplus X_4 \oplus X_5,$
$Y = Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5,$
$Z = Z_1 \oplus Z_2 \oplus Z_3 \oplus Z_4 \oplus Z_5,$
$F_1 = ((X_2 \oplus X_3 \oplus X_4 \oplus X_5)(Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5)(Z_2 \oplus Z_3 \oplus Z_4 \oplus Z_5)) \oplus$
$((X_2 \oplus X_3 \oplus X_4 \oplus X_5)(Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5)) \oplus Z_2$
$F_2 = (X_1(Y_3 \oplus Y_4 \oplus Y_5)(Z_3 \oplus Z_4 \oplus Z_5) \oplus Y_1(X_3 \oplus X_4 \oplus X_5)(Z_3 \oplus Z_4 \oplus Z_5) \oplus$
$Z_1(X_3 \oplus X_4 \oplus X_5)(Y_3 \oplus Y_4 \oplus Y_5) \oplus X_1Y_1(Z_3 \oplus Z_4 \oplus Z_5) \oplus$
$X_1Z_1(Y_3 \oplus Y_4 \oplus Y_5) \oplus Y_1Z_1(X_3 \oplus X_4 \oplus X_5) \oplus X_1Y_1Z_1) \oplus$
$(X_1(Y_3 \oplus Y_4 \oplus Y_5) \oplus Y_1(X_3 \oplus X_4 \oplus X_5) \oplus X_1Y_1) \oplus Z_3,$
$F_3 = (X_1Y_1Z_2 \oplus X_1Y_2Z_1 \oplus X_2Y_1Z_1 \oplus X_1Y_2Z_2 \oplus X_2Y_1Z_2 \oplus X_2Y_2Z_1 \oplus$
$X_1Y_2Z_4 \oplus X_2Y_1Z_4 \oplus X_1Y_4Z_2 \oplus X_2Y_4Z_1 \oplus X_4Y_1Z_2 \oplus X_4Y_2Z_1 \oplus$
$X_1Y_2Z_5 \oplus X_2Y_1Z_5 \oplus X_1Y_5Z_2 \oplus X_2Y_5Z_1 \oplus X_5Y_1Z_2 \oplus X_5Y_2Z_1) \oplus$
$(X_1Y_2 \oplus Y_1X_2) \oplus Z_4,$
$F_4 = X_1Y_2Z_3 \oplus X_1Y_3Z_2 \oplus X_2Y_1Z_3 \oplus X_2Y_3Z_1 \oplus X_3Y_1Z_2 \oplus X_3Y_2Z_1) \oplus Z_5,$
$F_5 = Z_1.$

Użycie schematu dzielenia sekretu sprowadza się do zapisania równań dla operacji realizowanych przez każdy z komponentów i ich zastosowaniu. Na przykład dla mnożenie w $GF(2^4)$ mamy następujące równania:
$(Z_3, Z_2, Z_1, Z_0) = (X_3, X_2, X_1, X_0)(Y_3, Y_2, Y_1, Y_0),$
$Z_3 = X_3Y_3 \oplus X_3Y_2 \oplus X_3Y_1 \oplus X_3Y_0 \oplus X_2Y_3 \oplus X_2Y_1 \oplus X_1Y_3 \oplus X_1Y_2 \oplus X_0Y_3,$
$Z_2 = X_3Y_3 \oplus X_3Y_1 \oplus X_2Y_2 \oplus X_2Y_0 \oplus X_1Y_3 \oplus X_0Y_2,$
$Z_1 = X_3Y_2 \oplus X_2Y_3 \oplus X_2Y_2 \oplus X_1Y_1 \oplus X_1Y_0 \oplus X_0Y_1,$
$Z_0 = X_3Y_3 \oplus X_3Y_2 \oplus X_2Y_3 \oplus X_1Y_1 \oplus X_0Y_0.$
Dla każdego iloczynu postaci X_iY_j po prawej stronie stosujemy rozkład według schematu przytoczonego na początku podrozdziału, pamiętając, że każde X_i oraz Y_i jest postaci $(y_{i0}, y_{i1}, y_{i2}, y_{i3})$.

Przykład praktyczny:

Niech dany będzie schemat dzielenia sekretu o czterech udziałach wejściowych i trzech wyjściowych. Wejście zadane jest za pomocą tabeli:

X	X_1	X_2	X_3	X_4	Y	Y_1	Y_2	Y_3	Y_4
1	1	1	0	1	1	1	1	0	1
1	0	1	1	1	0	1	0	1	0
1	1	0	0	0	0	1	1	1	1
0	0	0	1	1	0	0	0	0	0

Tablica 4: Dane wejściowe dla schematu dzielenia sekretu.

Dla schematu podanego na początku i obliczenia iloczynu XY otrzymujemy:

X	Y	XY	F_1	F_2	F_3	$F_1 \oplus F_2 \oplus F_3$
1	1	1	1	1	1	1
1	0	0	1	0	1	0
1	0	0	1	1	0	0
0	0	0	0	1	1	0

Tablica 5: Dane wyjściowe dla przykładu.

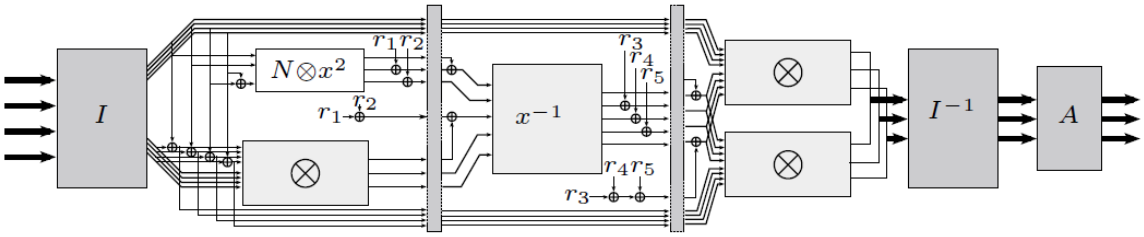
Jak widzimy, iloczyn XY dla zadanego przykładu przyjmuje wartość $F_1 \oplus F_2 \oplus F_3$. Zastosowanie przedstawionego rozwiązania pozwala na ukrycie wartości przetwarzanych w pracy układu.

4.3 Implementacja końcowa

4.3.1 Implementacja nr 6

Bazując na implementacji nr 5, wprowadzono mechanizm dzielenia sekretu. Maskowanie komponentów liniowych algorytmu realizowane było za pomocą zwielokrotnienia ich liczby w układzie, przez co zamiast jednego potoku obliczeń opartego na danych wrażliwych, powstały trzy będące ze sobą w zależności, ale z punktu widzenia atakującego zupełnie losowe. Maskowanie warstwy nieliniowej algorytmu polegało na zastosowaniu schematu dzielenia sekretu dokładnie opisanego

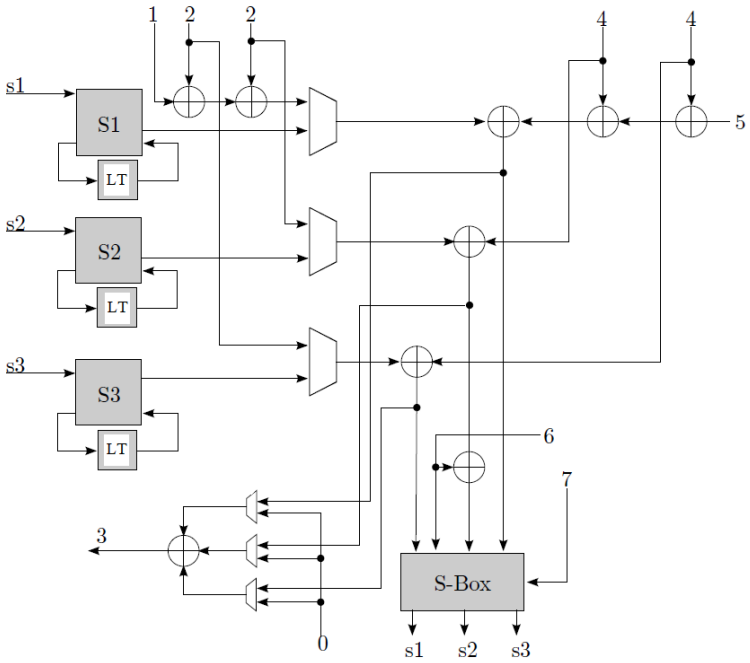
w [6], w czterema udziałami wchodzącymi i trzema wychodzącymi. Budowę skrzynki podstawieniowej wykorzystującej maskowanie przedstawia rysunek:



Rysunek 46: Zamaskowana skrzynka podstawieniowa [6].

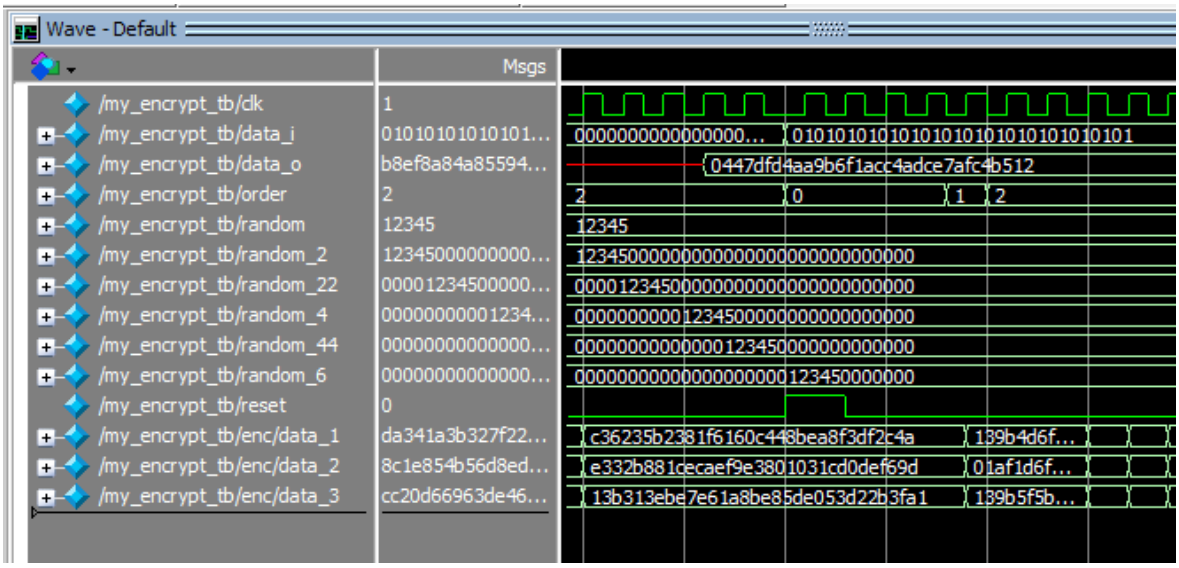
Wprowadzone w Sbox'ie zmienne $r_1, ..., r_5$ są losowymi wartościami wprowadzanymi na wejściu układu w celu zwiększenia złożoności.

Implementacja nr 6 swoje działanie opiera na zmodyfikowanych schemacie pokazanym na rysunku 21, a mianowicie:



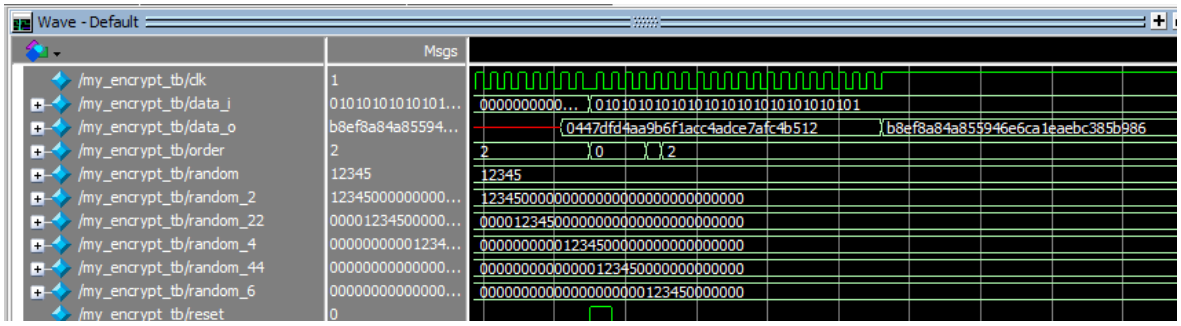
Rysunek 47: Schemat układu dla implementacji nr 6 [6].

Zauważmy, że na powyższym rysunku nr 47, dodane zostały dwa potoki obliczeniowe. Na wejściu nr 5 podawane są klucze rund. Wejście nr 1 to 128-bitowy tekst jawny, do którego za pomocą dwóch randomowych 128-bitowych ciągów dodawane są maski. Klucz rundy maskowany jest za pomocą analogicznych ciągów oznaczonych na rysunku numerem 4. Warstwa nieliniowa zgodnie rysunkiem nr 46 przyjmuje 4 udziały 128-bitowe, przez co w układzie na wejście nr 6 podawane są wartości losowe. Cyfra 7 oznacza 20 bitów dostarczonych do skrzynki podstawieniowej w celu realizacji losowości wcześniej opisanych zmiennych $r_1, ..., r_5$. Wyjście z Sbox'a to trzy udziały, które przechodzą do S_1, S_2, S_3, gdzie realizowane są na nich operacje liniowe (oznaczone na rysunku jako LT), odpowiednie dla zastosowanego szyfru blokowego. Po zakończeniu szyfrowania, wyniki trzech potoków są dodawane modulo 2 i wysyłane na wyjście nr 3.



Rysunek 48: Symulacja implementacji nr 6.

Zauważmy, że każdy z potoków, przedstawionych na rysunku nr 48 za pomocą zmiennych $data_1$, $data_2$, $data_3$, realizuje obliczenia na innych danych, dopiero w końcowej fazie po zsumowaniu otrzymujemy niezaciemniony wynik.



Rysunek 49: Wyniki otrzymane dla implementacji nr 6.

Powyższy rysunek przedstawia działanie implementacji oraz zwracane wyniki, które po porównaniu z rysunkiem nr 44 są identyczne.

Negatywnym aspektem zastosowania schematu dzielenia sekretu, jest zwiększenie zasobów sprzętowych potrzebnych do pracy układu. Analizując rysunek nr 50, możemy zaobserwować prawie czterokrotne zwiększenie wymagań w stosunku do implementacji referencyjnej.

Revision Name	Anubis
Top-level Entity Name	Anubis_2
Family	Cyclone V
Device	5CSEMA6U23C6
Timing Models	Final
Logic utilization (in ALMs)	9,970 / 41,910 (24 %)
Total registers	780
Total pins	2 / 314 (< 1 %)
Total virtual pins	918

Rysunek 50: Wymagane zasoby dla implementacji nr 6.

4.3.2 Implementacja nr 7

Implementacja nr 7 jest ostatnią implementacją wykonaną w celu zebrania w jedną całość wszystkich narzędzi wypracowanych w poprzednich implementacjach. Zastosowano w niej:

1. ukrywanie realizowane za pomocą:

    - dodania potoku obliczeń na zboczu opadającym,
    - umieszczenia dodatkowych rejestrów przechowujących dane zanegowane,

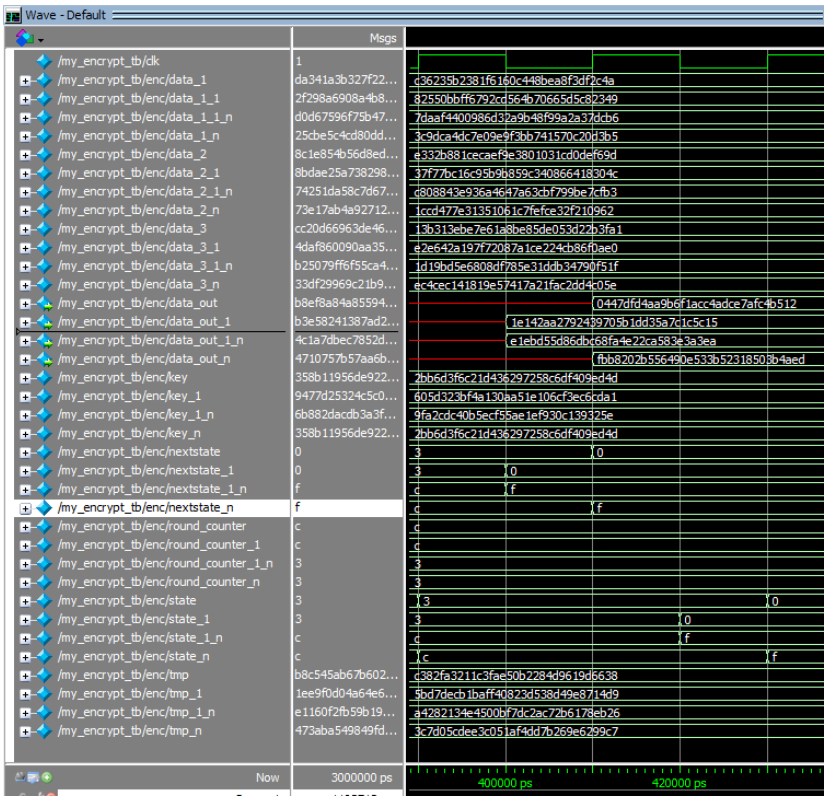
2. maskowania, zrealizowanego za pomocą schematu dzielenia sekretu, na podstawie implementacji nr 6.

Na największą uwagę zasługuje fakt zwiększenia wymagań w stosunku do implementacji referencyjnej. W tym przypadku, liczba niezbędnych komórek logicznych sięga prawie 20 000 jednostek.

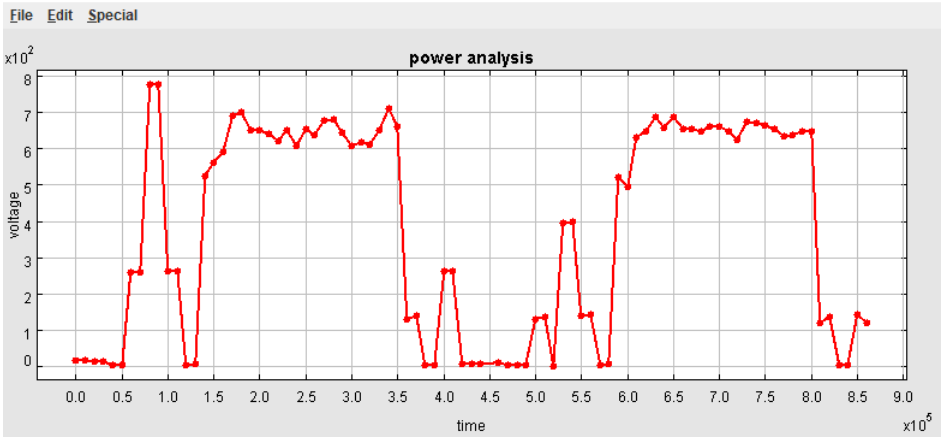
Revision Name	Anubis
Top-level Entity Name	Anubis_2
Family	Cyclone V
Device	5CSEMA6U23C6
Timing Models	Final
Logic utilization (in ALMs)	19,360 / 41,910 (46 %)
Total registers	1680
Total pins	2 / 314 (< 1 %)
Total virtual pins	1,302

Rysunek 51: Zasoby wykorzystywane w implementacji nr 7.

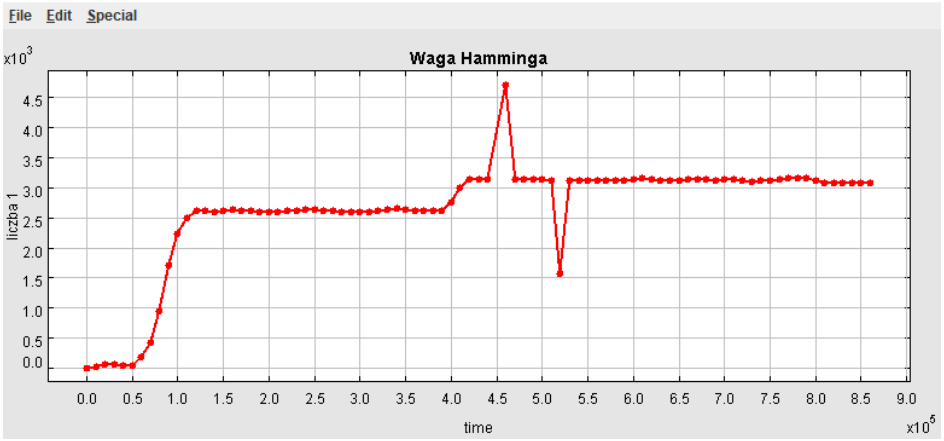
Na rysunku nr 51 widzimy, że liczba rejestrów wzrosła do 1680. Wszystkie one zostały zaprezentowane na rysunku poniżej:



Rysunek 52: Rejestry przetwarzane w implementacji nr 7.



Rysunek 53: Pobór mocy implementacji nr 7.



Rysunek 54: Waga Hamminga dla implementacji nr 7.

Właściwości otrzymane dla implementacji nr 2 i 3 przeniesione są jeden do jednego na implementację nr 7. Pobór mocy jest bilansowany za pomocą dodatkowego potoku obliczeń natomiast Waga Hamminga utrzymuje się na stałym poziomie w związku z zastosowaniem dodatkowych rejestrów. Z otrzymanych charakterystyk wynika, że otrzymano implementację sprzętową odporną na ataki z kanałem bocznym.

4.4 Porównanie i analiza wykonanych implementacji sprzętowych

W niniejszej pracy zaprezentowano 7 implementacji sprzętowych. Pierwsze trzy odnosiły się do algorytmu Anubis w postaci zaprezentowanej w dokumentacji oraz z zastosowaniem środków ochrony przed podstawowymi atakami z kanałem bocznym. Kolejne cztery operowały na zmienionej skrzynce podstawieniowej w celu wykorzystania schematu dzielenia sekretu do wykonania maskowania. Implementacja nr 4 miała taką samą budowę jak implementacja referencyjna z innymi wartościami dla skrzynki podstawieniowej, danej za pomocą *lookup table*. Implementacja 5 poszła o krok dalej, skrzynka podstawieniowa nie wykorzystywała sztywnego przypisania wartości, ale na bieżąco wykonywała obliczenia odwrotności w ciele $GF(2^8)$, za pomocą przekształceń na ciałach złożonych. Implementacja nr 6, wykorzystywała implementację wcześniejszą w celu zastosowania podziału sekretu. W ostatniej

implementacji, do zamaskowanych danych doszły metody ochrony zastosowane w implementacji nr 2 i 3.

W tabelach przedstawionych poniżej, implementacje od 1 do 7 przedstawiono w postaci dwóch grup:

- odnoszące się do algorytmu Anubis (I, II, III),
- implementacje szyfru Anubis z wykonaną modyfikacją (polegającą na zastąpieniu skrzynki podstawieniowej na tą dostępną w AES) (IV, V, VI, VII).

	Implementacja I	Implementacja II	Implementacja III
$85°C$	72,31 MHz	70,72 MHz	72,35 MHz
$0°C$	73,28 MHz	71,42 MHz	73,41 MHz

Tablica 6: Maksymalne częstotliwości taktowania zegara dla implementacji z pierwszej grupy.

	Implementacja I	Implementacja II	Implementacja III
ALM	2652	5127	5472
reg	524	1048	1300

Tablica 7: Zajętość zasobów dla poszczególnych implementacji z pierwszej grupy.

	Implementacja IV	Implementacja V	Implementacja VI	Implementacja VII
$85°C$	73,36 MHz	55,32 MHz	56,48 MHz	25,59 MHz
$0°C$	74,16 MHz	56,01 MHz	57,2 MHz	25,98 MHz

Tablica 8: Maksymalne częstotliwości taktowania zegara dla implementacji z drugiej grupy.

	Implementacja IV	Implementacja V	Implementacja VI	Implementacja VII
ALM	2624	4786	9970	19360
reg	524	524	780	1680

Tablica 9: Zajętość zasobów dla poszczególnych implementacji z drugiej grupy.

Jak widzimy, najwięcej zasobów zabiera ukrywanie informacji za pomocą generowania drugiego potoku obliczeń dla zbocza opadającego (prawie dwa razy więcej ALM w stosunku do implementacji referencyjnej). W przypadku dokonywania modyfikacji w algorytmie, zmiana skrzynki podstawieniowej oraz jej implementacja w ciałach złożonych także podwoiła liczbę potrzebnych zasobów. Stosunkowo opłacalnym jest wykorzystanie rejestrów przechowujących dane zanegowane, ponieważ dwukrotny wzrost liczby rejestrów (implementacja nr 1 i 2) osiągnięty jest przy małym zwiększeniu zapotrzebowania na komórki logiczne (rzędu 300 ALM).

Charakterystyki częstotliwościowe wskazują, że najsłabszym ogniwem jest zamiana skrzynki na tę, która opiera się na działaniach na ciałach skończonych. Mimo tego, wykonywane operacje mają stały pobór mocy, niezależny od danych wejściowych, przez co atakujący nie jest w stanie znaleźć korelacji pomiędzy przetwarzanymi danymi. Widzimy również, że im większa zajętość układu, tym częstotliwość pracy urządzenia jest niższa.

Podsumowanie

Bibliografia

[1] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. *Kryptografia stosowana*. Wydawnictwo Naukowo - Techniczne, 2005.

[2] Begül Bilgin. *Threshold Implementations As Countermeasure Against Higher-Order Differential Power Analysis*. KU LEUVEN, 2015.

[3] Benedikt Gierlichs. *Introduction to Power Analysis*. Katholieke Universiteit Leuven – COSIC, 2011.

[4] Federal Information Processing Standards Publication 197. *ADVANCED ENCRYPTION STANDARD (AES)*. 2001.

[5] Francois-Xavier Standaert, Loıc van Oldeneel tot Oldenzeel, David Samyde,and Jean- Jacques Quisquater. *Power Analysis of FPGAs: How Practical Is the Attack*. Universit´e Catholique de Louvain, 2003.

[6] Henrik Fegran. *DPA-Resistant ASIC implementation of AES*. Norwegian University of Science and Technology, Department of Electronics and Telecommunications, 2015.

[7] J. Daemen. *Cipher and hash function design strategies based on linear and differential cryptanalysis, Doctoral Dissertation*. K.U.Leuven, 1995.

[8] M. Masoumi. *A DPA Resistant FPGA Implementation of AES Cryptosystem with Very Low Hardware Overhead*. Iranian Journal of Electrical and Electronic Engineering, Vol. 8, No. 1, 2012.

[9] Maciej Nikodem. *Rozprawa Doktorska - Metody ochrony przed kryptoanalizą z uszkodzeniami.* Wydział Elektroniki Politechnika Wrocławska, 2008.

[10] Paul Kocher. *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other Systems.* in Advances in Cryptology (CRYPTO '96). Lecture Notes in Computer Science, Vol. 1109, pp. 104-113., 1996.

[11] Paul Kocher, Joshua Jaffe, Benjamin Jun. *Differential Power Analysis.* LNCS, vol.1666, 1999.

[12] Paulo S.L.M. Barreto, Vincent Rijmen. *The ANUBIS Block Cipher.* 2001.

[13] Philip Hodgers (QUB), Francesco Regazzoni (USI), Richard Gilmore (QUB), Ciara Moore (QUB), Tobias Oder (RUB). *State-of-the-Art in Physical Side-Channel.* SAFEcrypto, 2016.

[14] Shivam Bhasin. *Logic-Level Countermeasures to Secure FPGA based Designs.* Télécom ParisTech, 2011.

[15] Xinmiao Zhang. *High-Speed VLSI Architectures for the AES Algorithm.* Case Western Reserve University.

[16] Łukasz Dzieł. *Atak poboru mocy na niektóre implementacje sprzętowe pewnej klasy algorytmów kryptograficznych.* Laboratorium Badawcze Kryptologii, Instytut Matematyki i Kryptologii, Wydział Cybernetyki, Wojskowa Akademia Techniczna, 2010.

[17] Dawid Oswald, Ruhr-Uni Bochum. Id and ip theft with side - channel attacks, 2014. URL `https://www.slideshare.net/phdays/1300-david-oswald-id-and-ip-theft-with-sidechannel-attacks`. (stan na 13.03.2019).

[18] Ella Rose. Substitution-permutation network visualizer, 2017. URL `https://crypto.stackexchange.com/questions/45300/substitution-permutation-network-visualizer%7D,`. (stan na 10.03.2019).

[19] Julio Hernandez-Castro. Illustration of a round in a feistel network, 2006. URL `https://www.researchgate.net/figure/Illustration-of-a-round-in-a-Feistel-network_fig1_224645711`. (stan na 10.03.2019).

[20] Paulo S. L. M. Barret. Differences between rijndael and anubis, 2008. URL `https://web.archive.org/web/20160606112246/http://www.larc.usp.br/~pbarreto/AnubisPage.html`. (stan na 11.03.2019).

Spis tablic

Spis rysunków